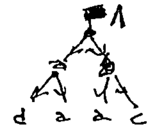
3.29.04
4AM

10 In || Lsrch, how is it clear (?). Well, drawing the ~~[illegible]~~ construction ~~tree of~~ the conds:



At (down) each nodes we have an assoc. pc., we also have a ~~[illegible]~~ cc needed to ~~execute~~ Execute that node. So at each pt, if it take ≈ ϕ time to compute pc, we can move along each branch. If the total pc after a ~~[illegible]~~ node that was just executed says more cc is left, we branch the node & work on the branches, until the (cc expended on that branch) > (pc of that branch) × total time spent thus far × ~~[illegible]~~ f (perhaps normaliz const ... may be unnecy, hvr) ← Did I (mistakenly) criticize OOPS on this point.

∅6 In random Lsrch we "cc ≡ ϕ" to calculate pc's: ~~Calculate a pc [illegible] for completed cond, w/ pc ∝~~. Jump to a cond w.p. ∝ pc of that completed cond || & work on that cond, for (fixed, universal) time Δ. This work will be shared betw the conds, if they have common prefixes. The idea is that we work on the cond for time Δ — taking advantage of any time spent on that cond thru ~~[illegible]~~ "cc sharing".

0

1(Somehow, we end up w. a factor of D, the length of cond ~~in~~ Dern..

Is the || Lsrch thru this (?) that the speedup over T≤2T Lsrch is by/ factor of ≈ (D of the soln cond),

If I do .∅6 — || for Random Lsrch, will I get the same "D" factor ~~[illegible]~~ of improvement over T≤2T Lsrch?

If the branching factor per node is k, than the total cc of all of the conds is

$$A(1+\frac{1}{k}+\frac{1}{k^2}) = \frac{A}{1-\frac{1}{k}} \quad (= 2A \text{ if } k=2) ; = \frac{4}{3} \text{ if } k=4$$

$= \frac{k}{k-1}\cdot A = \frac{k}{9}$ for $k=10$. So for reasonable k, we save ≈ D · 2 over T≤2T Lsrch

Perhaps the way OOPS does Lsrch, is it currently a factor of ~2 ~~[illegible]~~ over || Lsrch,

-0 And — he may have been doing || Lsrch correctly! — The No, he didn't save partial No!

21 runs on conds → A BRANCH (102.20)

Maybe not so time consuming! → (102.26)

Q's: 1) How to ~~save~~ store states in || & Random Lsrch. Sounds Expensive, Time Consuming; So spend ~~[illegible]~~ a Much time between storages.

2) In CFG discovery: I will initially ~~[illegible]~~ "discover" a Pos (or nonset) by noticing/certain class of words tends to follow (precede) a certain work. How do I define this class? — perhaps by simply telling how it was discovered! It is well defined! ✱ ~~The~~ undo/Res data is cheap, — to implement it, one may need some of its most frequent Members

→ Anyway, say we get this factor of ~20 in speed. Using || Lsrch:

One bad side is ↑ in Memory Cost: If we use T≤2T Lsrch, the ~~[illegible]~~ RAM ~~needed~~ is quite small (usually) —

→ (i.e. conds can be Monte Carlo Generated.)

6/24/04: I later found that if done right, T≤2T need not be slower by a factor of code Length (= Code "depth") So || srch would only ~~[illegible]~~ be faster by a factor of Maybe 2. Random Lsrch ~~[illegible]~~ has other desirable features, hvr.

●0 So we can just buy a lot of cpu's and use their L2's (secy cache). Perhaps ~~[illegible]~~ 256k, but Lsrch usually needs ~~& that~~

Hvr, if one has the Ram available, or if one can suitable implement: disc swapping — it might be

O.K. A fast disc can write ~~[illegible]~~ 150 Mby/sec. If each trial is 10 tokens long & each token takes 1000 clocks, and it takes ~~[illegible]~~ 20 by to store state of machine — so ≈ 10^{10} different sequences. — Token ≈ 1 byte 10 main tokens.

20 by / cond $20\times10^{10} = 2\times10^{11}$ bytes. = 200 Gbytes.

1 trial will take, say 10^4 clocks: = 10μs for a 1 GHz CPU. During this time 10μs ~~[illegible]~~ disc can write 1500 bytes — or 150 cond dumps.

1-

4TM

30 : 98.40 : A simple way of coding, using ~~L_t~~ L_t) : ~~L_t~~ L_t) is the L that we have after we know the t chars of original corpus. distance that predicts correctly

~~Using~~ C_t is the corpus up to time t. L_t is the sorted order of C_t. Knowing C_t and L_t, code the next symbol in terms of the context in L_t. We can use the min of distance ~~#~~ nearest context or the mean of the 2 contexts, or a Kernel weighting of possl. contins. ← width of kernel is act. Each method gives a pc. for next symbol. — We code it using Arithm Huffman. — This gives C_{t+1} and, inserting that symbol into L_t, we get

See 103.05 for a more exact defn. of the Kernel

.05 ~~&~~ L_{t+1} — from these 2, we code the next symbol, etc.

.06 So I can try any of these methods for predn.: ~~#~~ PPM & PPM* are modifns. of them — that recent Russian method is also probably a simple modifn. Another possibly "Big Deal" is

.09 Understanding & applying the Dirichlet to DF. PDF : Could Joy ~~may~~ discuss it. Marcus Got an interesting

.10 result.! Also the empirical result of (PhD thesis; Girl Suzan ~~[illegible]~~... Bunton)

.11 OF the effects of interest, the DDF seems largest: A factor of 2 reduction in wt. of the corpus.

While .06–.11 are certainly useful in improving BWT compressn, It's likely that ~~[illegible]~~ higher order grammars will be much more helpful:

SN Bunton says ~~[illegible]~~ her "semantically motivated Improvements to PPM attribute" is mainly excerpted from her thesis. Also, I think the thesis goes more into "Typing" the corpus words — to get hi compression.

1) Write up various Phase 1, ~~[illegible]~~ 1½, 2 models: Then discuss improvements,

2) Some immediate concerns: (a) Partly try policy for partial execution v.s. No trials until

.0 cand. is completed : ~~[illegible]~~ Latter can be faster by factor of (~~length~~ "depth of soln" no. nodes; no. ~~[illegible]~~) for || or random realizn. of TM.

(b) Just how to get some feedback from ~~previous~~ previous cands, available as soon as possl. to modify pc's of new cands. || Lsrch seems to make this ~~[illegible]~~ difft.

How important is this "quick f.B."? It would seem that in certain kinds of problems, this rate of f.B. could be a major factor in Prob. ~~[illegible]~~ Solving speed. The idea of each trial giving info used in next trial.

27 : 96.40 If time to compute pc << evaln. time: For || Lsrch & Random Lsrch, Recompute the pc. of each trial, each time:

Say we have insts ~~[illegible]~~ "a r a m c ..." as part of a ~~[illegible]~~ cand Trial, & we have completed ~~[illegible]~~ the last inst. "c".

0 ~~#~~ Next time around, we compute pcs of "a r a m c" & various ~~[illegible]~~ possl. token contins.

We then spend an amt. of time on each contin = ~~#~~ (the pc of the total trial up to that pt.) × T × const

T is ~~[illegible]~~ total time spent on problem thus far. If this Δ time is < 0, we don't work on that contin at that time. 96.34–.40 does this. : See 100.00 for more detail.

For Random Lsrch (w. ~~[illegible]~~ ~~[illegible]~~ computation compn. of pc's much faster than time to evaluate)

Recompute pc's after every 10 "successful" trials. (Also true for || Lsrch)

For each complete pc assigned to a cand, we Monte ~~[illegible]~~ Carlo wise ~~[illegible]~~ go to that cand w. its pc, & spend ~~[illegible]~~ fixed time, Δ, on it. Can we get sharing ~~sharing~~ of ~~[illegible]~~ cc on parts (prefixes) of cands ~~[illegible]~~, this way? The meaning of the "cc" ~~#~~ of a cand is unclear, if the cc is partly shared,"

3.26.04

4TM

Extends to loop

0:97.40 : In present 2-3-T pgm we've written, +∞ (≡255) is o.k. — inserting it causes no trouble. How, Inserting "ϕ" (≡−∞) does cause trouble. It may be fixable by using "≥" instead of ">" in the comparison function. Otherwise, it seems able to deal w. cands that are identical to cands in the ~~[illegible]~~ "L" corpus.

So set δ (i.e. Δ and δ) to ϕ, , δ to 255. See if our 2-3-tree can deal w. this.

If we generate cands ~~[illegible]~~ left to right (97.11) then we don't need the δ symbol into "L" corpus; ~~[illegible]~~ δ only occurs ~~[illegible]~~ as a predicted symbol.

So if we are troubled by "0" we can use "≤" or "≥" as our ordering ~~[illegible]~~ inequality in t. 2-3-Tree. — So "0" is o.k. but "255" causes trouble. (255 will never be inserted into "L".)

12 Say we use ϕ as Δ (≡δ): T. cand trial always starts w. ϕ: When we take ϕ as t. predicted symbol, t. cand ~~[illegible]~~ code terminates. I think they did this sort of thing w. the shifts of a ~~corpus to~~ normal textual corpus to be coded. I.e. only one termination/start symbol. Also note 97.14–.24

So perhaps t. method of insertion of cand. code data into L is "under control".

Assoc. w. each insertion we can have 1) What came next empirically: 2) Score of assoc cand. sequence! 3) "Date" of insertion (perhaps): This is t. order no. of t. cand. — is perhaps its address.

.19 We may want to delete very old data (?).

0 T. first BW paper took 400 clocks ~~[illegible]~~ per key inserted (or 400 clocks to "compress" one symbol ~~of~~ corpus. For cands 100 bytes long and 1000 cands in corpus 10⁵ × 400 = 4 × 10⁷ clocks — which is ≈ 1 second ~~[illegible]~~ for a 1 ns. clock. or 4 × 10⁷ × 10⁻⁹/4 for a 4 GHz clock. ≈ 10 ms for a 4 GHz clock. This is to ~~revise t. entir~~ re ~~[illegible]~~ sort t. entire corpus: — seems like a short time! This is 400 clocks per Token of t. cand. OOPS took 1000 or (10000?) clocks to execute 1 instruction — so generating a good "OOPS" trial should take much less time than t. trial executing t. ~~[illegible]~~ Unless, of course, t. trial has loops/ recursions, etc. which makes for very slow execution

25 A possl. diffy w. large analysis: It assumes that I create a cand, then test it — that I do not ~~test~~ execute insts as I go along (The 97.14 — .24 does discuss how to do this, it seems like a very incomplete discussn.). See 99.27 for good approach to ~~[illegible]~~ problem of ∥ testing. I'm not sure 99.27 is relevant to .25!

What I don't have a clear picture of is how to run t. system so that cands get relatively recently updated pc's for their "Tokens".

30 OOPS has facilitys for defining macros, or, I guess, recursive pgms. It doesn't seem to have facilities for making definitions — so this BWT ~~[illegible]~~ could help it a lot.

What needs to be done:

1) I may want to study OOPS again! Its use of t. Forth lineology. & its ability to ask for more Tokens could simplify t. system a lot. One of my complaints about it was that it didn't seem that ~~[illegible]~~ definition of subprograms could ~~[illegible]~~ boost its performance much; I don't remember just what my argt. was. look in t. latest "NIPS report": Comments on OOPS appear 2 or 3 places. (maybe just 2) Use Winedit to find OOPS in it.

40

2·26·04
3
4TM

.0 I'm really Concerned w. 2 parts of Phase 1 : T. first part is pretty much ≈Z141-like, using BW xfon for pcls. Ideally, They vary during each srch. & I may or may not make Improvements ("Refinements") on BW.

The second part ~~[illegible]~~ defines words, Then Ngmsts — which augments t. prediction accuracy of BW. finally, ~~[illegible]~~ we devise a grammar for ngmsts — CFG or CSG or any other kinds of Grammar rules.

A TM in this second ~~phase~~ part of Phase 1 will certainly be ready for Phase 2 — (if not already in Phase 2!).

.0 (94:37)R: If I use Polish notation, Grow cands ~~[illegible]~~ Rt to Left, Contexts are normal Lex order (not reversed).

#1 If RPN is used Grow cands Left to Rt, Contexts are reversed Lex order (as usual BW).

If I do not obey these rules: try it out & see if predn. is ~~better or worse~~ worse or same!

Q: How to put relatively short pgms into "rotated corpus".

4 ~~[illegible]~~ Say Δ & end δ is a ~~[illegible]~~ cand to be inserted. (δ Δ) is a null symbol at (rt end / left end)

→ What is lex order of Δ, δ? Do I need both?

If I do .11 (~~RPN~~ RPN) I start pgm. w. Δ, work along till I get to δ. Then stop. [N.B. in .11, I would ~~[illegible]~~ be able to execute ~~[illegible]~~ pgms as they were generated.   Δ 3 7 sum

Similarly for .10 (Polish).   ~~[illegible]~~ sum 3, 7 Δ

.0 If I do sum 3,7 Δ I would not be able to execute until Δ (~~[illegible]~~ end) occurred.

to go sum 3, mul 4,3. going ~~[illegible]~~ rt to left (←) I can execute mul (4,3) then t. entire expression, But going → left to rt. I can't execute until end.

.4 Hvr, as far as ~~[illegible]~~ predn of next symbol is concerned, one can go either forward or backward (.35)

.26 91.07 90.21 SN On second thot, that idea about using both forward & backward prediction on "L" is prob'y wrong! ← prob'y wrong, ~~too~~. We can use one or t. other, but not both. : On ~~second~~ third thot maybe using Both is o.k. "L" is not sequential info: It represents t. past up to now. PPM uses both directions of info on L for prediction (i.e. contexts, "both sides" (lexically) of t. context to be predicted. So using Both can be O.K.

.0 Try it out empirically: Using both should be ~~[illegible]~~ slightly better than using only "up" or only "down". See 99.00-.05 for why forward & Backward are O.k.

Perhaps that's a big reason for PPM's small (tho real) superiority over BWT.

The BWT is comparable to PPM*: PPM* normally (tho there may be a recent low cc version)

.34 uses much more ~~[illegible]~~ than BWT. .342 NB I considered taking t. mean of up & downstream predictors — a better way mite be to pick for each token t. closest of up or downstream

35: .24 I guess for Δ, δ — one symbol would be adequate. Take t. code of a cand. : put Δ on one end & do all rotations: Insert all rotations (including Δ symbol), into Lex sorted corpus.

Start at new cand code w. "Δ", end it w. "Δ".

Hvr., ~~[illegible]~~ when using Δ as a start symbol, we don't want to use contexts to t. Left of Δ ~~error~~.

This could be troublesome: If we use both Δ & δ, Δ can be −∞ & δ can be +∞.

(Spec 94.34): ~~Testing~~ takes almost all of cc. In this case, as t. pc's for cands change in a Mt. Carlo Lsrch, we will spend differing fractions of time on a cand. 'Consider say, "Generating cand takes cc=0". As pc's of cands change, we will spend varying amts of ~~[illegible]~~ cc on them. Superficially, this would seem to be O.K. — if a cand seems (v. Bad / v.g.) we would usually spend (little / much) cc on it.

Hvr, if "generation of ~~generating~~ cand. has cc=0" then we don't have factor of depth of cand" in superiority of || v.s. T≠2T Lsrch. In this "depth of cand" idea, I was Rembring that cands would execute during their generation — This would enable many cands to share part of their execution expenses (i.e. t. parts of the cands that were held in common).

A possl. Criticism of T≠2T: Suppose at a certain T level, cands begin to be "completed" i.e. complete execution — so we get Goves for them. These Goves will not affect t. pc's of cands until t. "next round". The round after that takes twice as much cc, so we really have "hyly delayed feedback" | Ideally, t. modn of a pc due to a very good or very bad Gove. should be made available as soon as possl. || or Random Lsrch seems to do this much better!

14 .... How to ~~Generate~~ Generate Random Cands using the BW xfm: Start w. a random symbol based on null initial context. ~~Then~~ We jump to this Lex point and get pd for next symbol, by randomly choosing to ~~[illegible]~~ move in up or down direction: One moves up until a new symbol occurs, then uses that symbol w. Pescape and goes on to ~~[illegible]~~ Lex order corpus to chose next symbol. w. pc = (1 − Pescape) one continues to t. ~~next~~ "new symbol" and chooses it w pc = "Pescape", etc. Pescape can be some constant or we may find a good way to vary it as we do successive "escapes". We will end up w. a terminating symbol — which commonly occurs in our corpus of short seqs of symbols (≡ cands). The "escape" probty will be modified by the Gove of t. particular cand. involved. [T. gore of a cand will be t. same for all of its rotations]

It could be possl. to generate random symbols and execute each as soon as they are obtained. When a new symbol is represented by a pgm, it's obtained ~~[illegible]~~ Mt.[redacted]Carlo wise - 1.4a. 19ff.

What was t. search routine used in "OOPS"? If used T≠2T. Generated tokens and executed them, then got another token ~~of~~ and when pgm asked for it. It would (I think) use total allowable execution time ∝ pc of that cand. Thus far.

It would be best to do || srch (or Random srch) until Pd begins to change. This doesn't occur until we ~~begin~~ begin to get useful results. T. "T" in || srch ~~[illegible]~~ is at a level where our "final soln" will be better ≥ T and 2T (probly).

see 97.27 following relevant discussion.

.34 Ideally, t. || method would work so: We keep in ~~[illegible]~~ RAM, t. state of t. system for as far as each trial has Gone. ~~We generate~~ For each ~~new~~ change of t. pd, we recalculate the pc's of cands, in the pc order, and we workout t. parts that need Work. We have to do t. ~~[illegible]~~ cand generation and part finding, in "together". — See 97.27

This is about what 97.27 does.

7·25·04
3
4th

<UNIFICATIONS>

: A Listing of ideas & Refs to ~~ideas~~ on Unifications in Prob. Solving techniques:

1) CORDIC (sp?) : A way to calculate trig, log, mult, div, ext funcs (& perhaps others) that is simple & fast. Perhaps can be extended to other funcs like Gamma funct, Beta funct, etc. Look up on Google. I have a book that explains much of it. Refs to Matrix triangulation, SVD.

2) "EM theory" ← Estimation Maxim. Many Max likelihood probs have many useful features in common.
A. Dempster, N. Laird, D. Rubin: "Maximum Likelihood from Incomplete data via the EM Algm" Journ Roy Stat Soc, B vol 39, 1977
See IEEE IT newsletter Dec 2005 p 13 for preliminary discussion.
Also IEEE IT Trans: Sept ~~1982~~ 1982 : L. Poraco : Max Like Ests for Multivariate observations, Markov Sources

3) Continuous ⟷ Discrete vars: In Rvms, ~~[illegible]~~ PSM's, Unitary xforms.

4) See my lists of PSM's & Optzn Methods: Attempts to unify in groups (Rains, Taylor: ·

5) Neural Nets, Fuzzy logic, ~~[illegible]~~ Radial Basis Functions .... Thomson ....
Also Units of ANN, high dimensional ... By A. Barron, 1993. ~~The~~ Paper by .. T. Poggio

6) GA, GP, Simulated Annealing ... [Monte Carlo Methods?]

7) Index theory in Math: { Michael Atiyah, Isadore Singer. } ← founded by. Abel Award.

3.2.24
4TM

33.8 At ~ ... lston
≈ 32×(1+.05).

00: 93.40 ; In ~~the~~ T-2T Lsrch, ~~the~~ T. dif'ty of 93.37 may not occur; We do a complete enumeration & testing of a trial before we start on a new one, when T gets large enuf. At such a time, t PD will begin to change rapidly — but these changes will be able to be (taken advantage of / utilized) subsequent cand trials — we will be in real "learning during Lsrch" mode!

—— But note that this is not the same as the "Incremental Lrng" that is done by sequences of problems.

So, it looks like there are really 2 quite difnt kinds of Lrng. taking place: ① During Lsrch problem soln ② Betw. problems.

Tho actually, I'm really confusing things! : In GA Mode, there is only 1 problem. Hvr, t seq's ~~can be regarded~~ for QATM can be regarded as a slowly changing GA problem. In normal GA, t problem changes as t population moves toward by Gorc. The GA is able to deal w. this "Change of problem": In a ~ way, GA should be able to deal w. t problem of finding a single optimum soln for $Q.A]_1^n$ as n "slowly" increases.

Hvr, we do want TM to start each trial from t. beginning, after each ΔGORC a/o Δn. So it will be able to take advantage of any updating of t. "Corpus".

However, even if we are doing T-2T Lsrch, at each point in time, certain cands will have been ~~tested~~ "tested" or "Timed out". Some of the "Timed out" cands would have been given more pc due to a Pd change after they "timed out". So we have to wait for t. next T-2T update before that is tried! — The completion of such "formerly timed out" cands will then occur early after the T-2T.

One Conclusion: that || Lsrch isn't good (normally) for updating Pd during Lsrch! The early symbols will use t. early Pd, & never changes. — Unless we recompute t pc's of various sequences. — I don't immediately see how to do that!

Perhaps draw up various poss. systems & see how reasonable they are!

Say start w. ~~immediate~~ continuous immediate update of corpus, with Gorc info used for wts.

★ Consider a simple growing Corpus w. no deletions of poor cands, but Gorc weighting.

30 See how it works w. T-2T. Also look at || Lsrch to some extent, since it is of value for random Lsrch & is, in general economical by at least a factor of 2 & perhaps by a factor of "t. depth of t. soln". — T. main dif'ty is that w. each symbol of a cand being assigned a pc, we have no way to "go back" & change earlier pc's in view of t. changed

54 → "Guiding Pd". Hvr, suppose that creating t. cand takes little cc (as I expect) & then 96.00

Another thing I haven't decided on is whether to generate the cands forward or backward — i.e. t cand is represented by a ~~finite~~ seq. of symbols. One can start at t. front & generate forward, or at t. end & generate backwards. — Difnt kinds of regys are found in corpus for t. 2 methods .32R 97.10

37 SN T. "Escape" mechanism may be a good way to deal w. t potentially infinite Corpus. — to assign pc's to elements of t. ① Does it amount to a certain specific D.f. on any ∞ of elements of t. lang.? ② Can t. escape mechanism (or variants of it) be used to cope w. other d.f.s on infinite sets — like sprid. on t. integers?

— 96.00

4TM

00: 91.40 spec; from t. QA corpus I'm thinking about — or t. other types of corpus in "Phase 1". In English, there are many words & ~~[illegible]~~ t. info in t. 2 word "digram" freqs requires an enormous corpus. In t. ~~made~~ predicting Models In Phase 1 there will not be so many operators/prim. functs which correspond to "words" in English (T. correspondence isn't perfect, however — English words do have some observable internal str., primitive operators (As ~~presented to~~ used by TM, will not.

On (Generating) Testing Cands: In GA, I had t. idea of removing Bad cands when they were in a too low percentile of t. Gore: ~~[illegible]~~ ~~[illegible]~~ Actually, this may not be so easy. If a cands of length N symbols, one must remove N bogs (all N ~~[illegible]~~ Rotations of t. cand.). Well, O.K.! ~~[illegible]~~ it takes about t. same sort of time as in sorting t. cand into t. first place. — So we could remove them.

12 Another way would be to keep all of t. data, but weight it on basis of Gore. Each prediction would have a wt. that's some monotonic function of t. Gore. ~~It would be best,~~ for

So no rejection of cands is ~~necy~~ nec.

To save space in .12 we might allow corpus to grow to size 'S', then take top $\frac{5}{10}$ ~~[illegible]~~ cands/say & reorder them.

A faster, less ran method to do much of this. Do batches of K cands, & keep corpus size at K. Say our 1st corpus size of K/exhausted cand. Sort them in Batch mode (B&W original paper tells cheap way to do this). Order the cands numerically in context order. This enables us to do a Binary search (log n time) to find "insertion point" for any ~~[illegible]~~ context ... so we can construct trial cands. Construct a first K cands. Discard all trials worse than worst of previous corpus. Sort original corpus w.r.t. Gore; This gives r, new "not bad" cands. remove the "worst r cands from old corpus. Insert the new r cands into corpus. This isn't t. "exactly best way", but is fairly good. Removing bottom r cands. takes as much time as inserting records. (This is not "batch processing") A better way would be to take t. r new cands & t. K−r original cands in t. corpus, & ~~[illegible]~~ Batch ~~[illegible]~~ sort ~~Sort~~ them.

27 On t. other hand, if we have insertion capability, (like 2-3-tree) we can look at each cand's Gore. If its ~~better~~ than worst in corpus, remove worst in corpus & insert new cand; else discard t. cand.

Since we don't have to be very careful about being sure all t. low Gore cands ~~[illegible]~~ have been cut out of t. corpus, we may be able to find good approx. ways to do .27

A serious diffy: The Pd will change ~~rather~~ continuously or abruptly (both updates of corpus & its evalns). So real Lsrch couldn't be done. I did write about how to do Lsrch when t. "guiding Pd" is changing: Its not so easy to do! A possible way: do T←2T Lsrch and change Pd's betw. T changes. As T gets large, This update will become very infrequent. A possl. way to deal w. it — do total Pd revision on T←2T boundaries, but during "a T" update Pd continuously or much more rapid that T←2T ~~[illegible]~~ points.

.37 This can be rather bad, hvr; All of the cands will have "Bad Beginnings" just after T←2T occures. ~~&~~ The Pd will improve during t. "T" cycle — so PC's are much better at t. end of a cycle — So V.G. at t. beginning of t. next cycle! → 94.00 spec

9·23·04

4TM

781-646-3703. Tom Ward.

30 191.40 [SN] BW uses "recency order" to assign pc's to symbols. Alternatively (or as a Mixture) consider "distance in time" of most recent occurrence of a symbol. Instead of "T no" BW uses "T: no. of new symbols introduced betw. 2 time pts" as a metric — This is a reasonable way to estimate "likelihood of a new symbol".

[SN] In using BW for constructing trial Cands (or for constructing Nonsense English), it's not nec'y to update the system after each char is created. In 2·3·Tree this would save lots of time. Many words would still be quite LARGE. B & W used 6 bytes per char. (did this include the character? — I think not ... but check), I used 20 or so.

Hvr, I need more than just "L" (the BW xfm (≡ BWT) of the corpus; I also need to be able to

09 tell where to insert a new trial string. Having an ~~[illegible]~~ "index", telling where the i-th ~~[illegible]~~ lexic string was,

> would make it possl to ~~[illegible]~~ find index of ~~[illegible]~~ trial strings by fast binary ~~[illegible]~~ comparisons.

11 Such an index table, for each corpus symbol, would need 2 bytes for 64k & 3 bytes for 16M corpus sizes. → (.24)

It may be that existing pgms for the "BW transform" can be adapted to give an index table output as well as just L. Or the fast Decompress pgm may be adapted more easily for this.

F: 91.36 [SN] Re: large Corpi: We could divide up corpi on basis of the first 3 or 4 symbols of the context. Can first do random sort of small corpus to find ~~[illegible]~~ contexts that divide ~~[illegible]~~ the distribution of contexts into, say 20 equal parts, Then go thru the corpus ~~[illegible]~~ 20 times — each time looking for a particular range of contexts — The contexts for each range can be sorted by 2-3 Tree or any other method able to deal w. random order of keys.

Seems that foregoing would give time ∝ corpus length, to sort it (!?).

> Say divide original corpus into 16 sections & with 15 "boundary keys" that define the edges of the sections. When a string comes in, we do $\log_2 16 = 4$ comparisons to find which section it belongs to. Then we sort each section individually. ~~[illegible]~~ These sections can be used with corresponding parts of "L" for prediction & cand. construction.

-4: (.11) → In practice, practically any "batch process" Sorter can give us an (.09) index to the corpus — — rather rapidly. ~~[illegible]~~ This is certainly adequate ~~[illegible]~~ We can easily get "L" from that sort/index. — & use it to create trial cands.

We will want a list of cands in ≈ Good order, so we can keep the best 1000, say, in the list that creates new trial cands. So we do need a way to remove arby ~~[illegible]~~ keys from list. Also ~~[illegible]~~ a rough ~~[illegible]~~ index of the G values (say

> 10 or 20 diff'nt G values, or a list that grows upward — but not by random.

We use this to keep the "top 1000" on the "Action list".

→ A very impt. problem! To obtain an inst ~~set~~/set of operators, ∃ concat of the cands to be useful. Work on e.g. "If α then β else γ" as a 2 or 3 arg't "function" or switch. How best to express it so sequential things will be meaningful? : F(α, β, γ).

3

3.22.04

4TM -

20 | 90.40 More exactly, ~~[illegible]~~ if x_0 ~~of~~ occurs at point ϕ & doesn't occur again until pt. n, we want to know t. total variety of symbol types occurring betw ϕ & n.

A not bad way to do this: As we encounter symbols, sort them in Lex order, and note whether t. symbol is new to the sorted list. Use a small Btree. Move along until all of the alphabet of interest has occurred. In Phase 1 induction using a 'AZ141 type "function" constructor, it shouldn't take too long for all of t. symbols to occur. They may be ~~[illegible]~~ or (ppm ~~language~~) units, but we don't want very many of them.

probably wrong! — see 97.26 also note 97.34R

.7 We run the forg. pgm. both backward & forward. The output of this pass of t. pgm → .16

.08 → SN IMPT In t. original BW paper, ~~[illegible]~~ 1994, p14 ~~[illegible]~~ Table 2: They used their initial system on a test corpus (like Book1 of Calg corpus) starting w. 1k corpus they mult corpus + size by 4 up to 64M (actually 103M) ~~[illegible]~~ bpc ↓ from 4.35 (1k) to 2.01 (103M)

The ~~differences suggested a min at~~ ~ 1.98. — Corpus had to be 1M ~~before~~ it gets lower 2.93.

So, BW/as is, isn't much good w. small corpi. In original BW paper they got ~~[illegible]~~ one char compressed ~~[illegible]~~ using only 400 clocks! Sounds v.g.!

→ on 81.19, I used at least 4500 clocks: 11x as bad! — But BW uses ~ Quicksort, which is much faster than 2-3T.

3 → One of the Main Ideas of ALP was to deal w/ small corpi → .38 .38

One problem w. my present 2,3 Tree pgm is that at ~20 bytes per byte of corpus, I could only do (at most) a 4.5k corpus. T. smallest Cal. corpus file is 21k by!

16 .07 , would be 2 lists of symbols in order of pc. ~~[illegible]~~ We could arrange t. 2 lists & use them directly to create trial cands. So this would simulate BW, but would sequentially get pc's.

As is, hvr, it will not fit into PB35 — But I can use "expanded memory" and "Dynamic Memory" & get much more — But very slowly! Which may be good enuf to start off. Multiplying CPU + Speed by 5 mite help a bit — but I really have to rewrite pgms: perhaps PBCC can do it. — see if its fast w. large Memory use.

It may be possl. to use protected RAM in PB35 in ~~[illegible]~~ Assembler Mode.

16M dram ÷ 20 = .8M filesize. — so I ~~can~~ could do Calg corpus.

One trouble (so far) w. BW is that I haven't yet found a way to modify pc because of grammatical info. Bunton does it for PPM, so look at her Refs. She did mention using (exp. words) as primitive symbols — which is only possl. after ALP has discovered words & their associated grammar. (see 87.12 ff)

For study, all I really need is a bunch of BW xforms of files. (I may have a usable c++ file that can do this). From that I can try PPM ~~[illegible]~~ (?) or other compression tricks.

+ A Q about original BW paper: They did a 104M file on a 25 MHz machine, in 1994! They probably didn't have ~600M of ram at that time: They mite have had a HDD w. that ~~[illegible]~~ size. Anyway, How did they do this 104M file? If they divided it up, the ↓ of bits/char means that it had to remember relevant info from previous batch. How was this done? Perhaps write them about this!

There may be a type of Quicksort that divides up a file into non-overlapping chunks & sorts each separately. → 92.14 does it .36

38: (.13) → SMALL Corpi problem BW seems to deal w. prediction as good as old Z141: It does use contexts in a v.g. way. It may be t. (a) Best Way to do induction of this sort for Phase 1 ... whether corpus is large or small! English (et. type of corpus in BW (.08-.13) is perhaps much diff → Spac 93.00

3.22.04

4 pm

89.40 , A major (Approval) criticism of BW ! That I don't see any way to improve it using t. construction of a higher level grammar.

, Try Google intel crypto(graphy) MMX. or intel mmx crypto (graphy)

[SN] In crypto & in lossless compression. Say M(s) encrypts ~~or~~ compresses ~~a~~ messages. $M^{-1}(M(s)) = s$ is always true, for any s. But $M^{*}(M^{-1}(s))$ should also be true for any random s.

If it is not, t. system ~~a~~ could be made more efficient.

→ Look at Method BZ2 uses to xmit L.

.8940 spec

On t. original B&W paper (1994) They use "bring to t. front" coding. It uses this w. t. ideas of .27-.29 ~~[illegible]~~ is easily xlated into pc's using "Bring to front" coding. We get a very fast prob'y distribution over poss'l symbols at each pt. | The probabilities may not be very exact, but t. real Q is, will it give "reasonable pc's" to important subsequences of symbols? (.087) → (11)

[SN] B&W ~~says~~ that ~~T. system works best for~~ corpi ~~of~~ at least "1 or 2k in length". → I would have to "prime" my Phase 1 w. at least a few k of symbols before using BW — "or any other induction pgm"! In GA, this means a population of 500 or 100 for 20 symbols/card.

(.087) Not as fast as I thot! The "bring to front" algm. has a particular ordering of t. alphabet that has to be updated/for each sequentially pt. in "L".

PPM or PPM* can be looked at as (effective complicated) ways to code L. In PPM* PPM, instead of having t. code of a symbol in L depend on recent codes for past symbols — it also considers (w. = wt.) codes for symbols in future.

To "update" the ordering of symbols and "symbol pcs": ~~[illegible]~~ The ~~[illegible]~~ ordering of the ~~a~~ symbol α, and its pc is: {How many types of symbols have occurred since t. last occurrence of α?} For α's of low frequency, this updates slowly: Also, in certain sections of L, α will have much lower freq. than at other pts. In general, t. less frequent α is, the more distant (on L) its influence is.

A diff'nt way of Getting t. pc's over all α_i: Use a broad, symmetric smoothing kernel over L. When a new pt. ~~[illegible]~~ in corpus has to be predicted, we "insert" its "context" — (via a sort) which means we find where it is on L. To get its pc, we then ~~[illegible]~~ use a kernel on L, ~~[illegible]~~ taking ~~data~~ into from both sides of insertion pt. (as in PPM)

See 97.26: Maybe kernel has to be either ~~[illegible]~~ all forward or all backward!

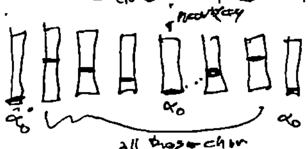
Also Note . 97.34R

I think width of kernel must be ∝ corpus length. Th. kernel must at once be ① as narrow as poss'l ② wide enough to capture all poss'l α_i types, (i.e. "escape"). In PPM, we mult pc ~~[illegible]~~ by an "escape constant" every time we ~~[illegible]~~ expand to a new context length.

[SN] ① Does this mean that t. escape constant in PPM ought to vary w. corpus size?

② In normal BW: It might be well to code each char of L by ⓐ forward motion ⓑ ~~[illegible]~~ backward motion: then average the 2 ranks: the function of rank that pc would have would have to be ~~normalized, since~~ rank can then have ½ integral values.

Q: Using "push to front" coding algm.: Say, at each point of L, we had an ordered char. list. (This would perhaps take much Memory). When a new key/α_0 is inserted, how much updating has to be done? The char lists for all keys out to t. next ~~[illegible]~~ for 2 α_0's have to be updated



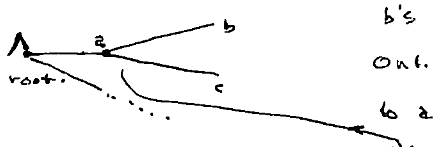
all those char lists have to be updated in simple way.

However, t. update isn't so simple: The amount that α_0 moves up in t. char. list depends on how many "new" chars (i.e. above α_0 in t. char list) have occurred since t. newly inserted key

So t. main problem is: How far are t. nearest α_0's? We don't need to "update" if we can answer this Q for all values of α_0. Unfortly, "How far" is not a simple Q (as we noted on (.36)!) → 91.00

3.21.04
4 PM

00 : 89.40 ; If we include only (say) downstream contexts, we ↓ size by factor of 2. It would probably still ~~compress~~ compress, but certainly, ~~to~~ more poorly.



If context b occurs, then if we escape to a, we remove b's possible continuations.
On the other hand, if context c occurs & we ~~escape~~ escape to a, we exclude "c's" possl. continuations.

So, perhaps it is impossl. to store at "a" the "excluded" versions of its possl. continuations.

The cct's at a will be the sum of those at b and c — (But "exclusion" doesn't deal w. arith subtraction — it does Boolean "subtraction": ~~on~~ whether a symbol has or has not occurred in contexts of contexts.

So even if you use Trees, we will not update exclusions! ~~That~~ Exclusions have to be computed for each key as it occurs.

SN) It looks like the best way to do predn. would be to wt. all observed contexts. Effectiveness in past predn. is an impt part of the wt. compn. Int. counter example given in ~~the~~ "Unbnd" cont. length for PPM" (v1995) p 4,20 They don't understand how to do this properly! My treatment of One Shot Lrng (OSL) in Z 14.1 analysis deals w. this in a proper way. T. wt. depends also on ~~total corpus length & ↓ as corpus length ↑.~~

SN) I'm beginning to understand BW method. It may be possl. to get pc's out of it & make it incremental. The method of "predn." may be run length encoding. — So code consists of symbol, run length pairs. — ~~Best~~ A good way to code is
.18 to study statistics of run length: ~~what~~ Are there correlns betw. successive run lengths?
20 — Do long runs tend to be followed by runs of 1 or 2? — Then return of long run
21. — on ~~first~~ symbol of previous long run.

A strangeness of BW: That ~~redundancy~~ log p ∝ length of corps + log length of corpus!
Other methods, log p ∝ length of corpus,
Unless ~~It becomes~~ b cost of ~~the~~ coding L is not ∝ length but less! — This might occur because we get longer & longer runs in L! That it should ↓ exactly as log (corpus length) is of interest, if true.

,27 How to get pcs of symbols from BW coding? L contains the predicted symbols, so at the proper pt. of L, we try all R (≡ radix) diff. symbols and see how the
,29 run length code ~~or~~ for L is changed: This will be a very "local" effect: easy to compute.
2 If the run length code consists of a seq. of symbols ~~with~~ alternated w. run lengths, we can decide on some pc as funct of run length. Symbols must change when they follow, so if p_i is pr. of symbols i we get ~~by~~ $p_i \cdot \left(\frac{1}{1-p_i}\right)$ ← for the next symbol.
Because of the (circular) torig topology every p_i may be mult by $\frac{1}{1-p_i}$ ∴ $\left(\frac{p_i}{1-p_i}\right) = \frac{1}{\frac{1}{p_i}-1} = \left(\frac{1}{p_i}-1\right)^{-1}$
I will see if I can find the best way to code/predict such a sequence.
See how BW does it: Note .18 - .21
This looks very promising & simple — also I may be able to improve it — also look at what others have said about improving BW.
→ Another trick: first ~~code~~ xmit. no. of diff symbols. Then xmit how many symbols of each type. From this info, the coding of L will make ~~certain~~ symbols get much greater prob'y as other symbols are "used up". → 90.06
2

3.20.04

ANN: Backprop analysis: .25

4th

So; ~~Each~~ The $N_{\alpha 1}$ node always pts to another node "on same level", That contain t. first char. in t. set of chars that ~~[illegible]~~ N_α points to. Call this "N_α^2", N_α^2 points to "N_α^b" (which is t. second char that N_α points to.; N_α^2 also points to $(N_\alpha^2)^c$

So: Each Node of 87.35 (like N1) points to a linked list of nodes that contain pointers ~~ptrs~~ to ~~N~~Nodes 2, 3, 4. Nodes 2, 3, 4 ~~[illegible]~~ assoc nodes with lists of where they point to. Each node also has ~~a~~ linked list of update info on cct's ~~#~~ >0 of each possl. symbol continuation.

Aside: T. foregoing looks expensive wrt. memory, & can take much time to make ~~[illegible]~~ ~~insert~~ (insertion / update) decision at nodes, since a potentially 255 way switch has to be evaluated. ~~#~~

It may be feasible to use my present 2-3 Tree sorter & store update info on contexts ~~[illegible]~~ as "auxiliary "Nodes"" in that system. An apparent difference betw. t. 2-3 tree sorter & t. "Tree" updater; T. 2-3 tree seems to update lowest context first! Hvr., t. exact mechanics of update in 2-3-Trees is not at all obvious! Where would info be stored? Before I began reading about Trie's, I considered using 2-3 Trees & counting contexts anew, ~~each~~ for each augmentation of t. corpus ... sounds very slow — but less memory used.

Q: In 2-3-T, could I use t. decision seq. for insertion as a "context definition"? (?)
~~#~~ Some diffy: The dcsn of insertion is in 2,3 radix form & often varies for various insertions.

0 (Maybe) WOOPS! The 2.3.T method may not get all of t. prefixes — ~~[illegible]~~ — maybe ½ of them! Consider t. ~~[illegible]~~ context: bba: aba ~~[illegible]~~ comes lexically before a, cba comes lexically after it. All 3 ~~[illegible]~~ have common ~~[illegible]~~ context ba; yet aba is rejected; only contexts after bba are considered! This seems to be very wrong!. It looks like if we have keys in lexical order,
24 then keys "close" to K0 will be both up & down (lex) stream, from K0. (.33)

25 SN on ANN: Backprop! Say t. ANN gives us a function & we test t. GOF of t. function on a fixed "Test set" of data. We randomly chose initial params for Back prop (i.e. no pcost) Then we get a seq. of functions terminating on some ~~#~~ very A.H. very good fit out training set. This is a 1 param seq of functions on t. Test set & has a "Best" point. If we repeatedly chose init conds & do runs, we will get many "good fits"
0 to test set. Picking t. "Best" is extreme SOP. Picking t. mean may not be bad: —— But I need to analyse this more!

3. (.24) Perhaps most serious effect of this; that computing "exclusions" is more complex than if contexts were all "upstream". A Q. is: How is it that Trie's are able to deal w. this effect, but simple lex ordering ~~is~~ is not? Well, Lex is a "linear" ordering of contexts, TRIES is not "linear". T. trie is a partial ordering that can be made into a linear ordering, but need not be. Lex/linear sort ordering can be obtained by inducing a Lex order on symbols. Tries needs no ordering on primitive symbols — only needs to know if 2 symbols are identical, or not.

→ T. impt idea is that ~~a~~ for a given key, contexts of it of a given length (< Max), will be found both
0 up & down stream (Lex) of t. key (.24)

3-19-04

PSG discy System
Applicability to ~~[illegible]~~ (A) Z141

.12 ff

4 TM

.0 Using a Trie Str., Each time a new key comes in, I update t. trie nodes w. data on freq[redacted]vencies of each symbol. Or, have t. Edges represent ~~[illegible]~~ symbols.

Still, we may want to put data at nodes.

Each Node can have list of edges emanating from it in lexo order — (or a cheaper faster access method).

→ 26

0.5

SN In a GA envt. & in perhaps all Lsrch (or other) envts, We will have a choice of several induction schemes for each a priori Cand. choice. The one we chose will depend on ① Mean Entropy obtained by t. induction scheme ② Mean cc of induction, E ③ Mean cc of final cand evaln (cc of "fitness function") (≡GORC)

If ③ is very large, we can afford to use expensive (by cc) induction schemes; In general, there may be a way to optimize over a set (of Hyper induction schemes.

2) 86.28 SN So here, we go back to the early Z141 & we want to discover Words, & we want to discover "good parses" for t. corpus. From this parsed corpus, we want to find adjacent word pairs that are predictive. One way of doing this is simply looking for compression obtained by ~~assign~~ defining t. pair names. Another is to invent classes of words (≡POS) and find Markov rules for t. POS's following one another. Essentially, what we do is parse t. corpus into words, then assign words to classes (POS); Then we have a corpus of a seq. of POS'. We then obtain rules about what follows what or, we invent ordered Pairs of POS's. — if ~~[illegible]~~ these classes are among old classes, we have a recursive rule!

So t. proj. may be a useful, achievable, way to get good cfg's (maybe CSG's!).

Probly nec. to periodically reparse t. corpus in view of newly derived reps (≡ POS', words)

→ Note that a word may be in >1 POS class. — (We would like t. classes to not overlap — i.e. be Mut. Exclusive) ......

