TUTORIAL - 7

Que1) What is the Greedy algorithmic paradigm? When should you make use of Greedy Algorithms in problem solving?

Ans Greedy is an algo paradigm that builds up a solution piece by piece always choosing the next piece that offers the most obvious and immediate benefit.

So, the problems where choosing totally optimal also leads to global solution are best fit for greedy.

These are simple instinctive algos used for optimization either maximized or minimized problems. This algo makes the best choice at every step and attemps to find the optimal way to solve the whole problem.

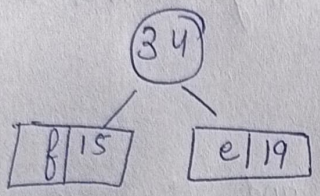
Que2) Analyse the time and space complexity of the following algorithms:

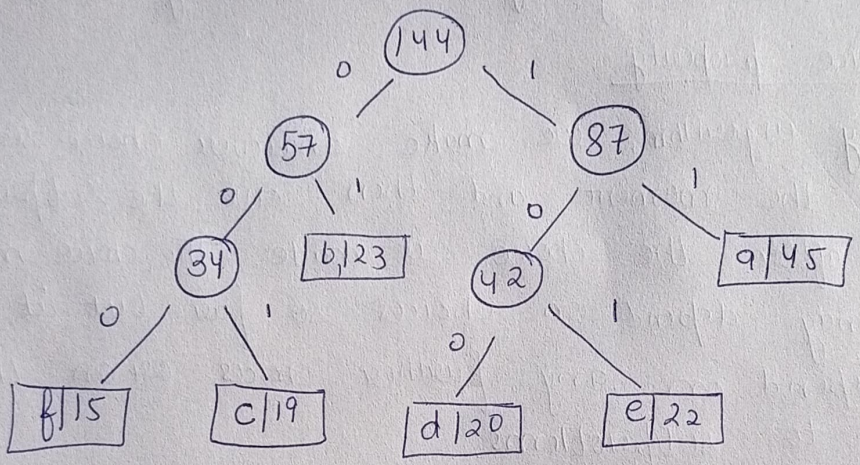
	Time Complexity	Space Complexity
Activity Selection	O(n log n) if input activities may not be sorted O(n) when i/p is sorted	O(1) constant. No extra space is used.
Job Sequencing	O(n log (n))	O(n)
Fractional Knapsack	O (n log n)	O(1)
Huffman Coding	O (n log n)	O(n)

Que3) A file contains the following characters & their corresponding frequencies as shown below:

a: 45, b: 23, c: 22, d: 20, e: 19, f: 15

We use Huffman coding for data compressions, generate the encoding for a, b, c, d, e, f using Huffman encoding and find the average length of a character after compression





a = 11
b = 01
c = 101
d = 100
e = 001
f = 000

Que 4) What data structure is used while implementing Huffman Encoding? What are the applications of Huffman Encoding?

Ans Priority queue is used for building the Huffman Tree such that nodes with the lowest frequency have the highest priority.

A min Heap data structure can be used to implement the functionality of a priority queue.

Applications

a) Huffman Encoding is widely used in compression formats like GZIP, PKZIP & BZIP2.

b) Multimedia codes like JPEG, PNG and MP3 uses Huffman encoding.

c) Huffman encoding still dominates the compression industry since newer arithmetic and range coding schemes are avoided due to their potential.

Ques) Prove that Fractional Knapsack problem and Huffman Encoding has the greedy choice property.

Greedy choice property

In greedy algorithm, we make whatever choice seems best at the moment and then solve the subproblems arising after the choice is made. The choice made by it may depend on choices so far but it cannot depend on any further choices or on the solutions to subproblems.

Fractional Knapsack

Example - Robbery

Want to rob a house and have a knapsack which hold 'B' pounds of stuff. Want to fill knapsack with the most profitable items. Fractional knapsack can take a fraction of an item.

Let j be the item with maximum V_i / W_i. Then there exists an optimal solution in which you take as much of item j.

Suppose that there exists an optimal solution if you didn't take as much of item j as possible.

If the knapsack is not full, add some more of item j & you have a higher value of solution.

We thus assume that knapsack is full.

There must exist some item ie $k \neq j$ with $\frac{v_k}{w_k} < \frac{v_j}{w_j}$ i.e in the knapsack.

We also must have that not all of j is in the knapsack. We can therefore take a piece of k, with ε, weight, out of the knapsack and put a piece of j with ε weight in. This increases the knapsack value.

