UNIT IV ALGORITHM DESIGN TECHNIQUES

Dynamic Programming: Matrix-Chain Multiplication – Elements of Dynamic Programming – Longest Common Subsequence- Greedy Algorithms: – Elements of the Greedy Strategy- An Activity-Selection Problem - Huffman Coding.

ACTIVITY SELECTION PROBLEM

You are given n activities with their start and finish times. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a time.

Examples:

Input: start[] = {10, 12, 20}, finish[] = {20, 25, 30}
Output: 0 2

Explanation: A person can perform at most two activities. The maximum set of activities that can be executed is {0, 2} [These are indexes in start[] and finish[]]

Input: start[] = {1, 3, 0, 5, 8, 5}, finish[] = {2, 4, 6, 7, 9, 9};
Output: 0 1 3 4

Explanation: A person can perform at most four activities. The maximum set of activities that can be executed is {0, 1, 3, 4} [These are indexes in start[] and finish[]

Approach:

To solve the problem follow the below idea:

The greedy choice is to always pick the next activity whose finish time is the least among the remaining activities and the start time is more than or equal to the finish time of the previously selected activity. We can sort the activities according to their finishing time so that we always consider the next activity as the minimum finishing time activity

Follow the given steps to solve the problem:

- Sort the activities according to their finishing time
- Select the first activity from the sorted array and print it
- Do the following for the remaining activities in the sorted array
- If the start time of this activity is greater than or equal to the finish time of the previously selected activity then select this activity and print it

Note: In the implementation, it is assumed that the activities are already sorted according to their finish time, otherwise the time complexity will rise to O(N*log(N)) and Auxiliary Space will rise to O(N), as we have to create a 2-D array for storing the start and finish times together.

Algorithm Of Greedy- Activity Selector:

GREEDY- ACTIVITY SELECTOR (s, f)

1. n ← length [s]
2. A ← {1}
3. j ← 1.
4. for i ← 2 to n
5. do if $s_i \geq f_i$
6. then A ← A ∪{i}
7. j ← i
8. return A

Example: Given 10 activities along with their start and end time as

S = (A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8 A_9 A_{10})

Si = (1,2,3,4,7,8,9,9,11,12)

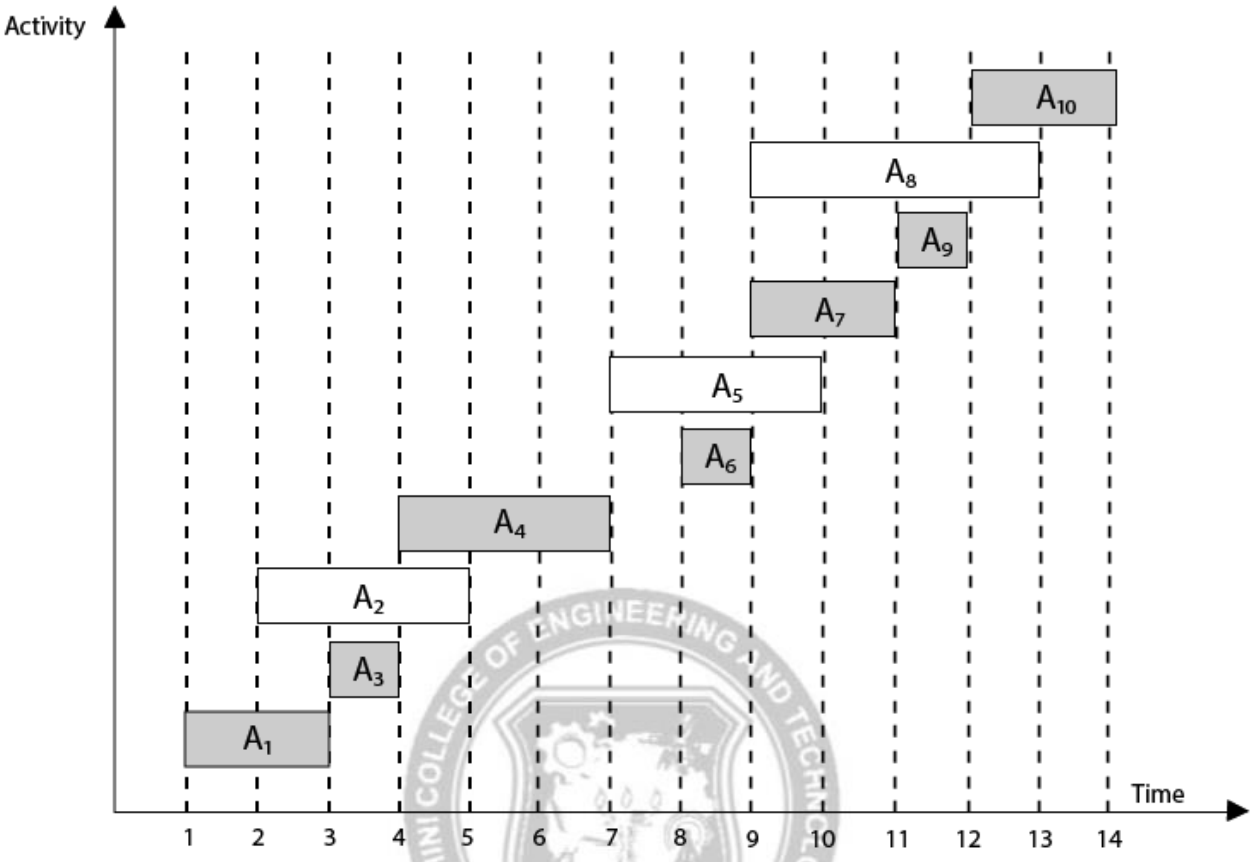
fi = (3,5,4,7,10,9,11,13,12,14)