→ .12 ff is a way to augment both PPM or Z141 and obtain grammars w. "these structures"

26 : (.05) → Each of certain edges will have a list of "case counts" for various chars that followed t. corresponding Context.

cct ← ≡ casecounts
Casecount

Def cct ← case count
CCt

One way to realize this is to have "Nodes" that have info on them i.e. ~~[illegible]~~ cct's.

Each node also has pointers that ~~[illegible]~~ link it to a few other relevant nodes.

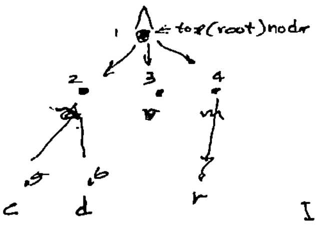
0 As t. corpus grows, t. no. of nodes grows.

Each node will have several addresses in it. If we are clever, we will be able to eliminate some of them or make them only a few bits long.

First design a not particular efficient system, then try to improve it.

O.h. Each N node has a list of some symbols & addresses of corresponding nodes.

.35



~~[illegible]~~ N_1 is root, it points to N_2, N_3, N_4

N_2 pts to 5,6 , 4 pts to 7

We keep Nodes ~~at~~ to ~~at~~ most, depth D.

Thus far we ~~[illegible]~~ don't yet discuss cct info.

If radix is R, each node has to be able to pt. to R diff't nodes.

40

3.18.04
4 TM

00 In normal tries, we have a branch factor of R (t. corpus radix) at each point. In a certain kind of trie, t. edges can have -> 1 characters.

.02 My 2-3 tree uses mem ∝ N (= corpus length). I could keep this constant by removing a data pt N₁ chars in t. past every time I added on a char in t. present. I would then have a data length of 2N₁.

05 (84.12) → **SN** I thot that I wouldn't have no use for a deletion routine for 2-3 Trees, but .02 suggests that I may want to delete data in that way. T. idea of deleting a data pt (a "pt" is N₁ symbols) is kind of weird! — Unclear in my Mind! Seems O.K. to delete a data pt at Now − N₁ & before, but otherwise, if a symbol in t. middle of t. corpus is deleted, the "comparison" of 2 ~~[illegible]~~ strings (for lex ordering) is screwed up.

10 Normal PPM only stores contexts of length < D w. D ~ ~~5~~, say. to consider contexts of arby length — They probly use a modified Trie. Using 2-3 Trees, I think very long context does not introduce ~~[illegible]~~ much Dif'ty. — insertion time is still ∝ ln N, ~~[illegible]~~. For Trie structure, would not space be ∝ N²? Teahan may discuss this. — look at his early chapter on PPM — (It may be = to his paper

.15 w. Cleary & Witten!) → (.30)

16 : 85.33 → ℤ only grammar we may need is a list of POS's [Pᵢ] (parts of speech) and a probty density on [Pᵢ Pⱼ] or equivly if Pᵢ occurs, what's probty that Pⱼ will follow? This is essential digram frequs. A more Complex Grammar would give probys of strings of >2 words ~~[illegible]~~ that were not (trivially) derivable from t² [Pᵢ Pⱼ] density.

20 So This looks like a very useful way (even at t. Beginning) to obtain a grammar for a corpus. We start with ~~t~~ a small [Pᵢ Pⱼ] density set, then expand / contract it to improve its predn. capabilities. Eventually, we begin to ~~a~~ add concat rules and we get eventually recursion rules giving POS's rxcos of arby length.

The ~~[illegible]~~ criterion of the design of t. POS's Pᵢ ~~are~~ that t. knowledge of t. density on [Pᵢ Pⱼ] maximally ↑ t. pc of t. corpus. From an initial [Pᵢ Pⱼ], we can ~~[illegible]~~ incrementally add on delete members of each POS in attempts to ↑ corpus pc. (A Greedy Strategy)

We mite be able to construct ~~the~~ Pᵢ classes from knowledge of some (many) probty word$_k$ → word$_\ell$ ~~[illegible]~~ observe empirical frequencies. → (87.12)

.8

29

30 (.15) → Another approach: Try to implement PPM myself, using "trie" structure.

O.K. say I have this Trie that represents contexts. T. root is Λ & is in position of symbol to be predicted. As we branch out in tree, we eventually get to points at which there is no further branching. At such points, we could store this data!

1) address of this ~~[illegible]~~ pt in t. corpus 2) How many times this prefix occurred, 3) The counts of each possl. continuation.

If we only store contexts of length ≤ D, Then each time a new symbol is added to corpus, We update the tree, by inserting t. most recent D bits (+ contin) into t. tree.

Alternatively. (& a Sanders Trie): Just put all D bit ~~[illegible]~~ suffixes of t. corpus in lexical order: Hvr That, in itself, doesn't give all that's needed. Also, I need the predn. statistics for each context. 40 (Computing) Counting them each time we augment corpus is probly too time Consuming.

40

4TM P123

00 SN Bill W.J. Teahan's PhD Thesis: 1998! "Modeling English Text" Teahan has at his website a large bunch of ~~[illegible]~~ zipd PS files of his papers: Much interesting work there w. Cleary & Witten: (See "web refs II html.txt in PS")

.03 He espouses using "Tags" (parts of speech of each word). This enables/total compression of ~1.48 bits/symbol. [(Compare w. ~1.29 that King & Cover Got for Humans!).
He thinks that by using t. enormous WWW corpus, ~1 bit/symbol would be obtainable! ☺

SN Dimitry Shkarin [2002]: Calgary Average of 1.92 bits/char — or 2.041 ! or 2.050 BPS. 2.104 is more likely. Still is much better than 2.29 which is "OPMD" ← by P.G. Howard
http://datacompression.info/ppm.shtml ← couldn't get the originals [illegible]
(I probably Howard) → "One step to practicality" in A. Storer and M. Cohn eds. Proc 2002 IEEE P.G. Howard.
D Data Compression Conf, pp 202-211 Ap 2002 < PPMII algm. ← I got this in PS ~ 3.17 orig .04
PPMZ2 [12] is ~~PPM*~~ —averaged— Got 2.139 BPS. ?

I have Shkarin's paper in PS ~ 3.18(orig).04! He describes improvements that he made in PPMZ (ZZ ← maybe PPM*?) that got its compression down from 2.139 ~~[illegible]~~ to > 2.041
(The ~~[illegible]~~ Cleary et al only Got 2.34 bps on the same (Calgary) corpus.

Teahan says that w. English (.03) one can do much better w. "words" & "word tags"

Gen. discussion: ① I need to understand the "Trie" data struc. & how it may used for context encoding. ② The "improvements" in PPM involve (partly) finding better groups for contexts! The way I would do it: New contexts are formed of concats of old useful contexts & parts of older useful contexts. T. various "improvements" of PPM may
0 have been in this direction. What about use of "words" + "tags" in English? — essentially partial parsing. This would probably much narrow t. set of words that could follow a particular word. That PPMC worked best for p ≈ 5 suggests that ~~[illegible]~~ word sequences (other than 2 or 3 char words) were not important regularities detected used by PPMC.

I'd like to try out ~~[illegible]~~ PPM & see if it's promising for "Phase 1" — then, perhaps, look at improvements, by others or by me.
[The idea of Eng/Grammar — main effect, That having word classes & a simpler grammar for t. classes ~~[illegible]~~ can signifly ↑ ssz for ~~[illegible]~~ rules about what follows what.
e.g. we have "adjective" "Big" which is likely to ~~be~~ followed by a noun. So this is a simple narrowing on what occurs after t. adjective. To realize that a word can follow "Big"
0 we need not have seen it follow "Big" in t. past — we only need know that own next word ~~[illegible]~~ must be ~~[illegible]~~ in t. "noun" class. We have to discover these classes by trial — by noting set (class) of words that come after a particular word & assign tentative class names to them. (This is one of my initial PSG-discy Methods). → 86.16

3 But anyway, by making a grammar of ways to improve PPM (using as instances, various tricks proposed) I write to be able to get a PPM to improve itself much!

Hr, PPM may already be good enuf to go thru. Phase 1 & get to phase 2.
I do want to know just how they use tries for PPM (& PPM*) — to see if it's much better than simply using 2-3 Trees. The structure of data in Tries seems very similar to t. Lexical ordering method used in ~~[illegible]~~
40 B trees.

4TM

T. main Jist of 83.27-.30 is that Z141 is probly t. best way to do it — That t. use of frequencies of algms is a better way of ~~[illegible]~~ doing the "time of first occurrence".

In General, I want to look at PPM in detail & see just how (& if) it approximates ~~Z~~ Z141. A main Q is that Z141 may be a poor Model for English. — that PPM may be better because it models it better. Z141 may model other Stochastic Sources better — These are Empirical Q's. → see 31.29 for Discn

. So read up on PPM again and see how it is really related to Z141.

SN On Definition of a key ~~[illegible]~~ in 2-3 Tree Pgm: I don't see that I'd ever want to do that! A key is simply an address of the sequence A[ ]. We may find a value is uncertain, so best tell it w. wild card (≡ any thing) or "≢ wild" or whatever, but a chg deleting seems to be not just what I want .... tho maybe there is a way! → 86.05

SN Did I really ~~test~~ SU ~~[illegible]~~ 14? I did show that it faster derived KYMX! Orderings & that t. perm put them in that order. ~~[illegible]~~ The KYMX! number is assoc. w. the Σ correctness of t. |Σ+C| ~~of~~ partiony. That all of the KYMX! were distinct was not shown. I could get t. pgm to test this, but instead look at 4! = 24 cases: they are in 2 K12 symmetric sets.

SN WEKA ← Easy to find via Google. is an organized set of ML lrng, induction, data mining algms. Pgms are on web, directly downloadable & Pgms are in Java w. "classes" of objects.

I might be used as a source of ~~[illegible]~~ PSM's to make a grammar of —

(There is a ~~Book~~ Book assoc. w. Weka, which one can buy, but its not a bc key.)

Perhaps the Duda, Hart, Stork book might be better in this respect.

Back to ~~[illegible]~~ PPMC, PPM*!

My impression of PPM*: To do prediction; find longest context that has occurred ~~at least~~ twice in past. If it is ~~[illegible]~~ "consi" (≡ "deterministic") find shortest consi context: Use it to predict. If it gives wrong prediction, escape down to a context that does give correct predn.

We can try coding in 2 ways: ① ~~[illegible]~~ Old MDL! ~~[illegible]~~ for known corpus, we ~~[illegible]~~ find ~~[illegible]~~ models within that corpus & get ~~[illegible]~~ t. params of each model using data within corpus ② More like the lower's version of ALP! Consider possl. continuations of corpus & make s.Del. codes for each of them.

In ① we get a pc for unknown continuation directly ~~[illegible]~~ In ② we code (t. corpus + each possl. continuation symbol) & compare t. various codes.

W.J. Teahan
J.G. Cleary
T. entropy of English using {PPM}-based Models. 1996

also same authors, 1997
~~[illegible]~~
Models of English Text.

3-14-04

4TM

33.30

00:82.22 [Re **33.19–30**, etc: My present impression: ⓐ is cost of new symbol — $\overset{\propto}{\approx}$ (no. of symbols thus far)$^{-1}$

(82:23–.34) R for long seqs. — this gets very small ~~[illegible]~~ Not nearly: In fact the first time that α occurs, gives us "the no. of symbols thus far". If α has occurred only 2 times & corpus is large, then we get small ~~[illegible]~~ "ⓐ"<(33.20). ~~[illegible]~~ ~~[illegible]~~ So, the idea that ⓐ is same for all kinds of "α" is wrong. For symbols in the normal "alphabet" we can't use:

.04 Its first time of occurrence as an ⓐ index. So actually, what ⓐ is, is the frequency

.05 of occurrence of the symbol, or of "α" !! This "freq. of occurrence" means that in coding the entire sequence the pc's of the other symbols in the alphabet would be reduced (as in my recent/ old. See letters to Wolff) version Z141. In fact, that work may be essentially correct: that the detailed Algebra may be wrong. The initial equations could be correct, but my "Approximations" could be wrong.

If PPM is better than that Z141 method, it is probably because English is not well approximated as the kind of "Simplified CFL" that Z141 is. (See 31.29 for brief discussion).

A Big difference betw. PPM and Z141 is PPM using "length of α" as a basis for a pc, v.s. Z141 using "product of pc's of symbols of α" for a pc.

"The length" function should ~~[illegible]~~ depend on Entropy of the sequence being coded — essentially the geometric mean of the pc's of the incremental symbols.

$\boxed{SN}$ The expression for the sum of ~~[illegible]~~ p.c's of coding corpus in various ways, looks perhaps identical to the expansion of $(1+x)^n$ using the Binomial Thm!

Say ~~[illegible]~~ ~~[illegible]~~ again α occurs ~~n~~ times in the corpus. If we use ~~[illegible]~~ α ~~k~~ k of these times, we can do this $\frac{n!}{(n-k)!\,k!}$ ways. Each of these ways has pc ↑ = ~~[illegible]~~ $\frac{p_\alpha}{p_\square} \equiv p_0$

So total pc of coding is $\sum_{k=1?}^{n} \frac{n!\,p_0^k}{k!\,(n-k)!} = (1+p_0)^n$. This is the ↑ in pc ~~[illegible]~~ ~~[illegible]~~.

by coding using ~~[illegible]~~ the α definition. $= 1+\frac{p_\alpha}{p_\square}$. ($p_\square \equiv$ product of pc's of symbols of α)

Unreasonable!: if $p_0 = 1$ we expect no ↑ in coding effcy!

Perhaps we want $\sum_{k=1}^{n} \frac{p_\square^{n-k}\, p_\alpha^k\, n!}{(n-k)!\, k!} = (p_\square + p_\alpha)^n$. — No!

.27 Well, it's Much More Complex! One (perhaps still, simplified) way: α is in ~~[illegible]~~ Z141, just another symbol. It's pc & the pc's of the rest of the symbols, depend on how many times α is used. This includes symbols got in α

.29 In each of the $\frac{n!}{(n-k)!\,k!}$ codes, k is different & p_α is diffnt & pc's of all other symbols ~~[illegible]~~ are different.

30 As is, the method May be better than PPM or BZ2, But it needs to be ~~[illegible]~~ revised, & probably approximated, — since exact form may be too time consuming. Even w. good approxns, PPM & BZ2 will prob'ly be much faster — Tho in certain kinds of induction problems (the most difficult kind, the most useful kind), the computation for "Goodness of fit" will take much more time, & so the ↑ accuracy of Z141 (if any) will be worth it!

So: ~~Make~~ Take Note of .27–.29: Perhaps Make a good ~~[illegible]~~ Bib. ref. ~~[illegible]~~ before I know what's wrong w. Z141 — If anything!

Z141 work: Then I want to try ~~[illegible]~~ PPM for Phase 1. If its good enough to get to Phase 2, then forget Z141! Otherwise, Go back and ~~[illegible]~~ test Z141 on various data to see if it's better than PPM. Use Maple to test Algebra, & "Approximations".

40

4TM

.00 Now that I have the Sorting pgm, Back to ~33.19 (Best way to code a corpus using subtrees)

.01 Assoc. w. this (33.19) idea is 31.22. However consider all ‖ codings, I think what happens is that if we mult. the no. of codes by the wt. of each, we get a "Binomial distribn" — which is ≈ Gaussian w. a particular width; so we can probly get total wt. in a simple exact or approxt. way. This is interesting in that we end up using not the actual frequency of the usage, but the "most likely value" of its use — assuming the usual stochastic CFG model. (31.29 suggests that this may be a poor model for English!) If so, it means that I'll have to use a different kind of coding scheme for English and data sources like it: Also, be able to recognize sources of this kind.........perhaps by "Context" (where they came from - how generated, etc.)

0 Re: 33.19-.30; Reading it now, I don't get a clear idea as to how it works! Think about it a while: Perhaps we just have a new alphabet and we have to code the corpus using it. Say α ≡ abc: Every time abc occurs, we can either write it as α or abc — this gives us many ‖ codes for the corpus. The codes/pc's for all sequences can be obtained by assuming the abc (etc.) symbols are coded in the "usual way" (i.e. this way or PPM or ZBZ or whatever). In each case we will ↑ pc in the code by using α instead of abc, but using too many of the possible occurances of α means we have fewer ‖ codes.

$$\sum \frac{n!}{k!\,(n-k)!}\,\frac{1}{\rho^k}$$

gives us the Σ pc assoc. w. k uses of α when n are poss. ρ is the ratio of pc's of abc v.s. α.

In a real corpus, abc may have a diffnt pc each time it occures — so maybe use average/approxn.

20 $\rho = \frac{pc_{abc}}{pc_\alpha}$: How do we find pc_α?

The pc of abc can, in each case, be determined from previous calcns. of pc of total corpus up to each point.

22 But still: how to get the pc of α? From 33.20-.24, ⓑ is a funct of length only. → 83.00 →

23 (& ⓐ usually constant.). The exact function of length is obtained by simple averaging over past, using a Lap's rule modified by "precorpus" idea.

SN | Re SN14.805 This 2,3 Tree pgm may work o.k. now, But in future, I may want way to delete selected keys. ~~Either~~ Add that feature now a/o write up documentation very carefully, clearly so I can add this feature later if necy. The pgm seems to be a simple inversion of the insertion pgm: i.e. Deleting a key can give 3→2 kid node (∴ minimal change) or 2 kid → 1 kid may propagate up the tree.

write this up very carefully! — I almost lost it!

There is the Q of how to designate the key to be removed. I may want all keys labled w. their Parent so I can easily propagate upward in the tree. (Also note .33 about pushing discarded Nodes on a STACK)

I think the deletion pgm may be slightly simpler than insertion, because we don't have to decide ~~where to put~~ just where to cut a 3 kid into 2,2 kids.

Another Q: how is pc assoc. w. "the value of length actually used" used? What value was "actually used" in each case? ↳ 83.00

33 Also when Nodes are discarded, they must be put on a stack so those nodes

34 can be reused when new nodes are added.

Also, the insertion pgm must be modified so "BMX=BMX+1" or "INCR BMX" usually is replaced by "If stack is not empty, Pop a BMX address from stack — else Incr BMX :"

4 PM ABele

30: 80.90! One Moral! Even tho' sorter may be working well, getting it to do what it is supposed to do as part of another system, requires careful checking!

→ Anyway 100 μsec for each JJ loop is 10 key insertions — so 10 μsec for 1 insertion:
10 μsec = 5000 clocks ! (500 MHz Pentium) seems not so fast — But it is in Basic and it should be runnable much faster in Machine language.
This 100μs/JJ loop includes time to generate each $\tilde{Y}$( ) array — but
I think this is << time of JJ insertion loop.

Timing for SN18B:

KYMX:	Time(sec)	Z = kymx-2 = N	Time/Z!	T musec time using 10 k insertions	$\frac{T}{\ln Z}$	$\frac{T}{Z}$	$\frac{T}{kymx}$	$\frac{T}{kmx \ln Z}$
9	.3285	7	65.178 μs					
10	3.2425	8	80.4 μs	82.47			8.2	3.965
11	33.78	9	93.08 μs	~~98.8~~ 99.04	45.05	9	9	4.09
12	380.63	10	104.89 μs	109.67 .76	47.62	10.9	9.139	3.97
13		11		126.1 .78 .84	52.54	11.45	9.7	4.04
14		12		142.9	57.14	11.90	10.2	4.08

N is incremented for each complete recalculation of Y( ) — it involves ($\overset{=Z}{KYMX-2}$) insertions. So we expect time to grow as Z ln kymx — and it does. So ~ 9μs/insertion = $\boxed{4500 \text{ clocks! for kymx} = 11}$
How it could use 4500 clocks is unclear! Perhaps try compiled version of pgm.
This is for only one insertion!! I could look at rate times between various events, using test CPU cycle timer. (PB35's MTimer has only 2 μs resoln.

While system tries to use primary, secondary cache: It may use regular memory, which is on the HP500's prob'ly 133 MHz = 7.5 ns, rather than 2 ns clock. 9μs/7.5ns = 1200 ~~many~~ memory accesses per insertion.
Then 4 μs per stage is $\frac{4000 \text{ ns}}{2 \text{ ns}}$ = 2K clocks per stage. This includes choices going down and (mainly) updating, going up.
I've done pgm that has input = KYMX, output Time in 2 ns units (clocks) of $\frac{\text{Time}}{(KYMX-2)\ln kymx}$
Using kymx fixed by pgm and running it I get, for kymx=10: 2232 to 22~~36~~.
If I do input/pgm kymx " " " " 2238

11 →	2208
14 →	2254
20 →	2239
16 →	2194
15 →	2260
13 →	2235
17 →	2263
18 →	2255
19 →	2302
20 →	2292
12	2319

Compiled version
10 → 2213

exe. pgm: 45k by:

10 →	2381.9
	2390.55
	2380.4
20	2240.9
	2241.6

SN18C.Bas prints $\frac{\text{Time in Cycles}}{kymx(ln(kymx-2))}$ — This seems more monotonic than other version, but difference is small — but why different results each time, is unclear!

		Compiled "exe" version
10 →	1981.65	1985.00
20 →	2089.01	2088.83
15 →	2070.75	2071.9

SN By omitting the 1Rc( ) function in just timing protocol — ~~[illegible]~~ 454K → 41.7K so floating pt. does take space But still, 41.7K is a lot for an essentially simple pgm! (Only 6.7K in Basic pgm.

from DOS (SN18C.EXE)		from E: 64M ram floppy
10	1976.53	1980
20	2088.176	2091.42
15	1997.95 ← ?	
15	2069.57	2070.06
16	2017.99	2014.2

3·12·02
4PM

SN18B.BAS (.26)
SN18B.Bas

V.G.! ☺

30: 9.40! So I want to find these errors! Get it to stop and print out $\vec{Y}$ when misorder occurs. Ah! I see one!

Y: 0 255, 10, 9, 8, 7, 2, 5, 1, 4, 3, 6 4,2 / 4,3 reversed
0 1 4 2 3 5 6 7 8 9 10 255

KYMX=7!
0 255 5 3 2 4 1
0 1 2 4 3 5 25
Actually lots of errors

KYMX=6 no errors
I have the runs numbered in sorder so: "Bad runs"
13-17, 23, 37-41, 47, That's it. 12 errors.
5

.08 13 5 3 2 4 1 → 0 1,2 4 3 5 ← using this data, SN14 would find a bad reason while Nodeparents.
14 5 3 1 4 2 → 0 1,2 4 3 5
15 5 2 3 4 1 → 0 1,2 4 3 5
16 5 1 3 4 2 → 0 1 2 4 3 5
17 5 2 1 4 3 → 0 1 4 2 3 5 ← / ← Also note travel output in SN14.
23 5 2 1 3 4 → 0 3 4 2 5

So either a hard to find discrepancy in the sort pgms or error in initialization!
For the 13th case in SN18A. The Node printout gave 3 as Mid kid, 4 as R kid —
so ~~order~~ order was correct, but somehow the UP, DN pgm got screwed up!?
Could update on UP() & DN() be wrong?
See if up & dn arrays are consis.

28 Y() = 0 255 5 3 2 4 1 + correct DN array!
1 2 3 4 5 6 7

	1	2	3	4	5	6	7
	0	3	6	5	7	4	1
obtained DN	0	3	4	6	7	5	1
Correct UP array	7	0	2	6	4	3	5
UP obtained	7	0	2	3	6	4	5

0 1 3
DN 3 2 4 6 5 7 1
255 5 3 4 2 1 0

obtained UP 0 5 6 4 3 2 Null
1 2 4 3 5 255

so both UP & DN arrays obtained are consistent

Look at SN4!

Node Printout for case 13 (.08)

	M	R	MN	Lk	MK	Rk	P	BT
1	7	5	1	1	7	5	3	1
2	0	2	3	0	3	2	3	1
3	4	3	1	1	4	2	0	0
4	0	6	4	6	4	6	3	1

30 This is clearly wrong since Node 4 is a 2 kid array (M=0)
So I should (at least) zero all Lk and M and P and BT previous to the JJ loop. This seems to fix it —
No errors ~~at~~ for KYMX=7

0346751 7023645
DN() UP()
SN14 gave a 0 here!

.35 SN18B.BAS has correction: works fine (I guess!)
With KYMX=12 it took ~ .1 sec to do one JJ loop. if I had "H" key ~~letter~~ pressed.
If I omitted key board — i.e. "INKEY" Key checking it did about 10k JJ pgms in 1.3 sec. : so ~130 μsec per loop.) seems very slow.
In ~8 minutes it examined all 3628800 (= 10!) permutations and found no errors (i.e. Y(UP(J)) was always > Y(J).)

3.11.04

4TM

00 Y = 1 2 3 ~~4~~ 5 6
0 255 4 3 2 1

JJ=3 Loop occures soon after start of JJ loop

in SUB1 going down tree:

@ ~~A~~A=2 A=3
@ BB=0 B=3
not nearly # into A&B

FNCA
IA(A, BB) ← This does it.
A & R(B) seem to be identical strings!

First jump to S1: its just logically. S1 choice ~~for~~ - it looks as if top node is has trouble ~~because~~ A ≡ R(B) ∴ seems wrong, since
→ ~~B~~ ~~Params~~ The Node params ~~[illegible]~~ were not saved! Lots of other things or ~~[illegible]~~ or reset, as is nary when the JJ loop starts,

PB35 statements Relevant:
1) ERASE (for static Arrays)
2) Restore (for DATA lists)

0

SN18 is where I'm trying to fix SN17!

Just above LL2 (creation of permutations), we have J=N: which is initial entry to LL2, But usual entrances from ~~[illegible]~~ bottom of pgm - that sets up ST( ) array, Just before J=N statement I want initialization & Restore of Node (params, etc -

At what point do we exit? to go to the reset?

For each mode of the ST( ) array, we have to do the entire JJ loop

Perhaps J=N is first time entry to Permutation pgm, - it can be very early: we then put LL2 earlier - to include ~~initialization~~ of Nodes ~~[illegible]~~ & set Y( ) ST( ):

30

((Decide which initializations are needed when - and ~~[illegible]~~ place "LL2" so that they are done when & only when they are needed.) Be sure J=N is 1st into place

I may still have to do lots of ~~x~~ modification of creation of Nodes & assignment Before LL2: of Node params.

At entrance to JJ loop: Instead of B = BTOP! first set BTOP to 1 —

~~[illegible]~~ Earlier: BMX=1, BTOP=1) should be initialization.

~~[illegible]~~ Before first entrance to "LL2" ~~[illegible]~~ routine, we want ST( ) ~~[illegible]~~ and YY( ) zeroed.

So put that next to "J=N"

N = KYMX-2 should be earlier; its true always | Y(1), Y(2) always! ∴ before LL2,

30 Read Node params should be done every time.

~~So: At~~ 3 categories

1) Always: N=KYMX-2; Y(1)=1; Y(2)=255 ✓

LL2 2) First time entrance to LL2: $\vec{ST}()=1$, $\vec{YY}()=0$ ✓

3) Subseq. entrances to LL2: ~~[illegible]~~ ~~BMX=1 BTOP=1~~ B=1

4) At entrance to JJ do loop: BMX=1, BTOP=1, B=1; set NODE1 params
Currency: B=BTOP just after each JJ starts

UP(1)=2: UP(2)=1

This seems to have fixed it SN18 seems to work.

Running KYMX=12, with ~~[illegible]~~ only $\vec{YY}$'s ordered sequ. printout — I noticed occasionally the ordered seq. would change a digit or 2 momentarily — So the sorting pgm makes occasional errors

90

4TH

0 77.40 1 On SN17.bas: Combining SN16 (N! permutation) & SN14 (23 tree) barrier?

ABCDEFGabcde
fghijklmnop

I have it "running" with "in key" barrier;

Before key press I ~~get~~ get (for hy input) $\boxed{0, 255, 2, 1}$ on screen

0 255 is fine. 2 1 would be for Y(input), but I should have 2 more symbols:

press key (down arrow) & lines as output on screen.

0 255 2 1 0 0 } repeated 4½ times on screen, — so maybe last few configs, hy my = 6 means 4! = 24 configs,

) small space

0 2 255

) large space

If I initial Y(K) = Y(K+2) is not there, I get "4321" as Y output.

I fixed rewriting of Y( ), but now, first outputs 0 255 43 (4 8)

as it did nothing about moving Y( ) first! ~~F~~ Part of trouble was my limit was N (≡ hymx−2): Also, I wasn't writing it backwards properly. Now output after first 0 255 4321 (ok.)

was

0 255 4321, 0 255 4321 ↑ why? OK, no print after first R0,

0 4 255 — may be ok, output is in order

0 25 4 321

0 4 255 and ∞ loop.

It supposed to finish JJ loop before going to next R0. I left = L2 instead of LL2 !

Now: 0 255 4321 output before keypress (OK.)

After key press

0 ~~255~~ 255 4321

no space 0 255 4321 } ok.

0 4255) space one space — ordered output but for only 1 keyinput (4)

) larger space (3 spaces)

0 255 4312 ← Y( ) ok. next happens got loop of screen filled (only) w. 0's ⇒

← But got to loop after this.

looks like it got Y( ) screwed up!

This is the L9 loop printing J = UP(J) : Y(J); it just continues until J = 2 (which never occurs)

Ok. at first, but prints out 0 4 255 as ordering; It shouldn't — it should do JJ up to 6 not just to 3. This is output permutation.

~~0 255 43~~ It shouldn't print 0 4 255

I tried SN14 w. input 0,255,4321 ~~##~~ keymax = 6 : Worked perfectly!

The first time it prints Y( ); JJ = 0 (ok)

Next time " " " " JJ = 7 Ok. — But printout of sequential Y( ) only has ~~[illegible]~~ unsorted Y(3)&(4) : how did it exit ~~the~~ JJ loop.

A bug! Node 1 was not properly initialized! I had "N=1" instead of "J=1"

Now ~~[illegible]~~ Its output is

How did it get past 5 (=4!)?

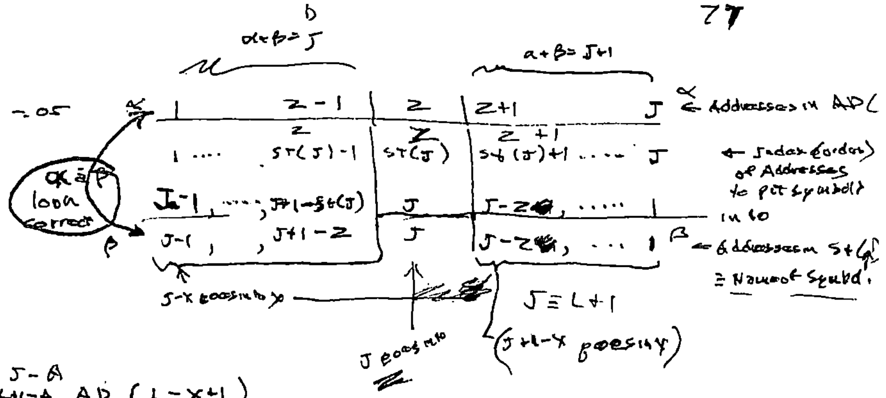
	JJ value		J	
0 255 4321	0		4	
0 255 4321	7	— ~~Y~~( )	7	— J is the index of the for next that prints Y( ).
0 1 2 3 4 255	7	— ~~[illegible]~~ ordered	2	
0 255 4312	7	· Next Y(6).	2	

It gets into inf loop at this pt — gives no printout.

4TM

The main book horror is 6. sven. of 75. 00 - .05

```
Z = ST(J) : D = J
For X = 1 to Z-1 :
  A = 1
  while ST(A) > 1
    INCR A
  Wend.  →  AD(X) = L - A + 1 = L+1-A   AD (L-X+1)      = J - A
  INCR A                  The X^th address.
  Next X
D = J
If X = Z Then SY(X) = J :        D = D+1
                                 instead of AD, use SY(mbol)
  Else  SY(X) = D - A                         SY(A) = D-X
```


D
α+β = J
α+β = J+1
α ← Addresses in AD(
1 | Z-1 | Z | Z+1 J
Z Z Z+1
1 .... ST(J)-1 | ST(J) | St(J)+1 .... J
← Index (order) of Addresses to put symbol(s) into
J-1, ..., J+1-St(J) | J | J-Z, ...
J-1, , J+1-Z | J | J-Z, ... 1
β ← Addresses in St( ) = Name of Symbol.
α = β look correct
J-X goes into X
J goes into Z
J ≡ L+1
(J+1-X goes in X)

L+1 ≡ J   J = L-1

12 For X = 1 to N : SY(X) = 0 : NEXT ; do this with initialization of ST(X) = 1

Z = ST(J) : D = J — using ST(J) & J probably o.k.

Try later.

```
4  A = 1
   For X = 1 to J
     While SY(A) > 0
       INCR A
     W END                     X = Z = ST(J).
     If X = Z Then SY(A) = J : Else SY(A) = D-X
                               : D = D+1
     INCR A
   Next X
   Print out ...... SY(X) ..
   GOTO L0
```

SN16 Includes SN15

Re: SN16.Bas! Maybe best to start working on SY( ) when D = J = 3 or 4

I left out A=1 ! Now seems to work w. N = 3

try N = 4 may possibly work.

SN16.B now/ seems works fine!

Try timing SN16 & SN14 faster (also time parts of SN16. & SN15 — all N! permutations; ← var radix)

(NB) Since I'm using ≈ 20 Bytes of storage / to deal w. 1 symbol (Mainly Node info), I can afford to store other info about each of mt. corpus: e.g. how often it's been used as most frequent, 2nd most 3rd most frequent) etc. We do this when the log N factor in key insertion time gets too big. (The perhaps Moore's Law mite help!)

— Actually Moores law is e^t & we are now concerned w. (n6.) but it is so slow that even the mildest poss Moore's law can deal with!

(SN) Moore's law can (deal w.) overcome any polynomial time f in cc !! (But the time necessary before M's law takes over depends much on poly exponent. $\frac{e^x}{x^k}$ when becomes > 1 (for x > 1 say) is a function of n. → $\frac{e^{\frac{x}{k}}}{\frac{x}{k}} \cdot \frac{1}{k}$) ↑ in x direction, ↓ in y direction.

3.10.04
4TM

SN15.Bas .16

$\Sigma = J+1$
$J+1-2-1 =$
$J-2$

.00 (75.40 ↓) L+1 in the $ST\binom{L}{+1} \overset{\pi}{=}$ Zero; and then integers ~~[illegible]~~ from $ST(L+1)-1$ down to 1

in the remaining ϕ's. (Draw a careful diagram before prog it)

$1, \ldots, ST(L+1)-1, \; \underline{ST(L+1)} \; | \; ST(L+1)+1, \ldots, L+1 \leftarrow$ (Addresses) $\quad L - X = ST(L+1)-1-1 \; ; \; X = L+2-ST(L+1)$ (L places

$L, L-1, \ldots L-(ST(L+1)-2) \; \underline{L+1} \; | \; L-ST(L+1)+1, \ldots, 1 \leftarrow$ Number (Address) in sorted ~~list~~ (Address) L places

$| L-ST()+2 |$ $\quad -ST(L+1)+1+L+1 \overset{?}{=} L-ST(L+1)-1+1$

$ST(L+1)-2$

$ST(L+1)-2$

$ST(L+1)-1 \; | \; L+2-ST()+1$
$ST(L+1)-1 \; | \; L-ST()+1-1+1$

$1, \ldots ST-1, ST, \ldots L+1$
$L, \ldots L-ST+2, L-ST+1 \ldots 1$

$L+2-2 = J+1-2$
$L+1 = J \quad L+2 = J+1$
$J = L+1 \quad L-2 = J-3$
$L = J-1 \quad L$
$J+1 = L+2$

6. The ST( ) array: Initialization & updating: — also AD(J) = ϕ at this point

~~for~~ J = 1 to N ST(J) = 1 : Next' N = 10 for initial case.

J = 1 before entering .12 for first time

12: ~~[illegible]~~ INCR ST(J) : ← If J = N+1 goto exit 2 Else.

If ST(J) = J+1 then ST(J) = 1 : ~~[illegible]~~ Incr J : GOTO .12

4 Else / goto Exit 1

SN ~~15~~ 15 ~~[illegible]~~ Starts ~~[illegible]~~ SN15

6 Lϕ J=2 ← experiment to see if "If J = INCR N" works O.K. ← It doesn't

Perhaps AD(J) = 0 at this pt

I.e. can "INCR x" be used as "1+x" in any expression e.g. x=9 print (INCR x)^.5

```
L1: ~~[illegible]~~ If J = N+1 Then ~~GOTO END4~~
                    GOTO END1                     (= INCR N)
      Else / If ST(J) = J+1 Then ST(J) = 1 : Incr J : GOTO L1
                                      INCR ST(J)
             Else ~~[illegible]~~ GOTO : ~~[illegible]~~ L$\phi$   At this point we will exit to the insertion pgm.
```

20 END IF ~~[illegible]~~
END IF

Have it print out SL(J) content for J = 1 to N, for, say N = 4 and all 24 cases.

END1 is for J = N+1 = 11, & is final end of Pgm — and it stops or prints out "END"

~~END2~~ Lϕ is end of this part for the ST( ) update : I goes to T. insertion

| sortn we do need to know value of $\underline{J}$ at exit → END2

L+1 = J in insertion pgm of (75.35-.40) L, 76.00-.10 :

Check on whether it works for extreme values of J : say J = 2 and J = N, & N+1.

J = N and N+1 seem o.k. J = 2 ; ST(2) is inserted — but first, SL(2) = 1 or 2 ; write

28 sort first, then try it ~~[illegible]~~ for J = 2 ; SL(J) = 1 ~~or~~ 2 or 3 .33

This Eureka was correct!

.12-.16 exits END2 with some ST(J) = J+1 ? so I want END2 above

30 Logit variable radix No. — ~~[illegible]~~ But it isn't !

If ST(J) = J !/ Then ! ST(J) = 1 ; INC J !/ Goto L1 ← this passes on overflows

Else

The final version of .16-.20 is **SN15.Bas** in C:\PB35 works fine w. ~~[illegible]~~ N=4 (& prob) N=5

Q: in SN15 do I want ST( ) min value to be 1 or ϕ? or does it make any difference?

33: .28 At exit, the ST( ) has in it $\underset{1}{0}, \underset{2}{0}, \underset{3}{0} \cdots \underset{J}{ST(J)}, ST(J+1) \cdots ST(N)$.

$ST(j)$ ~~[illegible]~~ $j \geq 1$ have all changed & we have to update them. Perhaps in SN15, when ST(J) ~~[illegible]~~ is " reset to 1, AD(J) is set to ϕ See (.14R)

T. ~~[illegible]~~ (initial) (first) values of $\underset{1}{AD(}$ ~~[illegible]~~ are $\underset{1}{N}, \underset{2}{N-1}, \underset{3}{\cdots} \underset{N}{2, 1}$.

If the initial values in AD( ) were all zero, our pgm would probly fill it correctly — putting N into AD(1), & N-1 into AD(2) etc.

40 |

3.9.04

4TM

```
00  Z:  Pgm to find S'th zero in AD(): For X = 1 to S
        A = 0 :
            while AD(A) > 0
                Incr A
            WEND                    INCRA
05      Next X      A is now t. 5th zero in AD() .
```

DIM AD(10), ST(10)

K 'is current Job name for svtn.

08 PR: (see 24.29).

S = ST(K) (S is present state, (25 ε k)

10 Find S'th zero in AD() viz. .00-.05 (write in line) — it is "A"

AD(A)= K (corresponds to S_α, t. AD symbol) (loop)

DECR K : GOTO PR: (.08)

Job K can directly incr ST(k+1) on overflow. If Job k+1 overflows we can incr ST(k+2) ect. We can propagate t. overflow as far as necy. If it overflows at Job 10, we enter special stop state.

Remember All Overflows occur on strings of jobs starting w. Job = 1

When overflow occurs from Job 1 to Job 2 we simply write 1 thru Z in t. first Z zeros of AD( ). This is a slight modifn. of rtn. .01-.05

Perhap all overflow is done by Job#1.

How do we do overflow in Job 1 ? (or Job 2?)

When we enter Job 2, if it is in state 2, we overflow & propagate that overflow as far as is necessary.

T. big problem seems to be: When does t. overflo propagation take place? — before or after t. rest of t. "Job J" ?

As soon as var. o.flo propagation gets to Level K, the k+1th level is first updated in AD(). Then when all lower levels are updated, They are updated in modified form.

The "carry program" is simply t. process of getting a Variable Radix no. in ST().

Once it is in there, we use standard way to insert symbols names into AD(). Every time a overflow string of length L occurs, we have to refill t. entire set of "lower L" "Jobs".

Actually, this isn't so diffy. ! — But first we have to update L+1th job. to 20 ↑ ↑

(40 R) I think in updating AD() after a overflowing pt Length L: First increment t. state of Job(L+1) & update S_{L+1} address.

Next, "Zero" L, then L-1 then L-2 ···; 2 .

I think that puts them in correct order for subsequent updating.

So I'm essentially doing t. Variable Radix Pgm. — But ~~I only change~~ when going from one permutation to next, I usually don't change more than a few symbols.

When I change a lot, it is in a fairly simple (fast) way!

So that's it! write Var. Radix counting pgm, then write update pgm using modifn. of .00-.08 inserting as we move along.

34 In theory, we insert from L+1 first, & we put it at t. $ST(L+1)^{th}$ place — hvr, it would save time in updating by doing the L+1th & t. L then.) th in 1 run of (x).00-.08 .

So first we insert the L^{th} thru $L-ST(L)^{th}$ (S_j's)! We insert t. integers L thru L - ST(L) in t. first ST(L) available 0's. Then we insert ...

4TM

00 + O.K. initial Input is list of addresses that haven't yet been filld by symbols. say the ~~[illegible]~~ Card. of these addresses is k.
If the soln is in state j, ~~[illegible]~~ it prints S_n in the jth of the k addresses
say input is [redacted] Like "~~4~~ k, ~~4~~ 1, 5, 8, 9" (the soln also knows where the 4 addresses, ~~1,5,8,9~~ 1,5,8,9 are) say there is a standard 10 word ~~[illegible]~~ "register" and 1, 5, 8, 9 are the first 4 words ("4" is perhaps the ~~[illegible]~~ 0th word of an 11 word register). If the state of the system is r, it writes S_4 in r'th word. if r is 3, that's the "8" position. I then send 3; 1,5,9 to the soln.
If the calld soln returns with a "done" signal, the soln itself goes to state +1 or state 4; If it goes to "state 5", this is a carry/overflow, — it goes to state 1 and
6 returns "done" to the soln. that calld it.

Initially, all solns are in lowest register state.
AD(J) J = 1 … 10 contains addresses of positions needing to be filld. (addresses = 1, 2 … 10)
~~k~~ k is a variable ~~telling how many~~ saying "first k addresses are relevant."
In AD(J), the J th element has "S_j" init. (S_j is J th symbol type).
I guess AD(J) is same for all soln inputs, but k will vary, also the state of the calld soln.

When a state gets ~~an~~ "carry" as input, it goes to its next state: if this generates a carry, its state → 1 & its output is "carry".

We need an array to store ~~[illegible]~~ the state of each soln.

20 + So we just enter the soln w. the ~~[illegible]~~ name of the job to be done; which is "k" the no of symbols to be permuted. From "k" it looks in the array. Because of the storage of ~~states~~ in AD( ) and k, we don't need a soln. — just a GOTO loop. At exit of routine, k is usually ~~decr~~ decr. Perhaps when it decr → φ the ~~[illegible]~~ soln ends w. output. So "if k = φ then k≠φ, .. ~~[illegible]~~ ~~first~~ exit w. a permutation. There will be another kind of exit when all permutations have been listed. It occurs when, in Job # 10, state → 11 (overflows) — I guess at next request it says "No".

~~[illegible]~~ ST(J) has state of J th job type. ST(J) has values 1, 2 … J

Perhaps AD( ) is also final output of permutation.

.29 So ~~the~~ we enter soln w. k. The soln looks in St(k) to find its present
30 + state: It puts S_j (or simply "J") in AD( ): It puts it into the St(k)th position in AD( ) that has a φ in it. (φ's mark unfilld positions). ~~If~~ If then decr's k & ~~[illegible]~~ GO TO PR (PR is name of soln),

If input is k, and "overflow", it increments its state ~~[illegible]~~ and changes AD( ) accordingly.

If input is k and overflow and ST(k) = k then its state ST(k) = 0 and outputs until it finishes is Overflow, as well as assoc. permutation.

4TM

00 So it looks like/usually not much change betw. successive permutations of the Symbols, S_j.

.01 Update rule: After one incr of t. pgm of (72.24–29)R we have a new IOV:

~~If~~ S_K is positioned first: Then ~~the~~ S_{K-1} to S_1 are positioned in standard order (not backwards).

