WEEK 4, LECTURE 7 ON 11 SEPTEMBER 2021 CS1.301.M21 ALGORITHM ANALYSIS AND DESIGN

BASIC GREEDY DESIGN

- Greedy choice property? What is the property we need to observe to make progress towards a solution.
- Once the first step is taken, can we recursively restate the rest of the problem so that an induction follows.

ACTIVITY SELECTION

Problem

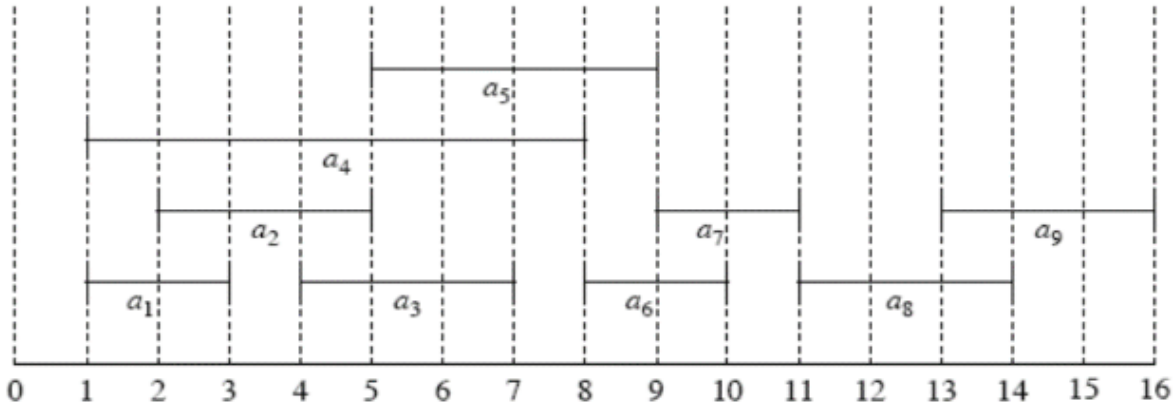
There is a set of activities $S = \{a_1, a_2, \ldots, a_n\}$ and each activity a_i needs a resource during the period $[s_i, f_i)$ which is a half open interval that is specified by a start time and a finish time. Select the maximal number of activities from S that do not intersect times.

Eg.

i	1	2	3	4	5	6	7	8	9
s_i	1	2	4	1	5	8	9	11	13
f_i	3	5	7	8	9	10	11	14	16

Maximum-size mutually compatible set: $\{a_1, a_3, a_6, a_8\}$.

Not unique: also $\{a_2, a_5, a_7, a_9\}$.



To design an optimum solution:

- Which property of an activity is guaranteed to show that it is to be a part of the answer.
- Can we apply this property to the remaining activities and create subproblems such that we can eventually arrive at a correct answer.

Solution

The activity that has the lowest finish time (say a_0) always belongs to a correct answer since it always takes up the least time.

Now consider the rest of the activities that are not intersecting with a_0 and then again pick the activity with the lowest finish time.

```
// assuming s and f are sorted in order of ascending finish time
ActivitySelector(s,f,n):
A = a[1]
i = 1
for m=2 to n:
    if(s[m]>=f[i]):
        add a[m] to A
        i = m
return A
```

HUFFMAN CODES

Problem

Say the alphabet is $\{A, B, C, D\}$ and a string T has 130 million characters according to these frequencies:

Symbol	Frequency
A	70 million
B	3 million
C	20 million
D	37 million

Using a 2 bit encoding (since there are 4 alphabets) 00 for A, 01 for B, 10 for C and 11 for D. The total is around 260 megabits.

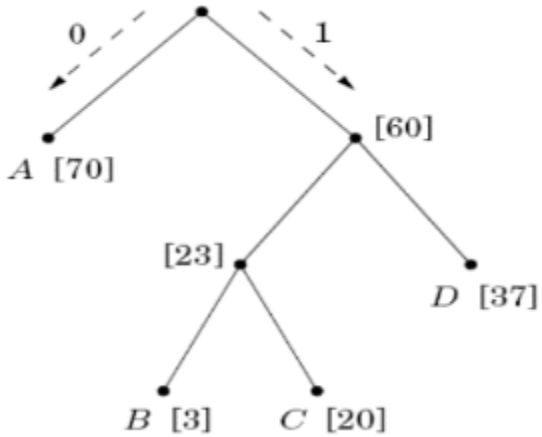
Can we encode this information using less bits?

Solution

Using **Variable length encoding** means that no codeword can be a prefix of another codeword i.e. one word cannot start with how another ends. This leads to the ability to represent an encoding as a *full binary tree*.

Eg.

Symbol	Codeword
A	0
B	100
C	101
D	11



> If any internal node has only one child, you may as well replace that word with the word obtained at the parent node since this will still maintain the encoding.

This encoding is a 17% improvement since the total size of the encoding is 213 megabits

Restating the problem

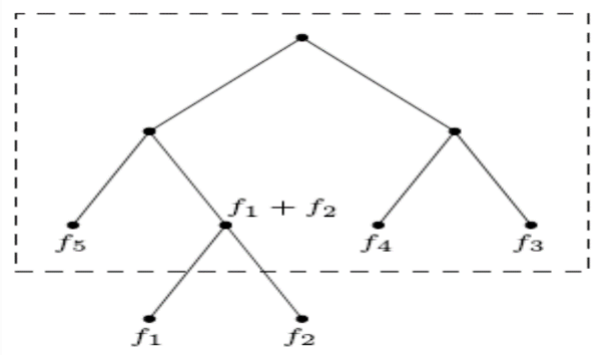
Given the frequencies of $f_1, f_2, \ldots, f_n$ of n symbols, find a tree such that each leaf corresponds to the overall length of the encoding. i.e. minimize this cost:

$$\text{cost of tree} = \sum_{i=1}^{n} f_i \cdot (\text{depth of } i\text{th symbol in tree})$$

Solution (again)

The two symbols with the smallest frequencies must be at the bottom of the optimal tree, as children of the lowest internal node. Since swapping the symbols at these leaves with the symbols with the lowest frequencies will always improve the encoding.

Any internal node has a frequency which is the sum of the frequencies of its leaves.



This tree has the same cost as a tree with $n-1$ leaves with frequencies $(f_1 + f_2), f_3, f_4, \ldots, f_n$. So now the optimum substructure property holds.

```
Huffman(f):
Build a priority queue H out of the frequencies
for k = n+1 to 2n-1:
    i = deleteMin(H)
    j = deleteMin(H)
    Create a node k with children i,j
    f[k] = f[i] + f[j]
    insert(H,k)
```

An observation

A more compressible code is also less random and more predictable.

If there are n outcomes for probabilities $p_1, p_2, p_3, \ldots, p_n$

$$\sum_{i=1}^{n} p_i \log \frac{1}{p_i}$$

The most uniform distributions minimize this value. This is called the *entropy* of the distribution. And the lower the entropy, the less efficient a Huffman encoding is.