Compute a schedule where the greatest number of activities takes place

Solution: The solution to the above Activity scheduling problem using a greedy strategy is illustrated below:

Arranging the activities in increasing order of end time

Activity	A_1	A_3	A_2	A_4	A_6	A_5	A_7	A_9	A_8	A_{10}
Start	1	3	2	4	8	7	9	11	9	12
Finish	3	4	5	7	9	10	11	12	13	14



Now, schedule A_1

Next schedule A_3 as A_1 and A_3 are non-interfering.

Next skip A_2 as it is interfering.

Next, schedule A_4 as A_1 A_3 and A_4 are non-interfering, then next, schedule A_6 as A_1 A_3 A_4 and A_6 are non-interfering.

Skip A_5 as it is interfering.

Next, schedule A_7 as A_1 A_3 A_4 A_6 and A_7 are non-interfering.

Next, schedule A_9 as A_1 A_3 A_4 A_6 A_7 and A_9 are non-interfering.

Skip A_8 as it is interfering.

Next, schedule A_{10} as A_1 A_3 A_4 A_6 A_7 A_9 and A_{10} are non-interfering.

Thus the final Activity schedule is:

$(A_1\ A_3\ A_4\ A_6\ A_7\ A_9\ A_{10})$

HUFFMAN CODING

Huffman coding is a lossless data compression algorithm. The idea is to assign variable-length codes to input characters; lengths of the assigned codes are based on the frequencies of corresponding characters.
The variable-length codes assigned to input characters are Prefix Codes, meaning the codes (bit sequences) are assigned in such a way that the code assigned to one character is not the prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bitstream.
Let us understand prefix codes with a counter example. Let there be four characters a, b, c and d, and their corresponding variable length codes be 00, 01, 0 and 1. This coding leads to ambiguity because code assigned to c is the prefix of codes assigned to a and b. If the compressed bit stream is 0001, the de-compressed output may be "cccd" or "ccb" or "acd" or "ab".
See this for applications of Huffman Coding.
There are mainly two major parts in Huffman Coding

Build a Huffman Tree from input characters.

Traverse the Huffman Tree and assign codes to characters.

Algorithm:

The method which is used to construct optimal prefix code is called Huffman coding.

This algorithm builds a tree in bottom up manner. We can denote this tree by T

Let, |c| be number of leaves

|c| -1 are number of operations required to merge the nodes. Q be the priority queue which can be used while constructing binary heap.

Steps to build Huffman Tree

Input is an array of unique characters along with their frequency of occurrences and output is Huffman Tree.

1. Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of the frequency field is used to compare two nodes in the min heap. Initially, the least frequent character is at root)
2. Extract two nodes with the minimum frequency from the min heap.
3. Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.
4. Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete.

Example:

character	Frequency
a	5
b	9
c	12
d	13
e	16
f	45

Step 1. Build a min heap that contains 6 nodes where each node represents the root of a tree with a single node.
Step 2 Extract two minimum frequency nodes from the min heap. Add a new internal node with frequency 5 + 9 = 14.

