Greedy Algorithms

Internal Scheduling — O(n)

Input: list of intervals $S = \{I_1, ..., I_n\}$ where each $I_i = [s_i, f_i)$

Goal: find a subset $S = S'$ of non-intersecting intervals of largest size

First attemp: dynamic programming

Subproblems: the optimal solution for intervals storing after f_i and ending before s_j

Recurrence: $DP(i,j) = maxDP(i,k) + DP(k,j) + 1 \quad [s_k, f_k) \leq [f_i, s_j)$

0 if no such k

Runtime: $O(n^2)$ subproblems $\times O(n)$ time/sub = $O(n^3)$

Second attemp: improved dynamic programming

Guess whether the last interval is used or not.

Subproblems: for $i = 1, ..., n$, optimal solution for $\{I_1, ..., I_i\}$

Recurrence: $DP(i,j) = max(DP(i-1), 1 + DP(k_i))$ where k_i is the last job to finish before s_i

Runtime: $O(n)$ subproblems $\times O(1)$ time/sub = $O(n)$ (actually $O(nlogn)$ because of sorting)

Greedy algorithm

Maybe we don't need to try all possible activities?

Turn out we can always take the first activity to finish!

Thm. For any list of intervals S there is an optimal solution that includes the activity that finishes first.

Proof. Take any optimal solution. If already include that activity, we're done. Otherwise, add that activity. It can collide with at most one other activitiy, so remove that other one. The number of activities remains the same.

After sorting the activities by finish time, the greedy algorithm takes

$O(n)$ subproblems $\times O(1)$ time/sub = $O(n)$

cur_fin = $-\infty$

for i = 1 to n:

    if s_i > cur_fin:

        print i

        cur_fin = f_i

Huffman Codes

Typically, we use 8 bits per symbol (ASCII).

If we only use ≤ 32 symbols, a, b, ..., , ?, !, then 5 bits per symbol are enough.

Some symbols are used much often than others (e, a, t, i v.s. w, x). We can encode them using fewer bits. To make sure there's no ambiguity:

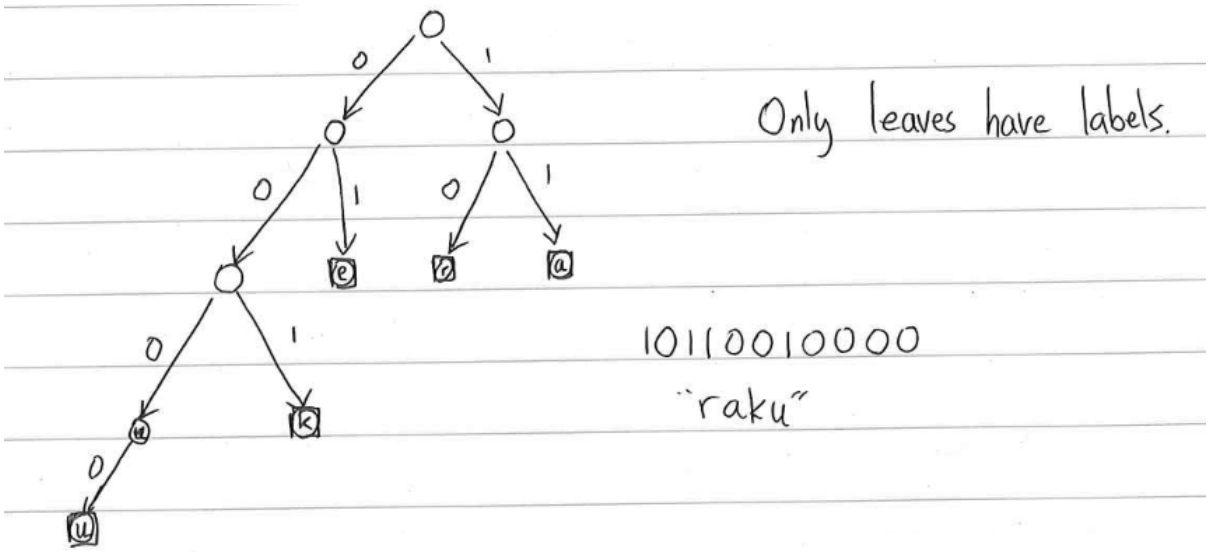
Def. A prefix code for a set S of symbol is a function $C : S \to \{0,1\}^* \quad s.t. \quad \forall x, y \in S, C(x) \text{ is not a prefix of } C(y)$.

e.g. a → 11; e → 01; k → 001; r → 01; u→ 0000

A file with 1 billion symbols with these frequencies: $f_a = 0.32, f_e = 0.25, f_k = 0.2, f_r = 0.18, f_u = 0.05$.

The size of the encoded text is $2f_a + 2f_e + 3f_k + 2f_r + 4f_u = 2.3$ bits/symbol, 2.3 billion bits.