(Probly if one did it "forwards", it would make no difference).

~~Sr~~ A_r ~~=~~ ≡ t. address of t r^{th} symbol, $(r = 0 \cdots R)$ ← Max value ≡ R!

A_r is t. inverse of t current IOVector) (IOV gives an ordered list of t. symbols $[V_S] = \vec{V}$

When an .01 occurs, we make a small table. A_r. for $r = 1$ to $k+1$. From this table, we can quickly update $[V_i]_1^{k+1}$ w.o. searching.

We can keep on stack; For t k^{th} level of stack: Addresses of $[S_i]_{k+1}^{R}$ addresses of $[S_i]_1^{k}$ ~~for the~~ just after D_{k+1} has carryd. After t carry, $S_i]_1^k$ all revert to values on stack

Not quite! 2 of t addresses have changed! — i.e. t. one formerly used by S_{k+1}, ~~[illegible]~~ i t one now used by S_{k+1}. So we minimally change things by assigning to t. symbol that gave up its D_j "place" to S_{k+1} the ~~[illegible]~~ D_j "place" that S_{n+1} is giving up: So we simply exchange these 2, "places"

Going Back to t Recursive PR: T. PR ~~[illegible]~~ has 2 states: in state 1, it will accept a finite set of S different symbols. its ~~output of~~ first state is t. same set of symbols in t. Null permutation (≡ no change at all). ~~[illegible]~~ it then goes to state 2: i waits for (output request) (that input). On request, its outputs t. next permutation of t. original input set of symbols. When it has completed ~~[illegible]~~ its last permutation, it takes an ~~[illegible]~~ aux output that says "I'm done"

PR ~~works this way~~ constructs these permutations this way! It picks ~~[illegible]~~ its input symbols to be permuted. ~~[illegible]~~ it sets it at one of the S values; The set of S−1 other values are "sent out" to be permuted by PR. When all permutations ~~[illegible]~~ on these S−1 symbols have been done, it fixes the next in the original set of S symbols, and asks for permutations on the remaining S−1 symbols.

The list of S symbols can be stored as a linked list so its easy to remove & restore symbols.

Perhaps as "output activity", have t. srtn write to t. relevant registers, its part of t. permutation. So t. "Job desc" of t srtn! Given a certain set of positions for a set of symbols $S_i]_{i \leq k}$, To ~~[illegible]~~ put out assignments for t. next set of $S_i]_{i \leq k}$.

So, normally, if t. srtn is given a set of symbols to assign ~~places~~ permutations to, it will assign one symbol directly & get t. ~~[illegible]~~ rest done as a srtn call.

T. srtn has as input, t. set of addresses that need to be filld: Also t. set of symbols; (This last will be t. no. of addresses, k ~~[illegible]~~ known since used, means t. symbols $S_i]_1^k$ are to be used. T. srtn also has another (occasional) output, when t. ~~[illegible]~~ permutation it has written is the last one in its repertoire. When t calling routine sees this, it carry's to a next initial symbol.

Try this on p 74!

3.7.04

4TM

00 Actually, t. recursive soln is very similar to t. "variable radix method"

~~Anyway, in PR,~~ When a pgm returns, it has info on whether it did a "carry" or not. (71.29) ≡ "limit state"

In t. ~~Radix~~ VariRadix Method ("VR" Method), we best translate the nos. into seqs by noticing only differences betw successive VR nos. — i.e. the largest digit that changed — we note t. "last carry" — or t. first non-carry. Every time we do a carry we incr a counter.

.05 OK: Try "Var Rad" method: D(J) (J = 1 ... 10 say) are t. values of t. digits; D(J) : goes from 0 to J, say we start w. all D(J)=0. Then: incr D_1 to get new 10 v. Then incr D_1 again & again: Each time D_1 is 1 & overflows, we have a carry to D_2 (.21)

S/N Interesting Math Q! Can ~~Rasa~~ this particular variable radix system express all nos. from 1 to N! − 1 ? (N is largest radix). The "J digit" X (x = 0 ... J) represents

.16 X·(J−1)! . so the VR no. 1 1 2 3 2 represents 1 + 1·2! + 2·3! + 2·4! 2 on

If all digits are at their max value, we have ~~...~~ 1·1! + 2·2! + 3·3! + ... + (n−1)(n−1)!

which = n! − 1, as we'd like it. (I found, empirically that $\sum_{1}^{n}$ ~~...~~ x·x! = n! − 1.

13	n	1	2	3	4	5
14	n!	1	2	6	24	120
		~~1~~	~~3~~	~~5~~	~~29~~	~~120~~
	n!n	1	4	18	96	~~...~~ 600
.17	Σ n·n!	1	5	23	119	~~...~~ 719 = N! − 1

which suggests, that t. system will be a complete representation. To find t. representation of a no., Z, we divide Z by N! & take remainder; divide remainder by (N−1)! ~~...~~ divide ~~to~~ that remainder by (N−2)!, ect. — t. Quotients ~~...~~ are the ~~a~~ VR representation of Z

So, given a no. betw. 1 & N! we can use t. method of (.13−.17)R to find a VR representation and from that a unique permutation of t. n integers.

.20 But I could spend much time on this math curiosity! Go back to (.05)

21 (.07) And so on to higher carrys. When D_9 carrys we stop, Halt, GOTO endd.

So, each time we ask for a new 10 vector, we just incr D_1 and do all needed carrys. We will sometimes have a ~~set of~~ K consecutive carrys, in which case we store k along with its 10 vector, $\vec{V}$. so "$\vec{V}$, k."

```
.24  j=1 → L1 : incr Dj : If Dj = j then Dj = 0;  ~~...~~  j = j+1  goto L1.
                                 Else ~~...~~ goto endd.
```

```
                This needs a bit of polishing, but will give a 10v.
j=1
L1:  If ~~...~~ J = R+1 ~~...~~ then GOTO ~~...~~ ENDD
     Else Inc Dj
          If Dj = J then Dj = 0; J = J+1; GOTO L1
          Else Exit          '(exit goes to end of pgm that
          END IF               ...
     END IF
```

Given the 10v. & possibly t. value of k.

.9 Enter w. ~~...~~ a 10 vector $\vec{D}$ ($D_1 \cdots D_R$)

.6 k+1 is t. smallest value of J ~~for which~~ D_j had "no carry". ~~...~~ So for all J's ≤ k D_j had a carry. This means that ~~all the~~ t. ~~...~~ set of t first k symbols has to be re arranged. Symbols of index > k don't change yet. ~~When t. first k ... Dj's carry, we~~ ~~change~~ for j≤k, all D_j → 0 and we insert the symbols into t. vacancies in j order: — j = 1 in t. leftmost vacancy, j = 2 in t. next leftmost ect. While the k+1 th symbol was no carry, it does change by 1 "vacancy jump". The remaining vacancies are for t. ~~...~~ symbols S_1 thru S_k in ~~d~~ linear order ~~If the k+1 th~~ D_{k+1} → $D_{k+1}+1$ and we have to find that position to t. ~~...~~ Rt of t. previous S_{k+1}

Dat

~~They are in linear~~ order, but perhaps backwards from their S_j ordering.

40

4TM

At input time, PR has just been "call'd": T. system is in one of "k states"
In ~~state~~, it is getting a new input 10 vector. ~~It~~ It ~~[illegible]~~ calls itself, ~~[illegible]~~ modifies the 10 vector w. a new addition, goes into a new state / Then uses this new 10 vector to call itself.

The system could be designed w/o a stack, but with 10 diffnl. srtns that "goto" one another: This might save a few clocks! Anyway, it may be easier to think about in this non-stack form.

Anyway, we input PR w. a 10V (10 vector): its in state k, r
nos. = 1 to 10
"k" means its input 10V has ~~k~~ ~~[illegible]~~ first k components filled out.
its job is to fill out t. rest of t. components in all possl. ways. r (r = ~~[illegible]~~ 1, … 10−k) is the no. of values it has tried for the k+1th 10V component thus far.

e.g. say the 10V = 7, 3, 1, 10, 0, 0, 0, 0, 0, 0 ; r = 0 so, before it calls, it fills in (k = 4)
a 2, which is t. smallest possl. value; Next time, it will do 4, then 5, 6, 8, 9; ~~[illegible]~~. (All nos ≠ 7, 3, 1)
it fills in 2 so 10V → 7, 3, 1, 10, 2, · · · , state is then 7, 3, 1, 10, 2 ; $\overset{k}{4}$, $\overset{r}{1}$ which it puts on stack when
it calls PR with new input 7, 3, 1, 10, 2 :
The system is now in state ~~[illegible]~~ 7, 3, 1, 10, 2, ~~[illegible]~~ $\overset{k}{5}$, $\overset{r}{0}$ it goes to state 7, 3, 1, 10, 2, 4 ; · · · $\overset{k}{5}$, $\overset{r}{1}$
& and calls PR w/ 7, 3, 1, 10, 2, 4, · · · as input.

When a state ~~[illegible]~~ goes to its largest possl. fill-in (~~[illegible]~~ when k+r = 10 e.g. ~~when the~~ 7, 3, 1, 10, 2, 8 ; $\overset{k}{5}$; $\overset{r}{4}$ goes to 7, 3, 1, 10, 2, 9 ; $\overset{k}{5}$ $\overset{r}{5}$) we have (contains all state info.)
a Return of t. 10V, 7, 3, 1, 10, 2, 9 and the ~~returned-to~~ state, which was 7, 3, 1, 10, 2,
goes to 7, 3, 1, 10, 4 + - + - + +

2 entrances to PR: one from CALL, & other from Return.

The "top" set of states is K = 9: ~~[illegible]~~ There is only 1 possl. value for the last non symbol; this is then filled in & t. resultant 10V is a final output of PR. We then have to "unwind" to get t. next output.

T. idea is that each "sub prog" asks t. "subsubprogm" to complete its partial 10V.
Each time ~~when~~ the "top srtn" returns a completed 10V, each ~~to~~ srtn below returns that 10V.

~~Try 4 levels: 4 vectors~~: Each level asks its call to give it t. "next" completed 10V.
Perhaps on "return", Info about completion of that higher level should be given. It would seem that this info would be necessary & sufficient. essential & adequate.
So each return has a 0 or 1 bit saying whether this is ~~[illegible]~~ its "final state" A ~~state~~ level can have a final state only if all states above it are "final".

Or, think of it as a variable radix number. For first digit, radix is 10, next is 9, next 8, etc.
~~For [illegible]~~ to list nos by that radix: From every such Var radix no, we can create a permutation: T. 10's digit gives position of 1, the 9's digit tells where 2 goes, (in remaining 9 positions) ~~and~~ 8 digit tells where in 8 remaining places, 3 goes, etc.

So start w. 1, 1, 1, · · · 1, 1. we "add 1", ~~[illegible]~~ so digit "2" 1→2 : Add again by 2→1 w. a carry
(10 9 8 3 2 1 position possl.) so ~~[illegible]~~ incr digit 3.

This will certainly work, but I'd like to be able to write t. much shorter PR self calling srtn. — also understand how to write such ~~progs~~.

3.6.04
4 PM

SN14.Bas (.20)

00 60.40 4k o.k. 5k runout! This is w. words small places except bytes in Y and BT.

(4.5k) o.k. 4.6k too big (out of Memory) So ~ 4.5k limit.

So, w. 100k By available and 20 By/char, 5k Chars is about ~~right~~.

Well, start out w. small corpus

I will need more Memory for ~~[illegible]~~ Statistical ~~[illegible]~~ Info. on suffixes

Its not clear how the system can start out if we use it for ~~[illegible]~~ suffix comparison.

One possy: First char is 0, next char is 255. Every suffix is > ϕ & every suffix is < ϕ, 255

Seems O.k. I can use the exact same initial node #1.

Y(1) = ϕ, Y(2) = 255. UP(1) = 2; DN(2) = 1 ! Same as present

So this is easier than I thot!

0 So we load our ~~[illegible]~~ Y( ) matrix, starting w. Y(3), up to ~~[illegible]~~ Y(4500).

I restored the FACA(A, B) so it regards Y( ) as a set of strings, The pgm ran —

Gave same output as before! (o) Not surprizing, since all symbols were different, comparison was by terminal symbols only

Substituting "T" for 55 in the input data seq. Gave error: apparently "byte" is not Ascii, but a "Unsigned" no. from 0 to 255.

O.k. so I need the Ascii, ~~[illegible]~~ looking (for ordinary 0 à 255); changing CR, LF to space. Other characters dealt w. in various ways. ~~[illegible]~~ Caps → l.c., Any remaining chars eliminated.

60 chars/line.
36 lines/page
2160 k/page:
So 2 pp = 4,320k.

First try to do PPM.

⊘ SN14.BAS is what I'm working on now.

(SN) I'd like to test SN14.B: One way is to input all permutations of 1 … 10, say. & see if it ends up in rite order each time: To list the permutations: use a stack recursively: Name of pgm is PR: The input to PR is a list of ~~[illegible]~~ k places with their occupants; also a new symbol (not in the list): Its output is a ~~[illegible]~~ new set of lists ~~[illegible]~~ of k+1 places with ~~[illegible]~~ their occupants (The old occupants remain same: new occupant has ~~any~~ each of remaining places.)

No: Its input is a ~~[illegible]~~ config. k out of ~~n~~ positions occupied by integers n down to n−k ~~[illegible]~~ k! ~~[illegible]~~ ~~[illegible]~~ Its outputs is a set of ~~[illegible]~~ ~~[illegible]~~ configs of ~~[illegible]~~ integers I thru k occupying remaining positions.

10 The way it works PR puts ~~[illegible]~~ integer k in each of the k unoccupied places: each of the resultant configs is given as input to PR. So PR simply calls on itself until it gets a ~~[illegible]~~ config. w. only one open place: It puts ~~[illegible]~~ "1" into that place and gives output (w.o. calling PR).

A way to do this: say ~~n~~ = 10: Input to PR is set of k integer giving "positions" of 1, 2, 3 … k; each "position" is from 1 thru 10: no repeats allowed. PR's output is a list of k+1 integers — same as before but w. a diffrnt k+1 th integer (There are 10−k possible ~~output~~ ~~[illegible]~~ values.

At first Glance PR has a 10 vector as input & has k, ~~k~~ vectors as output. We want ~~[illegible]~~ PR to be "clocked" so it presents the next ~~[illegible]~~ 10 vector ~~[illegible]~~ 2160 when we ask for it. So we can have an extra input to PR that asks for "next" output.

40

3.6.04
4PM

SN#3.Bas P3

SN 14.B May be in better shape.

SN15.B } List all n permutations of n integers.
SN 16. B }

```
  ELSE
    P=P(B)
    IF M(P)=0 THEN                    'P has 2 kids
        P(C)=P
        IF MK(P)=B THEN                'B is MKid of P (51.05)
          M(P)=MN(C): MN(P)=MN(B): LK(P)=B: MK(P)=C
        ELSE
          M(P)=MN(B):R(P)=MN(C):LK(P)=MK(P):MK(P)=B:RK(P)=C 'B is LKid of P(51.08)
        END IF
        GOTO L4
    ELSE
          INCR BMX                  'P has 3 kids-needs another Node
        IF LK(P)=B THEN                'B is LKid of P (51.13)
          P(MK(P))=BMX: P(RK(P))=BMX  'Updated w.r.t original P
          R(BMX)= R(P): MN(BMX)=M(P):MK(BMX)=MK(P):RK(BMX)=RK(P)
          R(P)=MN(C):MN(P)=MN(B): MK(P)=B: RK(P)=C
          P(C)=P
        ELSEIF MK(P)=B THEN           'B is MKid of P (51.20)
          P(RK(P))=BMX              'Updated w.r.t original P
          R(BMX)=R(P):MN(BMX)=MN(C):MK(BMX)=C: RK(BMX)=RK(P)
          R(P)=MN(B):MK(P)=LK(P): RK(P)=B
          P(C)=BMX
        ELSE                        'B is RKid of P (51.22)
          R(BMX)=MN(C):MN(BMX)=MN(B):MK(BMX)=B: RK(BMX)=C
          R(P)=M(P): M(P)=0: RK(P)=MK(P): MK(P)=LK(P)
          P(C)=BMX: P(B)=BMX
        END IF
       LK(P)=0: M(P)=0: C=BMX: B=P: GOTO L5
    END IF
  END IF
  GOTO L4
ENDD:
NEXT JJ

FOR J=1 TO KYMX: PRINT USING "###  "; Y(J),: NEXT
PRINT: PRINT
FOR J=1 TO KYMX: PRINT USING "###  "; DN(J),: NEXT
PRINT
FOR J=1 TO KYMX: PRINT USING "###  "; UP(J),: NEXT
PRINT: PRINT
FOR J = 1 TO BMX
PRINT USING "###  ";J,M(J),R(J),MN(J),LK(J),MK(J),RK(J),P(J),BT(J)
NEXT
PRINT "     M   R   MN   LK   MK   RK   P   BT  "
PRINT

J=1
PRINT USING "###  ";Y(J),
L9:
J=UP(J)
PRINT USING "###  ";Y(J),
IF J <> 2 THEN GOTO L9
PRINT: PRINT: PRINT
```

SN13.BAS p2

```
    B=MK(B): RETURN
  ELSE
        B=LK(B):RETURN
  END IF

L2:
 IF FNCA(A,RK(B)) THEN          'Two kids-Update Nodes
        LK(B)=MK(B): MK(B)=RK(B): RK(B)=A
     ELSEIF FNCA(A,MK(B)) THEN
        LK(B)=MK(B): MK(B)=A
     ELSE
       LK(B)=A
   END IF
   M(B)=MK(B): R(B)=RK(B): MN(B)=LK(B): PA=B
   GOTO L4

L3:
   BMX=BMX+1: M(B)=0: M(BMX)=0          'Three kids--New Node needed
   IF FNCA(A,RK(B)) THEN
      MK(BMX)=RK(B): RK(BMX)=A: PA=BMX: RK(B)=MK(B): MK(B)=LK(B)
     ELSEIF FNCA(A,MK(B)) THEN
      MK(BMX)=A: PA=BMX: RK(BMX)=RK(B): RK(B)=MK(B): MK(B)=LK(B)
     ELSEIF FNCA(A,LK(B)) THEN
      MK(BMX)=MK(B): RK(BMX)=RK(B): MK(B)=LK(B): RK(B)=A: PA=B
     ELSE
      MK(BMX)=MK(B): RK(BMX)=RK(B):  MK(B)=A: PA=B: RK(B)=LK(B)
   END IF
   R(B)=RK(B): M(B)=0: MN(B)=MK(B): LK(B)=0
   R(BMX)=RK(BMX): M(BMX)=0 :MN(BMX)=MK(BMX)
   BT(BMX)=1: P=P(B): C=BMX
   GOTO L5

L4:              'Determines DN(A), the Downlink of A
                'And      UP(A), the Uplink   of A

    B=A: P=PA

  LP1:
    IF LP(P)=B  THEN                'Go up tree, trying to stay
      B=P: P=P(B):GOTO LP1            'on left,
    ELSEIF  M(P)=0 AND MK(P)=B THEN  'Exit loop when B is not
      B=P: P=P(B): GOTO LP1           'leftmost kid of P
    END IF                    '

    IF MK(P)=B THEN                 'We want B=(kid to left of B)
      B=LK(P)
    ELSE
       B=MK(P)
    END IF

    WHILE  BT(P)=0                  'Go down tree, sticking to right
      P=B: B=RK(B)
    WEND
      DN(JJ)=B
      UP(JJ)=UP(B): UP(B)=JJ: DN(UP(JJ))=JJ
      GOTO ENDD

L5:     'We enter L5 with addresses for B and C

  IF B=BTOP THEN
     INCR BMX: BTOP=BMX: M(BMX)=0: P=BMX: P(B)=P: P(C)=P
     M(P)=0: R(P)=MN(C): MN(P)=MN(B): LK(P)=0: MK(P)=B: RK(P)=C
```

5.6.07
4PM

'This is SN13.BAS P1

NB: The ~~[illegible]~~ keys to be sorted Must ALL Be Different!
No Duplicates Allowed!

```
DATA 0,255,55,10,50,70,20,80,5,60,52,15
DATA 0,2,1,0,1,2,0,1
KYMX=11
HI=1000
DIM DN(HI) AS WORD   '"Downlink" Addresses for Linked List
DIM UP(HI) AS WORD   '"Uplink" Addresses for Linked Lis
DIM Y(HI)  AS WORD   'Y() stores seq to be predicted

Y(1)=0: Y(2)=255: UP(1)=2: DN(20)=1

DIM M(HI)  AS BYTE  'M=min of middle kid
DIM R(HI)  AS BYTE  'R=min of right kid
DIM MN(HI) AS BYTE  'min leaf
DIM LK(HI) AS BYTE  'Leftkid
DIM MK(HI) AS BYTE  'middlekid
DIM RK(HI) AS BYTE  'rightkid
DIM P(HI)  AS BYTE  'parent
DIM BT(HI) AS BYTE  ' 1 ->bottom node

YY=VARPTR32(Y(1))
DIM AA AS BYTE PTR
DIM BB AS BYTE PTR
SHARED AA, BB, YY, Y()      'Important Line!
BMX=1              'BMX is latest node defined
BTOP=1             'BTOP is top node="Root"

DEF FNCA(A,B)          'CA is TRUE if "A Comes After B"
 ' AA=YY+A: BB=YY+B
 ' WHILE @AA=@BB
 ' DECR AA: DECR BB
 ' WEND
 FNCA= ISTRUE Y(A)>Y(B)          ~~'Y(A)>Y(B) '@AA>@BB~~    'TRUE->-1:FALSE->0
END DEF

N=12: FOR J=1 TO N: READ Y(J): NEXT
M=1
FOR J = 1 TO M: READ M(J),R(J),MN(J),LK(J),MK(J),RK(J),P(J),BT(J)
NEXT
'FOR J = 1 TO M: PRINT M, M(J),R(J),MN(J),LK(J),MK(J),RK(J),P(J),BT(J)
'NEXT

FOR JJ=3 TO KYMX
A=JJ: B=BTOP


START:
   IF BT(B)=1 THEN          'Bottom Node
     IF M(B)=0 then         '2 kids-Insert Key,Update Tree
       GOTO L2
     ELSE          '3 kids-Insert Key->SUB4 Updates Tree
       GOTO L3
     END IF                                    'o
   ELSE
    GOSUB S1: GOTO START   'Not bottom Node-Go down Tree
   END IF

S1:
   IF FNCA(A,R(B)) THEN
      B=RK(B): RETURN
      ELSEIF FNCA(A,M(B)) THEN
```

3.6.04 SN13.BAS

```
4TM  0  5  10 20 50 52 55 60 70 80 255
     0
     --------------------------------------
.00       1   2   3   4   5   6   7   8   9   10  11

          0  255  50  10  60  70  55  20  5   80  52

DN appropriate  ~~10  4  2  7  5  6  3  1  9  10~~

output DN   0  10  8  9  7  5  11  4  1  6  3

      Ahh, the up's look good.
output up   9  0  11  8  6  10  5  3  4  2  7
```

So both seem OK — so it works!!

KYMX OK but KYMX+1 → ∞ loop! KYMX = 11

for "While J < KYMX", I got

```
J 1  9  4  8  3 | 11
  0  5 10 20 50 | 52
```

~~10~~

```
Do until J = KYMX                 J=1
                                  L9:  print (   )
   print  ;  J = up(J)                 J = up(J)
loop                              If J ≠ KYMAX GOTO L9
```

16000
8000

Seems to work fine! I tried a random permutation of the input string — got same output list.

.16 SN13.BAS seems bug free. It orders the addresses Y( ) w.r.t. their contents. The final printout is the properly ordered contents (say they are numbers.)

Next: 1) Get it to sort strings! The input pgm may be complex! For English text, I want to elim. CR, LF; — LF will be detected as always following CR; But changing CR to space may add some info, since the system can't detect 6 range of paragraph CR's. — I don't think BZ2 can detect periodicity. Also, perhaps all punct. but spaces. Use lower case only.

20 So Basic cry based on all Text files is a seq. of bytes. We can work on as bytes, or just use a pre processor pgm. that converts to space + Lcase. 2 spaces at end of sentence. — but all periods should not convert to spaces! — only so .␣ → ␣␣ would be not bad. would elim some periods used in numbers. — Perhaps scan text "by hand" first to decide on "Pre processing".

At first just keep it input normal ascii — Make sure 00 & 255 doesn't occur, ~~AAW@#0, 255~~ 0 can be obtained in custom keybd/ cyber protection systems. But these possibly infrequently, if ever, used on text: But filter it out first.

The sizes of Node params must be Word since the no. of nodes is ≈ the corpus size.

30 The only non-Word param is BT) (actually 1 bit) but we wouldn't save much by using < a word. So 8 words param + 2 up/DN links = 10 words/char of text!

Y itself could be 1 byte: But this does not save much for corpus of N, we will use (10 + 1 Y) × N × 2 bytes or 22 bytes/char; we could go down to 20 by 1 byte for Y( ) & for BT( ).

for 64k available, that's maybe ~ 3k bytes of corpus. Is only 64k available? Maybe more like 200k! In PB35: Nothing on screen but this statement!

Print Fre(-1) gave 100688 (bytes) so ≈ 100k.

" " (-11) " 16.777216 meg Bytes of available EMS memory. — I can use it but it's rather slow. Dynamic Arrays. In fact I'm using dynamic Arrays now.

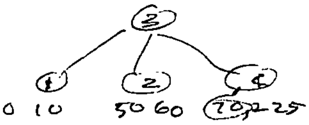
When I put HI 10000 (rather than 1000) it said "out of Memory". 64k all. 7x N.G.

67, 68, 69 are SN13.Bas, Y-sorting pgm → 70

40 — But this was with. Bytes used for Node Params, so prob 2x if I use words

3-5-02 SN 11) .20 SN13 : 24

4 PM SN 12)



0 A = 70

PA = 4

P = 4 3 3 2

A = 70 →

B = 70 4 2 60

P ⇒ B

```
If BT(B) ≠ 0 Then not a bottom node.
  while BT(B) = 0
    ~~P = B~~   B = RK(B)   Gets into ∞ loop.
  WEND
  B = RK(B)
```

BT (B) is not working for if B is a leaf. and of for it will be.

After transfer to down ^trees motion, BT(P) = 1, we ~~[illegible]~~ do UP(JJ) = B otherwise.

10 SN12 B25 seems to work O.K. w. KY = 4 (20 inserts but I'm still not 100% sure of 1 & LE routine.

≤ left no "IF ; end if" : Works only w. KZ = 4

~~So maybe~~ ≡ previous pgm |

~~Back~~ to 6 a 27 and uplinks! 52 gives φ as uplink!

Enter ~~[illegible]~~ ^uplink routine : B = 5~~2~~ ^2 ~~[illegible]~~ P = 2, PA = 2

B	52	2	8
P	2	6	6

was it the dirty Prstm going Down trees Trouble w. Leak ≠ Node ∴ "Parent" of leaf unclear?

Try while

Finally got Uplink to work, But Downlink shouldn't work!

20 ~~SN~~ 12 works

SN1~~2~~ now works

In SN11 I had a "If M(P) = 0 Then" should be If M(B) = 0) ~~[illegible]~~ uplink pgm

Both work : SN11 has simpler format Up link pgm (~~to~~ corresponding closely to Down link pgm)

SN12 has longer UPLink.....

B ← JJ → ·

24 SN13 ~~Be~~ at end of downlink : New uplink pgm. B ← (JJ) →

modular of SN11 PN ~~B~~(JJ) = B ; ~~[illegible]~~ ^UP (JJ) = UP(B) ; UP(B) = JJ ; DN(UP(JJ)) = JJ ← After UP (JJ) is computed.

~~start~~ initialize pgm w. ~~Y~~Y(1) = 0 ; YY(2) = 255 , UP (1) = 2 ; DN(2) = 1 → GO TO ENDD.

To print out : ^JJ=1 : If JJ ≤ KYMX (+2?) then print YY(~~[illegible]~~ ^JJ) ; JJ = UP(JJ)

```
While JJ ≤ KYMX (+2?)          using "/"
  Print Y(JJ) : JJ = UP(JJ)
WEND .
```

30

	1	2	3	4	5	6	7	8	9	10	11
INput	0	255,	50	10 ·	60	70	55	20	5	80	52
~~[illegible]~~ UP	0	0	6	3	0	0	5	0	4	0	7
~~UP~~ DN	0	0	4	9	7	60	11	10	0	70	50

1	2	3	4	5	6	7	8	9	10	11
0	5	10	20	50	52	~~60~~ 55	~~70~~ 60	~~80~~ 70	80	255

Well ; trouble is the simple DN ( ) pgm gets the value of ~~[illegible]~~ the Ring pointed to, rather than its address. This entire ^sorting pgm should be manipulating addresses rather than values (≡ Addr Content)

initial matrix has 1,2 init values than 0, 255

I made "a few changes" : Output looks reasonable!

40

3.5.04

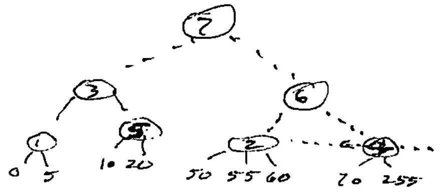
4TM

w. KYMX=7, even tho t. Tree Nodes were meaningless (syntax errors!) UP() & DN() seemed able to get rite ordering!

Computing the uplink can be done purely from down link info, recursively. If all up & down ~~links~~ links ~~are~~ for a seq. are known then to insert a new key, only its down ~~[illegible]~~ links need be known. ~~from the diff. of [illegible]~~ Say DN(A) = X and befor insertion Up(X) = U then UP(A) now = U and we can update all the other 2 Up & down links,

63.40 is a picture of 6 : $\left(\overset{KY}{7}\right)$ inserts 5 & 3 more nodes : just what happened?- how should it have been done?

KY7 !



2 & 4 claim 3 as parent.

Which is OK except that the parental claims of 2 & 4 are wrong. ——

3 was parent for 2 & 4 in KY6 so maybe it didn't get updated. !

3 was split into B & BMX : but BMX kids' parents not updated.

3 breaks into 2 w. "insertion" at L2 (57.13) /// | \

The names of parents of kids were not updated. The kids of BMX didn't have parental update. P(MK(P)) = BMX ; P (RK(P)) = BMX ← ~~for~~ P(P) = B case.

also P(RG(P)) = BMX ~~for~~ MK(P) = B case.

Worked OK. for KY7 For KY8 ; "80" inserted. This goes to Node 4, no big changes (new node(s)) M & R of N4 are ok.

Try KY9 $\overset{key}{(52)}$ breaks up Node 2 so 1 new node : (8) has kids 55,60 ;

~~2 ne~~ N2 has 50,52 ! so superficially KY9 looks ok.

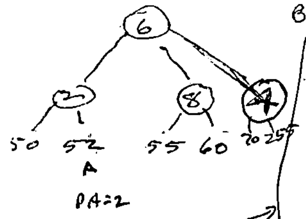
Try ky 10 ~~(4) 70 80 255~~ key = 15, ~~[illegible]~~ no new node(s) expected → N5

Nothing noted of interest.

Ah Bug : 52 (KY9) Up = φ ! should be 55 !

I'd like to know of bug in Uplink pgm, but there now ~~up~~ 2 or 3 line uplink pgm !

Maybe because L3 had no PA output. ; it ~~[illegible]~~ does : its $\overset{2}{BMX}$ or $\overset{2}{B}$



B = 52, PA ~~[illegible]~~ 2 2 , P = PA = 2

~~B = 2, P = 6~~
BMX 8
PA ~~[illegible]~~ 2
P ~~[illegible]~~ 2
B 52
A 52

enter L4

ok ↙
~~6~~
2

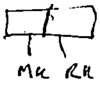
So MK(~~6~~) ≠ 2

A B, ~~P~~, PA, BMX

~~A~~ = 52

~~So apparent error~~ ~~MK(6) = 2, not as ~~8~~ a bug says =8~~ 80 → 52

Pgm said : R~~K~~K(6) = 4 OK.
MK(6) = 8

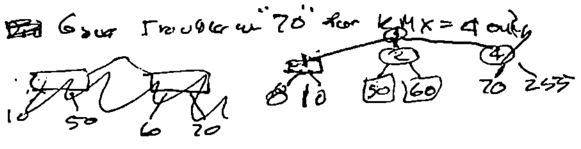


The second "while" has ~~[illegible]~~ P = 6

The previous DN() pgm shouldn't work either! — for same reason:

Gave ~~[illegible]~~ DN() = φ

I think the UPlink pgm is wrong! I changed it in SN12.BAS : see how it works! Didn't work for KYMX=9, insertion of "70" ; ~~[illegible]~~ Gave Trouble w "70" for KMX = 4 only 50,10,60,70 inputs.



3-4-04

4TM

Now if KYMX=8 80 is inserted, It just goes into ~~[illegible]~~ 24 Node#4 and converts it to 3k node.

KYMX=9 3 inserted: again no big deal 2k node → 3k node.

Try to insert 52 instead. It splits Node#2

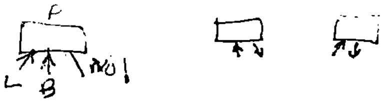
This is SN10.BAS that works thru 52 ; A fair Green Copy: So I'll Just Confine our Fix & ones

It will take fairly large no. of input new inputs to do much splitting., since nodes on levels ≥ 3 are 24 2 kid.

Sub4 seems to be working: try writing up link pgm.

```
~~Continue~~  Enter w.  (S=B)? ~~[illegible]~~  B=A : P=PA

  LP2 :
      ~~If RK(P)=B Then B=P: P=P(B)~~
      While RK(P)=B                          Go down/ tree, stay on (B+)
        B=P : P=P(B) ! GoTo LP2
      WEND
  inline: ( If MK(P)=B Then        We want kid to ~~[illegible]~~ of B
              B = ~~[illegible]~~ RK(P)
            Else B = MK(P)  ;{ MK(P)=B}
                                     | Go down tree. Sticking to left.
      While ~~[illegible]~~ BT(P)=0
    If M(P)=0 , Then P=B: B=MK(B)          stick to ~~[illegible]~~ left.
    ~~Else~~
                  P=B : B=LK(B)
      WEND
    UP
    ~~ZZ~~(JJ) = B                          Z → UP; ZZ → DN
    ENDD.
```



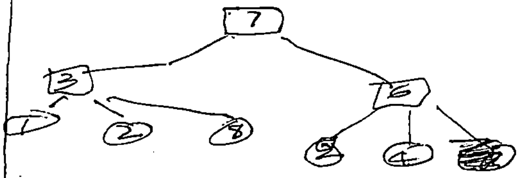
At exit from ZZ(JJ) Stick to Z(JJ)

First try fixing pgm so Sub4 is not a Subrtn, → SN11

I ran it and all UP pointers were OK, ~~except~~ last one: 52 pointed to 0 rather than 80

It worked fine for KYMX=8, but it got wrong on KYMX=9

For KYMX=9



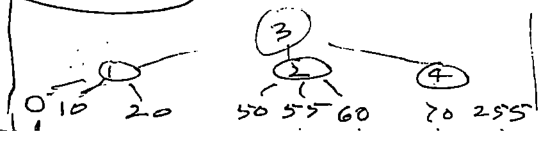
Bug: 1, 2, 4, 5 claim 3 as parent
3 has only 1, 2, 8 as kids

See if matrix for ~~MK~~ KYMX=7 is O.K.   No: 4 claims for 3 as parent. 1, 2, 4, 5 look ur, 3 says 1, 5 only

So KYMX 8 does show trouble in

System: try 8. still same trouble: 4 claims of kids for 3 as parent.) has 7 nodes   So: Bug Charge.
try 6 : seems O.K. only 4 nodes

SN11.Bas



This is KY6 : inp KY7 insert 5 which requires 3 new nodes.
watch what it does to get its where BUG is.

3-4-04

Subsantsu. B, C, ~~P~~ = P(B) F = BMX, PA (Parent of A) ~~[illegible]~~ P B BMX A

inputs sub & ~~R~~ P (loops)! B (≡A), P (= PA = Parent of A) ~~[illegible]~~

Subsanits w. A, PA,

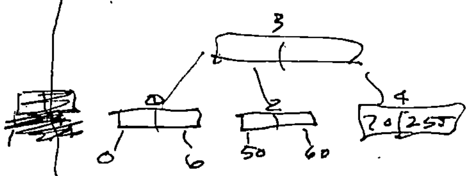
00 B = 4, P = P(4) It should have

OH! Sub4 can't work until 5 is done! Since ~~[illegible]~~ is not complete.)

So in SN10; (changes from SN9): at end of Sub 3 have just GOSUB sub5 (no subs)

at end of Sub5 insert "Go sub Sub4" ~~Subs~~ END

It didn't hang up, but 70 pointed to ϕ!

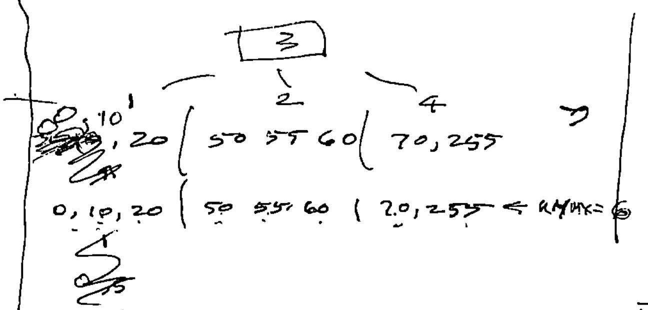


The Node output was correct, ~~but~~ 60 got rite downlink, 70 got ϕ downlink!

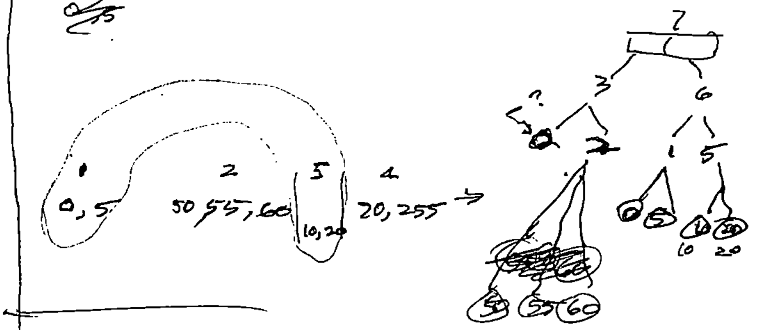
Sub5 seemed to miss jumps to

59

0



2



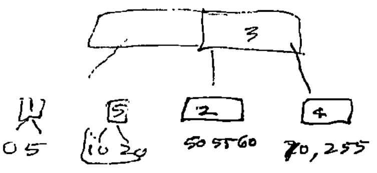
N4 has 3 as listed parent but 3 doesn't have 4 as listed child! (N4 has (70, 255) kids

0

KYMX = 6 was O.K. But when we want to KYMX = 7 with 5 as new key, it had to ~~create~~ many new nodes and it got screwed up. badly. The ZC( ) table was correct for 5, but ~~[illegible]~~ w.

KYMX = 6! .09 - .112

KYMX = 7! (5) inserted into



3 had 1, 2, 4 nodes as kids

~~It was~~ 3 → 0, 1 as kids (← 2?) Thus 2 ~~[illegible]~~ left kid should break into 2 nodes 6 (a new node) should have been having 1 and 5 nodes as kids

It didn't write in SUB3 (bottom node) but same [illegible]

But note that SUB3 & Sub5 have somewhat different probs & solns

4 decisions (SUB3)
i.s. 3 decisions (Sub5)

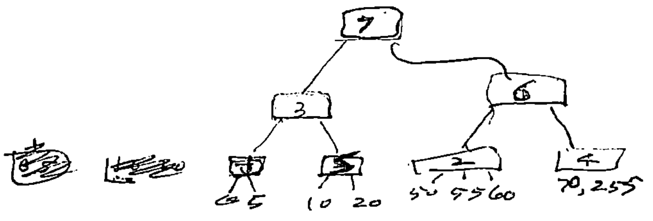
(breakup left node of 3 kid node) ~~was~~ was buggy in Sub5 (see 51.33)

I went over the 51.~~[illegible]~~20 section of Sub5: ~~[illegible]~~ found 1 error.

30 R K(P) = MN(P) should ~~be~~ RK(P) = B ! But this didn't repair Bug!

I did 2 things ~~[illegible]~~ M(P)=0 and ~~[illegible]~~(P)=0 were ~~[illegible]~~ after INKK BMX in the 3rd ~~sit~~ part of Sub5: I removed them to the end of that section, moving M(P)=0 wasn't important but moving LC(P)=0 was, since it was used in that region.

So KYMX = 7 gives O.K. results now. This is SN10. Bas



40

3 TM

I made some important Modifn of the system in 8.Bas:

It gets into a loop in sub1 and start

Somehow Got B=1 & RK(B)=0 ; yet it is initialized to 1.

① 1 1,2 KyMx = 7

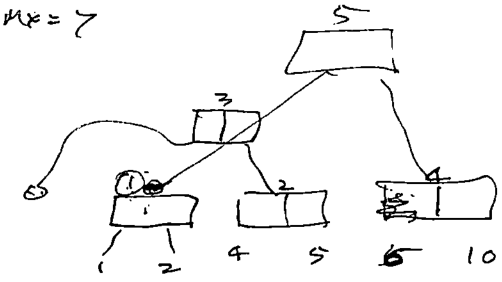
① 2 4,5

ny 3 ~~0,20~~ — no

① 4 10,5

ny 5 ~~6,4~~

① 6 3,5

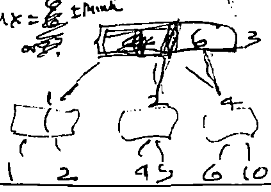


3 doesn't appear as a leaf.

1 2 4 6 5 10 (3)

the earlier (upto 10, but not 3.) structure was.

kyMx = 8 I think



So looks o.k.

To ~~get~~ insert 3, nodes 1, 2, 4 were invariant except for parents.

Bingo ~~step~~ fast until BMX=4, then slow down.

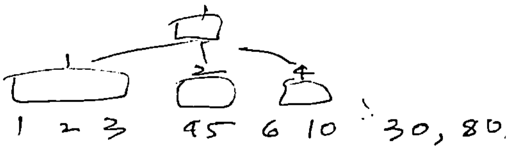
Watch BMX, A, B,

starts w. B = BTOP = 3 (But P=3 !?)

I ~~a~~ modified Sub ~~4~~ : [Instead of GoSub Sob5] I put "return"

GoSub5 is not at all reached!

Soln for 3 insertion



Start w. 0 & 255 in a 2 kid node.

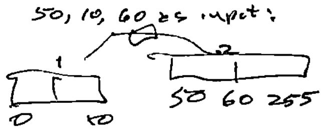
0, 255, 0, 0, 0, 255, 0, 1

SN9.Bas.

~~3 5 5~~ ~~3~~

50, 10, 60, 70, 55, 20, 5, 80, 3, 15

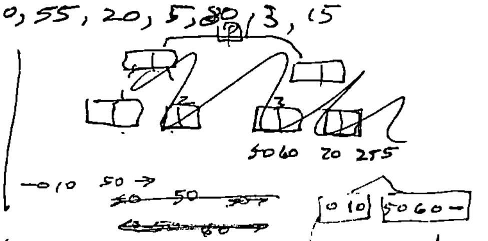
With 50, 10, 60 as input:



So it worked o.k.

KyMx = 3, 4 o.k. o.k.

4 no: it goes into loop!



0 10 50 60/ 20 —

For A 20 it goes into simple loop in sub 4? ① While BTK(P)=0

② P = B B = RK(B)

" " BMX=4 is o.k. ~~[illegible]~~ 20 they print) ③ WEND

During this run.

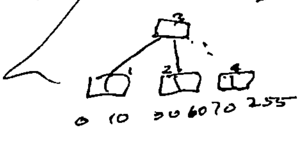
$\boxed{B=0}$

BMX=4

A = 70

How it got B=0 is unclear : which is the problem.

If has to add 1 more nodes, but goes into loop before any more nodes are added!



It enters sub4 w. B=2 (A=70, BMX=4)

B=2 ~~[illegible]~~ is ~~wrong~~ Node 2 — it's looking for 60 (which is <70)

So the bug is in SUB4

No: It should enter Sub4 at Node 4, then go up to 3, then down to 2 (it ends at node 2, starts on Node 4)

Now in ~~50~~ A=70 BMX 3, B=2 insertion = Sub3 BMX→4

On exit from Sub3, B still = 2! BMX=4 So a bug in Sob3!

