Ans:- Doukush 21037 ESE-1 (5th sem)

Part a)

set	S_1	S_2	S_3	S_4	S_5	S_6	S_7
profit	3	5	20	18	1	6	30
Deadline	1	3	4	3	2	1	2

Step1

Arrange descending as per profit

set	S_5	S_1	S_2	S_6	S_4	S_3	S_7
profit	1	3	5	6	18	20	30
Deadline	2	1	3	1	3	4	2

step2

$3+1+5+20=29$

profit = 29

Q1
part b)

Greedy method	Dynamic Programming
In a greedy Algorithm we we make whatever choice seems best at the moment in the hope that it will lead to global optimal soln	In Dynamic programming we make decision at each step considering current problem and solution to previously solved sub problem. to calculate optimal soln
In greedy method, sometimes there is no such gurantee of getting optimal soln	It is gurranted that Dynamic programming will generate an optimal soln as it generally consider all possible cases and then chose the best
A greedy method follows the problem solving heuristic of making the locally optimal choice at each stage	A dynamic programming is an algo technique which is usually based on recurrent formula that uses some previously calculated states

It is more efficient in terms of memory as it never look back or revise previous choices

It requires dp table for memorization and it increases it's all memory complexity

Q3

P.B b)

Kruskal's algorithm to find MST of an undirected graph

Step 1:- Sort all the the edges in non-decreasing order

Step 2:- Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not form

Step 3:- Repeat step 2 until there are (V-1) edges in the spanning tree

The algorithm is a greedy algorith. The greedy choice is to pick the smallest weight edge that does not cause a cycle in the MST constructed so far.

Dakksh
21037
CSE (5th sem)

Q4

0-1 knapsack using Branch and Bound

Branch and bound is an algorithm design paradigm which is generally used for solving combinatorial optimization problems. These problem typically exponential in terms of time complexity and may require all possible permutations in worst case. Branch and Bound solve these problems relatively quickly

problem statement for 0-1 knapsack
Given two integers arrays. val[0...n-1] and wt[0...n-1] that represent values and weights associated with n-items respectively. Find out maxm value subset of val[] such that sum of the weights of this subset is smaller than or equal to knapsack capacity W.

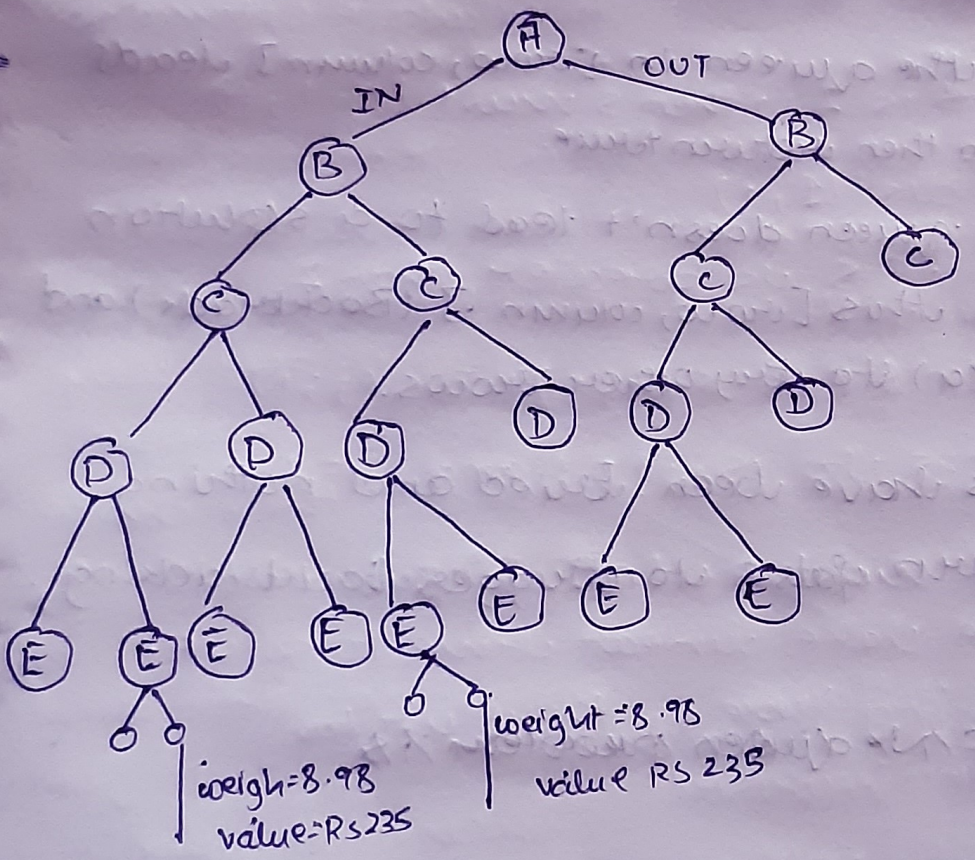
The Backtracking based solution works better than brute force ignoring infeasible solutions. We can do better if we know a bound on best possible solution subtree rooted with every node. If the best in subtree is worse than current best,

we can simply ignore this node and its subtree so we compute bound for every node and compare the bound with current best solution before exploring the node.