Huffman Encoding

Suppose that we have a 100,000 characters data file that we wish to store. The file contains only 6 characters, appearing with following frequency.

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

We would like to find a binary code that encodes the file using as few bits as possible.

We can encode using two scheme –

1. Fixed length Code
2. Variable length Code

A code will be a set of code words.

Que7) Consider a set of activities given below, along with the starting and finishing time of each activity, find the maximum number of activities performed by a single person assuming that a person can only work on a single activity at a time

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

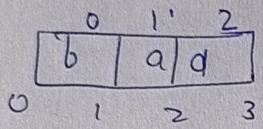
Activities (max) = 3

```
#include<bits/stdc++.h>
using namespace std;

struct Activity
{   int start, finish;
}

bool activityCompare (Activity s1, Activity s2)
{   Return (s1.finish < s2.finish);
}

void printMaxActivity (Activity arr [], int n)
{   Sort (arr, arr+n, activityCompare);
    Cout << "following activities are Selected";
    int i = 0;
    cout << "(" << arr[i].start << ", " << arr[i].finish << ")";
    for (int j = 1; j<n; j++)
    {   if (arr[j].start >= arr[i].start)
        {   cout << "(" << arr[j].start << ", " << arr[j].finish << ")";
            i = j;
        }
    }

int main()
{   Activity arr[] = {{1,3}, {2,5}, {0,7}, {6,8}, {9,11}, {10,12}};
    int n = sizeof(arr) / sizeof(arr[0]);
    printmaxActivity (arr, n);
    Return 0;
}
```

Ques) Consider the following jobs where every job has a deadline and an associated profit if completed with deadline. It is given that every job takes a single unit of time. Perform job sequencing such that you maximize the total profit if only one job can be scheduled at a time?

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

Ans



Total people = 3

Profit = 20 + 15 + 5 = 40

```
# include <iostream>
# include <vector>
# include <algorithm>
using namespace std;

bool compare (pair<int, int> a, pair<int, int> b)
    return a.first > b.first;

int main()
{   vector <pair<int, int> job;
    int n, profit, deadline;
    cin >> n;
    for (int i = 0; i<n; i++)
    {     cin >> profit >> deadline;
          job.push_back (make_pair (profit, deadline));
    }
    sort (job.begin(), job.end(), compare);
    int maxEndTime = 0;
```

```
for (int i=0; i<n; i++)
{
    if (job[i].second > maxEndTime)
        maxEndTime = job[i].second;
}

int fill[maxEndTime];
int count=0, maxProfit=0;

for (int i=0; i<maxEndTime; i++)
    fill[i] = -1;

for (int i=0; i<n; i++)
{
    int j = job[i].second-1;
    while (j>=0 && fill[j] != -1)
        j--;
    if (j>=0 && fill[j]==-1)
    {
        fill[j] = i;
        count++;
        maxProfit += job[i].first;
    }
}

cout << count << " " << maxProfit << endl;
}
```

Ques) When should we avoid making use of greedy approach in problem solving? Give examples to support your explanation

Ans It is not suitable for problems where a solution is required for every sub problem the greedy strategy can be wrong, in worst case even lead to a non-optimal solution

Example

(i) Dijkstra's algorithm fails to find or fails with negative graphs.

(ii) We can't break objects in the Knapsack problem. The solution that we obtain when using a greedy strategy can be pretty bad too. We can always build an input to the problem that makes greedy algo fail badly.

(iii) Another example is the travelling Salesman problem. Given a list of cities & the distance b/w each pair of cities, what is shortest possible route that visits each city exactly once and returns to origin city?

- We can greedily approach the problem by always going to the nearest possible city. We select any of the cities as the first one & apply that strategy.
- We can build a disposition of the cities in a way that the greedy strategy finds the worst possible solution
- We have seen that a greedy strategy could lead us to disaster but there are problem in which such an approach can approx the optimal solution quite well.

Que10) How can you optimize the approach used to solve the Job Sequencing problem? Write an algorithm for the same

Ans We can optimize the approach used to solve the job sequencing problem by using priority queue (max heap)

Algorithm

1. Start the job based on their deadline
2. Iterate from the end and Calculate the acutable slots between every 2 consecutive deadlines, include the profit, deadline & jobID of ith job in the max heap.
3. While the slots are available and there are jobs left in the max heap include the job ID with maximum profit and deadline in the result.
4. Sort the result array based on their deadlines