It should exit w. B = BMX : remembering where it put A! (i.e. BMX in this case

Sub3 needs 2 kinds of outputs : one for Sub5 (which is the present output)

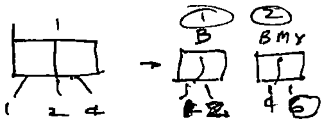
& one for Sub4 ~~(which it doesn't have)~~ that gives "P(A)" — well it does get it as "PA".

3-3.04

3 ~~[illegible]~~ Nite/Note.

>0:59.40 T. presence of KL's at 2aid nodes should cause no trouble in Sub 4; Going up, it checks on M(B) for 2/3 kids
Going down, it sticks to RK, so no problem.
In transition from up to down, it also looks at M(B) for 2/3 kids. } Later: Not True!

So trace it thru for initial



6 should get 4 immediately
6 has just been installd ~~at~~ start of sub4.

PA = 2 → P = 2 ; B = A = 6

~~At last~~ near end of Sub 4 instead of Z(A) = RK(B), I wrote Z(A) = B
I got 2 4's for A = 4 & 10 (!☺).

Hvr, I got

A	1	2	4	6	5	10	20	30
JJ	1	2	3	4	5	6	7	8
Z(JJ)	0	0	0	4	4	6	10	20

correct. wrong. rite

Z(6) = 4 is correct! I've been confusing A with addresses, because of earlier original version in which T. keys were strings.
I can easily fix this by using addresses of A's and Y(A) > Y(B) in the ENCA function.

Grace noticed "while BT(B) = 1" should be "while BT(B) = 0"
if B gets 10 as value, it will get BT(10) = 0 & ~~[illegible]~~ continues forever.
— ~~[illegible]~~ it does get into ∞ loop.

In stepping thru, B = 6, 2, 4 were apparent even when JJ = 4 (only 1 inserted A=6)
BMX = 2 : It said BT(2) = 0 ! (B2 has not yet been created — it has BT(3) = 0. Maybe I forgot to put BT = 1 for the new BMX! ← No, I did update BT(BMX) = 1 in Sub 3. We entered the "while BT(B) = 0" loop with B = 4, the leaf we want to point to. B(4) (B(leaf)) is meaningless in this context.

So this is wrong. I did:

```
while BT(P)=0
  P = B : B = RK(B)
WEND
```

} Seems to work o.k. — for insertion of "6" 1 2 4 ⑥.

The next key 5 doesn't work ritein Sub 4, hvr, — gave 4 as result rather than 6 ok it is rite! ; I forgot to change links. ! ~~1 2 3 4 5 6~~ 1 2 4 6 5 10 20 30 4

Since I only need downlinks, is there a way to not find uplinks?

1 2 4 5 6 10 20 30

→ Some other Bugs: Fix Lk for 2 kid Nodes. When M( ) is updated =0, also update LK( ) = 0.
Also the lowest & hyest ~~[illegible]~~ keys are a problem. Ideally they are ±∞
Perhaps have these 2 points in ~~the~~ initialization node ≡ Node 1.
How to do this in the sequence ordering is unclear; I could do it if there are single symbols that are > or < all symbols in the alphabet

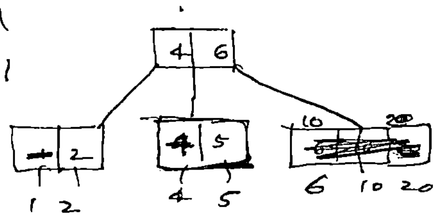
The uplink routine is very similar to the downlink.

So: what parts ~~[illegible]~~ are easy—ie store. Sub 3. Sub 5

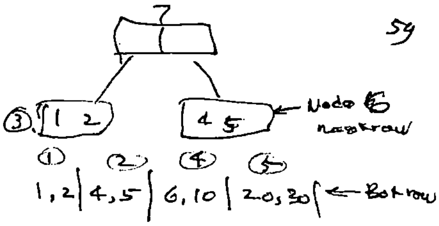
| 2 K. nodes created/ ~~modified~~ (never modified: If changed, they become 3 K nodes.)
from BMX does not need zeroing of M & Lk. From B and from P is impt
~~Actually only one critical place to [illegible]~~

3.2.04

4TM



→



looks like it did its thing but didn't
create a new BTOP !

Another error: Node 3 has 2 leaf kids but
M(3)=4 !! also R(3)=4 ← correct.

The M(3)=4 is due to not updating it as a zero (since it's a 2 kid parent)
The 3 kid parent it split from had M(3)=4 so that's the only error in ~~Not~~ N_3

The ~~Node~~ N6 seems o.k. —— Could the error in ~~[illegible]~~ M(3) ever matter to No 7B node? The updating of M(3) to zero was done; but then there were no other changes: i.e. no new top node) Nodes 3 & 6 seem O.K.

The output of Sub5 doesn't reverse at all: it's
0 20 6 0 4 5 0 0 with forest fooling around, I got
6 20 6 4 6 5 6 0 for N_6 [this is both its own M child & parent!] } i.e. Sortnam ?

Putting B=P: C=BMX at ~~or~~ recursive exit of Sub 5 gave 7 nodes
N_1 & N_2 are kids of N_3 : N3 and N6 are kids of N7 There are no 3 kid nodes.
Seems to look O.K.)

I may get into trouble ~~using~~ ~~[illegible]~~ not updating ~~[illegible]~~ Lk in Nodes w. 2 kids:
So far, the sorting is O.K. but the Z(A) part will not work!.

A very serious bug on input of Sub4! I say " B=A, P=P(B)
~~[illegible]~~ so P=P(A)! well A is a leaf ~~not~~ not a node: it "has no parent" that is listed as such, P(A) will try to find parent of Node "A", i.e. start out w. A=4; no such node: ~~[illegible]~~ At no point was the no. of node = A! so naturally we never got anywhere!

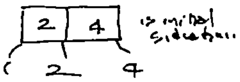
~~The routines~~ at the point of entrance to SUB4, the previous pgm has just ~~[illegible]~~ put A into a 3 ~~kids~~ node or a 2 kid node.
~~[illegible]~~ if 3 kid node, its parent is B: If 2 kid, it can have B or BMX as parent. Whenever we exit with ~~A Node~~ BT( )=1 —i.e. a bottom ~~node~~, we can say ~~what~~ who the parent, P is

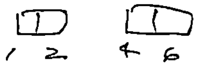
Sub2 & Sub3 are only routines going to Sub4.

Whenever A is put in final leaf row, we write PA=B or PA=BMX, depending on what the parent of A was. We then use PA in Sub4, as A's parent.
So Sub 4 will start with $\overset{S=B}{B}=A$, P=PA.

Sub2 & Sub3 have to have PA assigned. In sub2 (2 kid → 3 kid), PA=B can occur at end: in Sub3 it has to occur ~~whenever~~ whenever A is inserted: Woops! doesn't work! I added PA as a shared variable.
Going thru debug, I got B=10 — this was prob. value of A

In Debug, I'm watching A, B, PA, P, RK(B) : RK(B) is ultimately the ~~[illegible]~~ value of Z(A). I'm starting w. ~~[illegible]~~ inserting A=6 (the 4th key)

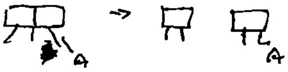


→ 

so it should get RK( )=4 : No way!
So pgm is way off!

3.2.04 58

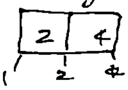
4th



6 just inserted into

this errouous output

50 : 57.40 Params of ~~#~~ #r=BMX node:
m ~~p~~ MN kl ranker p BT
2 4 1 1 2 4 0 1



.0	$\overset{2}{1}$	1	1	1	$\overset{2}{1}$	0	1	← B=1
0	6	$\overset{4}{0}$	0	4	6	0	1	← B=2
m	r	Mn	k					
			2	m	r	p	BT	

06 4.4601
3
MNerror

.1 2 46

0'21-1201 ← errors

R(1), ~~R~~ Rk(1)
MN(2)

Tracing thru ! Start! ! BT(B)=1 M(B)≠0 (=2) ∴ Go to Sub3

Sub3! BMX ; (1⇒2) I fixed 2 lines in sub3 in which statement were in wrong (temporal order). — That fixed Node(1) "B"

BMX still has wrong MN! O.K.! I saw error but didn't ~~patch~~ fix it enough.

I've fixed it for BMX. So both are O.K. now.

~~Try~~ ~~Search~~ ~~Get values of~~ Woops; They have BTOP = 1

So BTOP hasn't been updated it should go from 1 to 3 (but not via "2").

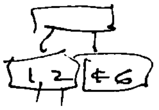
0 Sub4 changes value of B (it should be "ditto" back" (≡ be remembered).

⇒ Chang Sub4 to "Return" ⇒ sacrifices Kinder it

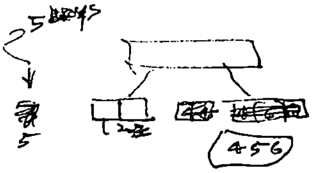
It does give 3 Nodes: "top node =

2 nodes	ok	ok	ok	ok	ok	no point	no error
0	4	1	0	1	2	0	0

~~So the five nodes~~ is ok. So we have



insert 5



Then insert 10 (6 keys, still 2 levels) which splits one node. Then insert 3 gives 3 levels. (7 keys.)

So 5 first ~~[illegible]~~ N_1 same.

	ok ok	ok	ok	ok	ok	p	BT	
N2	5 6	4	4	5	6	3	1	— So N2 is O.K.
	ok	r	MN			ok	ok	
	0	4	1	0 1	2	0	0	N3 ok
						p	ok	
20 W have N2	0 5	4	4	4	5	3	1	

1 2 456

Insert 10 456 should split 12 same.

It didn't insert 10

insert 10: N2: 0 5 4 ($\overset{new}{4}$) ~~54~~ 5 3 1 ← B only 3 nodes!

N2 is correct! But N_3 is badly screwed up. (looks like Top Node

	3 nodes ok	ok	ok	No	ok	ok	ok	
N3	4	6	1,	0	2	4,	0 0	Not updated! should be 1

3
1 2 4
1 2 45 :: 6 10

So N_3 o.k. but one error, N4 missing! O.K. I should print from ~~[illegible]~~ Node 1 to Node BMX not BTOP!

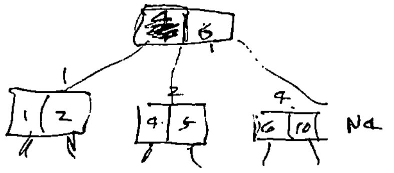
N4 0 10 6 0 6 10 3 1

LK(P)

This error is in update of P(LK): 51.25 says it need not be updated — its invariant! That was an update w only 3 statements (usually 4 are ord).

30 No, But it's not it its case 2b ~~[illegible]~~ 51.25!

That fixes it O.K. (0 → 1) other Nodes ~~are~~ unchanged by this "fix".



Could I have made this error other places?

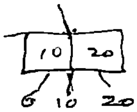
Well anyway : insert ~~[illegible]~~ 20 This just ~~[illegible]~~ gives ~~node 1 = 1 2 3~~ Node 4 → 6 10 20

Then insert 30 & see how it goes.

50 N4 becomes | 10 20 6 6 10 20 3 1 |

other nodes invariant. Worked fine

N4 : ~~6 10 20~~



Try 30 : This creates 3 new nodes! — so 7 nodes! KMX=8

woops! only 6 nodes listed. Nodes 1, 2, ~~3~~ invariant. only 2 non-bottom Nodes

40

4TM

Tentatively, try to put ~~[illegible]~~ A=3 into an empty B=1

Mistakes in pgm! (1) set M=0 for reading in M(j), R(j) etc.

(2) for ~~[illegible]~~ JJ loop JJ=1 to 1 ; Data for ~~[illegible]~~ Y(1)=7

This sounds unlikely!

The pgm should start w. 2 keys in (N. BMX) Next data from Y(3)

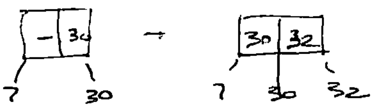
Y: ..7, 30, 32, 6, 4, 90, 3, 80

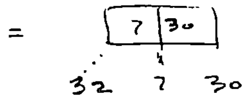
M r MN LK MK RK P CT ~~[illegible]~~

0 30 ~~30~~ 7 0 7 30 0 1 ~~[illegible]~~ ← initial BMX

→ 30 32 7 7 30 32 0 1 ← result should be

Got 7 30 32 32 7 30 0 1 ← Got.

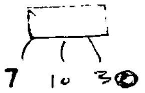




Looks like it put 32 in wrong place..

try inserting 10 instead of ~~32~~ 32

~~[illegible]~~ 30 10 7 7 30 10 0 1 ← got



4,4,0 MK(B)

4,2,0 RK(B)

→ The older pgm ~~B~~ SMNGA.BAS I had looking a LK input table from

input Node: m R MX | L M R

~~[illegible]~~ start ~~start~~! 0 2 1 | 0 ~~4~~ 1 2 0 1

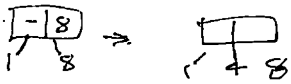
Got output 1 2 1 | 0 2 0 1

Input: A=1 putting in 1 is Bad! I is in a board.

Change ~~2 50 [illegible]~~ 2.60 8 input 4

Start 0 8 1 | – 1 8 0 1

Insert A=4 4 8 1 | 1 4 8 0 1 Seems right!



Using initial Node 1 = ~~start~~ 0 2 1 0 1 2 0 1 I put A=4 into 2 pgms ! output

α 4 2 1 1 4 2 0 1 — out of SMNGA.Bas ~~Sortnam6.B~~,

Wow! β 1 2 4 4 1 2 0 1 — out of Sortnam6.B ~~[illegible]~~ (More recent w. For Next loop)

In α looks like 4 was put in as 1 . 4 . 2

" β " " 4 " " " 4 . 1 . 2

A, MK(B)

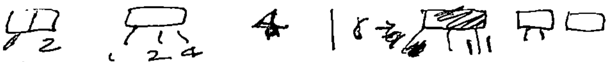
It got FN CA(4, MK(B)) = 0

T. Bug was in data of FNCA: I had Y(A) > Y(B) instead of A > B.

Putting in A > B got right result in old Sortnam6.B.

try fixing SNMGA.Bas. Now gives same ~~[illegible]~~ (correct) results as Old Sortnam6.Bas.

i.e. 2 4 1 1 2 4 0 1



I Got it backwards! SortNAM6.Bas has f. loop to insert A values,

I did 2 insertions first 4, as before, then 6. got

1 | 1 1 1 1 1 1 1 0 1 clearly wrong!

2 | 1 6 1 0 1 6 0 1

Try starting with and inserting a 6. — Got same silly result!

IN SUB 3 : I had to rearrange order of updates so BMX was always updated first (before B) was.

Also I forgot to write B(M) = BMX(M) = 0.

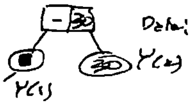
Am now working on SUB5 (I did Sub3) checking for M( ) = 0's

also ~~[illegible]~~ that BMX should be updated ~~before~~ Prior ... whatever

SORTNAM6.BAS Has all my corrections

2·29·04

4PM



	M	R	MN	LK	MK	RK	P	BT
Node no.	1	2	3	4	5	6	7	8
Data: 1	0	2	?	0	7	2	0	1

∞ :

```
N=1
For J=1 to N:
    Read M(J), R(J), MN(J), LK(J), MK(J), RK(J), P(J); BT(J).
    Next
For J=1 to 10: Read Y(J): Next
Data 1, 30, 32, 6, 4, 90, 10, 9, 3, 80

N=3

for J=3 to N
    (Run pgm): Next.
For J=1 to BMX
    Print using "##w"; M(J), R(J), MN(J), LK(J), MK(J), RK(J), P(J), BT(J).
    Next
    First, hnn,
    For J=1 to N, Print using "###ww"; Y(J); ~~Next~~
    Next
```

→ Types of srtns! (4) simple, ends in Return, no side exits — Z(A) compu.

5 is not a sbtn — it ends in Terminus as now. terminus → EXT.

~~3~~ does its "thing" then sbtn then 5 so it is not a srtn really, } BoM the Bottom node.

(2) It returns, no internal exits — true srtn.

1 " "Returns" ——— but actually calls sub5 which ends in Terminus.

Sub 5 goes up tree updating split nodes (for 2 or 3 kids): it always ends in EXT.

3 for 3 kids: splits node, and updates 2 resultant nodes.

So only 4 is needed as srtn: Others are simply GOTO's, that eventually go to End.

Maybe 1 is a srtn? → It is: It has an aux exit viz sub5, hrm

So 1 & 4 are srtns,

1 and 4 are srtns: 4 has no internal exit

1 has a possl. internal exit.

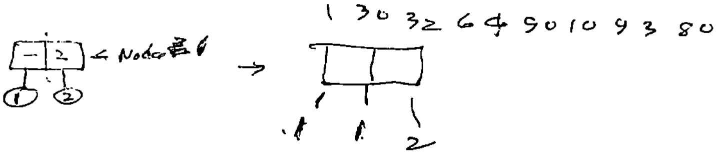
2, 3, 5 are not srtns.

Start is a loop.

Maybe change names of srtns & tables, so its clear which is which.

Tables must be followed by Colon:

Start: Node 1 = 0 2 1 0 1 2 0 1 —
M R MN LK MK RK P BT



A=1; B=1 try A=3

0 2 1 0 1 2 0 1

input 3 2 3 1 1 2 3 0 1
0

Start w. 1, 10 then add 5, 7

```
To do entire pgm!  For JJ = 1 to 2 or 3 ....
                       A = Y(JJ)
                         <pgm>
                       next
```

Put desired sequence in Y

2-29-04

4TM

00:54.40 54.36-.40 sub4 ! I have to Explicate: "while (leftmost kid of P is B)"

```
If ( L(P)=φ and Mk(P)=B )          (L(P)=φ is α, Mk(P)=B is β)
or Lk(P)>0 and Mk(P)=B             (Lk(P)>0 is γ, Mk(P)=B is δ)

If [(α AND β) or (γ and δ)]

If α Then If α Then — Else

If α Then
    If β Then x=1
    Else x=0
If γ Then
    If δ Then x=1
    Else x=0
End If

If Lk(P)=B
Else If
```

Perhaps just write up the condition in simplest possl. way: using PB35 language. Later, it will be relatively easy to write a fast Mach Lang Macro that does the Same Thing, but much faster, using compares, flags & Jumps.

If Leftmost kid of P is B.

```
loop
    If Lk(P)=B Then —
        B=P; P=P(B): GOTO Loop
    Else if Mk(P)=B AND Lk(P)=0 Then
        B=P; P=P(B): GOTO loop
    END If ;
        ↓
    Breakout of loop.
```

NO! ~~B = Second leftmost kid of P~~

```
If Lk(P)=0 then ?
    B = Rk(P)
Else B = Mk(P)
```

```
~~If Lk(P)=0 Then~~
~~    B = Rk(P)~~
~~Else B = Mk(P)~~
~~End If~~
```

B = second leftmost kid of P

NO! we just want it to be rt of B.

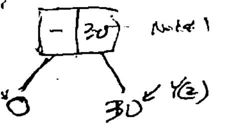
instead

```
If Mk(P)=B Then
    B = Rk(P)
Else B = Mk(P)
```

```
While BT(B)=1
    B = Rk(B)
WEND
Z(A) = Rk(B)
```

Start ~~subtract~~ sub4 with: B=A ; P=P(B).

Testing : initially:



This will be Sortnum5.BAS

Sortnum5.Bas will use correct FNCA data "BYTE"

So Y( ) will contain instead integers >0 <256

Change definition of FNCA so its a simple Sorting program with Magnitudes.

So

```
DEF FNCA (A,B)
    FNCA = ISTRUE Y(A) >Y(B)
END DEF
```

Say ~~[illegible]~~. I want to be able to insert successive values of Y into the Sort pgm. I fill up Y (by hand) Then use that pgm to sort up to N, say (N=3 to start)

The output will be the Z( ) array, and the contents of all of the Nodes up to BMX.

I can get 8 columns of data for each Node; 3 digits per column + 2 spaces = 5 per line : ×8 cols = 40 cols only, fine.

We can Run it on Grace's machine & printout on her old printer.

4TM

00 In Sub4 I only considered 2 cases: P had 2 kids; P had 3 kids:

I did not consider case that ~~P=~~ P = ~~[illegible]~~: BTOP : re. P had no parent.

~~Start w IF~~ sub4 start w. B & C. If B has no parent we have special ~~[illegible]~~ prom.

our NSub4 ~~[illegible]~~

```
If B = BTOP Then          'B has no Parent (Yet).
    [ α ]
Else P = P(B) ; Goto ~~Sub4~~, ~~Sub4~~ ~~(5.Bas old Sub4)~~ old suy4
ENDIF
```

No!

08 ~~ENDIF~~ α: INCR BMX. BTOP = BMX

P = BMX ; P(B) = P P(C) = P

0 M(P) = 0 ; R(P) = C ; MN(P) = MN(B) ; ~~M~~K(P) = B ; RK(P) = C ; BT(P) = 0 unnary

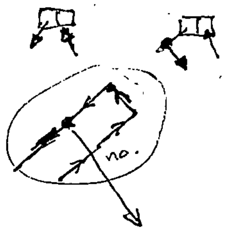
Do Sortnam 5. Bas. !

I made lines .08 -.10 part of Sub4

Note: T. prog itself now is called Sortnam5.Bas 2-28.04 In C:\PB35 directory on HP500.

Sub5 now consists of old SUB4 plus 54.08-10

18:52.80 Sub4 is worked on : 52.6 ff: 52.24 -.33 is a flow diagram for 2 kid nodes only:

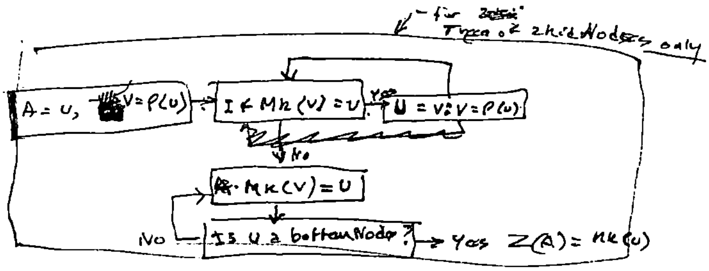


0 Rule: Go up, keeping to the Left. As soon as a node is found in which we are not left, go to the nearest left node and go down, sticking to ~~the~~ rt

P
B

A → P(A)=P if MK(P) = A then

```
A=U, ~~P(U)~~: if MK(V) = U, → ~~[illegible]~~ U = V, V = P(U)
       V=P(U)
```



Works for mixed 2 & 3 kid nodes

Go "up" (toward Root) along leftmost node: as soon as this is impossible.

0 Go down (away from Root) along rtmost node until you hit a Key : Put key in Z(A).

```
P ~~B = P(B)~~, ~~IF~~ B=P; P = P(B) if (Extreme ~~w~~ left kid of P = B then (B = P , P = P(B))
                              = P(P)                 α
If (α is false) Then B = MK(P) or B = RK(P) (whichever is ~~[illegible]~~ just left of)
→ If BT(B) Then Z(A) = RK(B)
    Else B = ~~[illegible]~~ RK(B)
```

warm start w. special key at "φ" (≡ -∞)

.36

```
While leftmost ~~kid~~ kid of P is B
    B = P; P = P(B)
WEnd ~~[illegible]~~.
B = second leftmost ~~kid~~ kid of P
```

```
While BT(B) = 1
    B = RK(B)
WEnd ~~[illegible]~~
Z(A) = RK(B)
```

10

2.27.04

4TM

How is srtage 51.00-.40 4TM ~~[illegible]~~, We enter w. B, BMX and P(B):
Then we do C = BMX to remember old BMX befor we change it.
then look at P(B) is P a 2 or 3 kid node (i.e. is M(P) = 0 or >0?
(2kid ↓ 0; 3kid ↓ >0)
If 2kid, we enter at ~51.01 and we have 2 cases: (2a) of 51.03 (CM(P)=B) and (2b) of 51.08. ~~[illegible]~~ CR (CR(P)=B). This 2 kid srtn ends - it goes no further.

Srtn ~~[illegible]~~ part for M(P)>ϕ starts ~51.14 : 3 possl cases:
(LK(P)=B / 51.14), (MK(P)=B / 51.20) (RK(P)=B / 51.23) ~~The output is P, BMX and P(B)~~)

We first do ~~[illegible]~~ we enter with B, BMX, P(B).
We first do ~~INCR BMX~~ C = BMX, then INCR BMX. So we have one new node, BMX, ~~one~~ and one old node, P(B); These are now both 2kid nodes and we want to insert B, C into them. — which is what srtn does.
We exit with B = P(B) and C = BMX,) ~~[illegible]~~ ← This is the update B ← Now P(P(B)) = P(B)
This srt part occurs on exit of Sub 3 of 48.25

→ More careful version of 51.00-.40 !

→ Enter from exit of Sub 3 of 46.25 ! ~~[illegible]~~ we have B, BMX ~~and P(B)~~ (C = BMX): B and C have to be inserted into node P(B)

```
Sub 4
IF M(P)=0 ~~[illegible]~~ Then          ' ~~[illegible]~~ ~~2kids~~ P has 2 kids  | See P51 for diagrams !
    If MK(P)=B Then                      ' B is an Mchild of P ) ~~Case 2a~~ (51.03)
        M(P)=MN(c) : MN(P)=MN(B) : LK(P)=B ; MK(P)=C
    Else  M(P)=MN(B) : R(P)=MN(c) ; MK(P)=B : RK(P)=C      ' B is an Rchild of P
                                                           ' case 2b (51.08)
    END IF
~~ELSE IF LK(P)=B Then~~
~~ELSEIF LK(P)=B Then~~                    ~~' 3kids~~
        ~~R(P)=MX(c); MX(P)=MX(B) : MK(P)=B ; RK(P)=C~~
~~ELSE IF MK(P)=B then~~
        ~~R(P)=MN(B); MK(P)=LK(P)~~
    ~~INCR BMX~~
    ~~IF MK(P)=~~
Else  INCR BMX                           ' P has 3kids.
    IF LK(P)=B then                      ' B is a Lchild of P
        R(BMX)=R(P): MN(BMX)=M(P): MK(BMX)=MK(P): RK(BMX)=RK(P)
        R(P)=MN(c): MN(P)=MN(B): MK(P)=B ; RK(P)=C
    ELSEIF MK(P)=B Then                  ' B is a Mchild of P
        R(BMX)=R(P): MN(BMX)=MN(c): MK(BMX)=C: RK(BMX)=RK(P)
        R(P)=MX(B): MK(P)=LK(P) ; RK(P)=MX(B)
    ELSE ~~IF RK(P)=B~~ ~~Then~~                ' B is a Rchild of P
        R(BMX)=MN(c); MN(BMX)=MX(B): MK(BMX)=B : RK(BMX)=C
        R(P)=M(P): RK(P)=MK(P) : MK(P)=LK(P)
    END IF
ENDIF
```

4TM

How. srtn as of 51.00-to fit in ~~[illegible]~~. We enter w. B, BMX and P(B):
Then we do C = BMX to remember old BMX before we change it.
Then look at P(B) is P a 2 or 3 kid node? (i.e. is M(P) $\overset{\text{2kid}}{=0}$ or $\overset{\text{3kid}}{>0}$?
If 2 kid, we enter at ~51.01 and we have 2 cases: (2a) of 51.03 (C M(P) = B) and (2b) of 51.08. ~~[illegible]~~ CR (C R(P) = B). This 2 kid srtn ends - it goes no further.

Srtn ~~[illegible]~~ part for M(P) > φ starts ~ 51.14 : 3 possl cases:
(LK(P) = B / 51.14), (MK(P) = B / 51.20) (RK(P) = B / 51.23) ~~The output is P, BMX and P(B)~~

We first do ~~[illegible]~~ We enter with B, BMX, P(B).
We first do ~~INCR BMX~~ C = BMX, then INCR BMX. So we have one new node, BMX, ~~one~~ and one old node, P(B); Those are now both 2 kid nodes and we want to insert B, C into them. — which is what srtn does.
We exit with B = P(B) and C = BMX.] ~~[illegible]~~ This is the update B → Now P(P(B)) = P(B)
This srt part occurs on exit of sub 3 of 48.25

→ More careful version of 51.00 - .40 !

→ Enter from exit of sub 3 of 46.25 ! ~~[illegible]~~ we have B, BMX ~~and P(B)~~ (C = BMX): B and C have to be inserted into node P(B)

```
If M(P) = 0 ~~[illegible]~~ Then            ' ~~[illegible]~~ ~~[illegible]~~ P has 2 kids
    If MK(P) = B Then                       ' ~~(Case 2a~~
        M(P) = MN(C) : MN(P) = MN(B) : LK(P) = B ; MK(P) = C
    Else M(P) = MN(B) : R(P) = MN(C) ; MK(P) = B : RK(P) = C   ' case 2b
    END IF
~~ELSEIF LK(P) = B Then~~
~~ELSEIF LK(P) = B Then~~                                     ~~' 3 kids~~
    ~~R(P) = MX(C) ; MX(P) = MX(B) : MK(P) = B~~ ; RK(P) = C
~~ELSE IF MK(P) = B Then~~
    ~~R(P) = MN(B) : MK(P) = LK(P)~~
  ~~INCR BMX~~
  ~~IF MK(P)~~
Else INCR BMX                               ' P has 3 kids.
    IF LK(P) = B Then
        R(BMX) = R(P) : MN(BMX) = M(P) : MK(BMX) = MK(P) : RK(BMX) = RK(P)
      - R(P) = MX(C) : MX(P) = MX(B) : MK(P) = B : RK(P) = C
    ELSEIF MK(P) = B Then
        R(BMX) = R(P) : MN(BMX) = MIN(C) : MK(BMX) = C : RK(BMX) = RK(P)
      - R(P) = MX(B) : MK(P) = LK(P) ; RK(P) = MX(B)
    ELSE ~~[illegible]~~ ~~[illegible]~~
      - R(BMX) = MX(C) ; MX(BMX) = MX(B) : MK(BMX) = B : RK(BMX) = C
        R(P) = M(P) : RK(P) = MK(P) : MK(P) = LK(P)
    END IF
ENDIF
```

2·27·04

4TM

00:51:40 On the linked list setup: We only need link to ~~lower~~ key.

Say we have just inserted a key, This setup will come $\overset{\text{note}}{/}$after Sub2 into after Sub3.

After insertion into 2 kid → 3 kid node: "B" is name of "Present 3 kid node.

A is name of inserted key. After each "~~Ablittle~~" ——— = A" expression,

.04 ~~If~~ unless A is an extreme left child, link to lower key is trivial and we can write suitable Z(A)=

If .04 is not true: / $\overset{\text{then}}{\text{A}}$ is extreme left child: If "P" is address of parent (P will be B or BMx)

Keep ~~P~~ P = N(PP, 7) is / $\overset{\text{next}}{\text{Node}}$ up tree.

If $N(P,1) = \phi$ and $N(P, \overset{mc}{5}) = A$

Going up: stick to left extreme: until impossible. Then ~~go~~ go down, pick branch just Lofted Address, then go down, Picking highest branch each time.

Say we just ~~put~~ inserted A into a new 3 kid node.

10 Name of Node B if $N(B, \overset{lk}{4}) = A$ then

$P = N(B,7)$: If $N(P,1) = \overset{+\infty}{0}$ and $N(P, \overset{mk}{5}) = B$, then $\overset{B=P}{P = P(N,P)}$

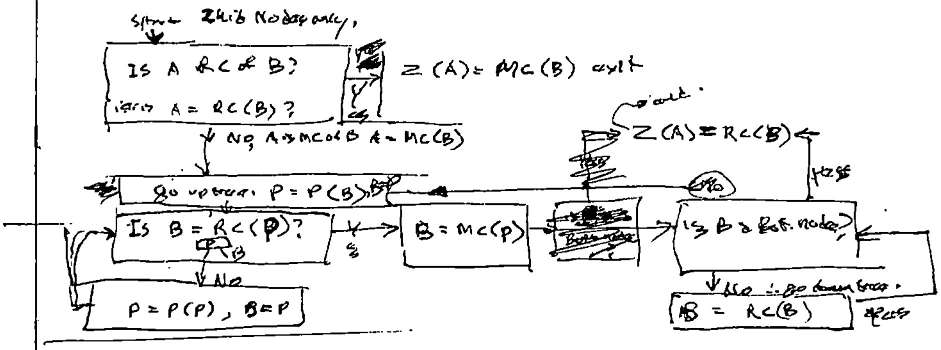
If $N(P,1) \neq 0$ and $N(P, \overset{rk}{4}) = B$ " " " "

Th. Fogg is confusing: Say all Nodes were $\boxed{2 \text{kid}}$. To simplify problem.

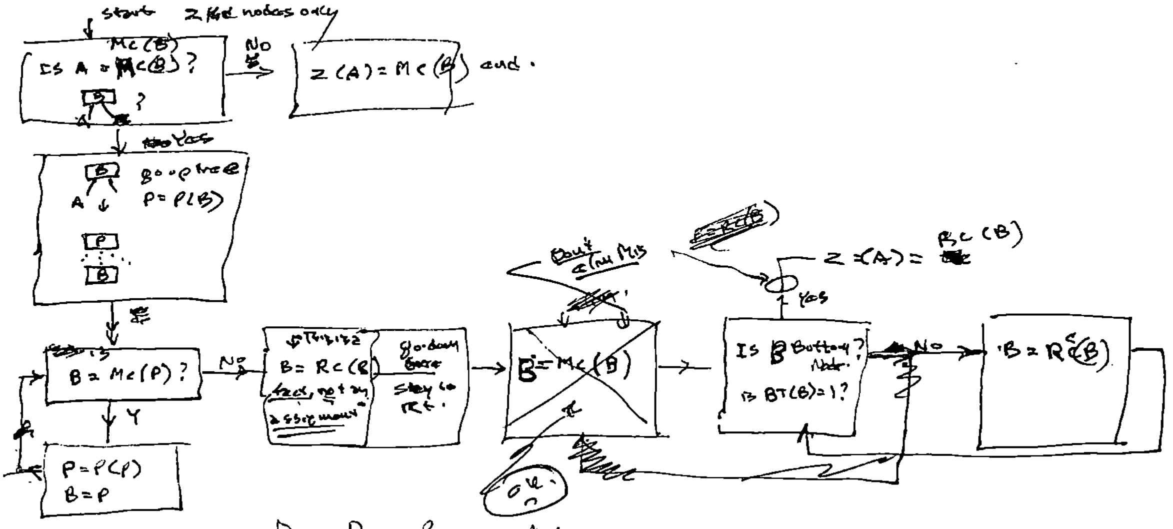
A's parent is $\overset{\text{Bnodes}}{(\text{B node})}$ $\overset{\text{loop}}{\alpha}$ If RK(B) = A then Z(A) = MK(B) Exit.

Else (i.e. MK(B) = A) then P = P(B) : If MK(P) = B then B = P ~~[illegible]~~ P = P(P)

20



.24



30

.33

Drop this for a while.

→ 54.18

90

4 TM
51.40

longest match.
KvUR: longest Common prefix

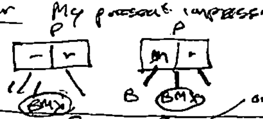
N(P,)
This is 2 kids,

00: I want to write up ~~[illegible]~~ Case ~~[illegible]~~ 2a of 49.24

I wrote $N(P,7) = N(B,7)$ (hvr. $N(B,7) \geq P$ It looks like ~~[illegible]~~ 49.25-26 is much in error — I think confused $N(\_,7)$ w. $N(\_,3)$

My present impression:

Case 2a:



defined by

Case 2a	Case 2b
$N(P,\overset{cm}{5}) = B$	$N(P,\overset{cr}{6}) = B$

only 4 update

Rosemary:

1	2	3	4	5	6	7
M	R	Mn	Lk	Mk	RM	P

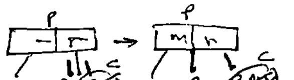
8 B (Bottom) node

New notation
$N(1000,2) \rightarrow$
8 Arrays line
~~N~~ $M(1000)$
$R(1000)$ ect.

.05 If $N(P,5) = B$ Then $\underline{N(P,\overset{m}{1})} = N(\overset{c}{\cancel{BMX}},\overset{min}{3})$; $N(P,\overset{r}{2}) = N(P,\overset{r}{2})$ invariant.
$\underline{N(P,3)} = N(B,3)$: $\underline{N(P,4)} = B$, $\underline{N(P,5)} = \overset{c}{\cancel{BMX}}$, $N(P,6)$ = invariant

$N(P,7) =$ ~~[illegible]~~ leave this alone ~~[illegible]~~ — invariant $N(P,8) = 1$ — invariant

.08 Case 2b



only 4 updates.

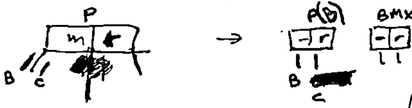
Else
[If $N(B,6) = B$] — this is only alternative so it's an "Else"]

10 $N(P,\overset{m}{1}) = N(B,3)$; $N(P,\overset{r}{2}) = N(\overset{c}{\cancel{BMX}},\overset{min}{3})$; $N(P,3) = N(P,3)$ invariant ; ~~$N(P,4) = N(P,4)$~~ $N(P,4) = N(P,5)$ 5 equs.!
$\underline{N(P,5)} = B$; $\underline{N(P,6)} = \overset{c}{\cancel{BMX}}$; $N(P,7)$ invariant, $N(P,8,)$ invariant

only border kid update.

.13 Ok. Next following ~~[illegible]~~ 3 kids Sub3 of 48.25: If $N(P,4) = B$ Then

This is common for all posts following Sub3 so it's in codedit on 46.15
B = P
c = BMX ; ~~[illegible]~~ INCR BMX

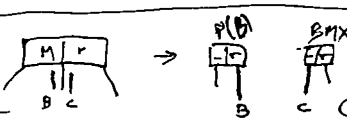


Next $N(P,\overset{m}{1}) = N(c,3)$: $\underline{N(P,3)} = N(B,3)$: $N(P,4) =$ None ; $\underline{N(P,5)} = B$: $\underline{N(P,6)} = c$ (4 updates)

No N(BMX,1)

first → Do N(BMX,) first $\underline{N(BMX,\overset{r}{2})} = N(P,\overset{r}{2})$: $\underline{N(BMX,\overset{mm}{3})} = N(P,\overset{m}{1})$: $N(BMX,4) =$ None $\underline{N(BMX,5)} = N(P,5)$ $\underline{N(BMX,6)} = N(P,6)$ 4 updates

Then $N(P,2)$ Because N(BMX uses "P" info ; N(P) update changes P info

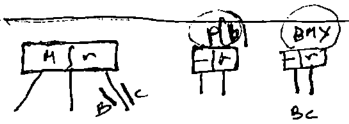


If $N(P,5) = B$ Then

20 ② $\underline{N(P,\overset{r}{2})} = N(B,3)$; $\overset{inv.}{N(P,3) = N(P,3)}$; $\overset{invar.}{(N(P,4) = \Lambda)}$; $\underline{N(P,\overset{cm}{5})} = N(P,\overset{cr}{4})$; $\underline{N(P,6)} = \cancel{N(B,3)}$ B

Do N(BMX,) first ~~[illegible]~~ → ① $N(\underline{BMX,\overset{r}{2}}) = N(P,2)$: $\underline{N(BMX,\overset{mm}{3})} = N(c,3)$: $N(BMX,4) = \Lambda$: $\underline{N(BMX,\overset{cm}{5})} = c$: $\underline{N(BMX,6)} = N(P,6)$

.22



This is Else
If $N(P,6) = B$
Since this is the only other poss.

Else

① $\underline{N(BMX,\overset{r}{2})} = N(c,3)$: $\underline{N(BMX,3)} = N(B,3)$: $N(BMX,4) = \Lambda$: $\underline{N(BMX,\overset{cm}{5})} = B$; $\underline{N(BMX,\overset{cr}{6})} = c$ 4 updates

.25 ② $\underline{N(P,\overset{r}{2})} = N(P,\overset{m}{1})$; $\underset{inv}{N(P,\overset{mm'}{3}) = N(P,\overset{min}{3})}$; ~~N(P,4)~~ $N(P,4) = N(P,4)$ Invariant. ; ~~$N(P,5) = N(P,4)$~~ ; $N(P,\cancel{6}) = \overset{Nq}{N(P,5)}$. 3 updates!!
$\underline{N(P,6) = N(P,5)}$ Then $\underline{N(P,5) = N(P,4)}$

Needed → $N(P,1) = 0$
$P(1) = 0$

Then $B = \overset{or\ B = N(P,7)?\ N(P,7)}{\cancel{E}}$; c = BMX : INCR BMX. for next round of recursion.

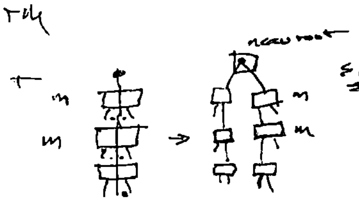
30 Grace suggests having 8 1 dim arrays rather than 1 2 dim array for N: The advantage is that each of the 8 arrays could have its own name 1 → M, 2 → R, 3 → MN ; 4 → LK 5 → MK 6 → RK , 7 → P , 8 → BT

~~P is used as an address variable i.e. an Array name.~~ ~~[illegible]~~

Unclear B = P : c = BMX seems right. INCR BMX
P(P)

40

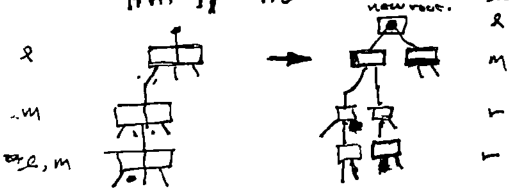
4 TM

00 

So before splitting, (updating) t. path down was $\overset{1}{m}$, $\overset{2}{m}$, (takes $\overset{3}{m}$ or r)
After " " " " " ~~m~~ r, m, m, m

So, working back up from t. bottom, t. path was t. "same".

Hvr, if ~~we~~ used a l decision going down ; ~~So~~ After splitting, t. decisions are all different.



.08

.09 { Hvr, while moving up to update nodes the decisions in going down at ~~[illegible]~~ each

10 node will be useful in deciding how to split a node.

~~[illegible]~~ — But I dont know if these decisions would be useful in getting "linked list" addresses.

— The going down decisions could be updated as one moves up t. tree, Hvr. This updating, while not absly essential, could be a useful check on t. pgm; as well as make it unnecy to recompute the key traces to get the above/below addresses for t. "Linked list".

16 [for t. linked list (For t. prediction Application) we only need addresses in t. ~~[illegible]~~ "below" direction: It may be good to reverse dalas of l m r so l & m are carried by t. nodes, rather than m & r — This may make it easier to find ~~[illegible]~~ "Below" addresses. — (But I'm not ~~yet~~ sure!) —]

So Do generate a Trace for each insertion path — it will be usable for Node updating.

SN It looks like this pgm will be very easy to pgm in MachLang! I can either use the

20 "FNCA" as a sbtn or since its very short, write it as a Macro ~~E~~: Since I dont have a macro assembler, I can simply do a ~~E~~ "Find and replace" for t. macro work!

If PB35 doesn't do find & replace, write pgm in Winedit. PP35 does do srch & replace (pp 37, 61, 67 of U.G.) ; But it takes a lot of time. To method in Winedit or Word pad is much easier, but I dont know if Word pad produces vanilla Ascii.

QBasic has a simple "srch & replace" but may be too "picky" about exact format:

use PB 35 or Winedit

The output of FNCA is a flag change, that I can use to control the branch implied by an "If statement". — So very fast!

Also, if I have a pgm that runs well in Basic, write

30 a Basic pgm that rewrites that Basic Routine into ML form! — particularly the long list of Assignment statements. → [or it may be just necy to check on pairs of ~~[illegible]~~ integer indices that appear in assignment statements]

SN in ML (or Basic) have t. leaving be nodes in which almost all addresses are t. same (i.e. the "A" of Y(A) of that leaf). Main thing is that items 1, 2, 3, ~~[illegible]~~ (of course) are all "A". 8 may be special 7 is parent

40

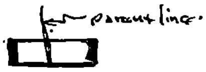
→51.00

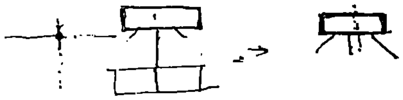
2.24.04 <span>mr</span> $\overset{MM}{}$ 12 3 <u>456</u> 7 8 bot kids.

4TM

00:43.40 ....In pgm, t. insertion of a key into t. bottom row of Nodes: When a 3 kid node was split, I think I assumed the <u>left</u> <u>node</u> ~~led~~ of t. pair had the parent of t. unsplit node. Is this ~~the~~ a <u>legit</u> way to do it? Or: if one of t. pairs of nodes has 2 leaves that were originally in t. 3 kid node, we must give <u>that</u> node t. old parent.