It's convenient to model a prefix code as a tree.



average bits per letter $ABL(T) = \Sigma_{x \in S} f_x \cdot depth_T(x)$

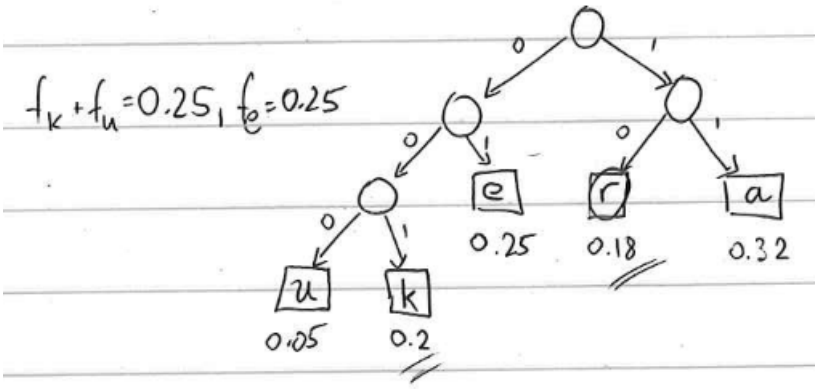
Claim. The tree corresponding to an optimal prefix code is full. [full = every node that is not a leaf has two children]

Proof. Assume by contraction that T is a binary tree of an optimal prefix code and is not full. Then, there is a node u with only one child. Remove u, and replace it by its child. Since this decreases some depths, and leaves the others unaffected, the ABL gets smaller, in contradiction.

Greedy attempt #1 [Shannon-Fano 1949]

Create tree from root to leaves, splitting S into two sets S_1 and S_2, with almost equal frequencies, and recursively building trees for S_1 and S_2.

$f_a = 0.32, f_e = 0.25, f_k = 0.2, f_r = 0.18, f_u = 0.05$.



2.25 bits/symbol; depth(k) = 3 > depth(r) = 2, swapping can improve

Algorithm is not optimal! Implementation is hard.

Huffman encoding

Observation 1: Lowest frequency symbols should be at the lowest level.

Observation 2: The lowest level always contains at least two leaves.

Observation 3: The order in which items appear in a level does not matter.

Claim 1: There is an optimal prefix code with tree T^* where the two lowest frequency symbols are in sibling leaves.

Idea for proof: first, using Observation 2, there are at least two sibling leaves in the bottom layer; next, by Observation 1, the two lowest frequency symbols must be in the bottom layer; finally, by Observation 3, we can assume they are in sibling leaves.

Huffman's greedy approach [1951]

Create tree from leaves to root. Make two leaves for the two lowest frequency symbols y and z. Recursively build tree for remaining symbols plus "meta-letter" "yz" of combined frequency.

$f_a = 0.32, f_e = 0.25, f_k = 0.2, f_r = 0.18, f_u = 0.05$

$f_a = 0.32, f_e = 0.25, f_k = 0.2, f_{ru} = 0.23$

$f_a = 0.32, f_e = 0.25, f_{kru} = 0.43$

$f_{ae} = 0.57, f_{kru} = 0.43$

HUFFMAN(S):

1. if |S| = 2:
2. return tree with root and two leaves
3. Let y and z be lowest freq symbols in S.
4. S' = S
5. Remove y and z from S'

6. Insert new symbol w in S' with $f_w = f_y + f_z$
7. T' = HUFFMAN(S')
8. T = add two children y and z to leaf w in T'
9. Return T

Time complexity:

Naively, if we don't use any data structure, Step 3 takes $\Theta(n)$ time, leading to run time of $T(n) = T(n-1) + \Theta(n) \rightarrow T(n) = \Theta(n^2)$

Better to use heap / priority queue, which takes only $O(logn)$ to extract minimum and add a new element. This gives $T(n) = T(n-1) + \Theta(logn) = \Theta(nlogn)$.

Correctness:

Claim 2: $ABL(T') = ABL(T) - f_w$

Proof: The difference between T and T' is that instead of the leaf w of T', T has two leaves y and z. Their combined frequency is f_w, but they are deeper by 1.

Theorem: The Huffman code achieves the minimum ABL of any prefix code.

Proof: By induction on number of symbols $n = |S|$.

Base case: n = 2, ABL = 1 and obviously optimal.

Induction step: assume holds for n-1.

By hypothesis, Huffman tree T' for S' of size n-1 with w instead of y and z is optimal.

Suppose by contradiction that Huffman tree T for S is not optimal.

So there is a tree Z such that $ABL(Z) < ABL(T)$.

By claim 1, we can assume y and z are in sibling leaves in Z.

Let Z' and Z with y and z merged into one parent node (just like we derived T' from T).

By Claim 2, $ABL(Z') = ABL(Z) - f_w$

$$ABL(T') = ABL(T) - f_w$$

Then, $ABL(Z') < ABL(T')$ in contradiction.