GREEDY ALGORITHMS

Use Case | Activity Selection

Input: n activities $a_i := [start_i, finish_i]$

Output: Maximum no. of non-overlapping activities.

Intuition: Commit to a choice, and keep going.

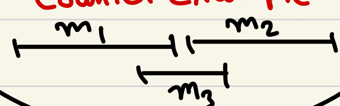
What are intuitive options?

Dynamic Programming?

Too Slow!

Shortest movie?

Counter example



Movies with least overlap?

Correct Answer: Prioritize movies that end the earliest

▷ GREEDY TECHNIQUES

▷ Greedy Always Stays Ahead (GASA)

▷ Define a correct measure of progress formalising that for every step i, the greedy solution G is better than any other valid solution Z.

▷ The measure of progress should explain why we chose this greedy solution.

▷ 10% rule: Give it a name & define it formally.

For the given problem

$F_i(Z)$ = end time of the i^{th} movie (chronologically) from Z

$\forall i \geq 1$, valid solutions in Z

$$\boxed{\text{89\% rule: Argue by induction that } \forall\, i \ \&\ \forall \text{ valid } Z \quad F_i(G) \le F_i(Z)}$$

For the given problem

Proof

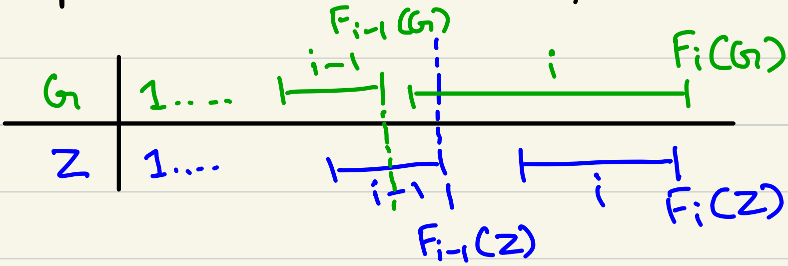
Base Case: $i=1$

$$\Rightarrow \boxed{F_1(G) = \text{smallest end-time of any movie} \le F_1(Z)}$$

Inductive Step:

Assume $F_{i-1}(G) \le F_{i-1}(Z)$

To prove $F_i(G) \le F_i(Z)$, consider the following



Let $F_i(G) > F_i(Z)$ for the above.

- But right after $F_{i-1}(G)$, G considers
  - $F_i(G)$
  - $F_i(Z)$

Since $F_i(G) > F_i(Z)$, G would choose $F_i(Z)$, which contradicts $F_i(G)$.

$$\boxed{\text{GASA often leads to a deeper understanding of how the greedy solution is better}}$$

▷ Local Swap (LS)

For a given greedy solution "G" and an assumed optimal solution "Z", the LS technique:

1. Shows that the first step of G is safely inter-changeable with the same of Z, i.e., swapping these steps has no negative impact on the optimality of Z.

safe to swap z_1 for g_1 ←----

$$\left.\begin{array}{l} G:\ g_1, g_2, \ldots \\ Z:\ z_1, z_2, z_3, \ldots \end{array}\right\} \text{Problem set } I$$

2. Swap the aforementioned steps, thereby reducing the size of original set of optimal steps from Z to Z_1 with the added first step from G.

$$\left.\begin{array}{l} G:\ g_1 \mid g_2, g_3, g_4, \ldots \\ Z:\ g_1 \mid \underbrace{z_2, z_3, z_4, \ldots}_{|Z_1| < |Z|} \end{array}\right\} \text{For the original problem set } I \text{, we get a smaller problem set } I_1 \text{, s.t. } I_1 \subset I$$

3. The reduced optimal set Z_1 corresponds to a reduced problem I_1 similar to the original problem I.
Now, the first step of I_1 is the second step of I

$$\left.\begin{array}{l} G: g_1 \,\vdots\, g_2, g_3, g_4, \dots \\ Z: g_1 \,\vdots\, z_2, z_3, z_4, \dots \end{array}\right\} \text{Reduced problem } I_1$$

z_2 → 1st step of I_1 or 2nd step of I

For the given problem

For a supposed optimal movie schedule Z and given greedy schedule G (least-end-time first)

$$Z: z_1, z_2, z_3 \dots$$

z_1 → can swap with g_1

▷ We can swap z_1 with g_1 as, the end-time for movie g_1 is less than or equal to that of z_1, which means that none of the movies in set $Z_1 [z_2, z_3, z_4, \dots]$ will overlap with g_1.

▷ We can repeat the above for all movies in Z.

▷ We can conclude that the greedy schedule G can perfectly displace Z without reducing optimality.

▷ HUFFMAN ENCODING

▷ Lossless data compression using variable length codes based on frequency of occurrence of characters.

Prefix Code: A coding system where no code is a prefix of another.