.04 (43.40) pac: X: 2 nodes. Hvr. these 2 nodes should fit in where the ~~[illegible]~~ original parent came from: I.e., t. original parent had a range of keys it would accept: After t. parent splits, t. 2 ~~[illegible]~~ kids have same <u>Total Range</u> as ~~[illegible]~~ <u>Unsplit</u> parent. So if we ~~[illegible]~~ know ~~[illegible]~~ where parent was attached to (upward) then we know where t. pair will be attached — same as w. keys on bottom row.

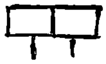
I can think of it as t. <u>parent line splitting</u> 

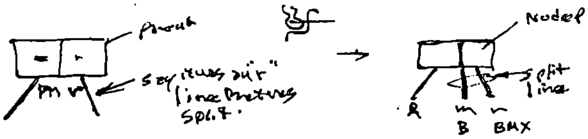
.10  when we split a node, one of t. pair ~~[illegible]~~ gets t. <u>name</u> of t. ~~[illegible]~~ presplit node, but this is of no import. The parentages of t. ~~[illegible]~~ twins are re-assigned (a ~~[illegible]~~ "split parent line")

OK, write one If! Even the non-splitting case seems diff!

.14 Case 1 $\boxed{B = BTOP}$: the node we split was top node. So we have to create a new 2 kid TOP NODE

.15 Case 2 ~~[illegible]~~ N(P,) is a <u>2 kid</u> ~~[illegible]~~ node i.e. $\boxed{N(P,1) = 0}$ (-00)

 ← P node From our "Path track" we know which "line" P was (<u>m</u> or <u>r</u>).

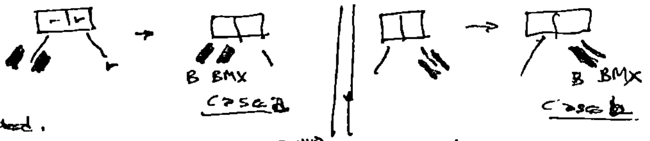


To <u>update Node P</u>:

4 5 6, are addresses of <u>kid nodes</u>

4 ← old m; 6 ← BMX, 5 ← B } or (vice versa)!

20 ....→ Perhaps By <u>convention</u>, B is on <u>Left</u>; BMX is on <u>Rt.</u> <u>I did this in SORT5.BAS</u> 46.~~[illegible]~~ .26-.29

2 possl. ways to ~~[illegible]~~ split. 

in both cases:

It's clear how $N(P, \overset{4}{\underset{6}{5}})$ are updated.

.24 case 2 sub Case a, $N(P,\overset{m}{\underline{7}})$ = ~~[illegible]~~ $N(B,\overset{3mm}{7})$; $N(P,\underline{1}) = N(BMX,\overset{3mm}{7})$; $N(P,\underline{2}) \overset{\checkmark}{=} N(P,2)$ ← invariant

$N(P,\underline{4}) \overset{\checkmark}{=} B$, $N(P,\underline{5}) = BMX$; $N(P,\underline{6}) = N(P,6)$ (invariant), $N(P,\overset{mm}{\underline{3}}) \overset{\checkmark}{=} N(B,3)$, $N(P,8) = 1$ ($\neq 0$)

.26 case 2 sub Case b I don't see any essentially new ideas needed.

There is a <u>peculiarity</u>! In t. <u>2 kid case</u> of sort5.bas <u>46.26-29</u>, there were <u>3 cases</u>, but here we only seem to <u>have 2</u>.! So it looks like this will be <u>essentially diff. from</u> the <u>Botton Row Node updates</u>! Discussd it w. Grace a bit. No apparent Bug in reasoning.... →

.30 + For 3 kid case: Write out one ~~[illegible]~~ (or 2 if necy) detaild cases. Later, see if there (33)

→ Also note, I <u>may</u> need that list that keeps track of t. path <u>to</u> insertion point, so I can trace backward (Also for/ Mike help linkd list.)

.33 (.30) are common update eqns that I can put in a "~~[illegible]~~ common ~~[illegible]~~ terminal statement set.

say BR(J) ← "Branch (J = 1 to ≤ $\log_2 1000$ = longest path say (15)) is ~~B~~ | BR(15) = unsigned ~~Integer~~ ~~[illegible]~~ – Byte. 0 to 255.

~~BR~~ = <u>1</u> for ~~left~~ $\overset{\ell}{}$, ~~[illegible]~~ $\overset{2}{}$ $\overset{m}{\text{middle}}$, ~~3~~ $\overset{3}{}$ for rt. — hvr, after update of nodes this path will not be correct! would it be adequate to ~~[illegible]~~ 1) ~~F~~ Update Nodes: includy new nodes 2) find 2 components (Above & below ~~[illegible]~~ leaves) ?

40

2.24.04
4PM

SORT5.BAS

46 ← 48

2→1 dim.

N.B: we only need t. Below addresses. (50.16)

```
DIM Z(1000,~~2~~) AS WORD   'Above and Below Addresses for Linked List
DIM N(1000,8) AS WORD   '8 Node Parameters: 1 m|2 r|3 smallest leaf,
                        |4 l-child|5 m-child|6 r-child|7 parent|8: Bottom Nodes=0.
~~[illegible]~~
DIM Y(1000) AS BYTE     'Y() stores seq to be predicted : for testing, we may want to put
DATA 0, 1, 2, 3, 1, 2, 3, 4                                  32 bit random nos. in Y. so Y
FOR J=1 TO 8:READ Y(J):NEXT                                  would be dword
YY=VARPTR32(Y(1))
DIM AA AS BYTE PTR
DIM BB AS BYTE PTR
SHARED AA, BB, YY       'Important Line!
BMX=1                   'BMX is last node defined
BTOP=1                  'BTOP is top node="Root"

DEF FNCA(A,B)           'CA is TRUE if A "Comes After" B
   AA=YY+A: BB=YY+B     'TRUE->-1:FALSE->0
    WHILE @AA=@BB
    DECR AA: DECR BB
    WEND
 FNCA=ISTRUE @AA>@BB
END DEF
```

→ see note →

change to "1" makes Bottom node.

common to all continuations of Sub3

```
START IF N(B,8)=0 THEN                      'Bottom Node
        IF N(B,1)=0 then   GOSUB  SUB2: END ~~C=BMX:~~ INCR BMX    'Two kids
.15     ELSE               GOSUB  SUB3: ~~C=~~Then Update Parent(s) 'Three kids
[redacted]
        END IF
      ELSE          GOSUB SUB1: GOTO START   'Not bottom Node-Go down Tree
[redacted]
      END IF

.20 SUB2 IF FNCA(A,N(B,6)) THEN          'Two kids
            N(B,4)=N(B,5):N(B,5)=N(B,6):N(B,6)=A
          ELSEIF FNCA(A,N(B,5)) THEN
            N(B,4)=N(B,5):N(B,5)=A
24        ELSE N(B,4)=A:
        END IF
        N(B,1)=N(B,5):N(B,2)=N(B,6):N(B,3)=N(B,4): [redacted] RETURN
25  SUB3   ~~BMX=BMX+1~~ : INCR BMX       'Three kids--New Node needed
                                                                        B      BMX
6       IF FNCA(A,N(B,6) THEN
            N(B,5)=N(B,4):N(B,6)=N(B,5):N(BMX,5)=N(B,6):N(BMX,6)=A      2 m    r A
          ELSEIF FNCA(A,N(B,5) THEN
            N(B,5)=N(B,4):N(B,6)=N(B,5):N(BMX,5)=A:N(BMX,6)=N(B,6)      2 m    A r
          ELSEIF FNCA(A,N(B,4) THEN
            N(B,5)=N(B,4):N(B,6)=A:N(BMX,5)=N(B,5):N(BMX,6)=N(B,6)      2 A    m r
          ELSE
9           N(B,5)=A:N(B,6)=N(B,4):N(BMX,5)=N(B,5):N(BMX,6)=N(B,6)      A 2    m r
        END IF
                                            no #1 for 2kid node
            N(B,2)=N(B,6):N(B,3)=N(B,5):N(B,1)=N(B,5): ← unnecy
            N(BMX,2)=N(BMX,6):N(BMX,3)=N(BMX,5):N(BMX,1)=N(BMX,5): unnecy
            N(BMX,8)=0:RETURN          min           m
                                                     c      Doesn't exist for 2Kid Node.
           ↑ Bott. node

SUB1       IF FNCA(A,N(B,2)) THEN B=N(B,6):          RETURN
             ELSEIF FNCA(A,N(B,1)) THEN B=N(B,5):    RETURN
               ELSE B=N(B,4):                         RETURN
```

seems correct! ~~[illegible]~~!

insert { C=BMX : B=P : P=P(B) } (P=P(P(B)))

New B"

old. B"

goes up one level.

~~Actually~~ : GOTO 51.00 seen : ~~[illegible]~~

No need to return, 51.00 is a loop w. exit possl.

B and C ~~[illegible]~~ are the 2 nodes that have to be inserted into P(B).

2.24.04

ATM

00 SN Random Notes.

.01 (42.28 / 42.12) 1) Make "C++ Library for Mach Lrng": Various tools that can be combined to make new tools & pgms to do ML. Some modules will be in assembly.

2) For fast comparison, use 4 or 8 - "MMX" type - parallel insts. i.e. "are t. next $\left(\frac{4}{8}\right)$ comparisons identical" ... etc.

.05 (42.29 / 42.12) 3) Can we (usefully) put Y( ) in t. secondary cache. Is t. primary cache ever big enuf to store a usefully sized Y( )? What is min Y( ) size for useful ML? If Y( ) is a bunch of unordered cand. derivs, we can make a kind of SUMAC that stores only a good representative "set" of cands. This is closely related to "GP" ideas of 42.00 – .30

10 — 42.13 in particular. Also Note 42.08 – .10

2.23.04

46
→ 48

SORT3A.BAS revised
Grace's revision

```
.00  IF N(B,8)=0 THEN                        'Bottom Node
       IF N(B,1)=0 then                          'Two kids
          IF FNCA(A,N(B,6)) THEN
                N(B,4)=N(B,5):N(B,5)=N(B,6):N(B,6)=A
             ELSEIF FNCA(A,N(B,5)) THEN
                N(B,4)=N(B,5):N(B,5)=A                                    Sub1
             ELSE N(B,4)=A:
          END IF
          COPY1   N(B,1)=N(B,5):N(B,2)=N(B,6):N(B,3)=N(B,4): GOTO---
       ELSE double unary
          BMX=BMX+1                          'Three kids--New Node needed
          IF FNCA(A,N(B,6) THEN
                N(B,5)=N(B,4):N(B,6)=N(B,5):N(BMX,5)=N(B,6):N(BMX,6)=A
             ELSEIF FNCA(A,N(B,5) THEN
                N(B,5)=N(B,4):N(B,6)=N(B,5):N(BMX,5)=A:N(BMX,6)=N(B,6)    Sub2
             ELSEIF FNCA(A,N(B,4) THEN
                N(B,5)=N(B,4):N(B,6)=A:N(BMX,5)=N(B,5):N(BMX,6)=N(B,6)
             ELSE
                N(B,5)=A:N(B,6)=N(B,4):N(BMX,5)=N(B,5):N(BMX,6)=N(B,6)
          END IF
          COPY2 N(B,2)=N(B,6):N(B,3)=N(B,5):N(B,1)=N(B,5):_               several updates
       double   N(BMX,2)=N(BMX,6):N(BMX,3)=N(BMX,5):N(BMX,1)=N(BMX,5):_    added to PAS
       unary    N(BMX,8)=0                  N(BMX,7) must be computed      version.
                                            later via Sub2.
       END IF
     END IF ELSE call ~~sub~~ sub
```

Block format easier to see: Also corrections in "copy2" & added updates: N(BMX,7) has to be computed by moving up the tree

In fact, the biggest sub is "update N(BMX,7)"

SubI is a simple search down the ~~tree~~ for ~~[illegible]~~ N(B,8) ≠ 0 nodes.

~~[illegible]~~ Sub1 = 41.13.

```
IF FNCA(A, N(B,2))  Then B=N(B,6)    Go to 46.00     rchild addr.
ElseIf FNCA(A, N(B,1))  " B=N(B,5)   "  "  "         mchild          } Sub3
~~[illegible]~~ Else B = N(B,4)      "  "  "
```

```
α   IF N(B,8)=0 Then   ← Bottom node.
       IF N(B,1)=0 then  ← 2 kids
          CALL Sub1 (2 kid update) → Exit (key has been inserted)
       Else
          Call Sub2 (3 kid update) → update parent(s) — go up tree
       ENDIF
       ) B(N,8) ≠ 0 (not bottom node — descend TREE of NODES.)
    Else call Sub3!
    EndIf
```

2-23-04

Sort3

```
IF N(B,8)=0 THEN                   'Bottom Node
  IF N(B,1)=0                      'Two kids
   IF FNCA(A,N(B,6)) THEN N(B,4)=N(B,5):N(B,5)=N(B,6):N(B,6)=A:GOTO Copy1
~~COPY1~~
     ELSEIF FNCA(A,N(B,5)) THEN N(B,4)=N(B,5):N(B,5)=A:GOTO COPY1
     ELSE N(B,4)=A:                              N(B,7)=N(B,4), N(B,8)=0
COPY1   N(B,1)=N(B,5):N(B,2)=N(B,6):N(B,3)=N(B,4): GOTO--- α
  ELSE BMX=BMX+1                   'Three kids--New Node needed
   IF FNCA(A,N(B,6)      THEN N(B,5)=N(B,4):N(B,6)=N(B,5):N(BMX,5)=N(B,6)
_
   :N(BMX,6)=A:GOTO COPY2
   ELSEIF FNCA(A,N(B,5) THEN N(B,5)=N(B,4):N(B,6)=N(B,5):N(BMX,5)=A_
   :N(BMX,6)=N(B,6):GOTO COPY2
   ELSEIF FNCA(A,N(B,4) THEN N(B,5)=N(B,4):N(B,6)=A:N(BMX,5)=N(B,5)_
   :N(BMX,6)=N(B,6):GOTO COPY2                 ~~N(B,7)=N(B,5)~~ No
   ELSE          N(B,5)=A:N(B,6)=N(B,4):N(BMX,5)=N(B,5):N(BMX,6)=N(B,6)
COPY2 N(B,2)=N(B,6):N(B,3)=N(B,~~4~~5):N(BMX,2)=N(BMX,6):N(BMX,3)=N
(BMX,~~4~~5):GOTO--- β
 N(BMX,1)=N(BMX,5)          N(B,1)=N(B,5) ✓
```

No: Parent of BMX is complicated!
N(BMX,7)=N(BMX,5)
N(BMX,8)=0

~~After each alternative position of A is~~ For each position in which A is inserted, we have to update Z(,1/2) ... linkages for linked list. It involves 4 ~~[illegible]~~ updated params.

If A is inserted as Min or Max of this bottom node, ~~we~~ update is more elaborate! We have to find the next upper or lower neighbor --- it is in an adjacent node (if any).

This is a complex prem, but we ~~[illegible]~~ really need these adjacent keys for parity checks!

r20 Assume we have a "trace" of ~~[illegible]~~ path to insertion point.

However, this linkage problem can be viewed as a separate problem to be done after the insertion and update algorithms are designed. ~~At least~~ If we have the "Trace" of r20, we can probably insert the linkage routine ~~in~~ after the "~~[illegible]~~" "Go to ---"s at top of page.

The next step would seem to be to update N(BMX,7) to parent of the newly created Node. Also / updating N(—,3) the smallest root of various Nodes. Also set N(—,8) to ∅ for Bottom, & to 1 for others.

< May set (in vert / toggle) meanings ~~meaning~~ of N(B,8) since ∅ is default. >

2 3 7 1 8
123 78
56

B(MX) BMX
7 = 5   8 = 0
1 = 5

123 4567

8,3,7 = 5 !

```
DIM Z(1000,2) AS WORD    'Above, Below Addresses for Linked List
DIM N(1000,8) AS WORD    '8 Node Parameters: 1 m|2 r|3 smallest leaf,
                         '4 l-child|5 m-child|6 r-child|7 parent|8 "bottom"=0
~~bottom=0~~
DIM Y(1000) AS BYTE      'Y() stores seq to be predicted
DATA 0, 1, 2, 3, 1, 2, 3, 4
FOR J=1 TO 8:READ Y(J):NEXT
YY=VARPTR32(Y(1))
DIM AA AS BYTE PTR
DIM BB AS BYTE PTR
SHARED AA, BB, YY
BMX=1                    'BMX is last node defined
BTOP=1                   'BTOP is top node="Root"

DEF FNCA(A,B)            'CA is TRUE if A "Comes After" B
   AA=YY+A: BB=YY+B      'TRUE->-1:FALSE->0
    WHILE @AA=@BB
    DECR AA: DECR BB
    WEND
 FNCA=ISTRUE @AA>@BB
END DEF

PRINT FNCA(3,8)
PRINT
```

For debugging

The function FNCA (A,B) seems to work O.K.! In assembly lang. it would be *very fast*!

The output of FNCA is a *flag* set and can be used directly to make the needed decisions.

2.21.04

47 AM

43

00 -+ Re writing 41.10, using "Block" If of P.Basic 3.5 — or possibly D word (32 bits) — we will start with strings of length up to 1000.

```
DIM  Z(1000,2) , N(1000,8) ,  AS word  16 Bits
                  unsigned 16 Bit integer
     Y?(1000)
       unsigned 8 bit integer.
Def N,Z as word
Def Y as Byte
```

Z is the set of links for the ordering of Rank(A)'s

Z(A,1) is the downlink address

Z(A,2) is the UP link address.

The 8 arguments of $\boxed{N(B,(\cdot))}$: 1: m | 2: r | 3 smallest key covered by this node | 4: l child | 5 m child | 6 r child | 7 parent node addr | 8 If bottom Node its = 0, If not, its = 1.

ABC

ABCDabc

BMX is largest B / address used on N(B,) thus far.

BTOP is the address of topmost Node: It is the entrance node for Sorting K(A)'s.

Def CA as BYTE (actually, only 0 & 1 are needed). See what comes out as expt.

ABCD

ISTRUE 1<2

returns TRUE

```
.10  -)- DEF FNCA (A,B)          ' A and B are addresses on Array Y( )
             ↑ key Address
         While Y(A) = Y(B)       ' Funct. Defined U.Guide P 136
           Dec A : Dec B         ' It would work much faster if A and B were "Pointers" (U.G. PP89ff)
         WEND
       CA= ISTRUE Y(A) > Y(B)    ' (I may want CA= - ISTRUE Y(A)<Y(B)
                                   So CA is always ≥0. I which case 00
       END DEF
```

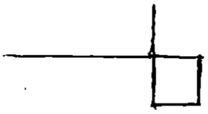
At start of Program, we have 2 Keys in place, and one Node:

	1	2	3	4	5	6	7	8
N(1,...) :	0 (0-∞)	X	W min	∧	W	X	← means no parent	φ

X < W are 2 keys.

BMX = 1 '(1 start Node defined)

BTOP = 1 (topmost Node in Tree = "Root").



∧ 5 6 A → 4 5 6

20 -{- ENTER pgm with Address of key to be inserted: it's "A", B = BTOP

456.

```
23 (from 41.19)  If N(B,8)=0 Then
                   If N(B,1)=0 Then
     If CA(A,N(B,5)) Then N(B,4) = N(B,5): N(B,5)=N(B,6): N(B,6)= A :   Exit
     Else If CA(A,N(B,5)) then N(B,4) = N(B,5): N(B,5)=A : N(B,6) = N(B,6) ← invariant.
                                                      └─ Exit
     Else N(B,4)=A : exit
```

∧ 5 A 6 → 4 5 6

A 5 6 → 4 5 6

N(B, m 1 ← 5, r 2 ← 6, l 3 ← 4) for

0 -{- On mapping the 3 subroutines of (sub 7 of 45.20-.40 into sub 5 for PUP Parent UPdate

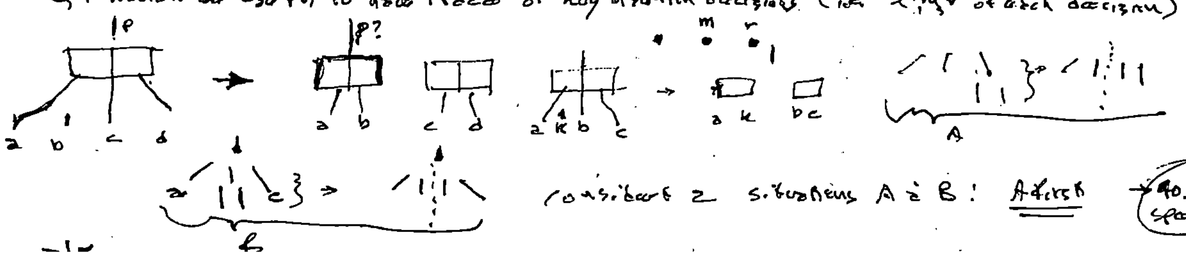
PUP: we start with address of Parent NODE / for N(B,) : N(B,7), address of that Node is "P"

~~In place~~ N(P,1) = 5-00, then ~~update~~ insert N(B,3) ← smallest last ~~in proper place, update N(P,) & exit.~~ of N(U,).

Insert N(BMX,3) ← smallest last. into N(P,), update params of N(P,) & exit. | Moreover, Insert BMX as a child on basis of N(BMX,3)

Q: would it be useful to have "trace" of key insertion decisions (i.e. l, m, r of each decision) ?

3 possible Parent Node types!
1) No parent
2) 2 kids
3) 3 kids



consider the 2 situations A & B: A first

40.04 spec

49.00 → standard

16 -|-

4TM

00 + SN On a ≈ "GP" routine using ≈ Bzz. We have a big population of cards that have been tested, using fitness function, F. ~~E~~ Each card has a "key". We sort the keys in 2 ways and make a linked list for each: The 2 ways: ① Lexical to generate trials ~~in~~ for Lsrch ② in F order, to discard bottom 90% of population (or just keep the top 1000 of population.

So Operations consist of using the Lex order to generate trials, then testing, (possibly Lsrch discard rule) then using F ordering to insert new good cards & discard old bad cards.. — some monotonic funct of F

08 It is, alternatively possible to keep top 10 000 cards, but weight them on basis of F (basis of F), so top cards given more wt. This can be done by inserting dups
10 + of good cards (as a function of F) or simply weighting pc's for new trials on basis of F values of population. {Meaning?} → perhaps ≡ lines .08 – .09 → 47.05

.12 We keep ~~table~~ of F values of cards.

13 In t. try: 2 ideas: Ordering on basis of F (Fitness Funct.), Ordering on basis of Aprp. ordering, Somehow these are mixed to choose new cards.

Unfortly, the apriori is always changing as we add, remove cards from corpus, So we are not really doing "Lsrch" (which requires constant apriori during srch)

An (≈ optimum) way to deal w. this: Say we have a list on which we match the "top 1000" or "top 10%" or whatever, in our "clique" of v.g. cards. Each time we test a new card, & we insert that card into the "clique", the pc of all cards in the clique are
20 + modified — So if we are doing Lsrch, we have to remember how much time we spent on each card in the clique & revise our "work schedule" (≡ allocation
.22 of cc to various cards). If cc_i are (cc spent on) card i thus far, (pc_i pc of) we work on card w. lowest $\frac{cc_i}{pc_i}$ ← At all times. As we work, these are larger & larger no of cards in same value, so we (?) show below them.

Schedule

I don't know how practical ~~it~~ to work schedule of .22 ff is. We could revise the pc_i's every time a new card is evaluated, or perhaps every 10, 100, or 1000 evals — or some other update criterion. — Perhaps
.28 when a big F card comes in. → 47.05 → on 47.05 this is almost SUMAC! →

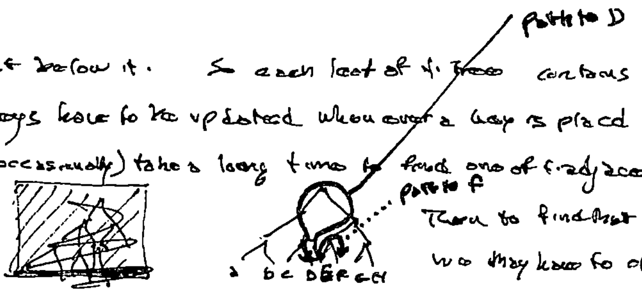
30 + On practical items in PB35; For fast access, an array of 64k bytes is as big as I can get: This means that 2 Bytes of 4 byte address are constant. So each Address component in N( ) is only 2 bytes. I might use a 4 byte pointer and an "Absolute" array (U.Guide, pp 101-102) (this type of array starts at edges of 64k blocks & has max size of 1 block.) However, an abs array is automatically "Dynamic" — which tends to slow/slows as RAM Disc.
I may try different kinds of Memory to see if I can speed up the "CA" (comes after) (fnctn. function), Pointers are discussed on U-Guide 89 – 92

One problem is: I don't know where to put an ABS array. Screen Memory starts at &H B800 try &H B700; But work this out later. I can modify the "comparison" function
10 + CA( ) "ComesAfter" later.

4TM

& above is just below it. So each leaf of the ~~tree~~ contains 3 addresses. Two addresses of adjacent keys have to be updated whenever a key is placed below them.

Hvr, it ~~[illegible]~~ will (occasionally) take a long time to find one of the adjacent keys: ~~[illegible]~~ i.e. if the tree is



Then to find that "D" is the key just below "E" we may have to climb high in the tree. In the present case, we ~~[illegible]~~ know all the nodes leading to E, and can immediately find ~~[illegible]~~ the path to the key just below E: starting at the top ("Root") node. There is a simple Algm. for obtaining the closest keys to a given key. The Algm. is: Go up the ~~[illegible]~~ branch of the "~~[illegible]~~ middle" key. To find the closest key above; take the first upper alternative branch and go down, ~~[illegible]~~ taking all lowest branch choices. If there ~~[illegible]~~ is no such choice, then ~~[illegible]~~ the "middle" key is the largest key in the tree.

+

Starting over on 38.25 we begin with K(A), the key to be inserted.

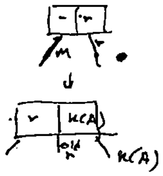
"X" is the name of the current root node. — Begin: B = X

```
.125 ⟶
.13  sith 1   If $K(A) > N(B, \overset{r}{2})$ then $B = N(B, \overset{\text{child address — These are Node addresses.}}{6})$ Go to .13
              If $K(A) > N(B, \overset{m}{1})$ then $B = N(B, 5)$ Go to .13
              $B = N(B,4)$ Go to .13.
```

(Insert at .125) If $N(B, 8) = 0$ then   N(B) is Bottom Node

select case!

```
.19   $N(B, m) = -\infty$  (N(B has 2 kids) If $K(A) > N(B, r)$ then  $N(B, l) = N(B, r)$
                                                                 and $N(B, r) = K(A)$
      ~~N(B,l)=K(A)~~
      ~~Else~~ If $K(A) > N(B, \overset{\text{lowest}}{3}$
```



→ 42.00

Strictly speaking A(43) say, is not an array element ~~[illegible]~~ we don't declare the dimension of A. We have a function, F(U,V) that can ~~[illegible]~~ determine if $k(U) > k(V)$. $F(U,V) = 1$ if $k(U) > k(V)$; else $F(U,V) = 0$.

Dim Z(1000, 2) : Z(A,1) is the ~~[illegible]~~ string just < K(A)

Z(A, 2) " " " " > K(A)

Y?(1000) ~~[illegible]~~ Y?(1000) is the seq. of symbols being predicted "Y?" means it's a BYTE

SN **BZ2** Uses a somewhat diffnt, (perhaps not as good), algm for compression, than PPM, — Perhaps probs not explicitly used. Could I perhaps get it into a form to create fountains of a string in rather than of "compressn"? This could save lots of time, & that's mainly what AD is about! [Another aspect of the problem, hvr,: if the testing of a cand takes much longer than the time to generate the ~~[illegible]~~ cand. itself, then time needed to generate the cands becomes unimpt. — i.e. we mainly want to get v.g. cands, & can spend much" time on ~~[illegible]~~ selecting them

2-18.04 Original 40

4TM

3 symbol string

"1" address = "-∞"

1	2	3	4	5	6	7	8
N	m	r	small ask leaf	ℓ	m	r	addr Parent	=0 also 1

) 39.32 → Bottom Node → 0 also → 1

.00 ∓ we first set A=3 then we start. to start off A=3, B=1.

~~If N(B,A) then~~

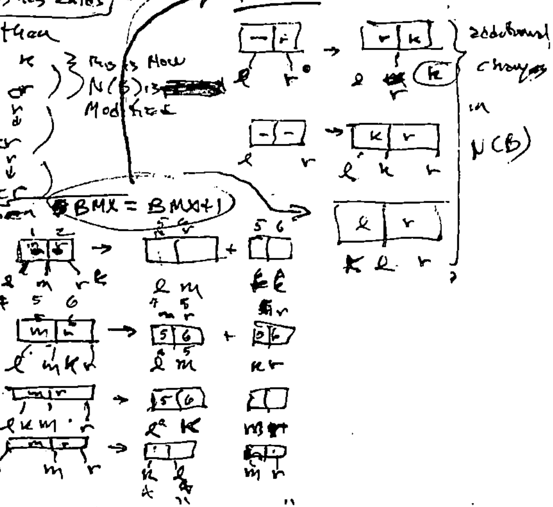
```
.03 (loop) If N(B, 8) = 1 then  If k(A) > N(B, r) then B = N(B, cr))
            ↳ not bottom node.  else If k(A) > N(B, m) then B = N(B, cm) else
                                  → B = N(B, cℓ)
                                                  loop to .03 (.03)
     else (i.e. if N(B,8)=0)
```

1 2 3 4 5 6
m r L cl cm cr
Min leaf

7 8
PA BN=0 Bott
(parent add.) BN=1 not bot.

```
10  If N(B, ℓ) = -∞ then        (i.e. N(B) has 2 kids)
       If k(A) > N(B, r) then ... N(B, cℓ, cm, cr)   } This is How N(B) is Modified
       If k(A) > N(B, ℓ) then N(B) → N(B, cℓ, cm, cr)
       else                   N(B) → N(B, cℓ, cm, cr)
    If N(B, ℓ) ≠ -∞, (i.e. N(B) has 3 kids)  Then BMX = BMX+1
    If k(A) > N(B, r) then N(B, cm, cr) and N(BMX), cm, cr)
                                  5   6                5   6
    If k(A) > N(B, m) "
ℓmr If k(A) > N(B, ℓ) "
456 Else
```

BMX (Bmax) is largest B value thus far



.20 SN Each Node is also assigned a level no. The bottom row of keys is level 1"

.21 The level no. of a Node never changes. This info may not be used in Pgm, but in Debugging. → lvl. see (.31) for possible use.

When "In t. search for where a key goes" keep track of ℓ,m,r branches as well as "parentage". ~~[illegible]~~ Both are needed in updating

For the "update" srtn, list the inputs needed for t. srtn, and list t. results, and outputs needed for t. next call of ~~[illegible]~~ this update srtn. → Parent can be (a) Null (b) 2 kids (c) 3 kids. ≡ No parent.

On entrance to "update srtn", one knows (1) address of Parent node to be updated. (2) The "Min leafs" (N(B, 3)) of the 2 Nodes needing revision (or reinsertion)

(3) ~~[illegible]~~ The address no. of t. last Node created (we increment this to create new nodes if needed).

(4) The ℓ, m, or r info relating Parent to present (unmade) child. ← (Maybe not needed) or fact that parent does (not) exist.

30 +

.31 (.21) (5) At each time, There is a register (Variable) that gives ht. of tree node. When a new top node is created (when a parent is null) Then this variable is incremented; — also the address of t. "input node" (to the tree) is updated (Note .20 - .21)

34 → SN On obtaining keys in order list of keys that are "<" a given key. When t. Given key is sorted, we keep track of the branches in ~~[illegible]~~ path to it. Those branches can be used to "Backtrack" via a stack, to find keys below "Given key".

(.34) May be one of t. biggest disadvantages of 2-3 TREES, for my particular applicn. — (So it shouldn't take much time to get the "closest 1000" keys (below) t. Present key

Well, one good way (That would triple amt. of RAM needed for system); — Make a linked list.

40 . As soon as a key is inserted in the sequence, give it the addresses of the keys [illegible]

4T/M

+ ok. This does t. bottom nodes:

01:38,29 → If N_A is not a bottom node, A ← [New address obtained by comparisons of 38.27] and Go to srtn. 38.26: This loop inserts K^{new} ~~into~~ into t. tree.

So, we have this loop that inserts k into t. tree by going down t. branches until it gets to a "Bottom node" — Then we do srtn. 38.29-40. to insert Key into t. bottom node. (We ~~may~~ May create a new bottom node).

36.31-40 on ~~doing~~ updating the bottom Nodes, is repeated w. little modifn. in t. general updating

~~Srtn.~~

.06: 38.30 Update of ~~the~~ Tree! After the "end ~~of~~ 38.30, we have inserted "K" into tree, and updated the bottom set of Nodes: Next srtn. to update t. rest of. Tree!

+ ~~Instead of exiting 38.30~~ If bottom node A has m = −∞, we don't have to update tree any more. ~~[illegible]~~ (Not exactly! : If we have inserted a key at the lower edge of Node A, then The "smallest leaf" param of Node A has to be updated — to be k. This has to be done whether m of Node A = −∞ or not! ~~it doesn't depend on~~ whether Node A has 2 or 3 kids.) For the 2 sub nodes (let $\bar{m}$ and $\bar{r}$ be the smallest elements in them (respectively).

If Node A originally had 3 kids, so it is now split into 2 Nodes: We next do A ← parent of A. If A is now m = −∞, ~~then we don't~~ have to split. NodeA. We do have to update N_A, hvr, we set $m = \bar{m}$ and ~~r~~ $r = \bar{r}$ → end of update.

If A ≠ −∞, then we do have to split ~~[illegible]~~. Node_A.

$\boxed{\text{SN}}$ for alphabetical symbols, we can use ~~[illegible]~~ $\boxed{a \equiv -\infty}$: We then start sorting for + ~~sorts > 1 symbol long: in which case they always come after a~~ Actually, we can set −∞ to be t. string consisting of t. first symbol in t. sequence. — it has a regular address, so it's easy to compare it to ~~[illegible]~~ other strings. This "−∞" is just "conceptual", hvr, ~~and probably I will replace it by something faster (the more involved, "conceptually)~~

If Parent of A has m = −∞, we stop update; if m ≠ −∞, we split parent & look at whether parents of parents have m = −∞, etc.

We do have to be sure to update the "min leaf" of ~~every~~ parent. node we update or create.

For (initial) pgm! Each key has an address; Each Node has an (address) ≡ number. ← its number in the sequence

~~8~~ 7 params; maybe ~~8~~ only 7 params, ~~not 8~~.

The params of a node are m, r, smallest leaf, (keys or Nodes / addresses of children), (node address of parent). ← but not for
3.

+ While we are running pgm, we have an ordered list of nodes we used ~~to get~~ to the sort. Not point. This may make it unnecy for a node to carry address of its parent.

.32 $\boxed{\text{pgm start}}$ K(A) is t. key of address A. N(B, j) is t. content of Node B, 4 m r smallest leaf

j: 1 2 3	4 5 6	7	8
m r smallest leaf	child addresses,	parent address	is this a bottom node?

Since t. no. of Nodes in t. tree is ~ ln (length of seq) ← No! total no. of Nodes is ≈ N t. no. of keys having many params per node does ~~[illegible]~~ take significant Memory. Other than t. seq. itself (memory ~ N) t. storage for t. ~~[illegible]~~ Tree that gives t. ordering, is only ≈ 8 $\log_2 N$. So essentially, no memory needed.

Dim $K(\bar{N})$ say N = 10k. I may be able (in PB 3.5) to put an array of bytes.

Dim $N(\log_2 N, 8)$

We start out with a binary node; we have ~~[illegible]~~ 2 keys pre sorted: This is for k=2 and k=3.

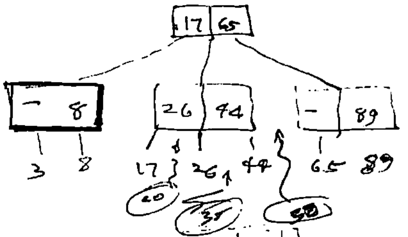
~~[illegible]~~ We start t. sequence with a or ϕ. (k=1).

2.18-04

4PM

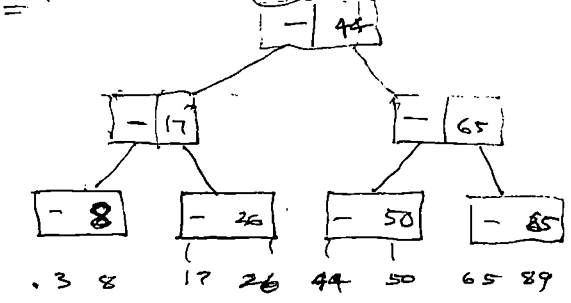
00 Consider Tree:

.02



To start we have 4 nodes.

If I insert 50!

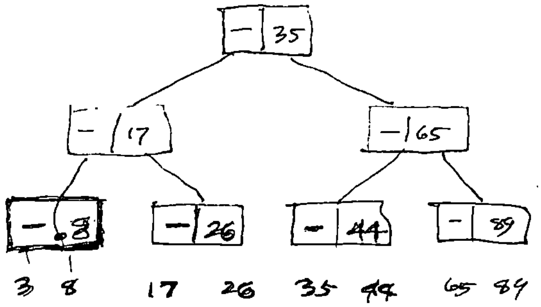


~~If~~, on t. other hand, if 35 were ~~[illegible]~~ inserted instead of 50, ~~[illegible]~~ then 35, rather than 44 would get into top node.

.10

Korma Curry Pasta
Coconut Almond
by Patak

.12 Alternately if 35 were inserted



.17

So, All of t. nodes in the ~~up~~ parental update path have to be examined — perhaps.

20 Actually, we create pairs along t. update path until we come to a 2 child parents — then stop. only the high parent need be modified (?).

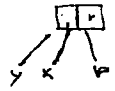
Inserting (20) into tree of .02 would put (26) into top node in tree of .12

Bottom line: 3|8 17 20 26 44 65 89

10k = 100×100 : 20 yrs.

r h
k r

l, r



.25 Pgm ① Go to N∅: or just input address of node ∅ into Srtn. ② address = A. ← = root

.26 ② Srtn ②: "A" is input = address of node. "K" is number or comparison index of key.

27 Compare k w. 2 indices of Node A (a 2 index node has t. left index = −∞)

K is then in branch $\overset{L}{1}$, $\overset{m}{2}$, or $\overset{R}{3}$ (k is never identical to an index: all sorted keys are different) → if N_A is not a bottom node, go to

.29 If N_A (node A) is a bottom node, Then If ~~[illegible]~~ m = −∞ (only 2 kids) modify

30 ~~[illegible]~~ m, r so that k has been properly inserted — end

.31 Insertion: Modify! 2 cases k>r; k<r; If N_A is a bottom node, it will have 2 kid values; a and b (r = b to start)

a. b → a k b → k|b ; k a b → a|b ; a b k → a|k } depending on k's value w.r.t. a & b. All 3 possys are legal & an error.

Find insertion key! If m ≠ −∞ then insert ks 31-40 and ... go to Srtn 39.08.

Bottom node → 39.00 → a creates ~~[illegible]~~ Srtn 39.08 which updates rest of Tree nodes

A bottom node also has a special param: its "smallest child" — which may be used in updating its own parent node(s).

So we have updated tree when key went to 2 kid node.

Next if bottom node has 3 kids:

l | r
r = c; l = b

a k b c
a b k c
a b c k

3 possys, unless bottom node is leftmost node in which case k a b c is poss. However, whether it's a leftmost node or not, t. x forms of (39-40)

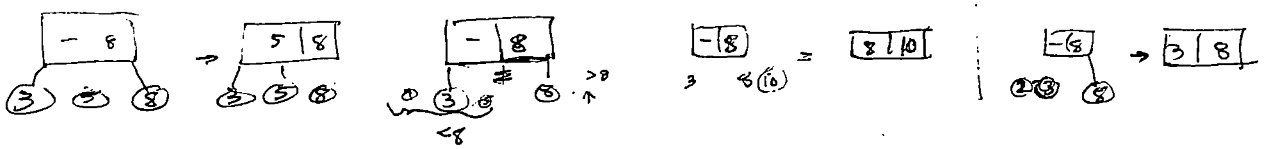
38 k a b c → b|c
39 a k b c → b|c
a b k c → k|c
40 a b c k → c|k

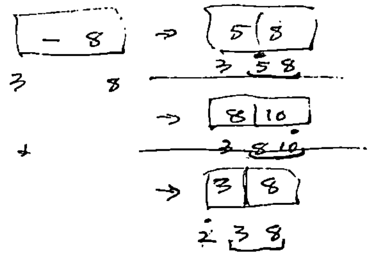
modify old node; 1 new node created →

k a b c → k|a
a k b c → a|k
a b k c → a|b

2.15.04
4TM

00:36.40. A new tack: Every leaf but the lowest, ~~[illegible]~~ seems to appear in a node (as a boundary threshold) once & only once. — So perhaps all we have to do is to see how the mapping changes w. an insertion. If we use the convention of labeling $\boxed{-\,|\,r\,low}$ for nodes w. 2 kids; then insertion simply changes "−" to ~~[illegible]~~





A notable constraint: the numbers in the nodes may move ~~[illegible]~~ from one node to another, but they always stay at same "level". — {Perhaps NOT see 38.00:17}

I could start programming it now. A node will have ~~at least~~ 5 params: 3 ~~[illegible]~~ pointers to kids & [m low, r low]) or (just r low). ; (It may also have 1 pointer to its parent.)

A node is split when it has 4 kids. If the ~~parent~~ Parent of a split node had 2 kids before, then is end of update. If the parent now has 4 kids, then split parent, and so on.

While the tree tells which nodes to split, it does not tell how to tell ~~[illegible]~~ ~~[illegible]~~ the assignment of m low, r low to the nodes that have been modified or created.

Perhaps have 3 types of nodes: Ø kid (= leaf); Parents w. ② or ③ kids

For mlow, rlow updating; we enter a node from the bottom, w. certain info: we know the lowest leaf of the nodes we come from. We know whether we are l, m, or r of the node we enter.

~~[illegible]~~ At first, just update by constructing new nodes and their 2 or 3 assoc addresses. Do this for the entire path ~~[illegible]~~ ~~[illegible]~~ Next, go thru the ~~[illegible]~~, path (again upward toward the root) putting in the mlow, rlow values! — ~~[illegible]~~

Then find way to do the 2 update ~~[illegible]~~ operations in ~~one~~ a single path.

$\boxed{SN}$ on "1-2 Trees": Objection to simple construction is that worst case insertion can occur if items are already in sorted order: Insertion for ~~[illegible]~~ kth item is time ~ k. So for n items $\frac{n(n+1)}{2} \sim n^2$. However, we can represent any no. by a binary representation of distinct powers of 2. ~~Each power of 2 corresp~~ If a 1-2-tree is of depth d, and the total no. of elements is n: then n will be sum of 2^{r_i} ~~[illegible]~~ for $i=1,2\dots$. For power r we have r "binary decisions" and d-r "unary" "decisions".

Trouble is, when we add 1 to n, ~~the represe~~ (insertion) the representation can change radically. e.g. ~~7 = 1+2+3+4; 8 =~~ $7 = 2^0+2^1+2^2$; ~~8~~ $7+1 = 8^3$. ~~[illegible]~~ This involves big representation changes in ~½ of the cases!

2.13.04

4TM

00:35.40 + So t. Q is what are all possl. modes of key insertion, & which of them imply over imbalance & How can they be balanced? +

A node must be unbalanced by ≥ 2 levels before rebalancing can be done.

Survey on Btrees 1979: ACM compsurv pp 121-137 (1979) D. Comer,

(1972) R Bayer first publication (so would Knuth 1973 have it?

Introd to Algms: Cormen, Leiserson, Rivest 1990: Discusses properties ~~of~~ Red-Black trees

(They are related to B trees

2-3-4 Tree: a Btree of order 4 having 2, 3 or 4 children.

2-3 tree " " " 3 2 or 3 children.

I have 3 files on 2-3 trees in PS: Study Room! ← These data structures called "dictionaries" involving insertion, deletion & sequencible occurs.