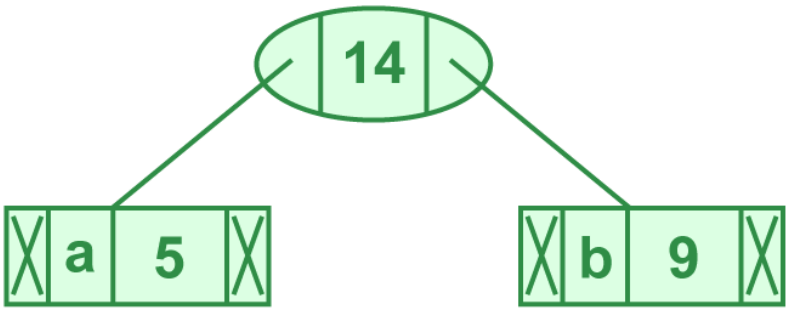


Illustration of step 2

Now min heap contains 5 nodes where 4 nodes are roots of trees with single element each, and one heap node is root of tree with 3 elements

character	Frequency
c	12
d	13
Internal Node	14
e	16
f	45

Step 3: Extract two minimum frequency nodes from heap. Add a new internal node with frequency 12 + 13 = 25

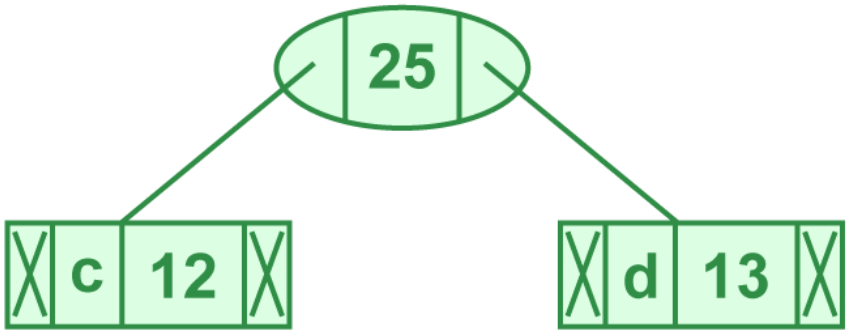


Illustration of step 3

Now min heap contains 4 nodes where 2 nodes are roots of trees with single element each, and two heap nodes are root of tree with more than one nodes

character	Frequency
Internal Node	14
e	16
Internal Node	25
f	45

Step 4: Extract two minimum frequency nodes. Add a new internal node with frequency 14 + 16 = 30

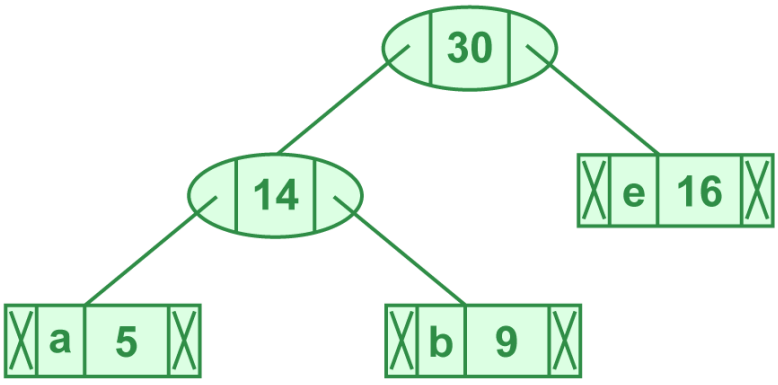


Illustration of step 4

Now min heap contains 3 nodes.

character	Frequency
Internal Node	25
Internal Node	30
f	45

Step 5: Extract two minimum frequency nodes. Add a new internal node with frequency 25 + 30 = 55

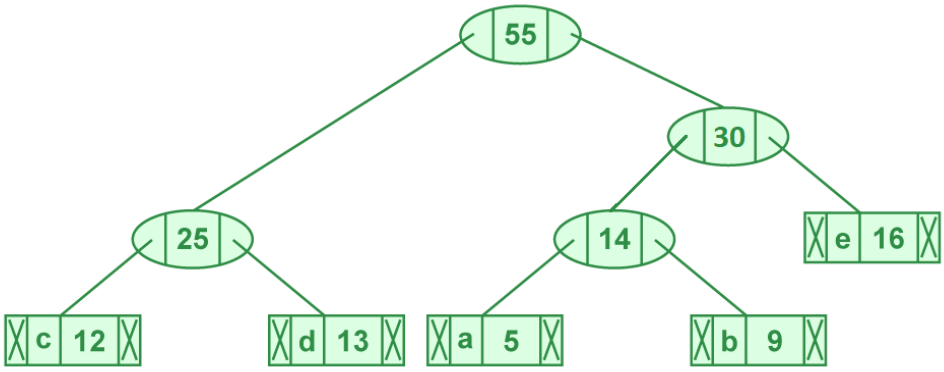


Illustration of step 5

Now min heap contains 2 nodes.

