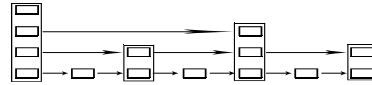


Skip Lists

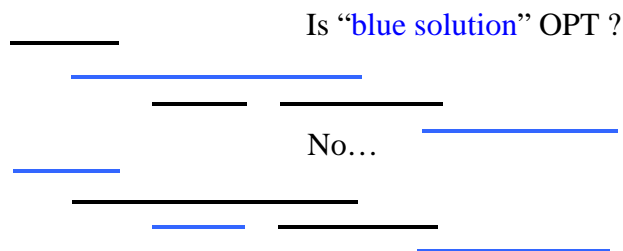


- Analysis:
 - » $2 \log n$ lists.
 - Probability that a given node is in list $(2 \log n)$ is $1/n^2$
 - Probability that any node is left in list $(2 \log n)$ is at most $1/n$
 - Expected number of nodes in list number $(2 \log n)$, i.e. last list ?
 - » Expected total number of nodes is $n+n/2+n/4+\dots = O(n)$.
 - » Search: find range in last list, examine this range next one down, etc.
 - » Same intuition as [binary search](#).
 - » Each range is expected to be constant length:
 $O(1)$ work per range, $O(\log n)$ total.
- Works best if there are **no deletions**.
- Does not work if deletions are **malicious**.

128

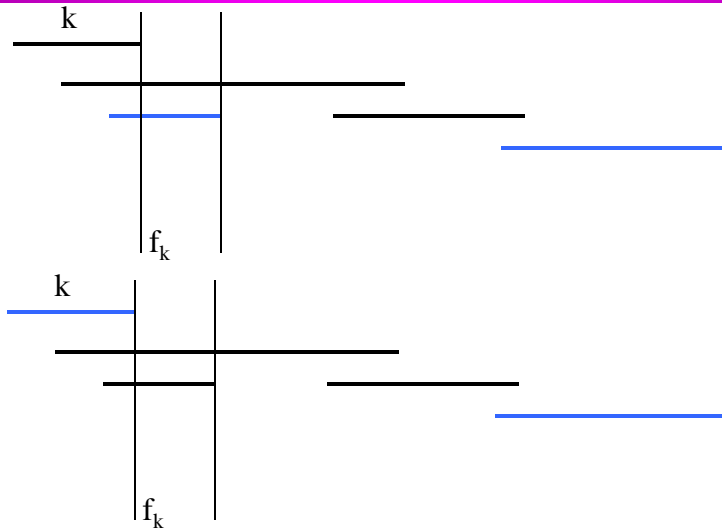
Greedy Algorithms

- Problem: set S of n activities
 s_i, f_i start and end of activity i .
- i compatible with j if intervals **do not intersect**.
- Goal: find max # of compatible activities.



135

Substitution claim



136

Substructure

- Let k have smallest f_k and let A be OPT solution.

» **Case 1: k in A .** Claim: $A-k$ is OPT for $S' = \{i | s_i > f_k\}$ ← Jobs that start after k ends

Assume not. Let B be OPT for S' , $|B| > |A| - 1$
But then add k to B and we get better than A

» **Case 2: k not in A ,** finish time of 1st job in A is AFTER f_k
⇨ replace it with k !

» Thus:

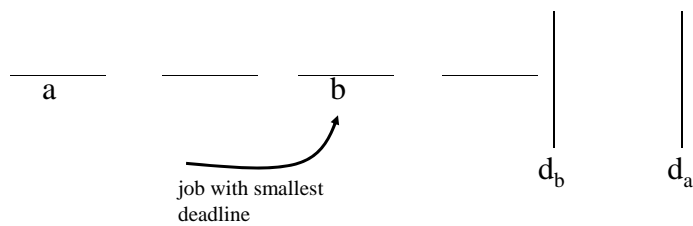
- There exists OPT A with k being first job
- $A-k$ is OPT for "remaining jobs", i.e. $S' = \{i | s_i > f_k\}$

» Thus we compute k , commit to it, compute S' , and repeat

137

Another greedy algorithm

- Task defined by (duration, deadline), eg. HW.
Goal: find a schedule if one exists.
- Assume that there exists a schedule
Claim: then there **exists** a schedule with
first job = job with smallest deadline.



138

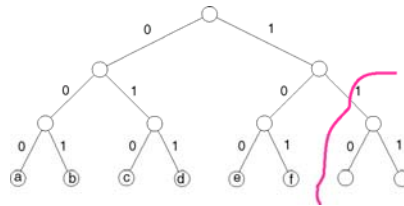
Summary

- Take **locally best** choice and **commit** to it.
- Main issue: proof that we can commit **without losing our chance** to get an optimum solution.

139

Huffman encoding

- Idea: represent **often encountered** letters by **shorter codes**.
- Prefix code: a code for **x** is not a prefix for any code-word for **y**.

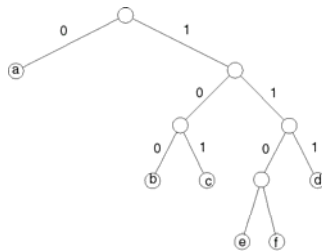


- In this example: $c=010, e=100$

140

Huffman encoding

- Assume that **a** is a very common symbol.



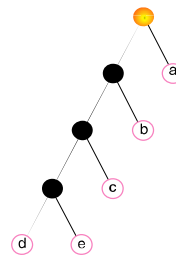
- Now: $a=0$
 $b=100$
 $e=1100$

141

Huffman encoding

- Assume we know **symbol frequencies**:

50 40 5 3 2
a b c d e

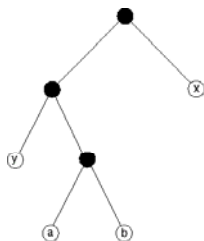


- $50 \cdot 1 + 40 \cdot 2 + 5 \cdot 3 + 3 \cdot 4 + 2 \cdot 4 = 165$, **1.65b/symbol instead of 3!**

142

Generating optimum encoding

- Claim: Let **x&y** be lowest freq. characters.
Then there exists code where **x&y** differ **only in 1 bit**.
- Consider **a** and **b** being the "deepest" symbols in the tree sharing parent.
(Is it possible that deepest symbol does not have a sibling?)



$$\begin{aligned} \text{WLOG } & \left. \begin{aligned} f(a) \leq f(b), f(x) \leq f(y) \end{aligned} \right\} \\ \text{also: } & \left. \begin{aligned} f(x) \leq f(a), f(y) \leq f(b) \end{aligned} \right\} \end{aligned}$$

\Rightarrow exchanging $x \leftrightarrow a, y \leftrightarrow b$ can only help!

- So does this mean that we do not need any other codes?
» [Hint: consider a sequence $abcabcabc\dots$]

143