0 + "In 1970 Hopcroft introduced 2-3 trees as an improvement of balanced binary trees (later generalized to Btrees By Bayer & McCreight (1972?) - Simplified by Bayer to for red-black trees +

2 3 Trees file 1 & 2 3 Trees file 2 have diagrams of insertion ~~[illegible]~~ path.

Best read file 1 first.

The sub title $\boxed{\text{2-3 Trees V.G - files 2}}$ is very good.

3 nodes: m low, r low | lowest key in mid. child, lowest key in R.H. child

2 nod ~~(m low only)~~ "r low" is a better desc. | lowest key in mid chi ~~[illegible]~~!

— So nodes can grow to t. rt only. (& hyear) (except for leftmost node that can grow ~~[illegible]~~ both rt & left)

Re: Updating nodes after a 2→3 insertion: Each node has (at least) 2 params: Its 2

0 + thresholds. An "update attack" comes from below along 1 of 3 paths, having info to revise t. node that it approaches. The node "state" & t. "attacker" state are combined to give a new node state and "update attacker" state. +

SN In each "search for place to put key", if we keep track of path, we can use this info to guide (update) ~~revision~~ path. & this may not really save time, but might make pgm easier to write. — It means that ~~we don't have to keep track of~~ each child doesn't have to know its parent (tho each parent has to know all of its children) so there is slightly less updating that has to be done.

Perhaps express each node as a ternary number. (maybe use base 4 if more convenient.) So each node has a known place in Memory, & it has (at least 2 assoc. params. —)

30 + Thus, ~~since each~~ child ~~node is known~~ Given a parent, the Ternary notation gives all children: we need know only the lowest leaf of each child. If a parent only has 1 child, the other unborn child can have a r low = ∞, so it would be irrelevant. +

NB: In t. keys of interest: No 2 keys (contents) are ever identical, because they are all of diff't length,

— So we know that a new key will ~~be~~ always be between 2 keys or above or below all.

Doing it "conventionally" At each Ternary point, we have 1 or 2 nos. stored; 2 kids: r low (or r low & m low = ∞) | 3 kids: r low & m low

At first Glance, this Ternary approach may be diff't: As the tree grows, the ternary names of t. nodes change (But in a simple way.) — They get shifted. But don't completely abandon this approach.

40

4TM

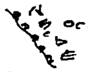
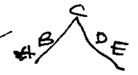
00 (~~34.40~~ 5Pac) , T. very worst case is if t. keys have been already sorted, so they come in ~~a~~ in Lex order. In this case, it takes n ^at the time^ comparisons to insert each key ∴ time ∝ ~~[illegible]~~ $\frac{n(n-1)}{2}$ +
i.e. the monotonic branches ~~[illegible]~~ remain as such, only if each new insertion comes at t. tip (leaf) of that branch.

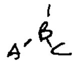
.04 In t. case of a growing string corpus, could such a (Bad) case occure? (06)

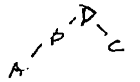
(SU) One could alternate search order up v.s. down for successive keys — ~~[illegible]~~ this would reduce ~~[illegible]~~ sort time by factor of only 2.

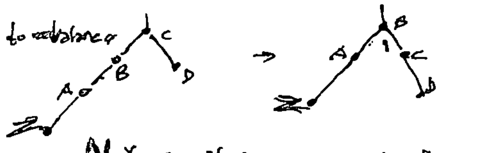
.06 (04) HM! In a situation like .04 – (growing seq. corpus) It is easy to sort it ~~backwards~~ forward The keys are already in Lex order. — Hwr Trying to predict corpus backwards would not find that Lex order useful!)

10 , In 04 t. only way to get it would be seq. like abcdef … z
or a ba, cba, ccba ~~[illegible]~~ t. new letter added to string must be ≥ t. previous letter, so string has to be monotonic, but not necessarily increasing for ~~each~~ character. This would occur in a very unlikely corpus! If this is t. max no. of repetitions, r, max string length is n·R (R is t. radix = alphabet size).
It would perhaps occur in DNA string,)

When  occures, we'd like to rebalance it up as  So and where B has no child —

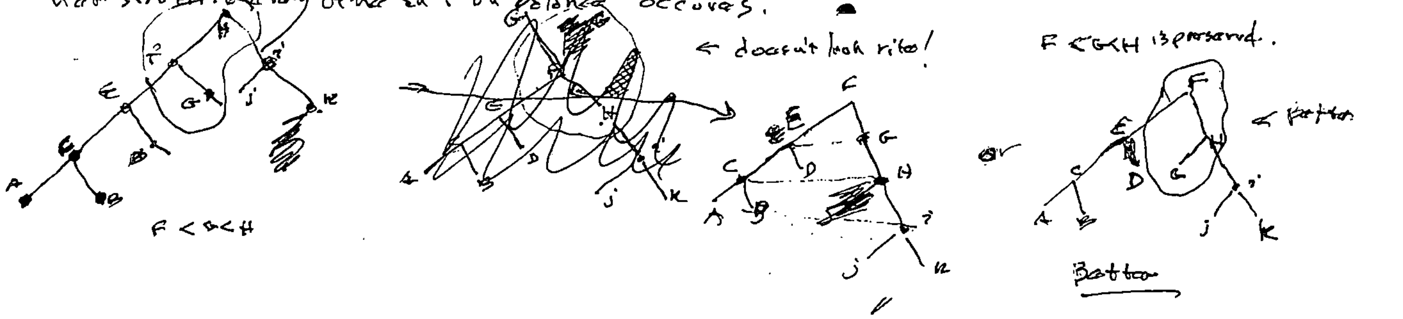
Becomes  would this fix it? I'm afraid not! ~~[illegible]~~ (If t. next D was > C w. no ~~[illegible]~~ child of D)

20 , we get  , Maybe we should do this modify off not be trying to "Balance" t. net ... which is what "B trees" ~~may~~ may do
Perhaps keep record of how much "wt" each part of t. net has; total no of children, ~~[illegible]~~
Actually, tree balancing can be a "local" problem! Each subtree tries to keep itself at minimal "Max depth" — so that the branches sprouting from each node are both of same depth (±1).



Note that Balance has to be > 1 depth off before rebalancing can do anything.

Note: it's easy to keep track of total wt. on each branch! A counter at t. node
30 , increments each time that node is compared to a key. ~~The node counter just has t~~ +
Better than a node counter! Keep record ~~of it~~ at node of difference between no of keys that were sent to left v.s. rt. child". we can "update" net structure every time an "^adequate^ un balance" occures.



40

2.8.04

4TM \$M:00

$(f(x-i)-f(x-i-T))^2$

$(f(x-i)-f(x-i-T))^2$

00: 38.32 26.20 ; Re: "Closeness" method of 26.13–.20: Here we consider $(f(x-i)-f(x-i-T))^2$ wtd over exponential past by $e^{-\frac{i}{k}}$ (26.17). We can optimize k, the width of the "window" or use various other means to measure "closeness" — e.g. $(f(x)-f(x-T))^\beta$ where β might be 1 (median smoothing) or other values. Also "outlyer rejection".

{Also try fractal closeness functions. So: TM would have to try all kinds of variations of "closeness functions" in order to be able to use the Z(4) (predn using "definitions") for predn. (By "fractal" I mean $\int_0^k (f(x) - f(\alpha x-\beta))^2 dx$ is small for some α & β. — "Scale Similarity")

2.12.04

10: ; GA v.s. BZ2: Does GP (w. Lisp ~~Tree~~ Tree Crossover) have any advantages or disadvantages over BZ2 Lsrch? How does Degeneracy (≅ symmetry) look in GP? Is it a factor?

Well, in both cases a good subtree will get a good score if its empirically good — The q is - how much apri wt. does ~~of~~ symmetry get? In both cases, look at the overall operation of the system.

SN To what extent can the ~~s~~ symm affect be ~~somehow~~ approximated by any symmetric operators more wts? — say 2x wt. for "+" or "×". {This doesn't take into account the $\frac{2n!}{n!}$ effect}

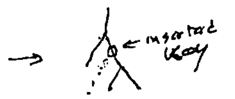
---

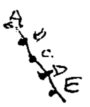
20 ; Re: Sorting: One could have a Binary Tree of codia markers & update it periodicly. Perhaps partial updates possible (I was thinking of this as an improvement of the $n^{\frac{1}{2}}\cdot\sqrt{2}$; $n\cdot n^{\frac{1}{2}}$, $n\cdot n^{\frac{1}{3}}$ etc. systems. So it would have an ~ n ln n speed, but insertion time would be ~ ln n. Total time spent on update might be appreciable. Say we update every time corpus doubles in size. This would not then, be a big time sink.

The forgg. would be ok. if insertions occured uniformly in the set of keys. (which is poss(.)). If not, we may want other ways of modifying the tree to deal w. growths ~~in~~ in certain regions & not in others.

Another way (this looks very much like a "BTREE"). At all times the data is in form of a tree. Each node has 3 addresses: 1) Parent 2) 2 children.

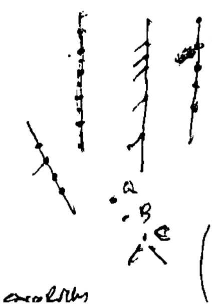
30 To insert a key: Compare w. top node: If key ("low") compare w. (low) key child, loop till done. If key is between parent and child, we insert it into net: it has only 1 child, hvr. We keep track of mean no. of comparisons per insertion (using x windo to average). When mean becomes too hy, we make a new, perfectly balanced net. (Say no. of keys in net = 2^{integer}.) When is true (periodicly) keep track of total no. of comparisons since last ~~[illegible]~~ "re treeing" to determine if we will "retree" or not.



A kind of "Bug" in forgg: say we had a string of keys w. only 1 child each. i.e. 

I new key has value betw ~~A~~ B & C (Above B but below C) Make it a child of C. If C has 2 children, insert the new

40 ~~key~~ key betw. B & C. I'm not sure of this, hvr. — Go over it carefully



3-24-04 Mar 2004 33 2/3

GOLOMB'S PUZZLE COLUMN™

Is this relevant to overlapping genes problem?

OVERLAPPING SUBSETS SOLUTIONS

Solomon W. Golomb

1. a. By statistical independence, the expected number of overlaps is $M = (\frac{a}{N})(\frac{b}{N})N = \frac{ab}{N}$.

b. $pr(k) = \frac{\binom{a}{k}\binom{N-a}{b-k}}{\binom{N}{b}} = \frac{\binom{b}{k}\binom{N-b}{a-k}}{\binom{N}{a}} = \frac{a!b!(N-a)!(N-b)!}{k!(a-k)!(b-k)!N!(N-a-b+k)!}$.

c. $\frac{pr(k+1)}{pr(k)} = \frac{(a-k)(b-k)}{(k+1)(N-a-b+k+1)}$.

2. a. $M = 9$.

b.

k	$\frac{pr(k+1)}{pr(k)}$	k	$\frac{pr(k+1)}{pr(k)}$	k	$\frac{pr(k+1)}{pr(k)}$
0	11.2344	4	2.0403	8	1.0284
1	5.4855	5	1.6586	9	0.8999
2	3.5703	6	1.3865	10	0.7959
3	2.6136	7	1.1829	11	0.7105

c. The *mode* is 9 (same as the *mean*, in this case), since $pr(k)$ is *increasing* up to $k + 1 = 9$, but *decreasing* thereafter.

3. a. $pr(k) = \frac{\binom{90}{k}\binom{810}{90-k}}{\binom{900}{90}} = \frac{(90!)^2(810!)^2}{k!((90-k)!)^2 900!(720+k)!}$

At $k = 9$.

$$pr(9) \approx \frac{(2\pi)^2 \cdot 90 \cdot 810 \cdot (\frac{90}{e})^{180} \cdot (\frac{810}{e})^{1620}}{(2\pi)^{\frac{5}{2}} \sqrt{9 \cdot 81^2 \cdot 900 \cdot 729} (\frac{9}{e})^9 (\frac{81}{e})^{162} (\frac{900}{e})^{900} (\frac{729}{e})^{729}}.$$

b. Except for a factor of $\sqrt{2\pi}$ in the denominator, all the irrational numbers disappear. (The powers of e cancel completely between numerator and denominator. Fortuitously, 9, 81, 729, and 900 are all perfect squares; and everything surviving involves only powers of 3 and of 10.) When all the smoke clears, all that remains is $pr(9) \approx \frac{10}{27\sqrt{2\pi}}$.

c. Numerically, $pr(9) \approx \frac{10}{27\sqrt{2\pi}} = 0.1477564$.

4. a. $Pr(9) = e^{-9} \cdot \frac{9^9}{9!} = 0.13175564$.

b. The largest source of error in 3.c. was using Stirling's formula to approximate 9! in the denominator of 3.a., which gives 9! ≈ 359, 536.873. This is only about 99% of the true value (9! = 362, 880). This "correction" would only reduce the estimate in 3.c. to $pr(9) \approx 0.146$; so 3.c. is almost certainly a better estimate than 4.a.

c. Since $\frac{Pr(k+1)}{Pr(k)} = \frac{\lambda}{k+1}$ for the Poisson distribution, if we take $\lambda = 9$ and $7 \le k \le 11$, we find

k	$\frac{9}{(k+1)}$
7	1.125
8	1.000
9	0.900
10	0.818
11	0.750

which are fairly close to the values in 2.b. (The values will not be as close for k farther from λ.)

5. $Pr(25) = 5.712 \times 10^{-6}$ when $\lambda = 9$. (The true value of $pr(25)$ is about 2.2×10^{-7}, and is actually much smaller than the Poisson approximation.) The student's intuition was correct.

3-24-04

Dec 2003

33⅓

GOLOMB'S PUZZLE COLUMN™

Overlapping Subsets

– Solomon W. Golomb

A former student in my undergraduate course in combinatorial analysis recently wrote to me with a question. The 900 students in the graduate program he is now attending are partitioned into 90-student sections (for manageable class sizes) in each of several courses. These partitionings are supposedly performed randomly, and independently from one course to another. Yet he estimates an overlap of about 25 students between "his" sections in two of these courses, which seemed highly improbable to him. He sought my assistance in addressing this issue.

1. Let's generalize to the following problem: From a set S of N elements, subsets A and B are formed, independently and at random, with a elements in A and b elements in B.

   (a) What is the expected number M of overlaps between set A and set B?

   (b) What is the probability $pr(k)$ of exactly k overlaps between sets A and B? (Use binomial coefficients in your answer.)

   (c) From your answer to 1.b., obtain a fairly simple expression for the ratio $\frac{pr(k+1)}{pr(k)}$.

2. For the case $N = 900$, $a = b = 90$,

   (a) What is the value of M?

   (b) Evaluate $\frac{pr(k+1)}{pr(k)}$ for each k, $0 \le k \le M+2$.

   (c) From your answer to 2.b., what is the *mode* of the distribution $\{pr(k)\}$? (That is, for what value of k is $pr(k)$ biggest?)

3. Stirling's approximation formula for $n!$ says $n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$, as $n \to \infty$, where $e = 2.718\ldots$ is the base of natural logarithms, and $\pi = 3.14159\ldots$.

   (a) In your answer to 1.b., substitute $N = 900$, $a = b = 90$, and then substitute Stirling's approximation for each of the factorials (in each of the binomial coefficients) for the case $k = M$.

   (b) Simplify the expression in 3.a., by cancellation between numerator and denominator.

   (c) What numerical value does 3.b. yield for $pr(M)$?

4. The Poisson Distribution with parameter λ, given by $Pr(k) = e^{-\lambda}\cdot\frac{\lambda^k}{k!}$ for integers $k \ge 0$, is often used to approximate other distributions with mean equal to λ.

   (a) Using the value of M from problem 1.a., what value does the Poisson Distribution give at $\lambda = k = M$?

   (b) The value of $pr(M)$ in 3.c. used the Stirling approximation to $n!$ Which approximation to the "true" value of $pr(M)$, from 3.c. or from 4.a., do you believe is closer?

   (c) How does $\frac{Pr(k+1)}{Pr(k)}$ with $\lambda = M$ compare with $\frac{pr(k+1)}{pr(k)}$ in 2.b., for k in the interval $[M-2, M+2]$?

5. Use any approximation method to evaluate $pr(25)$ for the case in Problem 2. Was the student's intuition correct?

erard J. Foschini Named Bell Labs Fellows (continued from page 8)

'hese people represent the best of the best in the Bell Labs R&D mmunity," O'Shea noted. "The consistently excellent work of ese individuals and their colleagues is the type of role-model &D that is needed to bring Lucent again to the forefront of the mmunications industry." A new class of Fellows is named each ar based on accomplishments in the previous calendar year. Past nners include such luminaries as Dennis Ritchie and Ken Thompson, creators of the UNIX™ operating system; Roy Weber, creator of toll-free calling technology; Nobel Prize winner Horst Stormer; and Federico Capasso, co-inventor of the quantum cascade laser. Profiles on the new Fellows will appear in future issues of LT Today and Bell Labs News.

UNIX is a registered trademark of The Open Group.

2-7.04 Summary of Z141: "exact form" (.19 – .30) 33

4TM

→ see pp $33\frac{1}{3}$, $33\frac{2}{3}$

00 : 31.28 ; Out. overlap of α's ! If α don't overlaps, its easy to count t. possl. parsings. Say there were $\underline{\underline{k}}$ ~~sm~~ simple overlap pairs. How many parsings would become illegal ? A parse is illegal if one or more illegal pairs occures. Say α occures n times. ~~Ther~~ There are $\frac{n!}{m!\,(n-m)!}$ ways to distribute ~~[illegible]~~ m α's among n positions, if we don't consider illegals. ——— How many illegal positions are there?

Another approach: There are k positions that can be occupied in one of ~~2~~ ways. So ~~[illegible]~~ if ~~#~~ ℓ of t. k are occupied we mult wt. by ~~[illegible]~~ 2^ℓ

~~[illegible]~~ : α occures ~~m~~ times ; ∴ there are k "illegal" pairs — out of n possl. — or ~~[illegible]~~ k "special" positions.

n ~~legal~~ positions in total. Of these n, k are "special", having ~~wt~~ wt 2. ← multiplication

How ~~many~~ much/total wt. ~~for~~ for these special cases?

10 There are $\frac{n!}{\ell!\,(n-\ell)!}$ ways to distribute ℓ α's in n places.

~~There are~~ to distribute (m−ℓ) α's on (n−k) ~~[illegible]~~ positions ("≡ ordinary", non-special) : Remainder $\frac{(n-k)!}{(m-\ell)!\,(n-k-m+\ell)!}$ ways.

Woops! t. "−1" should not be in Σ sign — not putting it there multiplies it by k (probably) totally.

this is really ≅ frequency of symbol, α — see 83.04–.05

$$S_0 \; \sum_{\ell=0}^{k} \frac{n!}{\ell!\,(n-\ell)!}\,\frac{(n-k)!\,(2^\ell-1)}{(m-\ell)!\,(n-k-m+\ell)!} = n! \sum \frac{(2^\ell-1)}{\ell!\,(n-\ell)!\,(n-k-m+\ell)!}$$

Subtracts occurrences in which no special cases ~~were~~ are not worried about.

.17 $= f(n, m, k)$.

———

I probably should make this clearer; give examples

.19. Dropping "overlap" for t. time being : T. ≈ "Best" (≡ pc) way to do ~~[illegible]~~ ≈ Z141 prodn !

20 1) T. "definition" cost of α is in 2 parts : ⓐ cost of "new symbol" (which mite be

.21 $\tilde{\alpha}$ (no. of symbols thus far)$^{-1}$) ⓑ a function of length of α. T. data cost is product of

.22 these 2. ⓑ is computed from 2 sources of data : first apriori data: each length has a historical ~~historical~~ pc & an assoc wt. – This is t. current "precorpus" : Second, t. regular corpus in

.24 which we do t. same seq. w. modified "straight" Lap's rule (modified in view of "pre corpus")

The ~~H~~ codes are computed & added in. These codes are always for the known corpus plus t. next symbol). [Perhaps ⓐ usually cancels out because all possl ~~▪~~ variations of α have same ⓐ : T. code not using "α" does not have this ⓐ factor, hvr.] ← False ! see 83.01

To compute t. pc of a single code using α: Just use α w. pc. as a normal "new" symbol, in all alternative (including various "overlaps" if any) codings of t. corpus. This replaces "t. pc of α" with t. "pc of coding via t. individual symbols of α" at

30 each point (we code using α) of t. corpus. → $\boxed{83.00}$ ← 82.22; (82.23–.34)R V. Good!

83.27–.29 is a counter Argument to this ~~#~~ "ll coding scheme

So .19 – .30 is currently my best realization of Z141. The calculations needed for ll codes need to be done, hvr. I thot of using TLU (table lookup) in computer codes; but this is not as fast as I thot ! Present CPU's use 2 / many "caches" : Primary is at ≈ cpu clock speed & is 16k by or 32k by (was 64k by for one of the V.g. AMD machines). Secy (or L2) cache of ~ 256k or 512k. w ~ 7 cpu cycles access time.

If "table" were in $\boxed{L2}$, it mite be possl. to get ~~[illegible]~~ ~ 1 cpucycle mean access times, but perhaps not ! I'd really have to have some idea of what t. caching algms were befor ~~[illegible]~~ any guessing !

Otherwise, it may be easy to do approximations of functions like (.17).

40

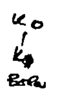
4TM

n	n·lnn v.s.	$n^{1\frac{1}{2}}$ v.s	$n^{1\frac{1}{4}}$
10	+ 23	31	18
100	460	1,000	316
1,000	6,907	31,622	5,623
10,000	92,102	1000,000	100 000

I write routine to $n^{1\frac{1}{3}}$, etc, ~~both~~ automatically, but I'm not sure its very efficient memory-wise.

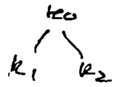
Also, perhaps I can design a ~~[illegible]~~ B tree that would work O.K. for this applicn.

Start w. one key₀: key₁ comes in. / We then have

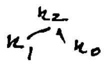


$k_1 < k_0$ $k_1 > k_0$ or the other way around, depending on ordering of k_0, k_1.

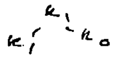
k_2 comes in: If $k_2 > k_0$; we have



If ~~k₂~~ $k_1 < k_2 < k_0$



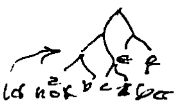
If $k_2 < k_1 < k_0$



— only 2 bonds ~~[illegible]~~ modified.

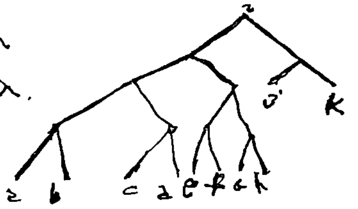
It would seem that if one had an ordered binary tree, that one could always insert a new item & maybe only change 2 "bonds".

Hvr, while that's probly true, the resultant tree would not be "balanced" — the branches would not be of = length, so the search would not be of max speed.



Perhaps periodic rebalance: Say we got ✓ at one point: Its only slightly "unbalanced"

say



Break! — In the usual BZ2 they sort a set of strings of fixed length. Using that sorting method, how much extra time needed to insert 1 more string in the Bunch?

or n	v.s. lnn	v.s. √n	v.s ∜n
100	4.6	10	3.13
1000	6.9	31	5.6
10k	9.2	100	10.
100k	11.5	316	17.7
Meg	13.8	1000	31

00 (4JM spec 30 ~~28.~~ 28.30) : In particular, say we have available M bytes for this Matrix. If we have D dif'nt
symbols & we only put t. first D' into t. Matrix, & t. Matrix is of dimension ℓ,
what is t. tradeoff betw. D' and ℓ so as to get minimum mean time per insertion?
~~[illegible]~~ It will vary a bit w. corpus size (since t. no. of symbols slowly ↑ w. corpus size —
— for English text ---- solution will be dif'nt for dif'nt Corpi)

Perhaps best just get a Btree pgm. on t. net — Tho t. details of t. pgm. seemed rather
complex, T. idea seems conceptually easy! With t. various levels in t. tree, hvr,
it would seem like t. cost of "log n" ~~[illegible]~~ for insertion time, would keep it to large!

Another way: using ≃ 28.09ff, I could keep a ~~[illegible]~~ linked list of ≃ ~~[illegible]~~ $\frac{n}{2}$, $\frac{2n}{2}$, $\frac{3n}{b}$
points in t. list, — which I do a search on first (This is ≃ "Btrees")

10 SN SUMAC ∃ "BZ2" Say we do ordinary Lsrch & get large set of pgms for "Corpus up to now"; We then get larger corpus & using t. same set of pgms, we get an ~~effectively~~ ~~[illegible]~~ narrower distribution, within that set of N. As t. corpus extends, the "good" ~~[illegible]~~ codes will slowly wander outside t. original "N". T. Backtracking idea was originally used to ~~[illegible]~~ "refresh" the "N best", w.r.t. t. extended corpus. ⌐ say N of them.

Could we use BZ2 to do this "≃ "Backtracking"? The Corpus would be t. set of codes best for t. augmented corpus. BZ2 would give trials "near" those codes.
Unfortly, ~~these~~ set of codes will probly have a common begining, then branch. The tree structure of t. pgms after t. "common ~~[illegible]~~ trunk", will be extrapolated by BZ2: T. Q is, would it be extrapolated any difrntly from t. way we normally do trials to
20 extend codes? My impression is that it mite well be quite difrnt & perhaps much more effective in getting hy pc codes for t. augmented corpus!

.22: 30.36 → AH!: perhaps/ very imp't idea in coding using ~~pgms~~ definitions of pgms: Say we code a (corpus + next symbol)
using defn "α". If α occurs n times in t. corpus, we can code corpus 2^{n-1} dif'nt
~~[illegible]~~ ways (all of which terminate in α). They will each ~~[illegible]~~ have their own pc, but
t. total pc will be >> an individual α's code. Also t. ratios between t. pc's of t.
next symbol — obtained using dif'nt ~~[illegible]~~ suffixes, will be much difrnt from that
.27 obtained by not using this ‖ coding. Note this is a v.g. way of ~~[illegible]~~ dealing
.28 w. t. ambiguity of (30.11–.13) when t. α overlap! → 33.00 Also Note 82.01
.29 In English, hvr, it may well be, that t. "~~Best~~ single" parsing is ~~[illegible]~~ much
30 ~~[illegible]~~ more likely than others, & that ~~[illegible]~~ using that particular Bias
would be v.g. for English text. — That t. "sum over ‖ parses"
would be better for other ∃ kinds of corpi. ~~[illegible]~~ If ~~[illegible]~~ ~~[illegible]~~
"T. sum of all parses" (~~did~~ does) not work well for English, this strongly suggests that
English is not a Stochastic ≦ FL.

.35 Recently, I was thinking that using t. length of α as its "definition" would be
cheaper if used later in t. corpus — when, presumably, regys in t. corpus would make
t. direct defn of α, cheaper. I hadn't figured out how to do this, hvr. — also, its
probably only a small difrnce & only impt. for smallish corpi… ~~[illegible]~~ where t.
difference tends to be small, anyway: & for large corpi, t. difrnce is larger, but less signif't.,
40 relative to t. total r cost savings via many uses of α. — So perhaps .35 is not impt.

4 TM SM .37

00 + After t. coding of t. 1st symbol, old pc's will be Mult. by $(1-\frac{1}{r})$ {since $\frac{1}{r}$ was t. pc of t. newly defined word that was used to code corpus} T. next time t. new word is used, it makes this factor $(1-\frac{2}{r})$ — so we have these factors $(1-\frac{1}{r})^{\ell_1}(1-\frac{2}{r})^{\ell_2}(1-\frac{3}{r})^{\ell_3}\dots$ in the pc calculation (I'm not certain to the exact details à formula, hvr), Also t. factors giving t. pr of each time t. new word occurs $\frac{1}{r}, \frac{2}{r}, \frac{3}{r} \dots$ — (Woops! $(1-\frac{m}{r})^{\ell m}$ can't be right, because m can be only bigger ... ≥ r!)

.06 A way to think about 29.29 ff! ə is a poss. symbol in t. alphabet that has (at t. beginning) "not yet occurred": so t. alphabet has, say 27+1=28 symbols. After ə has occurred, it's coded just like any other alphabetic symbol. After ə occurs t. next symbol gives K (which is a corresponding ... over t. integers). We then

.10 + code t. rest of t. corpus, using t. symbol "ə" for t. string that we've defined. +

.11 < Note ~~that~~ this gives some ambiguity in coding, i.e. ə's occasionally overlap! → (t. normal PPM method does not have this diffy) > Because of this ambiguity, we want

.13 to choose t. ə's backwards, starting from t. end of t. corpus → Hvr. see 31.22-.28 for v.g. way to deal w. overlap!

This coding method does give a pc for t. (tentatively) augmented corpus; we get t. pc's for various augmentations, using various poss. contexts. — from this set of predictors à their wts, we get a prediction for t. next symbol.

To shorten this process, we may be able to use approximations, à omit many contexts that have clearly much less wt. than t. [redacted] highest context.

.19 T. Q is: ① ~~Is~~ Is t. coding method of .06 .10 .195 legit.? ② does it give any large pc for t.

20 + correct symbol than **PPM** or PPM* or various improved versions of PPM? +

③ Perhaps Most impt: Is .06-.10 generalizable to things that PPM is not?

④ An interesting point: .06-.10 uses (potentially) all poss. contexts in ||: Is this any better ~~than t. usual PPM?~~ → #② (.195)

.06-.10 differs from PPM a little in how "ə" is dcrbd. PPM is perhaps more economical in its dcrn, but it only dcrbs 1 ə, à doesn't have benefit of || dcrns.

PPM uses several escape chars to dcrb "ə"; .06-.10 used t. equiv of 1 escape à a "length" dcrn.

T. sequence of escapes in PPM could be coded as a single symbol (or perhaps 1 escape, followed by a "length" symbol (as in .06-.10) — but in PPM t. range of length is much smaller ~~[illegible]~~ At t. present time, I don't

.30 see how I can restrict t. length of ə (as in PPM) except, perhaps probabilistically — since ə can grow larger as corpus ↑ in size. Also PPM has various constraints on ə that further reduce its cost — .06-.10 doesn't seem to have such constraints. +

At present, it seems like t. methods PPM uses to dcrb. length of "ə" à to dcrb. ə in general, are very ~ to those of .06-.10 à to Z141 in general — T. big difference being PPM chooses "t. best" single "ə" à also uses "restriction" info to choose ~~to~~

.36 t. decision on what "ə" is best. → 31.22

.37 SM SN In discn of General idea of Context as prediction see how $X(t+1) \approx f(X, X-1, X-2 \cdots)$ + noise fits in.

40 +

4JM

00 A possl. wting ~~mite~~ ~~be~~ amount that a given context compresses t. corpus — but this criterion depends on ~~[illegible]~~ what t. basis of comparison is — how many bits/symbol if th. context is not usd. If we compare 2 contexts that are usd for t. same (or same no. of) symbols, then ~~[illegible]~~ this effect cancels out, — but not if it doesn't! E.g. say context a is usd 10 times and has pc = pc_a for those 10 times (product of pc's) ; " b " " 5 " " " " " pc_b " " 5 " (" "). If, w.o. using contexts, pc = ~~[illegible]~~ P_0 per symbol of corpus, context a ~~[illegible]~~ increases our pc by $\frac{pc_a}{P_0^{10}}$; " b " " " " " $\frac{pc_b}{P_0^5}$. } the relative ~~[illegible]~~ utilities of ~~[illegible]~~ a v.s. b is $\frac{pc_a}{pc_b \cdot P_0^5}$.

So it depends much on $\boxed{P_0}$.

10 For each symbol done by context a, ~~presumably~~ there is a pc for context a, but also ~~[illegible]~~ there's a "raw" coding cost. Unclear what it'd be: it could be a simple ~~[illegible]~~ code based on symbol frequs, or a more sophisticated code.

Hvr, consider "T. coding of t. present corpus": (here "a" occurs ~~[illegible]~~ 5 times ~~[illegible]~~ only).

14 ——— a ——— a | a | a | a α β a |

.15 We code before t. "a"s as we have been doing in t. past. The first time "a" occurs, it's not used for ~~[illegible]~~ contextual predn., so we can ~~[illegible]~~ find t. pc of t. non-a coding of a,

~~Unfortly, t. coding of that first a interval will often not have "edges" at a's "edges"~~ ~~[illegible]~~

Since every ~~[illegible]~~ primitive symbol in t. corpus is incrementally ~~[illegible]~~ coded & given a pc,

.20 In .14 we can find the pc's of each of the "a"s in t. corpus ("a" is ~~[illegible]~~ usually not a single symbol but a longer string). Hvr, its not so easy to compute t. ↑ in pc due to our defing "a": Some of t. ~~[illegible]~~ past occurrences of "a" were usd to make predns — i.e. it was usd as a "context" ~~for~~ for predn

Go back to ~~[illegible]~~ : At t. end of t. string of .14, we want to know t. pc ↑ brot about by t. defn. of "a". For cost of defn, use the pc of t. components of "a" in its last use (those found to be better than earlier defns?). (Maybe we need to use t. penultimate defn use of "a", since its this last use that we are now evaluating! Another possy is that we must use t. first occurence of "a" to obtain its pc (as in $\boxed{.15}$)) •

29 One way to do t. coding (I'm not sure this will work). We code: {T. coded

30 corpus up to now plus one symbol}. We do it by defining a word (context) This includes the extra symbol. There is a symbol in t. alphabet, ~~[illegible]~~ I, that occurs only one time — it says that we are defining a word. The integer, k following I, tells how many characters long t. word is. (T. word defined is ~~[illegible]~~ t. last k chars befor I. The subsequent pc of that word is given by Lap's rule. ~~[illegible]~~ If R was ~~[illegible]~~ original radix, I gave R → R+1. As soon as t. new word was added, it became legal R ~~[illegible]~~ → R+2, but I became illegal, so ~~[illegible]~~ R+2 → R+1. So R+1 is t. constant radix, but t. pc of t. extra word changes during coding (as do all other ~~[illegible]~~ symbol pc's)

If I occurs after t. ~~[illegible]~~ rth symbol, its pc will be, say $\frac{1}{r}$.

40

1.29.04
4PM

SM(.31)

00:27.40 SN In comparing w: Human predn. of English: The escape to "symbol never occured" is ~~extremely~~ rarer — tho for any finite corpus, it will occasionally occur : i.e. for any finite corpus, almost always there are "words that have not yet occured". Otherwise, after a word has started, using a dictionary will often give pc = 1 for most of t. following characters.

On SORTING This is (apparently) needed for BZ2 a/o PPM.

I'd like a sorting pgm in which its easy to insert a new "record", and also easy to find files before a given file (say all files within 100 "~~down~~ up stream" of a given file — i.e. "earlier" lexically.)

Another operation is getting t. "next symbol d.f." for ~~[illegible]~~ each length of t. associated ~~contexts~~ +

.09 One simple (tho slow) method involves: We have ~~[illegible]~~ corpus = string of symbols that grows.
10 , Each symbol has sequential address. We then put "addresses" in lex order", by
having ~~each addr~~ at each address: a Lex forward ~~[illegible]~~ address & a Lex "backward" address.
.12 This enables one to insert new addresses in t. system.

Trouble is, if length of corpus is N, it takes on t. average, N comparisons to insert a new address.

One way to decrease this time: We have "k markers" in t. corpus that mark $\frac{1}{k}$ intervals of t. lexical ordering of t. corpus. So instead of starting at t. top & doing ~~[illegible]~~ $\frac{n}{2}$ comparisons; One does $\frac{k}{2}$ comparisons to find what k interval t. new insertion is at. — Then we jump to middle of that interval & go up or down, lexically, to find rite place. ← woops! — No! we don't know where middle is! so go to a "k marker".

.19 After each insertion, t. positions for 1 or more "k markers" may have to be updated. ^ I think only 1 (?)

20 + This method speeds things up by a factor of k.

We could also do a multi level "k" by having each k interval have "sub k' markers". — So we could, w. enuf levels, end up w. a B TREE (Balanced Tree).

It would seem that .09 – .12 would be easy to pgm & use. Meanwhile, I can be thinking of better ways to do it. "B Trees" may be v.g. — More complex to pgm. The things being sorted are as in .09 – .12 t. addresses of ~~[illegible]~~ successive symbols in t. corpus — that represent "shifts" of t. corpus.

Re: The insertion trick of .19: It would be easier (only 1 update per insertion) (at most & usually no update), if we stored an r dimensional array, the addresses of t. lowest (or highest) key with each r symbol ~~[illegible]~~ suffix. A 27^3 array only has ~20k elements — $27^4 \approx .5M$: still quite achievable. So this could be easy to do & speed up things a lot) It works well if t. symbols are ~~not~~ about = frequency — otherwise, it still works but we don't get as much gain. If we only use most frequent symbols in matrix we can get them most of t. time — which is very good! 27^5 = 14M : O.K.

30 → (31.00)

31 :26.20 SN : ~~SM~~ SPM Study Audio (lossy often) Compressn. ~~[illegible]~~ Many "tricky" Modern Schemes.
32 See how to Adapt to SM — similar for (self direct) or cross predn. → (3.4.00)

On Context in general: In 1 dim context, repeated (Bernoulli) contexts are are always completely nested & i.e. are linearly ordered & easy to describe, by an integer, say.) In >1 dim., contexts are partially ordered by t. ~~[illegible]~~ "covers completely" relation. In 2 (or more dims) ~~[illegible]~~ contexts t. partial ordering gives many "nests" of sets of nested contexts. Within a "Nest" t. criteria used in PPM compression can be used: But how to decide betw. different nests, or how to weight them
.40 is not clear.

SN Also Note "Fractal Context" occured before but in form of shrink or expansion of ~~present~~ exact present context: Any affine xfmn, say: also add noise.

1.28.04
4TM

00:(spec 25.40), using ~ BZZ : also by better ideas of what a Context may be.

From 90Sul 14.20 For the first problem I have a number for my input string.

Apparent T. soln. I'd like is " If Num then Push : for +, −, × only, its pc = $\frac{1}{4\cdot5\cdot3}$

.03 . . Hvr. another soln is Push ; which is St → op, ← No! op →

I'm not clear on how Push is used. Does it push whatever the input point is at?

.03 is prob'ly wrong. The only way to get "Push" is viz If ... Then ...

we then have chose betw num, x_s $+_s$ $-_s$ and # (end). There is no cond (num OR x_s OR $+_s$ ...)

```
   ↓ initial symbol
St → If Cond Then
   → Call (number of string in Memory): nos. from 0 to n; n ↑ as TM solves problems, stores stuff
10 → St : St      (concat)
   → do St until cond {= cond or do}
Cond → num ; $+_s$ ; $-_s$ ; $x_s$ ; #   ← end = blank tape
op → push $+_{op}$ ; push $-_{op}$ ; push $x_{op}$ ; push stop
```

To end a pgm, "If # Then stop" must occur in the pgm — usually at end.
Putting it at the end of all pgms would do no harm! TM could easily lrn this viz BZZ. (sequence)

.17 so (If # Then stop) $= \alpha$ could be first pgm in TSQ. It is the correct response to a blank input tape.

My first machine put all problem solns in Memory: A QATM in Phase I would use the set of solns as corpus, & would need no "memory unit"

20 One Q. I wrote about is: should we use a Bag of solns or a Set of solns?
"BAG" means that if we use a soln. N times, it gets wt. of N in Bag. I probably want this, but it would seem like there are situations in which it is not desirable.

Use "BAG" to start out.

To start, we insert a blank tape a few times & "α" (.17) gets reasonable wt. (= pc).
Next we put a No. on tape. correct response: Push no. on tos, then stop.
~~[illegible]~~ "If No. Then push α" (stop) $= \beta\alpha$: so $\beta\alpha$ occurs a lot.
" β If $+_s$ then $+_{op}$

It looks like BZZ might be able to lrn things w. rather hy pc's (small cj's!).

30 Perhaps try a simple version of BZZ (To start.)! Then perhaps later use an improved version, & see if it makes much difference.
With this BZZ: Try it on a simpler GA problem: Say Koza's n input multiplexer (ΣN input "If"). See if I can find a way for it to discover the loop giving a general recursive soln. for all n.

# Two possi. ways to do it: ① Use QATM to give n (Q) ; pgms (A) pairs for n input mux.
Do it for n = 1, 2, 3, 4, ...k say: see how long it takes to get soln. ~~[illegible]~~ for k+1, for TSQ up to k — (for various k).
② ~~Same~~ Same as ① but TM always tries ~~to soln. [illegible]~~ to find a function relating n to pgm for n. ③ given ~~[illegible]~~ n, pgm n pairs, up to n = k, TM tries to find pgm k+1 as a funct of pgm k.

6.7.90 Paul   This is a dup of a dup!   ± 5

NITS 266   DUP   14

7/19/03 for refs

dynamics associates

Paul's Grammar w. defns:

st =? "statement"?
yes:
Maybe unnecy - it would just go to make 'st' long

statement

.103 st → If cond then ~~op~~ op else.

→ call (number of ~~st~~ to defns)

→ st ; st

→ do st untill cond {end do}

~~→ [see 17.21 for loop inst.]~~ → loop : st : If cond endloop

end

can be a st however

This enables a statement to consist of several statements — I guess a multi-line prgm

is this? / this?

$+_s$ / 7 / 3   → top

$\frac{1}{4}\ \frac{1}{6}\ \frac{1}{4}$

.08 cond → Num ; $+_s$; $-_s$; x_s | #

if # is a condition, It has to be linked to something (stop)

.09 op → Push | $+_{op}$, $-_{op}$, x_{op},

.10 Num → $\{0, 1\}^*$ finite (binary string.)

";" a concat symbol? .... prob.: Then I don't think it needs to be on of "17": its "16"

$[0, 1]^*$ ↓ 15 (until is unnecy).

.12 Alphabet = Σ = Σ [If, ~~Then~~, (;) do, ~~until~~ end do, Num, $+_s$, $-_s$, x_s, #

No! (If, then, else) is one word!

Push, $+_{op}$, $-_{op}$, x_{op}, 0, 1]   17 symbols.

Looks like 14 symbols.

[IF, call, DO, Num, $+_s$, $-_s$, x_s, #, Push, $+_{op}$, $-_{op}$, x_{op}, 0, 1]   5   15 symbols

Perhaps don't need until "14".

(No division!) ← Actually no trouble adding division: ÷ ϕ is no problem.

first try seq. soln. :

.20 If Num then push. (pc = $\frac{1}{4.5 (4)}$ = ~~80~~ 80   Not "3"?   from input list

This prgm ends up w. number onto tos. which is what we wanted.
Top of Stack

"If" examines Input list;

"Push [redacted] from inputs. object on input list at pointer position, onto stack, increments pointer index.

If Num then Push: "Num" is not a terminal ∴ illegal.

But still, let's expand our idea of Grammar: Σ?

So "Num" is not a terminal, but a language (≡ set of strings).

25 $\boxed{\text{If}\ +_s\ \text{then}\ +_{op}}$ $pc^{-1} = \cancel{96}$ 80 $(\cancel{96})^{-1}$

$\frac{1}{4} \cdot \frac{1}{6}$ ≡ α_+ $\frac{1}{4}$ = $\cancel{96}^{-1}$ 80^{-1}

Def If we want our grammar to have defns: we have a special symbol bd Between defns. & at the end of all defns., [redacted] ~~the end of all defs is "ed"~~

~~between defs; "bd".~~

If the grammar ~~[illegible]~~ begins w. "ed", then there are no defns.

~~If After the last defn, we don't use "bd", we use "ed".~~

Grammar → ~~[illegible]~~ Def, statement

Def ~~→ # [illegible] statement~~

→ ~~statement~~ st bd Def

→ bd

bd

$(st\ bd)^*$

~~statement~~

4TM

00 + On n dim (vector) context: I thot of this in terms of ngmsts. If $S_{u,v}$ is T. ngmst of all possl. seqs from u to v, one can prbly create t. ngmsts corresponding to contexts in 2 dims (& prbly more) by using ngmsts of this sort (w. suitable combination rules) & ngms. (also note (.10) Z) +

.05 On ~~[illegible]~~ region contexts for continuous variables. t. sequences s(t) & r(t) are within pc distance d, if we can go from s to r ~~or~~ r to s w. $\overset{\text{binary}}{\text{info}} \cdot 2^{-d}$.
How it is used: we use PPM (≈ Bzz), but 2 contexts $\boxed{\text{are "identified w. wt } 2^{-d}\text{"}}$ if t. 2 $\overset{\text{context}}{/\text{sequences}}$ are apart by distance, d. Of course one simple inter distance measure is ~~[illegible]~~ total square difference — which tells how much info is needed

10 + to "correct" ~~[illegible]~~ s(t) to produce r(t). Another way s & r could be related is that r(t) = s(t − T), where T is a "cheap" number (like a lattice spacing) +
(see .00ff for more ideas on this).

