Page No.
Date

Assignment - 7

Ans 1. Greedy is an algorithm paradigm that involves solving the problem in stages, as we get input we see if that can be fit in results.
It is used for optimization (either maximized or minimized) problems.
Eg- Fractional knapsack.

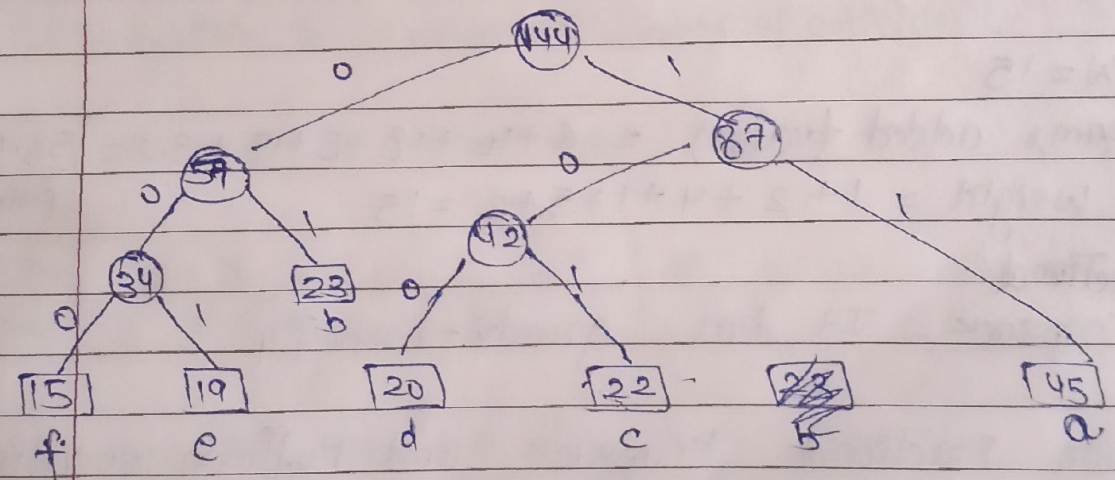
Ans 2. i) Activity selection - Time complexity = $O(n \log n)$
Space complexity = $O(1)$

ii) Job sequencing - Time complexity = $O(n^2)$
Space complexity = $O(n)$

iii) Fractional knapsack - Time complexity = $O(n \log n)$
Space complexity = $O(n)$

iv) Huffman Encoding - Time complexity = $O(n \log n)$
Space complexity = $O(n)$.

Ans 3. a : 45 b : 23 c : 22 d : 20 e : 19 f : 15.



a = 11
b = 01
c = 101
d = 011
e = 001
f = 000

Total length = 48 + 8
= 56

Page No.
Date

Ans4. Priority Queue is used for building Huffman Tree such that nodes with lowest frequency have the highest priority. A Min Heap data structure can be use to implement priority queue.

Applications:-

- Conjunction with cryptography (making transmission of data safe & secure.
- Data compression

Ans5.

Value	10	5	15	7	6	18	3
Weight	2	3	5	7	1	4	1
Value/weight	5	1.6	3	1	6	4.5	3

Value	6	10	18	3	15	5	7
weight	1	2	4	1	5	3	7
value/weight	6	5	4.5	3	3	1.6	1

W = 15

items added (value) = 6 + 10 + 18 + 3 + 15 + 3.2 = 58.5 profit

weight = 1 + 2 + 4 + 1 + 5 + 2 = 15

Ans6. Fractional Knapsack - It has greedy-property

Ans6. Both Fractional knapsack and Huffman encoding has greedy-property because both problem algorithm involves solving the problem in stages, as we get input, we see if that can be fit in result.

- Fractional Knapsack — The basic idea of greedy approach is to calculate the value/weight for each item and sort the items on basis of this ratio. Then take the item with the highest ratio and add them until knapsack is full.
Hence, it is optimisation problem in which we maximise the profit.

- Huffman Encoding - Greedy algorithm is an algorithm that follows the problem solving mechanism of making the locally optimal solution at each stage with thinking of finding a global optimum solution for problem and in huffman encoding in every stage we try to find the prefix tree binary code and try to minimize expected code word for optimum solution for compress the data.

Ans7. This problem will be solved using activity selection algorithm to maximise number of activities.

we sort the table w.r.t end time.

task	1	2	3	4	5	6
Start Time	1	2	0	6	9	10
End Time	3	5	7	8	11	12

The task that end first take least time and must be included to maximise the numbers of activity

Task that should be included are 1, 4, 5

Page No.
Date

Ans 8. This problem will be solved using job scheduling with deadline.

Profit	20	15	10	5	1
Deadline	2	2	1	3	3

We sort the table wrt profit

profit	20	15	10	5	1
Deadline	2	2	1	3	3
task	1	2	3	4	5

Since maximum deadline is 3, therefore 3 task can be performed.
We will pick task of maximum profit and see whether it can be performed within given deadline or not.

2	1	4		
0	1	2	3	

task no 1, 2, 4 will be performed to maximise the profit

profit = 20 + 15 + 5 = 40

Ans 9. Many algorithms uses the greedy approach to solve the problem because greedy approach make algorithm faster. But there are many situations where greedy algorithm become ineffective.

Eg. Dijkstra algorithm on negative edge cycle.

Eg - 0-1 knapsack problem where object are not breakable.

Page No.
Date

Ans. Job sequencing algorithm is used maximise the profit by perform certain tasks within given deadline.

Job sequencing algorithm uses greedy approach to maximise the profit.
Based on the last deadline we can find no. of tasks can be performed. We pick the tasks that give maximum profit and see whether it can be performed within given deadline. In this way, we can desired max. profit by using greedy approach.

Algorithm.

1. Take vector of pair(profit, deadline) job.
2. Sort the pair w.r.t profit
3. Take integer maxEnd, and assign it to zero.
4. Take integer i and assign it to zero.
5. Check if i is smaller than total no. of task → yes continue
   else move to step 10
6. check if job[i].second is greater than maxend → yes continue
   else move to step 8
7. maxend = job[i].second;
8. Increase i by 1
9. move to step 5
10. Take an array of maxend size fill(End max)
11. Assign Initial its all element with -1.
12. Assign i to zero.
13. check if i is smaller than total no. of tasks; Yes-continue
    else step 19
14. Take an integer j and assign j = job[i].second - 1
15. While j >= 0 and fill[j] != -1 decrement j else
    move to 16
16. If j >= 0 and fill[j] != -1 continue else move to 18
17. fill[j] = i, counter++, profit += job[i].first
18. Increase i by 1
19. print profit.