Example

	a	b	c	d	e	f	Cost
%	45	13	12	16	9	5	N/A
Fixed	000	001	010	011	100	101	3
Huffman	0	101	100	111	1101	1100	2.24

2.24 ---> Lower average cost

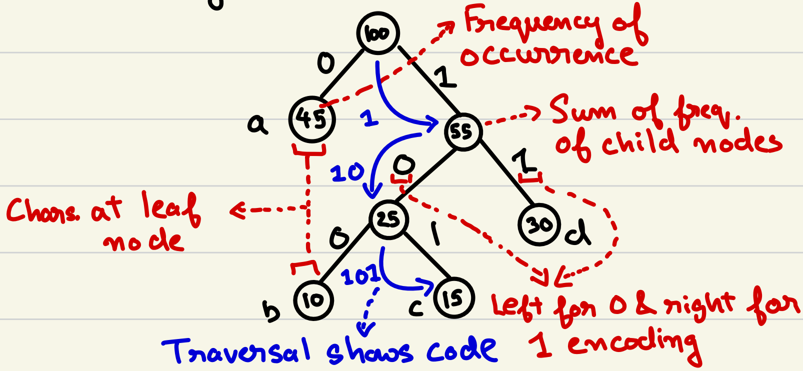
single bit for high freq. number

All prefix codes

▷ Huffman Trees

Huffman trees help generate and represent huffman codes

- ▷ Binary trees with "0" and "1" encoding for left and right child respectively (or opposite per convention).
- ▷ All internal nodes correspond to sum of values of child nodes, where child nodes have character frequency.
- ▷ Encoding obtained by root to leaf traversal



Char	Code
a	0
b	100
c	101
d	11

Constraints of Huffman Tree

- ▷ Valid codes only correspond to traversal from root root to leaf node
- ▷ All internal nodes must have 2 child nodes.

▷ Generating Huffman Tree

Step 1: Generate a min heap of the character frequencies where each value is a node with the key being their resp. freq.

Frequency nodes

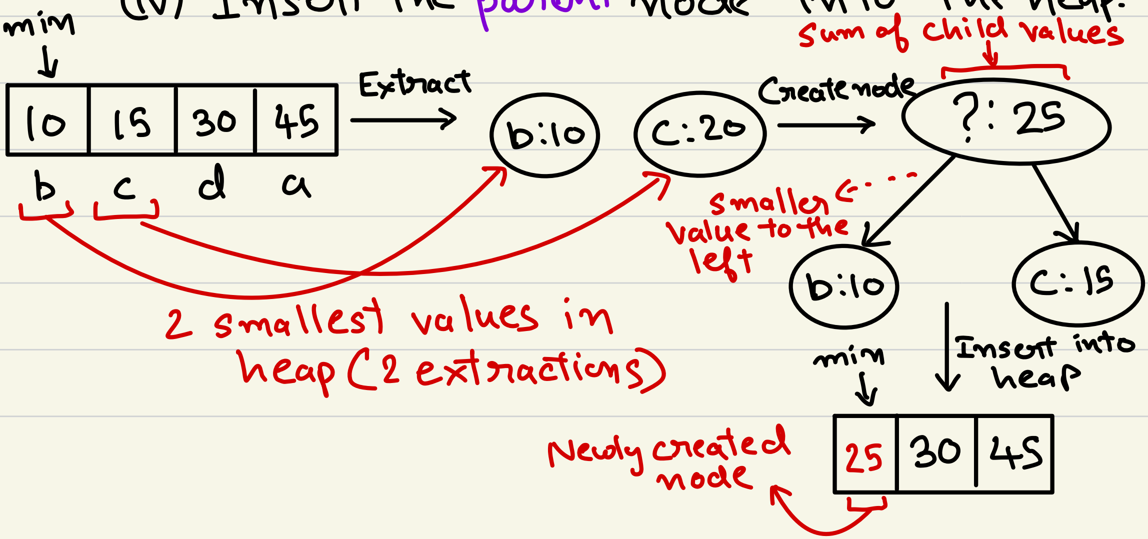
10	15	30	45
b	c	d	a

Step 2: (i) Extract 2 nodes from the heap.

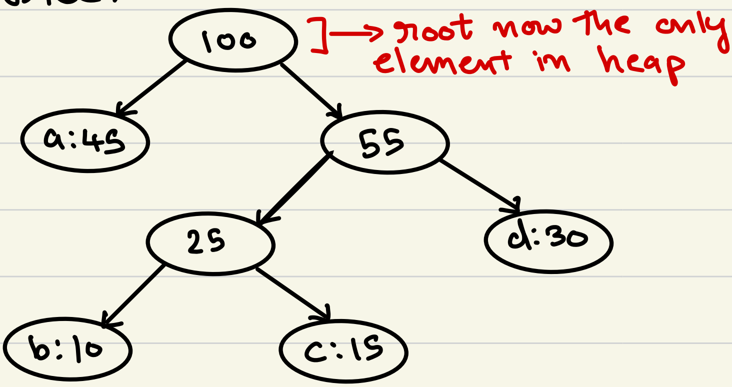
(ii) Create a new node & set its value to the sum of freq. of the 2 extracted nodes

(iii) Assign the smaller char. node to the left child of new parent node & the other to the right child

(iv) Insert the parent node into the heap.



Step3: Repeat steps 2&3 until we have only a single value in the heap, which is the root of the huffman tree.



▷ Generating Huffman Codes from Tree

For every character...

1. Traverse from root node to that character's leaf node.
2. Append "0" to your code string if traversing the left child else append "1".
3. The code string at the end of the traversal is the corresponding Huffman Code