.13 $\boxed{\text{SM}}$. One way to use context of the forgy (.05ff) kind for time series predn: Say we have a finite corpus length L. Consider discrete time, tho continuous time is about t. same).

Not → $D\left(\frac{1}{L}\sum_i \int_0^\infty dt\; e^{-\frac{t}{T}} \left(x(t) - ~~[illegible]~~ x(t-i)\right)^2\right)$ ← May be not exactly what I want! ~~[illegible]~~

(No!) Discrete time again. Compare x(t)'s history w. its history as of T ago. The value of T ago had a prediction of Y(T). give this prediction wt. of

.17 a function of $\sum_{i=0}^{\infty} \left(x(\overset{\checkmark\text{ now}}{t-i}) - x(t-T-i)\right)^2 e^{-\frac{i}{k}}$ (k is some smoothing constant — we may want to either optimize or sum over all (or many) k values)

20 + [i.e x(t − T + 1) = Y(T)] || We $\tilde{\text{a}}$ average ~~p~~ wtd. predns for .2 = 1 | L. → 34.00 spec 28.31 on SM +

& (Also consider fractal compression for (SM) → see (28.40))

.22; 25.16 ; (.05 — .20) suggests a view of context giving t. penalty of $\frac{1}{D^k}$ (of 25.15) ; ← No: that penalty is quite dfnt!
1) T. context that has longest length of match w. current suffix, is given most wt — or least dcrn cost. 2) Contexts ~~that are~~ k symbols shorter than t. longest require $\log_2(D^k)$ bits to dcrb them. — & so they get that wt. ! On second tht, this seems WRONG! ~~[illegible]~~ If t. longest match prefix is of length L, then I'd give wt. D^{-k} to a prefix of length L − k. Actually it only takes t. cost of integer (L − k) to dcrb that context. — since ~~there are~~ D^k such contexts in all we should give out $\sim D^{-k} \cdot D^k \sim 1$. to such a set of ~~[illegible]~~ contexts,

30 + — tho $D^{-k} \cdot D^{+k} = 1$ we also have to add in t. cost of specifying k — which is t. pc of integer k (or D − k if it is smaller). — A possl. pc of k might be $\frac{1}{L+1}$, because k can be any integer betw. L & ∅. — A very nice way to code this 1 out of L; Intipret t. ~~[illegible]~~ context of length L − M has given a pc = f(M) to t. correct continuation — on t. average. So we can ~~[illegible]~~ use M to code that context length. +

Another possy is to find average pc of correct contn, associated with a context of length k ; then renormalize because we know 0 < k < L.

1·27·04 4TM

? 740 E 90th t Near St. Clair Ave. ?

E. Cleveland Hayden ave

00: TSQ [redacted], PSM analysis, etc.

On 16.24:26 I listed a few sources of PSM's: To be added: T. writing of a "TSQ" by an ordered listing of capabilities: (perhaps obtained from an Elementary Alg. text) Each capability is (perhaps) a kind of PSM. Try inserting ~~more~~ (capabilities/PSMs) Then try factoring.

.06 SN In more recent version of "Z141", The contexts/ Always overlap — which makes them Not "indep". BZZ takes advantage of this dependence. Z141 ignores it — making it, perhaps, much weaker. Is there a better way to deal w. this "dependence"? — As t. long that t. context ↑ we have progressive "specialization"?

.10. — Perhaps best viewed as "Nested BAGS". — i.e. monotonic ↑ of case counts as we ↓ "specialization", PPM (≈ BZZ) uses ~~longest~~ string that contains symbol to be "predicted". In t. joint apriori space for pc of contexts of a Nested Bag of strings, there is a certain constraint. (17)

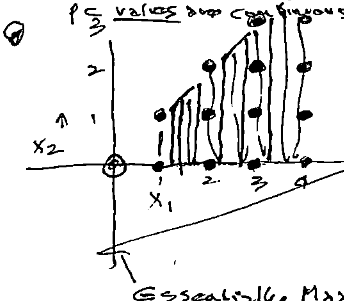
.12 When we neglect this "Dependency" then if a given ~~context~~ ENTIRE context string has K difnt. symbols that have followed it in t. past, & there are D legal digits in t. Alphabet, then ~~that~~ string & has

.15 a pc penalty of $\sim \frac{1}{D^K}$ ..... $\sim \frac{1}{D}$ for each time a new symbol was added to its "repertoir".

.16 So [redacted] context strings w. "small" repertoirs" are given more apriori wt. → See 26.22 for a difnt way to get wts!

.17 (.12) Just how can we characterize this constraint (using ideas of .10 – .12) in pc space, rather than the "case count space" of .10 – .12? Recall that t. usual Lap's rule is obtained by ~~integrated~~ integrating over a [redacted] uniform apriori of pc's. Well, an easy way to deal w. this is to have a case count

20 + space (integers), & have a P. Density over that space (it is a discrete point space, but t. pc values are continuous). Consider just 2 (directly ~~dependent~~) dimensions of that space.



||||| is region where pc > 0 can occur. Actually, this could be ≈ a case for Max Ent. Method (Max ent. w. constraints) but ALP should be able to do it more exactly. I think I did do an integration for uniform apriori w. ≥ 2 radix. — t. integrations was difft., but I think doable in "closed form" (i.e. exactly)

Essentially Max Likelyhood w. ~~aprips:~~ & constraints. $\prod p_i$ = Max (p_i are pc's assigned to events that did occur)

30 → I'm pretty Much "Ready to Run" w.r.t. TSQ/design ~~[illegible]~~ (PSM factoring) etc. ~~[illegible]~~ A "PSM" certainly doesn't have to start "at t. beginning": I can (& have in t. past) Considered any TSQ that goes from one "Set of knowledge" to another "SOK". Start writing TSQ's w. "Gaps", errors, ad hoc solns., etc. — then use t. "Repairs" of 15.20 ff. to fix them.

Start off w. 90 Pa01 14.00 – .40 This is that Fortrish lang that I used for ANL in Saarb.

. I thot of 2 diffys in t. TSQ that I wrote/analyzed: 1) That pc's of new probs & too rapidly due to my ~~[illegible]~~ throwing all prob solns into Common Memory — so retrieval (P cost) was $\sim \frac{1}{N}$ N being no. of probs. 2) T. only abstractions I put into Memory ~~[illegible]~~ were entire problem solns, no "sub-functions" or "subabstractions".

. I may be able to deal w. both of these problems now — partly by ~~27.40~~ spec 27.00

4TM

.00 On Methology: **Working Style:**

In t. (distant ⓔ) past, I would work on any part of a problem that came to mind, with certainty that when I needed to use t. soln. to that part, the soln. would come to mind. Unfortly, this has not been true for quite some time, ā much of my past work is <u>not accessable</u> to me — except perhaps by laborious searching thru my notes.

.05 Some ways to deal w. this difty:

1) When I have an idea about a sub-problem: Don't work on it unless ⓔ I have a clear idea on how t. soln. fits into t. whole picture, ā can make reference to the rite point(s) in t. "grand scheme"

.08

2) Devise some kind of "Data Structure" that outlines t. ~~[illegible]~~ "Large picture" of t. problem

.10 ~~Rel~~ with arbity fine grain, so I can tie references to various parts of the scheme (i.e. t. partial works of .05–.08

3) Don't <u>exactly</u> do 1), but befor working on a problem, DO outline a <u>bit</u> where it should be "attached".

4) The "4 ll PaRs to TM" of 16.24–.40 (Summarized in 19.31–.40) seem like t. beginning of "places to <u>put/attach</u> "<u>work done</u>"."

5) In ④ there are a few more ideas: Getting old A.I. probs w. "solns" from various sources. Also TSQ's that I've prooted. Find ways to <u>factor</u> t. explicit or implied PSM's.

.17 Getting these "factors" leads to construction of a simple Bernoulli Grammar of t. factors. A Bern grammar can (w. derivs a/o BZZ) have t. pc's of conc's depend on "context" {local context}.

.19 A more <u>powerful grammar expands</u> t. idea of "context" to include

.20 characteristics of t. problem that condition which PST's are best to try (e.g. "OOS"

.21 "R" recognition function.) — But <u>any</u> $\overset{S}{P}$-funct on strings defines (t. most general)

.22 kind of "<u>local</u>" context.

6) From 19.35–.40 — I <u>suspect</u> that PSG discovery will <u>not</u> be a very impt. problem for finding good PSM language: that <u>Bern</u> or <u>BZZ</u> type Grammars can be used — but the things being concatenated will be strings, pgms, procedures, "Methods". <u>Or</u> <u>I might</u> use a <u>CF Lang</u> like <u>Basic</u> or <u>Forth</u> and where t. <u>Grammar is</u> (<u>known</u>, prespecified <u>fixed</u>), ā t. problem is to discover useful functions (W. or W.o. "Side effects"!).

7) Perhaps enuf of Prob-Solving <u>Theory</u>: Start work on TSQ construction, Problem soln., Heuristics collection, PSM collection, ~~PSM fact~~ factoring <u>PSM's</u>.

.30 From $\boxed{\text{factoring of PSM's}}$ will come ideas on how to make good PSM Grammar

OB - OP Algebra .28

~~1.24.05~~ 1.24.04

4TM

00 It is mainly t. multiplicity of possl parses that gives rise to t. definition's q in pc.

In English text, hvr, this would tend not to be true. An alternative parsing for spoken text (i.e. w/o stated spaces betw words), would result in serious "parsing" "pun" ambiguity, (i.e. "parsing puns").

I could use Z141 in t. manner of BZZ. Just use t. exact formula for t. pc of t. corpus, for ~~using~~ each legal following symbols for ~~each~~ ~~past~~ suffix that has occured ≥ 1 time. This could be time-consuming, but I could run it overnite, or not use such a big corpus. ~~Mainly I want to compare it to t. other methods.~~ If it seems good, I can work out faster approxns. (e.g. 22.31) The main thing about Z141 is that it is well adapted to other kinds of regularities — e.g. numbers

→ Also, perhaps usable for PSG discovery

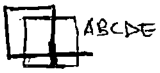
Note that t. best compression method (in Bell, Moffat, Witten 95: ps\1.7.04); PPM

10 was slowest by a factor of maybe 20, than "Huffword" t. next best compressor.

.11 No → In the Z141 method, we will have to compute t. pc of all possl. following symbols — (not just t. ones that occured) This is so we can normz t. pc. So this slows us down by a factor of maybe 20! (If we use an English dictionary, many beginnings of words will only have a few possl. contins (i.e. pc > 0). « dictionary economy — see .14

.14 → In the Z141 method, we will often give zero pc to most continuations for most suffixes since those continuations have never occurd in past. So actually only a few pc's of contnn. will ~~[illegible]~~ have to be computed in most cases. ← This is true for English or other Nat lang texts that have "words" in them. For other kinds of preda, we may, indeed, have to consider all poss. values of t. next symbol.

20 (21.13 / 21.16) → PSM & TSQ's: 21.00 – .16 certainly relates them closely! Once I have "factored" a PSM or Heur, I should be able to write a TSQ to acquire it — {perhaps not so simple — but I would be well on my way to being able to write a TSQ to acquire t. ~~[illegible]~~ PSM/heur. That PSM's have many "modules" (≈ concs) in common, makes it easier to write TSQs for many of them. After TM has found a lot of PSM's (~~not~~ necessarily in factored form), it already has a kind of Barn grammar for combining them to give cand heurs. → side Note 24.17 – .21



.26:2040 Note that a PSM will normally consist of a "Recognition" part on when to apply it — and an "Operator" part that is to be applied (t. Ob-OP Algebra).

.28 SN t. Ob-Op algebra suggests a Boolean Belief Net (BBN): A "BBN" could consist of a set of Obs only: T. output of several obs can be t. input to a new ob. In BBN's these

.30 will be ~~[illegible]~~ s-obs (stochastic observations). A ob maps from strings/reals to True, False. A Op maps from (string/real) to (string/real). A s.op is a conditional pd on (strings/reals) A s-ob is a ~~co~~ pd. on (strings/reals). A (s.ob) is a kind of (s.op), having no string/real output: only a pc output.

.34:2037 When a string is a program, then concat may have special meaning. If a string is a self del. pgm, then any ~~[illegible]~~ suffix leaves it invariant.

— So for concat to be useful, the first part can't be self del.

1·23·04

4TM

00:2640 . T. General Conclusion of (21.30–40) is that as stated, t ~~[illegible]~~ Univl. D.F.'s do not ~~nearly~~ completely solve t. problem of induction. There is, of course, t. choice of Ref. UMC. — but this seems to be in addition to that! Consideration of 21.36–38 mean that one could put both kinds of bias into t. "Ref UMC. choice" : based on "successful (choice experience) in t. past." The way in which this is better than previous models of induction, is that we understand, much better, what t. problems are.

:06 Hvr. in Sol 64a : (All paus method) one can avoid this problem by using Partial recursive "PEMS". As one of t. C.B. we do approach t. Univl. D.F. (≡ UDF ≡ UPD) | That this doesn't consider partial functs, makes it subject to criticism: of 21.30–.32

.07 Hvr. One could still bias one's search for approxns to t. UPD by not doing trials in an "obvious" [redacted] — simple-order — in fact t. usual srch methods of Humans (used in Phase 1 & in Phase 2) ~~could~~ bias results considerably.

10 So, If one insists on simple srch in "obvious" order, then choice of UMC ~~[illegible]~~ gives all t. Ambiguity/bias in UPD approxn. | Since one I never uses t. "obvious order" the bias is in both choice of reference UMC & in choice of search method, that give bias, & perhaps certainty that t. method need not converge to ~~[illegible]~~ U.P.D. ?

. Well conv. term ~~does~~ itself isn't even dependent on choice of UMC! It just says that if t. UMC has a (short / long) dern of pd that generated data, t. error will converge (fast / not so fast).

[W.r.t. search techniques: It may be that a certain class of techniques will always eventually get arby close to UPD. If so, I'd like to characterize such srches in various ways so I ~~[illegible]~~ could perhaps try to confine myself to them.

20 [A search method is "complete" if for any P.R. / FEM cand, (or any input string to t. UMC) one can show that eventually it would be considered by t. srch method. ~~[illegible]~~ Any "complete" srch routine on a universal machine does eventually find any ~~model~~ generating model, so t. conv. term holds for that (srch / UMC) combination]

Good

Good!

⟹ SO T. conv. term is true for approxns: If t. ref. machine can derb. t. data source: If t. srch routine must eventually (derb / find) t. data source. for a universal ref machine & a complete srch heuristic, we will eventually get t. small error specified in Sol 78T3.

1·24·03 SN On old Z141 & new Z141 & AZ : T. original Sol 64b version was wrong (at least) because t. didn't consider ll parsings — (or if I did, I didn't do it right). In recent work (Around correspondence w. Wolff), when I added up ll parses for different r = 1/R, I assumed certain factor in each term was indep of r — it was not.

30

31 A better way to do these sums: Most have a peak & a S.D. about that peak — I can very well approx t. sums from knowledge of t. μ & σ² of t. distribution.

.33 Also, while my method was not correct for .335 α coding w. new definition; recoding w. new defn loop to α (.335), it is O.K. to get ~~[illegible]~~ do one pass of t. .33 loop to get t. pc of t. last symbol in t. corpus.

T. method might still be adapted to finding t. "words" in t. corpus (Wolff's problem) if I did as Wolff did.... reparse ~~[illegible]~~ entire corpus after each new defn.

I did recently Consider using definitions that have fairly hy pc & are of fairly certain parsing in t. corpus. ~~[illegible]~~ Hvr, after each such definition, it's necy to do Wolff's reparsing.

Note, However, that one could ... ... for certain definitions → 22.00

1.22.04 INDEX: NOTE on writing: 18

4TM

19.37 spec → On PSM factoring & TSQ writing: That they are closely related is suggested by 19.35: i.e.,
.0:20.40

"One of Main problems in TSQ writing is suitable factoring of solns. into commonly used parts" PSM factoring & TSQ writing are even closer than a superficial similarity.

In both cases "commonly used parts" is essential — so in both cases the "parts" have to be entities that have much use elsewhere ... otherwise they have excessive rcost & aren't really "heuristics". Any "heuristic" by definition, ~~is~~ is the has been found, statistically to be of frequent utility use (in the circumstances that "call" "it", a/o are "called" w. extremely hy pc by "logical reasoning" — (deduction)).

In both analysis of PSM's & analysis of pgms to solve specific problems, I have to ~~[illegible]~~ ~~[illegible]~~ break up the PSM or ~~s~~ soln. pgm into "reasonable, reusable modules" — & that's the main thing.

If the modules are truly generally useful, I should be able to devise a ~~seq~~ set of problems that uses each module (≡ a TSQ). (This last isn't so clear! A module can be common to many problem solns, yet not have a set of problems leading to it!) → 23.20

.13 To make a TSQ leading to ~~a modules in~~ acquisition of a new module, one must find a
.14 problem in which that module is the only New (hy rcost) module in the soln. to that problem.
.15

.16 .14–.15 is in the rite direction, but I'm not sure it's entirely true/adequate/nacy! → 23.20
1.23.04

Put this index on computer for easy printout of updated index

.18 SN I need to make an index of impt. ideas that I often refer to: some ideas.

← Also, try listing ideas in order of
1) Importance
2) Frequency of "reference to".
(Give 1 mark each time I reference it)
or put date of each time I reference it!

1) ~~#~~ many Several ways to list cands in pc order

2) T. [redacted] desn of Search ANL tsq.

.3) " " " " linear eq. solver, linear → quad → cubic solver.

4) List of Optzn techniques; w. links betw. them. Preparatory to writing Grammar for them

5) "How to solve problems" for humans/machines.

6) List of Probs in TM: Solved & Unsolved (P~30 of some notebook) ~50 problems.

7) The "Stack" of ideas partially worked on that I want to get back to.

8) Soln. of "Chess" problem. [Hyperorder Univl. d.f.: Not nearly mathematically correct! ☹ – But used by humans: probly very impt.

9) (Perhaps) List of "Reviews", w. some impt. main pts. of Each.

SN In SOL 64a: "T.a ALL PEM's Method": Th. error was that "All pems" were not recursively enumerable. As a result, when one selected a search scheme a/o an

32 ordering ~~sch~~ scheme for cands, one always omitted an ∞ of them.

Do all approxs to SOL 78 T3 (t. conv. thm) have this (Bad) property — or is that way of approximating the univl. d.f. a Really Better way?

22.06 –.07 solves this problem.

.35 In the "conv. thm" I think I did consider alternative ~~appr~~ Approxn Methods —

.36 — that one could bias the rel. pc of 0 & 1 an enormous amt. One "apparently" "good" bias method would simply use Levin Srchs in simple pc. order —

.38 (No "heuristic srch") — Any bias would depend on choice of Reference Unc.

So: A possl. (tho rigorously untenable!) way would be to use ~~[illegible]~~ orderings of "PEM's" or ordering of trials for "the conv. thm" of types that have worked well in the past

D-ngmsets are D-Languages

0:19.40 On creating new ngmsets (from old ngmsts): for d-ngmsets: we can combine them by Boolean operations to produce new ngmsts.

When a d-ngmst is defined as the Range of a function (e.g. a CFG or other ~~[illegible]~~ Grammar. Not necc generative.) ~~[illegible]~~ : We can combine any d-ngmst a_1 by ~~any function~~ a ngmst defined ~~[illegible]~~ as Range of function $f_1(\cdot)$ as the set of strings $f_1(x_i)$ where $x_i \in a_0$.

~~We can do~~ For ~~the~~ ngmsts defined by functions, we can cascade these functions to obtain new functs w. assoc Ranges → ngmsts.

Here we talk of "functions" that map strings to strings.

.08 For ~~[illegible]~~ s-ngmsts, The commonest kind of data, is ~~[illegible]~~ a ~~proby~~ distribn. specified by proby distribn on strings — essentially a stochastic Language. ~~By~~ (composing / cascading) such functions we obtain new ngmsts (≡ p.d.'s on strings).

We can mult/2 s.langs (π. pc's of) to obtain the analog of AND-ing of 2 p.d's. — Perhaps Renormz

" " add " " " " " " OR ing of " " — " Renormz.

"Not" has no apparent analog, but if P_1 & P_2 are 2 p.d.'s on strings, then $P_1(1-P_2)$ is of interest & is normalizable if P_1 is.

$P_1(1-P_2) \neq (1-P_1)P_2$ ~~[illegible]~~ (also $\neq$ in D-ngmst case).

So P_1+P_2; $P_1\cdot P_2$ & $P_1(1-P_2)$ have meaning.

$P_1(1-P_2)(1-P_3)$ may have meaning.

18 ~~[illegible]~~ $\prod_{i=1}^{\infty}(1-P_i)$ may have meaning. It converges when $\sum P_i$ converges. (But in any case $P_0\prod_{i=1}^{\infty}(1-P_i)$ always defines a pd)

Hvr. .18 is usually not a P.d. (i.e. its usually not normalizable) — if ~~[illegible]~~ there are regions where all P_i's are very small, (.18) will converge to $> \phi$ in those regions & ~~if~~ if those regions are very large .18 will be unnormzable ... but this matter has to be looked at more carefully: we'd have to be sure that .18 was large enuf in a large enuf region, so that .18 was unnormzable.

$(P_1)^n$ is of interest: It sharpens P_1 — so for big n we have a very narrow d.f.

$(P_1)^{\frac{1}{n}}$ for large n broaden the P.d.f.

Monte Carlo generation of ngmsts. For both d & s ngmsts ~~we can put~~ defined by functions, we can put in random input & get a D-ngms or s-ngmst out.

If we want a uniformly distributed output for d-ngmst; We can use a uniformly distributed input ~~[illegible]~~ standardly if no 2 inputs give same output.

30 → Gen. Remarks on ..00ff: ① Instead of Ngm SETS! Have NGM BAGS! — the freq. of the items is retained! So one can get Bern Prediction from a single NGM Bg.

~~[illegible]~~ d-ngmbg is close to s-ngmst (a wtd ngmst)

② New ngm (Bgs? / sts) from old by Concat not mentioned ..00ff! Concat enables ~ CFG's

We might Genz NGm ~~[illegible]~~ st to $[a_i, b_i]_i$ If $[c_j]_j$ is a diffnt ngm bg then $[a_i c_j b_i]_{i,j}$ would also be an ngmst: How to Genz Concat to Ngm bgs is unclear.

~~[illegible]~~ also $[a_i\, d_j\, b_i\, c_k\, c_i]_{i,j,k}$ would be an ngmst. These last 2 operations enable

.37 CFG's. The problem is finding heuristics for good trial ngm sts of these kinds → 23.34

(2½) We might also get (effects of) Recursion by: x=a: Do i=1…∞; $\underset{\text{ngmst}}{x} = x\hat{}\underset{\text{gmst}}{a}$ OD

This is like: x=a For i=1…∞: x=x+a: Next.

40 ③ [illegible] → 2326

1·21·04 ~~[illegible]~~ MSST.DFMT.DLL 119~~[illegible]~~84 116K ↑ Jay

4TM Windows (system) / Aug 9 198

10: 18.40 ;, In general : If I find a certain concept, or heur or "idea" to be obvious/simple, then I must learn to be able to modify t. H. ~~[illegible]~~ or devise TSQ for TM so that it feels same way.

e.g. In ANL, I had idea of "quantity", which eventually morphed to "eval".

I never did actually implement this. In SAAB ANL, this ~~[illegible]~~ concept was perhaps somewhat realized by push/pop to t. Stack.

Conceptually, I had t. idea that there . . . ?

08 SN An impt. Heur: Say TM xfms a problem₁ reversibly into a new problem₂. If TM knows how to solve Prob₂, it should do that; Or if Prob₂ looks like its easier to solve than Prob₁, then work on Prob₂

The concept of "reversible xfmn" is important. (used in linear → quadratic → cubic eq. soln)

Example .08 : Solving Equs : ① When eq. is in "linear form", it is solvable ② when eq. is simpler (fewer terms, funcs), it is usually easier to solve. ③ Putting equs in linear, quad, cubic forms makes them solvable.

20 :17.28 SN Paradox about ~~Symmetry~~: Say t. alphabet has only 4 symbols: a, b, +, *.

So pc of symbol is ~ $\frac{1}{4}$. How can we get 4 of ~4n by adding new symbol?

Well, we only get ~~[illegible]~~ × 4n if we have at least n difrnt symbols ~~[illegible]~~ plus + and * (az)

→ Still, this seems to give pc's > 1 occasionally! If t. ratio is > 1 this means that having more mult args is more likely than less .... so recursively, ∞ args is best (Unlikely!)

Well, we have to add at least 2 symbols, * & a_n ; * ~~as~~ pc is ~ $\frac{1}{2+n}$ ~~[illegible]~~ (assuming only +, *, a, [az] are legal. pc of ; pc of a_n is ~ $\frac{1}{2+n}$; so pc of 2 symbols is $\left(\frac{1}{2+n} \cdot \frac{1}{2+n} \cdot 4n\right)$

or about ~~[illegible]~~ $\frac{4}{2+n} \sim \frac{4}{n}$ for large n — so pc does not get > 1, but we ~~[illegible]~~ may get an effective pc for the * symbol alone to be 4, but it must be assoc w. another a_n, which (for larger n) gives pc of $\frac{4}{n}$.

30

31 :16.40 : out 4 11 parts to TM at 16.24-.40 : ① & ② involve collecting & factoring (PSM's hours). from various sources. (A.I papers, books, reviews, old classic ~~Real's~~ Peal's book on heurs, t. Duda Hart Stork book, ANL, algebra ...)

③ Is writing TSQ's : ANL, elementary Alg, & soln of various diffys, symbolic integration, perhaps symbolic soln of diff. equs, etc : ③ is impt. to Program me to write TSQ's.

.35 ~~[illegible]~~ The "factoring" in 1), 2) is closely related to TSQ writing. It's a skill I must have a/o develope. In my developement of it! Make notes on t. process so eventually, TM can do it. — →21.00

.37

.38 → ④ PSG (Grammar) discovery : May not be as difficult in Practice as in Theory! : I.E., in Practice, one may usually have some good ugmct or parts of them (18.14R) — To drop and then to obtain new ugmct from old, is not { Actually, a B22 "Grammar" may be usually good enuf for most "PSM Invariance" : T. problem is only to "factor t. PSM's well" }

1.20.04 IMPORTANCE of Phase 1 for training TRAINER in TSQ Writing .12 10:08p philiz 10 12 13

PSG DISCY: A NEW TRICK 1.14

4 TM

(spec 17.14) (17.30) : BZZ for Recursive function discovery. A recursion method is discussed in t. "IDSIA ~~[illegible]~~ report" In terms of t. AZ lang. (maybe in t. assoc Appendix). Its likely That BZZ could use that formalism to dscr recursive functs.

Another way is how it was done in OOPS (~ Forth lang.).

Is There a difrnce betw Lisp & Forth in t. way functions are executed? In Forth with a partial defn, t. machine can execute some of t. instructions & then terminate, loop, or ask for more inpt. In Lisp, we put in t. entire expression ending in a "end" symbol; Only ~~at that~~ when t. whole text is available, does Lisp begin execution. (.22) .07

1.21.04 SN Since GA (or BZZ-GA) now is a more or less Adequate Phase 1 device; I could use it, to design Phase 2. So main problem would be 16.24, .26, .27: (1) Collect good A.I. probs w. heurs for solns. (2) List good PSM's ~~[illegible]~~ (≡ heurs), (3) work on ≈ PSG discy.

12 [An impt. part of Phase 1, is mainly training me to discover, identify, heurs & write TSQ's for them. It trains TM, too, but not nearly in th. critical area of discovery & extrapoln. of grammars for PSM's. 14

14R [Also Note on Grammar Induction: After TM has Discovered (or Been Given) a few NGMCTS — Maybe partial — It is usually much easier to discover new ones & ~~expand~~ t. old ones.

An Interesting Note! After I insert a grammar for PST's into TM, it will discover new PST's via that Grammar. As such They will ~~be~~ in "factored form", which tremendously simplifies ~~[illegible]~~ T. extrapoln. of such a grammar.

Were I to insert a new PSM into such a TM, w/o adequately "factoring" that PSM, TM would have a "hard time" integrating it into its Grammar — If it could do such a thing at all!

2 (.27) SN Thots on "Saarb ANL" & Recursion: After trying "eval" for +, −, ×, ÷, I want it to be able to learn $7\times(3+2)$ i.e. $\text{eval}(7\times(3+2)) = \text{eval}(7+(\text{eval}(3+2)))$. Then I want it to lrn more ~~general &~~ complex evalns. & eventually be able to lrn a recursive rule for evaln. In t. Saarb pgm, I may have had a "do" loop w. "until" as "Stop" condition. — so this partial form of recursion may not be so diffclt.

SN On TSQ writing in general. I guess t. idea is to devise a Conc. Net for a Difflt problem, then try to find ways to ↑ t. Condl. pc's of t. Concs that have to be lrnd. This last is something I didn't fully realize in 5019, until "Scaling" problem became Apparent.

A possl. "Top Goal" for a TSQ: To lrn linear Regression: This involves (at some "level of understanding") how to solve Simult. equs. — But also t. idea of optimization (?)

The "Evaln. of a general Alg Expressn" was something Saarb ANL did, — but w/ excessively rapidly ↑ cost for each new idea. I really understand "Context" ~~better~~ now — perhaps go back & fix it up — perhaps use ~BZZ!

To add new operators $\sqrt{\ }$, $\sqrt[3]{\ }$, sin, cos, tan, $\sin^{-1}$ ~~[illegible]~~ e^x, $\ln x$, would ↑ span of ~~problems solvable~~ expressns. evaluable: But prob. first I'd want it to lrn to solve linear, then some nonlinear equs. Also simult. equs.: several ways: 1) By "subtraction" or other analogous ways 2) substitution 3) Matrix inversion (mix of 1) & 2))

1·20·04

4TM

0:16.40 : $\boxed{SN}$ I want to use BZZ for function generation in lex order. I thot that it would be nice to know each function as a separate string, and consider all shifts of it, then put shifts of all strings in corpus in lex order. Then use PPM algm.

.02 ——

A simpler way would seem to be to just make a long string consisting of all functions, with special symbol between functions. This would amount to English text in which sentences (corresponding to functions) are separated by the sequence of 3 symbols, ., space, space. Look into details of these 2 apparently different ways to deal w. this kind of "chunking" into functions or sentences.

It would seem that .00 – .02 would take less memory, but this depends on how we ~~store~~ "store" the shifts. In PPMC we only need to store substrings of length $\lesssim 6$ or so, but in PPM* we store substrings of arby length. .115 ↓ For functions, we will probably not use very long strings, (but possibly long strings of multiplications or additions due to hyp "mod accuracy"/ symmetry redundancy) ← <This last reduces the cost of * or + symbols, but the cost of the symbols being multiplied or added, will still be the same.> → (18.00)

14

15: (3TM ~~558.15~~) 558.15 No such address. 3TM 459.15 = 4TM 7.15 is relevant

$\boxed{SN}$ for the ~~[illegible]~~ symm of .115 → .14; a bit stronger! The redundancy factor ↑ much more rapidly than the cost of the $\overset{mult}{(*)}$ symbols! → $\left(\frac{2n!}{n!}\right)$ ← I think this ~ 4^n for large n:

$$\frac{2n^{2n}}{n^n}\frac{e^n}{e^{2n}}\sqrt{\frac{2n}{n}} = \sqrt{2}\,\frac{2^{2n}\cdot n^{2n}}{n^n}\cdot\frac{1}{e^n} = \frac{4^n n^n}{e^n}\cdot\sqrt{2} = \left(\frac{4n}{e}\right)^n\cdot\sqrt{2}$$

~~[illegible]~~ ?

so function ratio is ~ $\frac{4n}{e}$

19 $$\frac{n+2\cdots 2n+2}{n+1\cdots 2n} = \frac{(2n+1)(2n+2)}{(n+1)} = 2(2n+1) = 4n+2$$

20 $2n+1$ ~~[illegible]~~ $\frac{2(n+1)!}{n+1!}$ = $\frac{2n!}{n!} = (n+1)\cdot(n+2)\cdots 2n$ | $\frac{2(n+1)!}{n+1!} = (n+2)(n+3)\cdots(2n+2)$

ratio = $\frac{(2n+1)(2n+2)}{n+1}$ as .19 = $2(2n+1) = 4n+2$

This "factor of e" error is familiar! perhaps examine old Z141 acidity in liter. 24R

$x! \approx \frac{x^x}{e^{x-\sqrt{x}}}$ ·

23 $$\left(\frac{4n+4}{e}\right)^{n+1} \Big/ \frac{4^n n^n}{e^n} = \frac{4n+4}{e}\left(\frac{4n+4}{e}\,\frac{e}{4n}\right)^n$$

24 $$= \frac{4n+4}{e}\underbrace{\left(1+\frac{1}{n}\right)^n}_{=e} = 4(n+1)$$

Could I have missed a ~ trick in my analysis of Z141? (when I got what seemed to be a spurious $\frac{1}{e}$ factor) ∴ Hvr. if I need that result, I should test approxns on various exact trials. → (~~[illegible]~~)(19.28)

28

Q: I ~~thot~~ I might need ≈ (AZ144) for "definitions" i.e. I used recursive definitions.

30 ~~There may be~~ ways to get recursion & still ~~[illegible]~~ use the BZZ. → (18.00)

@ $\boxed{SN}$ Re: King/cover got ~~1.25 bits~~ 1.29 bits/symbol for best/human predictor

1.25 " " " "Committee".

This is << the ≈2 bits/symbol of the BZZ. Try predicting symbols — via Humans! — see what techniques I use that are beyond BZZ's. Perhaps have group of people try to predict next symbol. Get Grace, Alex, Alice ☺, Murray, "Science Group", Dan R. For each char, the S. will be given a list of legal possys: also a dictionary to help choose. Usually, the big problem will be the first few letters of a word.

1.19.04
4TM

ABCDEabcde

Another approach: that the dern. of neonatal brain is << 100 Mby due to ~~redundancy~~ ~~Genome~~ Genome

200! 15:40 : to derbit pgm mite be larger (tho this would ~~not~~ Not amount to negative compression! since "understanding" is parallel codes of corpus).

The reason is, that "Understanding" means "Alternative Ways to Code the data", & there are an ∞ of ways to do this. Useful limitations will be nec. necessary to use each "understanding" of a proposed problem: Presumably, TM will have a v.g. function that assigns an 'understanding" to problems as Conditional PC: "Conditional" is problem.

Another Aspect of "State Storage": That we store state ~~...~~ infrequently & store changes to the state after each problem. When each prob. is Given to TM, TM updates wrt that problem: At this point, it will know which parts of itself have changed, & store changes on disc. We then store maybe every 100 or every 1000 (total states of system)'s.

To retrieve, we do update changes since last state.

10 In Summary, the ideas of 15.20 ff on "TSQ repair", ~~would~~ make it a lot eas ~~ier~~ yet to write TSQ's. With 12.30-.40, 14.00-.06 (Alternative Paths to TM), I should have no diffy devising TSQ's: The ideas of 12:30 "Alt Paths..." — Summarized (1) ~~These~~ are all "Phase 1" approaches — designed to get to Phase 2 (2) We solve all 3 kinds of probs: Induction, OZ, INV. (3). We obtain Probs from AI lit. (with (partial) solutions) & Heurs a/o by considering reasonable "paths of Lrng" in Algebra, calculus, ect. (in which case I will be devising solns & heurs.

For the "discovery" part, use ≈BZ2 a/o GA a/o {GA augmented implemented} by BZ2.

We can use PD1, PD2 (since this is Phase 1) (15.00 is good)

One Big problem is / my making grammar(s) for ~~...~~ PSMs of all ~~...~~ 3 kinds of probs

Another ("Advanced" - needed for Phase 2) is discovery of CFG's. 14.40 ff looks like Good approach

20 (Not necly / Discover CFG's but / something much like them. Maybe a certain subclass of CFG's)

(Also General discovery of Recursion (13.24-.27, 14.30-.38))

So: Main II Projects:

.24 1) Collect ~~...~~ A.I. papers w. good probs, good Heurs for solns.

3) Design TSQ for Alg, Calc, Maple drills .... Do not TSQ then "repair" using 15.20ff → use ≈ BZ2 to "Learn" them TSQ

.26 3) List Good PSM's for all 3 problem types — they will have many features common (for all 3 types). Try to factor them & devise Grammar.

4) Work on PSG discovery: Discovery of Recursion (13.24-.27, 14.30-.38) Read Koza's PhD thesis for good ideas.

This should be applicable to devising, discovering Grammars for PSMs (.26). Also Note discovery of recursion in OOPS (Forth-like lang), which may be able to use ≈ BZ2.

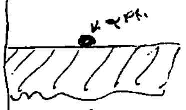
30 Notes: Relation ① & ② In trying to find out how to discover heurs in ① I will be factoring the heurs: so ① & ② are necessarily closely related.

5) Try using BZ2 a/o GA on various Lrng problems. Perhaps try BZ2 on known Problems worked on by GA. Note again that GA is a kind of automatic TSQ.

see 19.31

use MS word or some other w. p. that can do equs.?

For 2D "objects" (≈ BZ2



contact region of y n-1 2 contacts ~~...~~ ~~compact~~ connected bits including a

1.18.04

4TM

→ TSQ REPAIR .20 ← Good

CFG discovery .00 ← Also Good

00:14:40 In line w. this, consider defg. approach to CFG discovery:

One defines, initially ngrams by considering y.set of ngms that often proceeds (follow) a specific single ngm. After several small (finite) ngrams have been defined, one trys ~~to~~ to accept various concat rules will norm e.g. say A & B are 2 ngrams & C is a third: we ask — Is A⌢B → C reasonable? We also ask if A⌢B → A is ~~reasonable~~ reasonable --- a recursive rule.

In the Corpg, its necy to ~~define~~ (conjecture) ngrams. This is contrary to the spirit of BZZ, which does not use data (normally ~~within~~).

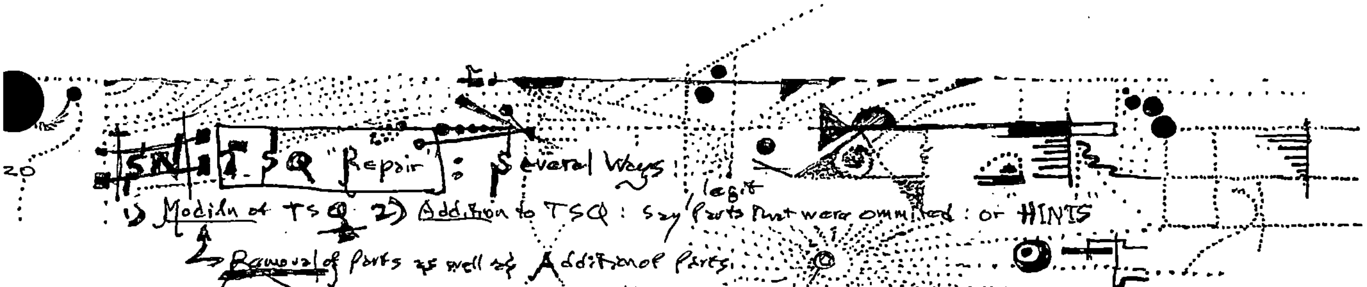
(Predictive by Partial Matching.) → PPM

Bell, Moffat, Witten 1995 in PSF 1.7.04 — page 28

In a review of things like BZZ the Authors (~~Moffat, Cleary, etc~~ Witten) said that 2 bits/symbol for English was about as good as "the community" ~~has~~ has Gotten: Using other regy types like CFG's, ~~did not improve this greatly~~ knowledge of corpus, etc, did not improve things much.

They didn't say that! They said that conjecture that CFG a/o corpus knowledge would help, had not been verified — Not that they were false!

— So the bit/symbol of Shannon's Guessing, was still fairly distant (a factor of 2 in further compression.). Hvr, for a MTM corpus (and perfect MTM predn) the compression will be much higher (I guess ..... much will be 0 bits/symbol, but much will be random choices — or a few possibilities that ~~where~~ inherent in the choices involved in the subject matter. — i.e. the entropy involved in the defn of the MTM problem.



SN ~~4~~ TSQ "Repair": Several Ways:

1) Modifn of TSQ 2) Addition to TSQ: say legit parts that were ommited: or "HINTS"

↳ Removal of parts as well as Addition of Parts.

③ Modifying Primitive set of insts ④ "Wireing in" all or part(s) of soln(s) to problem(s) — ideally, in adequately "factored" form, so TM could learn Maximally from that Soln., but this can be a fairly ad.H. soln. — in which case, the problem is poor little (mg.) to has place (?)

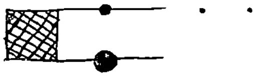
Using these 4 repair ~~methods~~, it should be easy to write TSQ's: Then as I learn more about TSQ writing, I can go back & "re repair" the TSQ better.

Actually, some TSQ's can be very bad — in the sense that they lead in the wrong direction, so they get to a local peak & can't continue. — An enormous amt. of Backtracking is then needed. So I will have to keep complete record of TM's training plus Occasional storage of system states. I can arrange to store system state that occurs befor each problem Soln. that took a long time. Hvr. modern disc storage storage may make it feasible to store all systems states (betw. problems). It may be necy to have a special mechanism for this: e.g. Store state in a special fast RAM & while TM is working on problem, load that RAM onto disk. If the state is as much as 100M; w. a 1000G hard disc that's 10K states storable, which seems like far more than enuf!

Proby 100M is not far from what a person could know: 2 bits/sec × 3×10^7 sec/yr × 50yrs = 3000 bits = 400 Mby

400Mbye would be only 4k states storable. I don't think this is a serious problem. i.e. I can store essentially the complete history of TM,

Hvr., on second thot: while 500MB maybe "one man's input", the amount of info needed → 16.00

1.17.04  14

4TM

The NIPS conf proceedings On-line | That Online A.I. jounal ← A.I. encyc — More recent. also That "A.I. encyc." I have

.00: (Spec 12.40): 7) Work on Various TM-Type problems in t. Literature (e.g. That Duda Hart Stork Review Book) see if I can find ways to solve them in a "unified manner" — This would be (essentially) my building up a grammar of PST's. Try using BZZ in many approaches — see if its better than t. approach used in t. original problem.

Next, see if I can find a function that looks at a problem & assigns (probabilistically) PST's to it.

.06 For General prob solving — see to what extent a "Phase 1" approach is "Adequate" → see 16.10 for Summary

10

20 SX An impt. project: To Index my Notes: How to do it: Go thru a yr. or so & list topics giving a brief descn of each category. Then try to Make a scheme of topics & their relations — to help find things.

Some Impt. Things to Look for:

Ideas on 1) CFG discy

2) GA tricks, extensions

3)

13.27 [13.27]

30 ON TSQ's & BZZ: One early idea in ANL is t. idea of Eval (expression). We teach TM eval 3+7, 9×12, etc. ⇒ eval(~~#~~ sum 3,7), eval(mul ~~#~~ 9,12). The parens tell what is t. argt of "eval". Next. eval(~~#~~ sum(8, sum(3,7))) Here we want TM to realize/discover That this last = eval(sum 8, (eval sum 3,7)). There is an impt idea here — t. idea ~~#~~ that sum(3,7) is a kind of "object". We want to make it easy for TM to (linguistically) realize that certain things can be regarded as "objects" & belong to classes & t. classes ~~[illegible]~~ of these objects can be manipulated in ways that correspond to t. way individuals can be manipulated.

38 Eventually, These ideas should lead to (13.24-.27 recursive) ~~[illegible]~~ definitions.

→ 15.00

1.16.04 13

4TM

S-Functs w. Continuous params — find by Lsrch .35

$\frac{10.2}{3} = \frac{20}{3}$

$2^{\frac{20}{3}} = 1000^{\frac{2}{3}}$ = 100

80 × 10 msy × 100 hrlon

.00: 12:40 : Genl. discn. : Main Bottlenecks in TM That I want to work on;

1) Is writing TSQ's really t. biggest problem? If so, I'd want to write some, Then see if t lrng. Them can be made feasible

2) I plan to use Lsrch w. a BZZ to find solns to ~~prob~~ many kinds of problems: To what extent is a BZZ adequate? | Can it really deal w. Function traces?

2.5) How bad is t. symmetry problem? One could do Lsrch w.o. considering it, but t. loss in speed would be considerable. Any partial solns. can be directly translated into speedup of srch. Tho this bears directly on t. efficacy of BZZ, it also bears on any Lsrch technique — & possibly ~~any~~ probability directed srch scheme

See 44.00 for rev. of various cmptideas

.10 3) In 12.31 I had this idea of working on Grammars for Inductive method, OZ method, INV prob solver
PSTS PSTS PSTS,
The (I) may be able to find common features in various PST's, & make a CFG or CSG or HMM. We want TM to be able to extend this technique. Are there any CFG discovery techniques that are any good? I've written a lot on PSG discovery, but I don't know where it all is .... I think I had some good approaches. | I have lots of Literature on CFG discy, that I could read for inspiration.