character	Frequency
f	45
Internal Node	55

Step 6: Extract two minimum frequency nodes. Add a new internal node with frequency 45 + 55 = 100

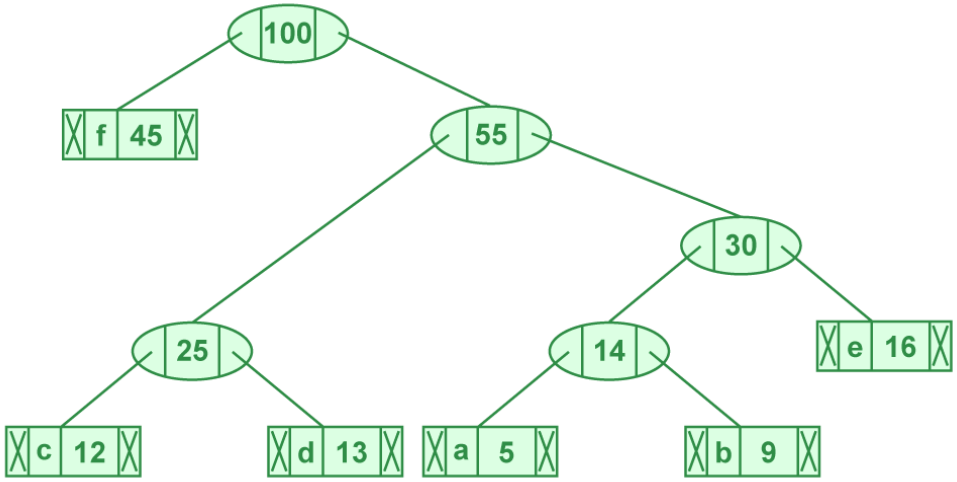


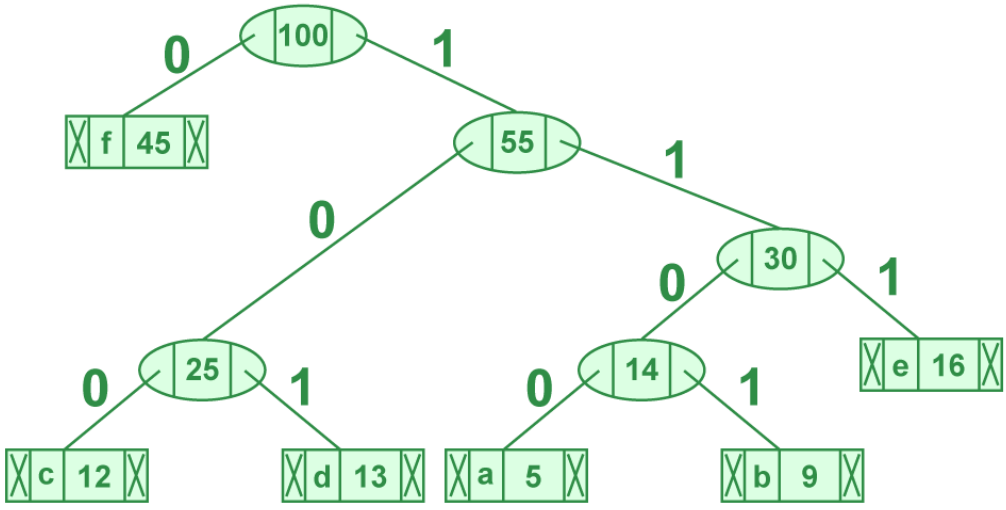
Illustration of step 6

Now min heap contains only one node.

character Frequency
Internal Node 100

Since the heap contains only one node, the algorithm stops here.

Steps to print codes from Huffman Tree:
Traverse the tree formed starting from the root. Maintain an auxiliary array. While moving to the left child, write 0 to the array. While moving to the right child, write 1 to the array. Print the array when a leaf node is encountered.



Steps to print code from HuffmanTree

The codes are as follows:

character	code-word
f	0
c	100
d	101
a	1100
b	1101
e	111