Eg:-

Knap sack capacity
W = 10

A	2	Rs40
B	3.4	Rs50
C	1.98	Rs100
D	5	Rs95
E	3	Rs30



Branch and Bound is very useful technique for searching a soln but in worst case, we need to fully calculate the entire tree. At best, we only need to fully calculate one path through the tree and prune the rest of it.

Q4

A)

Backtracking Algorithm

1) Start in the leftmost column

2) If all queens are placed return true

3) Try all rows in the current column
Do following for every tried row

a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.

b) If placing the queen in [row, column] leads to a solution then return true.

c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.

4) If all rows have been tried and nothing worked return false to trigger backtracking.

Time complexity of N* queen problem is
O(N!)

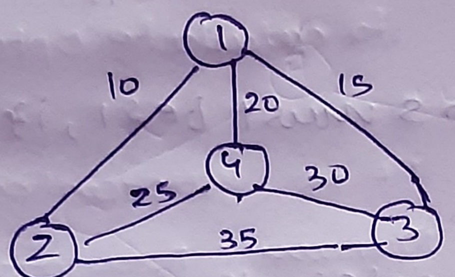
Time complexity of N-queen using backtracking:-
Solution N Queen problem using backtracking checks for all possible is the worst case, average case and the best case time complexity of an algorithm.

So ~~for~~ average and worst case complexity of the solution is O(N!). The best case occurs if you find your solution ~~without~~ before exploiting all possible arrangements.

~~if you need, all~~

~~Q3~~:- Q5:-

A TSP tour in the graph is 1-2-4-3-1. The cost of the tour is 10+25+30+15 which is 80



The problem is · Famous NP hard problem.

Dynamic Programming solution

Let the given set of vertices {1, 2, 3, 4 - - n}

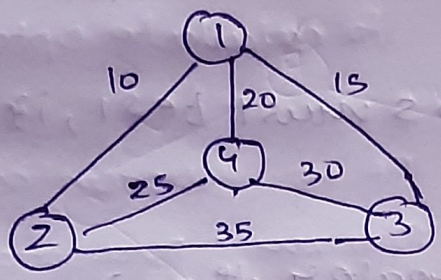
Let us consider 1 as starting and ending point of output. For every other vertex i (other than 1), we find the min^m cost ~~path~~, path with 1 as starting point, i as the ending point and all vertices appearing exactly once. Let the cost of this path be cos(i), the cost of corresponding cycle would be ·

cost(i) + dist(i, 1) where dist(i, 1) · is the distance from ~~i~~ i to 1. Finally we return the min^m of all [cost(i) + dist(i, 1)] val

So the average and worst case complexity of the solution is O(N!). The best case occurs if you find your solution before exploiting all possible arrangements. if you need all.

Q:- Qs:-

A TSP tour in the graph is 1-2-4-3-1.
The cost of the tour is 10+25+30+15 which is 80



The problem is famous NP hard problem.

Dynamic Programming solution

Let the given set of vertices {1, 2, 3, 4 ... n}

Let us consider 1 as starting and ending point of output. For every other vertex i (other than 1), we find the min^m cost path with 1 as starting point, i as the ending point and all vertices appearing exactly once. Let the cost of this path be cost(i), the cost of corresponding cycle would be.

cost(i) + dist(i, 1) where dist(i, 1) is the distance from i to 1. Finally we return the min^m of all [cost(i) + dist(i, 1)] values.

To calculate cost (i) using dynamic Programming, we need to have some recursive relation in terms of sub problems. Let us define a term C(S,i) be the cost of the minm cost path visiting each vertex in set S exactly once, starting at 1 and ending at i.

We start with all subsets of size 2 and calculate C(S,i) for all subsets where S is the subset then we calculate C(S,i) for all subsets of S of size 3 and so on.

If size of S is 2, then S must be {1,i}

$C(S,i) = dist(1,i)$

Else if size of S is greater than 2

$C(S,i) = \min \{ C(S-\{i\},j) + dis(j,i) \}$

where j belongs to S, $j != i$ and $j != 1$

There are at most $O(n^2 * 2^n)$ sub problems, and each one take linear time to solve. The total running time is therefore $O(n^2 * 2^n)$.