How Critical is CFG discovery? How bad is it if only a BZZ is used (≈ "Bernoulli Grammar").

4) In (3) can I really get some good Grammars ~~for PST's~~? . .

TRE For function (usdata) would BZZ work equally well backw or forwards?

20 SN On BZZ: I had Idea that ordinarily "definitions" were unnecy if BZZ was used! That defns. saved time in srch, but commited one to particular Pause. — So Defs. really not essential (except for speed & Communication w. Trainer).

24 What about Recursive defns? I had a way of doing them in "AZ" that BZZ wil ~~be~~ be able to realize, ~~but~~. — But it seemed like a not so good way of doing Recursion. — i.e. not as simple as normal "Lisp" recursive defns. — Tho perhaps AZ could do recursive

27 defns same ~~way~~ Lisp does! I'd need special symbols for recursive defns. → (14.30)

SN On GA & BZZ: In our corpus for BZZ, we may want to weight t. past examples! Related to Gore! If for induction perhaps wt = pc. This would give more wt. to

.30 "short" cands (hy pc) & result in parameters giving short cands (which is desirable). — i.e. No Bloat!

SN Possl. reasons for working on TSQ's (or even ~~GA~~ problems from GA community) first — would be to get to know just what kinds of regys TM has ~~[illegible]~~ to discover. Go thru Literature (e.g. That Pattern ~~dsc~~ finding book by Duda Hart Stork), ect to find both problems and PST's.

3 TM:453.40

.35 SN ∧ S-Funct w. Continuous params : Do ~~[illegible]~~ discrete params in usual pc order for Lsrch. For each cand, for ~~whatever~~ remaining time is available, first do random trials. If we seem to have gotten "on t. Hill", start using Non-linear Optzn (Quadratic ~~[illegible]~~ local approxn). Since "Expected value" of pc of a random trial ≡ the continuous factor of t. pc of t. cand. ← (This is Diffent from t. SUMAE Method of 453.26 — 454.07, — which seems v.g. Hvr. look at discn of 452.24ff for/ideas for objections to (.35))

1.13.04

Rev

SM .20

4TM

.00: 3TM448.40 : SN We can cascade functions, so t- ~~[illegible]~~ domain of one function is t. range of another.

03 SN I was thinking of making a "Grammar" for a large set of impt PSS's!

Do so for prediction methods in general : including methods of finding them

05 & clues ~~[illegible]~~ ("obs") on a corpus that suggest which to try. (.20)

SN Both Optzn & INV probs can also have a Phase 1 (as well as a Phase 2) — i.e. in Phase 1, .02, we look for ~~[illegible]~~ a single Universal Algm. for OZ probs. T. algm. looks at the problem defn & decides what to do (ob - op). Similarly w. INV probs.

In both cases, new (partial universal soln) cands are "like" successful old (cands for Unvl. soln)! This

.10 "similarity" can be via BZZ or any other induction method. In both cases t. relevant "corpus", is t. set of successful trials thus far. In Phase 1, th. system doesn't get have any idea of what "optzn" is — (This is a Phase 2 "Breakthru")

I would expect that reasonable PSG discovery should ~~[illegible]~~ t. bits/symbol a lot; The idea is that using simple ngm contexts, we miss a lot as compared to ngms + contexts (≡ POS (parts of speech)) in CFG's. ∈ POS's enables to get a much larger SSZ for productions than does simple ngms."

Initially CFG's wouldn't help much, because long ngms would convey meaningful longish term context info, & early CFG's would not. CFG's would perhaps have to be augmented by so understanding of t. text. Another Q (Such as to be answered by t. Cover-King paper .... Only t. very best human predictor got ~ 1 bit/symbol. The second best predictor was much worse! How much worse?? ≅ 2 bits/symbol??

Domain

.20 : (.05) One ~~[illegible]~~ to try this in would be SM. To list a large no. of SM predn methods & devise a grammar for them. — But the Corpus of this Grammar would have ~~to be~~ wtd. by the success (pc) in past predn. ~~[illegible]~~ This would be a useful "study problem" in several ways! ① ~~[illegible]~~ SM predn is an area I'm a bit familiar w. ② Predn. of all kinds is a very basic problem in TM ③ I might get some ~~[illegible]~~ predn. methods good enuf to use ④ In ~~[illegible]~~ predn, t. wts of various cands. is always clearly defined, so t. corpus consists of suitably wtd. cases. ⑤ I might be able to get a TSQ by finding very old SM data (where Predn was easy), & gradually give more recent data. — One trouble is that t. data (old data in particular) may be very buggy!

.30 So: Some possl. paths to TM!

.31 1) Work on ~~[illegible]~~ Grammar of (SM Induction methods) a/o Optzn methods a/o INV soln. methods, perhaps use ~ BZZ for search (?) : unclear how to ~~do~~ use ~BZZ here & since, the Grammar would be a CFG or CSG or HMM perhaps: Not a "Bernoulli Grammar"

.34 which is all that BZZ can predict. (Tho, try to extend BZZ to CFG's : look at other ways BZZ has been extended! E.g. Raster Graphics.

2) Try to devise TSQ for Alg, say or just to Lrn Defns in "Maple"; Do "Phase 1" w. ~BZZ a/o GA

3) "Phase 1" does use a particular method of induction — so it should be included in (1) (.31-.34).

4) Since ② uses "Phase 1", It can also ~~be~~ done ~~[illegible]~~ by GA

5) Try doing ~~[illegible]~~ well known problems done by GA by ~ BZZ

6) Try GA for Phase 1 (∈ in 2).

Spec 14.00

13.00

1.12.04

TM4 ← use "4TM"; this page is 4TM ~~465~~ ||

418 was 3TM
419 was 3TM3
463 was TM4
464 will be 4TM

I could alternately keep old address 463. so TM4

00: 462.40 it is put into t. sorted list (w. a special tag so it ~~could~~ can be removed when we generate next cand.)
The next symbol has context lengths of 0, & 1: we make best choice and modify sorted list (in a reversible way) — etc. .... Needs Work!

SN Re: [redacted] "Symmetry" Problem: In || version of Lsrch! We mite be able to properly Cross-Reference Cands: So when a "symmetrizable" expressn occurs, ~~this always ex~~ if it is recognized, it is then expressed in standard (say Lexical) form & t. standard form gets extra cc.

While this trick may be possl. for complete derns of Cands, it will not work so well while
08 we only have partial derns. But we don't execute a cand until it is completely derived! (?).
This) such is true for certain Lsrch models — not for all; Perhaps restrict ourselves to
0 Lsrch in which [redacted] execution of a trial occurs only after t. entire cand. has been generated!

In || version of Lsrch, one has to complete dern of cand before working on it — otherwise One can't find where it is, to work ~~on~~ executing it! so .08 is not a legit objection !

It is perhaps possl. to recognize several ~~of dif~~ forms of equivalence/symmetry in Algebraic a/o Logical expressns. To recognize and ~~get~~ them all (or some) of them into a standard (perhaps Lexical) form, may not be so easy!

Alternatively to Monte Carlo, one could deterministically generate all cands in ≈ ~~...~~ order of pc's by an exhaustive "width first" search, using a CB bound (generate all strings w. pc < CB₀! Test any that are complete, for proper length of time. Then CB₀ → CB₀ × 1.1, say & repeat until done. ~ to T ← 2T Lsrch, but we use "< 2" & we don't waste
20 any time by repeating trials.

It mite be possl. to do t. T ← 2T Lsrch. Each round ~~independ~~ of T ← 2T discards info from previous round(s). There is a standard form for each symmetry. All variants of that symmetry are ~~...~~ discarded & t. standard form is given extra wt.

Equivalent symmetries tend to be in same "Round". ●

5 For $a(b+c) = ab+ac$ | $\overset{5 \text{ symb.}}{\times a + bc} = \overset{7 \text{ symb}}{+ \times a b \times a c}$

a(b+c) has much higher pc., so it would occur in earlier round, but would get extra wt. ab+bc would be discarded. So in ~~...~~ T←2T Lsrch, we want to make t. "standard form", a form of Max PC (if ~~...~~ equivt. forms do not have same pc)

Factorization (in .252) can be difficult if a, b, c are complicated expressns,
30 Also, there can be ~~a~~ several ways to do this "factorizn" e.g.

a(b+c) + cd = [redacted] ab+ac+cd = ab + c(a+d). Since these are both equivt., they both add ~~...~~ pc to t. same Cand.

In looking at ab + ac; it may be diff'lt to notice that t. 2 "a" expressns are t. same. — In general, t. ~~Q~~ of identity of 2 arb. expressns can be quite diff'lt ~~if~~ t. expressns are Long. . .

1.12.04
~~TM4~~
4TM

"Quartz" ~ 57 sec ahead of "Realistic" 1.25.04 20:00 hrs so 10 days 12"
"Quartz" ~ 45 sec ahead of Realistic ≡ 1.12.04 | 22:00 hrs or ± 1.2"/day
(10pm)

BookMark (≈ Rev): .15 (458.00 - .11; 462.12 ff.)

00: 461.40: Hvr., DS looks more promising. I will have to use ~~exact~~ $\frac{PM_1}{PM_2}$ (or just PM_2) as wt. of suffix. If it works, I then have to find good, fast approxns. Using tables can help. Also for values too large to be in tables, find well fitting fast, functions by "heuristic srch".

At present it mite be well to try ≈ BZZ on some GA problems to see if the approach is promising. If it is, I may want to go back & improve "≈BZZ".

A major problem seems to be "symmetry" detection/evaln. Perhaps for depths of ≤ 5, say this will not be hard. — But that paper on " GA's used fixed length chromosomes of length 50 to 250! Perhaps they were mainly redundant (tho Ray said they used

○ — MDL to resolve (goodnesses / priorities) of ~~to~~ difrnt models: they would ↓ [illegible] effective chromosome size.

12:458.11 SN For BZZ on (or GA corpus!) QATM corpus! Each past soln. function is a separate word — they need not be same length. We do all rotations of all of the words & put them in [illegible] lex order.
It mite be best to use "BTREE" management of LEX files, since it's ~~easy~~ to insert / delete files.

15 SO: Summarize present expected path: 458.00 ~~[illegible]~~ - .11 → 462.12.
A main idea is that by working on GA problems, I can easily find problems & compare ~~[illegible]~~ the efficacy of my own methods with those of others. If "BZZ" looks v.G., I will try it on QATM & try to get to Phase 2 (OZ & IND probs). If GA is better than "≈BZZ" then I should use GA on QATM & try to get GA to work on

20 Creation of Phase 2. IN Phase 2, we have created a function that looks at a problem & suggests ~~a~~ a PSM. After we have this function, we can use it to improve itself. — we ~~no~~ would no longer ~~need~~ GA!

So the slowness of GA is only a "constant cc" factor in the history of development of TM.

Essentially, I am (initially) replacing GA by Lsrch, using BZZ to guide Search —

28 As w. most Lsrch, summation of identical trials is a problem, & the "Symmetry Problem" is a serious case of it.

30 Some Serious problems in .15 ff! ① The symmetry problem of .28 is serious: It may be possl. to use BZZ to keep track of Symmetries. A non-intuitive effect that for ~~[illegible]~~ the mult. of n difrnt nos. there is a # of pc of ~~[illegible]~~ $\frac{1, 2, 12, 120, 1680}{1, 2, 3, 4, 5}$, 2^{n-1} ~~[illegible]~~ (n: 1 2 3 4 5; 2 4 8 16; 1 4 24)

Could one get rid of this effect by using a suitable notation? Using usual $2^{n-1}n!$ ~~[illegible]~~ with parentheses & Polish, gives this wild effect(

2) How to generate the possible funcns is unclear. Using Mt Carlo is possl. But can one ~~better~~ do deterministic ~~tree~~ Srch? — Using Computed pc's to guide, switch branches?

SN See Knuth's books on methods of putting in Lex order: tho prob'ly there has been much progress since. Perhaps try ~~Google~~ SORTING problems. See Knuth on "BTrees".

→ One approx way: Say one has a corpus of n "solns", these are in "shifted" Lex order. ~~We make~~ For new cand we start w. ~~[illegible]~~ null context probs; As soon as a symbolic item

1.11.04
~~TM4~~ CIAP : .01

00: spec (459.40 460.90) ; This is beginning to look very promising! → .20

.21 SN CIAP: E. Drexler lists ~ 3 conditions under which Nano tech could result in a catastrophe (≈ "Gray Goo"). (I forget what they are but ≈ 1. Extreme ignorance incompetence of administrators 2. Real wild Malevolence by (Govts Admin Engrs) 3. Total technical incompetence)
Anyway; This argument applies to Countries' Admn. — it is an arg. that we would never have wars! — I.e. These conds often hold!)

Try to find that Drexler Refrence! I tried "Drexler gray goo" on Google: — try Safety of Nanotechnology, Drexler

1). "T. Sci community Metaphor." Wm A. Kornfeld, Carl Hewitt. M.I.T. AI Memo no. 641 Jan 1981 — I guess this is the Sci: Community as a kind of Intelligent Info. System.

.10 Drexler's "Engines of Creation" on the web: See chapter 1) for much stuff relevant to AI & CIAP

There is a lot of discussn in Drexler's on line "Engines of Creation" ← That is somewhat updated On AI.

Hvr, note that I wrote letter to Wolff on 1/24/00 : (corrected: (i.e. $x > e$) in 1.31.00 letter to W.
On 1/31/00 I noted that Eq(4) (w. $\frac{z^R}{\sqrt{R}}$) was valid only if $x > e$ & a fortiori for $x >> e$ ($x \equiv \frac{\epsilon}{\varphi} \equiv \frac{\text{"defined" pc of } \gamma}{\text{"undefined" " } \gamma}$!)
So look betw. those determining notes:
on TM Z141 89.99 29.27, I may have understood how the "e" factor in $z \to \frac{x}{e}$ for large x.

.0 : .00 ↗ It may be possl. to use exact formula for $\frac{P_{M2}}{P_{M1}}$ using summation over various $\frac{\epsilon}{\varphi}$ values of pc of each symbol
How much time it would take is unclear:
Alternatively, we mite do the older Z141 coding of t. corpus, trying to find good definitions;
finding largest $\frac{\epsilon}{\varphi} = x$ may be enuf to decide on which ngm to define. Use an approxt. method to decide if $\frac{P_{M2}}{P_{M1}}$ is > 1, for stopping: If it seems close, use t. exact formula for stop criterion.

2 Approaches to Induction with words : Oldest ① Define Words "DW". Used in old EMOL Z141 & Sol 64b.
② Recent: Define suffix (only): "DS" (≈ BZZ). DS looks more promising now. T. idea of Parsing is not so attractive! Tho eventually I may want to "DW" because definitions speed things up.
— Tho define things only w. large SZZ, when its quite clear what t. parsing is.

.0 Re: DW: I dare way to go thru corpus? corpus, defining finding /word of & many max $\frac{\epsilon}{\varphi}$, then defining it, then reparsing (using all current words). — It may not be nec'y to reparse after every new definition For termination criterion use $\frac{P_{M1}}{P_{M2}}$ — at first use approxt formula (which I can find various approxns w. various speeds & accuracy) then use exact formulas as end approaches. 20° 8pm 36

In DW, I may want to apply t. new definition for only part of the "apparent" words: Only a fraction $\epsilon - \varphi$, will actually be correct. By expressing all of t. newly discovered words I grossly "over parse" & eliminate substrings that would be useful in later word discovery! Maybe may parse (a randomly chosen) fraction $\epsilon - \varphi$ of them. (462.00)

1.10.04
TM4

18	4.776387E+8	3.684211
19	1.767263E+9	3.7
20	6.564121E+9	3.714285
21	2.446627E+10	3,727273 ← previous pgm.
.1	810.308284086724	
.2	10.9196293557682	
.3	3.09367726355712	
.4	1.7926755880658	
.5	1.35914091422952	
.6	1.16864039367441	
.7	1.07454411432124	
.8	1.0272203295235	
.9	1.00576716482809	
1	1	$\approx 1+\frac{1}{2}(x-1)^2$ for x close to 1
1.1	1.00441078988958	
1.2	1.01577807659597	
1.3	1.03209944642706	
1.4	1.05206820518637	
1.5	1.07479696586068	
1.6	1.0996628522104	
1.7	1.12621624317712	
1.8	1.15412468558553	
1.9	1.18313733617821	
2	1.21306131942527	

Power Basic does 15 signif figs in "low res"! ?

Single prec = 4 bytes 32 bits

$2^{32} = 2^{30} \times 8 = \underline{\underline{10^9}} \times 8$

double = $2^{64} = 2^{60} \times 16$
$= 10^{18} \times 16$ —

So 15 sig figs is double prec.

doing pgm w. y! instead of y#
& gave same 15 digit results. !

$\overset{\uparrow}{x}$ $\qquad x\, e^{\frac{1}{x}-1} \equiv z$ $\qquad 1.1 \times e^{.9-1}$ $\qquad (1+\epsilon)\; e^{\frac{1}{1+\epsilon}-1}$

Pgm is 4TM 459 This is interesting if true — i.e. there is a min at x=1 & $\Big|\ \frac{1}{1+\epsilon}-1 \approx$
$\qquad -\epsilon+\epsilon^2-\epsilon^3$

$z > 1$ if $x \neq 1$ Hvr, z gets very large for small x !

$1+\epsilon = e^{\epsilon-\frac{\epsilon^2}{2}}$
so $e^{+\frac{\epsilon^2}{2}}$

Anyway, this suggests useful codes for $X < 1$: i.e. for ngms w. unusually low freqs!! $\Big\|\ z \approx 1+\frac{(x-1)^2}{2}$ for small $(x-1)$.

Hvr, I suspect error in computer algebra! → TMZ141

I could try it with $\quad \xi \to s\alpha$

i.e. use my "exact" formula & see if it gets reasonable results.

$\alpha \to a \quad .1$
$\to b \quad .2$
$\to c \quad .3$
$\to \gamma \quad .4$

$\gamma \to (.1/.4 = .25)$ a .25
$(.2/.4 = .5)$ b .5
$(.3/.4 = .75)$ c .75

TMZ141
1.31.00: 3.01–.12 — This is where I got the idea that x must be > e for the approxns to be Valid.

Also ;

1.10.04
TM4

Poss. Soln. to Z141 "Paradox" .13 .15

:0: 457.40 . Help code t. corpus.

01 Perhaps a superior way to do this: Use ⟨t. method of Z141 that I wrote Wolff about⟩ to compute the savings in pc obtained by defining Sα & using it to code t. corpus — including all difrnt ways to code t. corpus. ~~[illegible]~~ — that ~~[illegible]~~ terminate in ~~[illegible]~~ Sα

This savings should be t. wt. of Sα.

Use .01 to get wt. for all other ~~[illegible]~~ possl "S" values.

To do this, we have to know ~~[illegible]~~ case count of Sα & case count of each symbol in Sα at that pt in t. coding.

There was something intuitively unreasonable about that analysis: Did I ever resolve it? There was an apparently spurious "factor of e".

0 — It was something like: ~~e~~ say symbols α,β,γ had freqs of $p_{\alpha,\beta,\gamma}$ ~~[illegible]~~. t. ngm αβγ occurs n times. from if αβγ were uncorrelated, it would occur m times, so it would seem that $\left(\frac{n}{m}\right)^n$ would be wt. given to ~~[illegible]~~ ngm αβγ if n is large.

13 I think I got $\left(\frac{n}{em}\right)^n$ instead! or $\frac{1}{e}\left(\frac{n}{m}\right)^n$?

(I didn't find any GOOD way to take advantage of $\frac{n}{m}<1$.)

More exactly: Say φ = pc of t. ngm of interest as t. product of pc's of its symbols (i.e. t. ngm is not defined)
$\epsilon = \frac{R}{M} = \frac{\text{no. times } \gamma \text{ occurs}}{\text{No. of symbols in corpus}}$.
If $x \equiv \frac{\epsilon}{\varphi}$ then t. ↑ in pc due to t. definition was $\approx \propto Z^R$ & $Z \approx \frac{\epsilon}{\varphi \cdot e}$ if $\frac{\epsilon}{\varphi} \gg 1$

15 Could this have been true because of H codes using t. ngm αβγ? Say αβγ were indep. ↑ so t. paradox is how factor of e arose

By summing over all parallel parses of t. corpus, one can, indeed get an apparent "compression" by using t. defn of αβγ.

0 If αβγ occurs n times in t. corpus, there are 2^n parallel parses of t. corpus (i.e. each of t. n αβγ's can be either parsed as "$\overset{\text{ngm.}}{\alpha\beta\gamma}$" or "α, β, γ." .... Each of these parses will have a difrnt pc ...

.20R 0/24/04 Note: If γ occurs R times, then we get most wt. by not doing t. substitution R times, but ≪ R times — because of ~~[illegible]~~ t. no. of codes: If we only do R substitutions, there is only one way. If we do k substitutions they have $\frac{R!}{k!(R-k)!}$ ways.

An apparently simpler way to show that Z^R w. $Z \approx \frac{\epsilon}{\varphi e}$ can't be right: Consider t. case $\frac{\epsilon}{\varphi}=1$. T pc's of t. corpi should be t. same, w. this factor indep of N (t. corpus size) (The "~~[illegible]~~" constant factor is t. ~~[illegible]~~ pc of defining γ).

1 Actually, if $\frac{\epsilon}{\varphi}=1$ then t. eq. used in Z141 was Z^R w ~~x~~ $= \frac{\epsilon}{\varphi} = 1$ then $Z = x$ exactly & we get $1^N=1$ as expected. Err, ~~[illegible]~~ since $Z = x\,e^{\frac{1}{x}-1}$, $Z > x$ if $x < 1$, $Z < x$ if $x > 1$

Turns out Z has a min at $x=1$ (!) — see page 470 for table — so for $x=1$, $Z=1$ but for $x>1$ or $x<1$, $Z>1$!

30 ~~[illegible]~~ This would ∴ give good codes for $\epsilon > \varphi$ or $\epsilon < \varphi$ (unusually hi or unusually low freqs)

The funny thing about $Z = xe^{\frac{1}{x}-1}$ is that for large $x = \frac{\epsilon}{\varphi}$, $Z \approx \frac{x}{e}$, which does seem unreasonable. — But check t. original eqns exactly, w.o. approxns (other than t. v.g. approxn for x!). Actually do sums. Also check t. conds for validity of Approxn! ← (there are 2 conds listed. $(\epsilon-\varphi)(L-1) \ll 1$, $(\epsilon-\varphi)(\epsilon-\frac{1}{L}) \ll 1$) I don't know if $\epsilon-\varphi<0$ is "legal"!

This method does seem to give a predn for t. next symbol using ngms, & one doesn't have to "reparse" at all —

If it really did work for ~~[illegible]~~ unusually low ngm freqs, this might ~~be~~ give a signift. ↑ in English (& most other) compressn!

460.00
461.00

1.8.04

TM4

TSQ's :.00:

Pancakes
C. Gau
~~[illegible]~~ Hoochi Shrimp.
Eggplant.
Rice.

458
6

.0 : I had been thinking of starting t. TSQ using Long list of meanings of terms in MAPLE

.01 like add, evaluate (numerically / literally), solve, ~~differentiation~~, integrate, ect.

An Alternative, Rich source of Problems & TSQ's ~~[illegible]~~ is t. body of work on Genetic Algs.

Every problem done by GA could be doable — & probly better — by Univl. d.f.

G.A. is particularly good because a single problem yields a TSQ. — of successively better "fitness funcs".

It would seem like an easy ~~[illegible]~~ way to start. I could compare ALP results w GA results — in speed of convergence & (perhaps different) — in cc per soln & perhaps ability to avoid local ~~[illegible]~~ extrema.

.10 The ~~[illegible]~~ TSQ of .00→.01 could also be done in ll w. t. GA-associated problems.

.11 I could try BZ2-like methods on both kinds of TSQ's. → 462.12

Hvr, I still need to finish my "Review" that tells just how I expect to proceed, ~~&~~ what t. big problems are & some suggested Solns. for them.

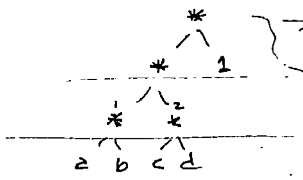
This "Review" would make a good talk/paper.

.20 : 442.40 What seems like a very serious problem (whether I use "BZ2" or not) is the symmetry/redundancy problem: Say I have several promising functions in t. "corpus" that I want to extrapolate.

Each ~~[illegible]~~ symmetric sub-function can have a Redn. Number (DN) that tells how ~~[illegible]~~ ~~[illegible]~~ ~~[illegible]~~ many equivalent ways there are to get that sub-function.

Hvr, its not so easy: A particular node can have several DN's — depending on how far back

.25 you want to trace its inputs. Consider



→ If we only use t. top 2 levels, t. redundancy is 2: * 1 & 1 *
→ .25R

.26 If we use top 3 levels there are many ways to get t. same sub function.

essentially, t. tryle product

~~* * * *~~ 1 , * * * * 1
* * * * 1
* 1 * * *
* 1 * * *

Consider product of 1, 2, 3!

~~[illegible]~~

$* \cdot (* \cdot \cdot)$
$* (* \cdot \cdot) \cdot$ } 2 × 3! = 12

.30

* * 1 & * 1 *

not (2 × 2!) (!).

So 442.26 may be wrong!

442.73 looks better.

n	1	2	3	4	5	
n!R(n)	1,	2,	12,	120,	1680	← observd.
n!	1	2	6	24	120	
? → R(n)	1	1	2	5	14.	
ratio ~~[illegible]~~ of n! R(n) ~~[illegible]~~		2	6	10	14	

So 2 levels : DN = 2

3 " : DN = 12 ← certainly seems right.

4 " : DN = 120

5 " DN = 1680

N = no of "mul" symbols.

N	1	2	3	4
R(N)	2	12	120	1680

$$F(n) = \frac{2n!}{n!} = (n+1)(n+2)(\cdots)(2n)$$

→ 4TM 17.15

1.6.04
TM4

457
5

18 rings ~ end of this page. } 500 pp β : 24:12 1.7.03
453 12 rings
443 2 rings
458 18 rings (458 has been written ~~the~~ 457)

The time has come

20 : 455.40: has occurd in t. past, & t. d. f. of symbols that have followd it. Then we have various lengths of ~~~~ that precede t. symbol to be predicted —— BZZ runs into this problem. Hm. BZZ tries to pick a size that has been "easi" in predicting t. desired symbol.

Lengths

In my own older work, I would use all methods of predn, & weight them. This may get rid of t. "zero frequ" problem! I'm not so ~~sure~~ sure now (in view of how BZZ works) just how to assign wts to different ngm predns.

.05 Hws, it gets rid of part of t. ~~~~ freq = 0 problem, but not all of it —— i.e. it still doesn't deal w. symbols that occur for t. first time in t. entire corpus. ......

Say we have a sequencial corpus: to predict next symbol: Using postfixes of ~~~~ length ∅ and increasing, we get prediction distributions for ~~each~~ postfix. For longest postfixes, we ~~~~ usually get only 1 predn, and for longer, we get "no previous occurrence". We could use wtd. mean, but what wts? We want an Algm. that usually gives v.g. values for total pc

0 of corpus. If we use t. "straight rule", we will get predictions of symbols (w. pc > 0) if those symbols have ever occurd in past of corpus, since there will exist at least one ~~~~ context (= postfix) (maybe null postfix) that gave a freq count of ≥ 1 for that symbol following that ~~~~ context. Hwr, for symbols occuring for t. first time in t. corpus, all ~~~~ contexts (including t. null ~~~~ context) will give pc=0. [very bad!] If ~~~~ n symbols have occurd thus far, & none of them is t. new symbol, then a ~~~~ reasonable guess for t. pc of ~~t.~~ new symbol is ~ $\frac{1}{n}$. This "$\frac{1}{n}$" is for t. null ~~~~ context. If a context has occurd k times & t. new symbol hasn't followd, perhaps "$\frac{1}{n}$" should be mult by $\frac{k}{n}$. → $\frac{k}{n^2}$. — But I'm not so sure of this

One view mite be that if a context occurs k times w.o. t. symbol following, that its pc should be ~ $\frac{1}{k}$ (usually which is > $\frac{1}{n}$).

10 On t. other hand, if t. context predictors are concentrating, then each context could not know t. symbol pc would be ~ $\frac{1}{n}$ — from its own experience, t. value of $\frac{1}{k}$ would be t. best it could do! One way to do this: ~~for~~ Each context has its own "prior corpus", & it ~~~~ (usually) has ~~a~~ one extra symbol in it, that has ~~never~~ occurd before.

24. [SN] Even if t. Algm could predict that "~~a~~ new symbol (not ever seen before) would follow" it would not be able to tell just what that symbol would be! — It would have to actually get t. right symbol for ~~~~ lossless Compressn. Algm. To do this ~~~~ say pc of "new symbol" was $\frac{1}{n}$; ~~~~ & there are ~~~~ ℓ total symbols that have not yet appeared so that never occurd before in t. entire corpus pc of any particular new symbol is $\frac{1}{n}\cdot\frac{1}{\ell}$.

Every time a new symbol occurs, it'll have that pc for the null context, that will be used to predict it. The $\frac{1}{n}$ & $\frac{1}{\ell}$ ~~seem~~ diff to relate "$\frac{1}{\ell}$" because all unknown symbols have = aprip.

0 "$\frac{1}{n}$" Mite ~~be~~ improved by modeling (fitting) t. rate at which new symbols have been introduced in t. past.

31 So: A method of ~~t~~ "ALP Coding": Consider all suffixes that have occured ~~once~~ before. (There will be a max limit on ~~of~~ length — all shorter suffixes have occured ≥ 1 time.) Each ~~~~ suffix will ~~~~ give a pc for t. next symbol, based on its relative empirical freq. in t. past, & considerations of .24 – .31 if t. predicted symbol has never occured in t. past for that suffix.

The wts. of t. various (suffix) predictors: Not easy to evaluate!

An Old fashioned way: $s\overset{\beta}{\alpha}$: we have suffix s, to ~~~~ get pc symbol α v.s. β will follow:

We look at ~~~~ case count of ngm sα: If it is > 1 then we can use it for pred. Otherwise no..

The wt. of t. pred s→α depends on how much compression we got by defining sα & using it to (459.00

20: 455.03 ; So we start w. a non-so-smart social animal. A mutation occurs that enables it to do induction better & also generate simple lang rules, & learn lang rules ~~[illegible]~~ ~~[illegible]~~ for langs that others may invent. There is some selection for ability to learn lang., because it enables the community to prosper & multiply. From this rudimentary (induction) lang.) enabler, further mutations enable expansion of lang generation & ~~lang~~ acquisition — These are assoc w. ~~[illegible]~~ induction skills in other ~~[illegible]~~ areas. — ~~So~~ This mutates further to enable ~~more a~~ better induction & more complex lang. — So language & induction skills mutate/cross together to advance ~~lang, & induction skills~~ both kinds of (related) skills.

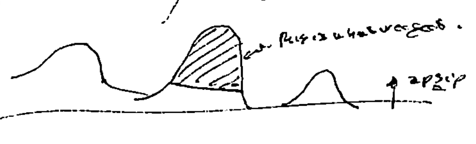
It may be that the development of ~~a~~ the ability to learn certain simple inductive tricks was the key thing that started it off.

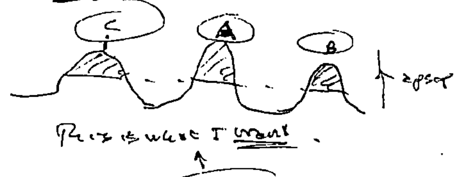
The process isn't yet very clear in my mind, but the basic idea is that induction skill & language skill, grow, develop together — partly by mut/cross & selection for good lang & good induction — & partly by social, & individual development of induction & lang. — These ~~[illegible]~~ mut/cross developments may have been rather fast because there could be much selection for them in a Social Group. e.g. Men who could communicate well could become leaders & have more offspring.

On Sumacs: A possibly Good form: To store several "Central pts" that good representatives "Cluster About". I had been thinking of using the "Best 100 codes thus far" as a representation of Sumac — & today this [illegible]: (tho w. Backtracking, we [illegible] much of work.)

A better way would be to select more carefully, points that cover the space better. Perhaps we want (1) A "best bet" "central" model (2) Other models that are "distant" from the central model, but having a apsp >> that which we would expect; i.e. their $\frac{aprip}{apsip}$ is unusually hi, Aprip is ~~[illegible]~~ info distance from central model, "Apsip" is ~~[illegible]~~ empirical pc evaln.

Well, actually, what occurs ~~of my~~ storing "100 best cands thus far" These are cands w. hyest ~~[illegible]~~ apsips. that have been found. ~~[illegible]~~ We'd like to keep "100 best" but the 100 best are normally ~~[illegible]~~ obtained by a ~~[illegible]~~ search that ~~[illegible]~~ moved out in "Info closeness space" from the cand of hyest apsip.





This is what I want.

I may even want to store the central pts (A, B, C...), then, when new problem occurs, start from A, B, C & search out from "them". (not just from "A" (the "Best cand").

Lang Lrng by infants: 00

00: ⓔⓡ ~~Lang. Acquisition~~ by Human children: ~~Human~~ Infant brains do have "special Mechanisms" to enable lrng lang. — But this is not so remarkable! The mechanisms for ~~lrng~~ general

lrng existed in human brains. Then / human Language was developed so as to take

03 advantage of those lrng. Mechanisms. → (456.00) → →

Chimps, monkeys use ~~[illegible]~~ & are able to interpret Body Langs ~~[illegible]~~ and facial expressions. Monkeys have a great variety of crys, but seem to have not developed a very complex lang. .... But this last should be checked by seeing what ~~[illegible]~~ communication skills & ~~[illegible]~~ assoc. inductive skills, monkeys have.

— Also Dolphins /Whales?: Some Birds can imitate human lang. Can they learn to understand to any extent? Crows seem very smart — But don't do imitations. What about Blue Birds? — They

10 can imitate ("Mocking Bird") but are they smart?

ABcdefghijkl
mnopqrst

In the review the d.IND → s-IND is a most impt., interesting problem. 431.14 is an early relevant remark. ~~[illegible]~~ pp 446.23.40, 448.00ff are relevant pints (up to now) in the Review

on d.IND → s.ind. 431.14
[Review on [illegible] 446.23-40 448.00 ff

20 — T. Discussn of 448. ~~[illegible]~~ .25 .29 is not so clear! There are 2 ways to use ngmcts for induction — well, perhaps ~~[illegible]~~ mainly 1 way: To use the Ngmst N for induction: Whenever the suffix of the corpus matches a ngm ~~[illegible]~~ in N in all ~~[illegible]~~ but the last symbol, Then we predict that last symbol for the corpus. Different ngmsts can give different predns! The pc assoc w. each predn. will be ~~[illegible]~~ the pc of the assoc. ngmst.

[Aside: trouble w. this: If a long / randomish prefix of corpus ~~[illegible]~~ ~~matches~~ Matches a string of some ngms (for all but last symbol), we would expect good predn. (i.e. hy pc): How., the an/ngmst containing long randomish ngms will probly have a small aprpd ⇐? On the other hand, a simple long ~~random~~ seq as a single ngm, is a purely A. H. def, so it is unlikely, even if it was. — ~~[illegible]~~ If a long Ngmst occurs twice, it has a pc >> than

30 that of 2 random occurences of ~~the~~ ngms of that length.] } so this looks like not serious difficy.

Say we define an ngmst. — it has a certain apriy assoc. w. coding cost — What is the ~~[illegible]~~ "case count" of that ngmst wrt a specific corpus? How is case count used in (≅ Lap's rule) predn.?

→ The ngmst defn is maybe like a "precorpus" having 1 occurence of each ngm. in the ngmst. This would enable "Lap's rule" by using the "straight rule".

But consider the ngm 111, how many times does it occur in the corpus 011110?

3 times? 1 time?

~~It may be that~~ The way we use the "case count" for predn., depends on how "case count" is defined! ⓒ Also, probly on the model of how the data was generated! — Well, if we are trying to use the string α to predict the following digits we may ~~[illegible]~~ want to look at every time α → (457.00)

00:453.40: T. way this is done: — ~~there~~ 2 ways: T. corpus for guessing new (functional pgms) is

.01 ① T. ~~set~~ wtd set of pgms (wtd by success) of t. present SUMac

② T set of sets of successful pgms for t. seq. of corpa of t. past & their successive successful SUMacs

② is like ~~[illegible]~~ t. "PD₂" of "Phase 1" of TM.

① (:.01) Seems like a new idea. In either case, one can get t. new trials from t. corpus (by

.07 either mut/cross of ~~[illegible]~~ pgms in t. Corpus) or ~~by~~ BZ2 extrapolns of pgms.

08 SN In BZ2 (& other ~~[illegible]~~ methods of Ngm (Ngmst?) induction): To create new (cand trial): ~~[illegible]~~ t. New (cands trials) are functional, so one can Build them up Backwards by looking

0 — at t. Backward sequences of previously successful pgms. One diffy is that usually one knows t. arguments of t. desired pgm — & if one generates t. function "Backwards", T. arguments are inserted Last! Otherwise, w. "forward" Generation of (functional pgms) t args are inserted at t. beginning w.t. ~~[illegible]~~ "set of legal symbols"

A Better way to approach this: Look at functions that I want to feel are "related"; find a way to "match them up" — possibly using Lexical Sorting in various ways. Also, the "partial Matches" used in Genome Analysis mite be useful: ~~[illegible]~~ Wolff's ~~techniques~~ in this area may be v.g. (but I find his writing Very hard to understand.) → He May have improved, hvr. Also, There is a lot ~~that has been~~ written on this (string) Matching stuff; perhaps Li Vitanyi's stuff (DNA anlysis. ~~Genetics~~)?

10

21 SN

Away back, I asked Q of whether an Ngmst was t. most general kind of d-induction!

Answer: IT IS. For each possl. predicted symbol, in d-induction, there is an (ngmst) of contexts that will reliably predict it as being (Token i). For predn, each Ngmst must have a wt, so, for small set, one can tell which is more likely. Hvr. each ngm in an ngmst has same wt.

For s-induction, each Ngm can have its own wt. So we can have an (infinite) set of wtd, (soft) ngmsts that can (soft) s-predict. A s-ngmst is typically described/defined by a machine or algm.

A d-ngmst mite be dscb as t. range of a machine or algm. — So we can easily xfm this kind of defn of an ngmst from d-ngmst → to s-ngms.

A way to think about it: An ngm ~~[illegible]~~ can be defined by a pgm & an input to t. pgm. This defines t. pc of t. ngm as pc of pgm x pc of input + punctuation cost.

And (d-Ngmst) (w.o. input) is defined by pgm alone: it is Range of t. pgm: t. set of all possl. outputs.

A pgm w.o. input also defines a P.D. on t. ngmst defined by t. pgm (for a d ngmst) — This is t. P.D. defined by t. pgm ~~[illegible]~~ plus t. punctuation plus t. pc of input needed for each output ngms.

Try to think of examples of ~~a useful~~ an inductively useful (d-ngms) Being xfmd to → an inductively useful (s-ngmst)

In early work on ANL I did try some partial solns involving ~~[illegible]~~ s-induction (~100% soon) but ~50% soon — which was better than ~~[illegible]~~ (previous try) .... so a useful step + look at the development!

How is it related to d-induction → s-induction? Perhaps we should start NTM w. s-induction!

Grms 6 18 58 5 ℓℓ
5 12 453 10 ℓℓ
7 443

12 Bomps
7→12 23:00 1-1-03
D21.03 (for all 443)

00 : 452.40 : The there may be conditions (small ssz + perhaps) in which 451.33 – .40 (t. direct Mt. Carlo Method) would be best.

What vare (or whatever!) do we get if we spend cc = CJS on the Mt. Carlo Method?

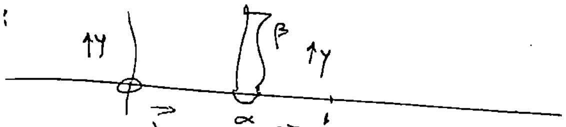
03 For n data pts: $\sigma^2 = \frac{\sum Y_i^2}{N} - \frac{(\sum Y_i)^2}{N^2} \cdot = ~~\frac{1}{N^2}\left(N\sum Y_i^2 - (\sum Y_i)^2\right)~~$

$$\frac{\sum_{i=1}^{N+1} Y_i}{N+1} = \left(\left(\frac{\sum^N Y_i^2}{N}\right)\cdot N + Y_{N+1}^2\right)\Big/ N+1 = \left(\frac{\sum^N Y_i^2}{N}\right)\cdot\frac{N}{N+1} + \frac{Y_{N+1}^2}{N+1}$$

0 $$\frac{\left(\sum^{N+1} Y_i\right)}{N+1} = \frac{\left(\sum^N Y_i\right)}{N}\cdot\frac{N}{N+1} + \frac{Y_i}{N+1}$$

$$\left(\sum^{N+1}(Y_i)\right) = 2\left(\sum^N Y_i\right)\cdot Y_{N+1} + Y_{N+1}^2 .$$

Consider:



Say we do a large no., n, of trials:

$\frac{(\alpha N)\cdot\beta^2}{N} = \alpha\beta^2$; $\frac{\alpha N\beta}{N} = \alpha\beta =$ mean. So $\sigma^2 = \alpha\beta^2 - \alpha^2\beta^2 = (\alpha-\alpha^2)\beta^2$.

$\sigma = \sqrt{\alpha-\alpha^2}\cdot\beta$: if $\alpha \ll 1$ this is $\sim \sqrt{\alpha}\cdot\beta$.

We are interested in the mean, $\alpha\beta$ and $\frac{\sigma}{\mu} \approx \frac{\sqrt{\alpha}\,\beta}{\alpha\,\beta} = \frac{1}{\sqrt{\alpha}}$ which is ~~[illegible]~~ always > 1,

.18 if α is small, $\frac{1}{\sqrt{\alpha}}$ can be quite large. — But this seems to be indep of N!

.03 – .18 is all wrong! Rather say: In N trials, what is probly of just m hits in the α region?

20 T. probly of each hit is α. I think its $\alpha^m(1-\alpha)^{n-m}$. $\frac{n!}{(n-m)!\,m!}$ = probly of m hits in n trials.

$$\frac{n!}{m!\,(n-m)!} \approx \frac{n^n e^{n-m} e^{m}}{e^n m^m (n-m)^{n-m}}\sqrt{\frac{2\pi\, n}{2\pi\cdot 2\pi\cdot m(n-m)}}$$

$$\approx \left(\frac{n}{m}\right)^m\left(\frac{n}{n-m}\right)^{n-m}\cdot\sqrt{\frac{n}{2\pi\, m(n-m)}} = \frac{\sqrt{n}}{\sqrt{2\pi}}\cdot\left(\frac{n}{m}\right)^{m-\frac{1}{2}}\left(\frac{n}{n-m}\right)^{n-m-\frac{1}{2}}$$

26 : 452.29 : A nice way to do this: Say we have a large corpus: Get a bunch of discrete / models & parts of it.

Use ssz = 1 or 2 (small): The continuous / Df's are broad & easy to find for all discrete models. So get each (r of f) peak & its vare. Then ↑ ssz by 1 or 2 & refine peak & vare. Easy to do because we are not far from it to start. Use that "locally linear" optimum quadratic form

30 to find successive approxns. It may be good enuf to get just 1 new approxn. for each increase in SSZ. Note that we always retain data pts from earlier trials: — (tho we may discard some of them if they get too far from t. present apparent peak.).

32

33 Each soln for $\boxed{SSZ = N}$ gives a v.g. apriori for both t. $\boxed{\text{Discrete}}$ & $\boxed{\text{Continuous}}$ dfs ~~for~~ $\boxed{SSZ = N+1}$

$\boxed{\text{This looks like SUMAC!}}$ Both discrete & Contin d.f.'s can save enormous amt. of time in Search!

Hvr. as a SUMAC, it isn't very instructive as to how to do it for other corpi types. ← (Be not so dismal'd!)

Actually, it is ~ to SUMACs in a useful way: We have t. Sumac which is a wtd Σ of many pcms — This is given really & also m .26 – .33. For trials of t. (slightly) augmented corpus, do trials that are "close" to t. set of discrete (& continuous) functions in t. SUMmarizing machine (≡ Sumac).

This is not SUMAC.
4TM 13.35
459.00