(1) Discuss the components of Greedy algorithm.

Greedy approach: It is a popular design technique used for solving optimization problems. The global solution of the problem is obtained by combining locally optimal decisions.

Components:

1) Objective function - It should be either maximized or minimized based on given problem.
2) Generating multiple candidate solution - It may have N inputs or candidate solutions.
3) Selection procedure - It is based on some greedy locally optimal criteria.
4) Feasibility check - It determines if the selected item is feasible as per constraint
5) solution check - This checks whether the partial solution together constitute a global solution.

```
Algorithm Greedy (a,n)
{ // a[1:n] Contains the n inputs
   Solution = ϕ;
   for i=1 to n do
   {
      x = Select (a);          // Selection procedure
      if Feasible(solution, x) then  // Feasibility check
           Solution = Union(solution, x); // Solution check
   }
   return solution;
}
```

(2) Define the term (i) Feasible solution (ii) optimal solution (iii) Local optimality

(i) Feasible solution:

⇒ Any subset solution that satisfies a set of constraints is called feasible solution.

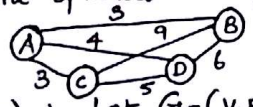
(ii) optimal solution:

⇒ Any feasible solution that maximizes or minimizes the given objective function is called optimal solution.

(iii) Local optimality

⇒ The best decision at a particular point of time irrespective of any future consideration.

③ What is meant by MST? Find the optimal cost for the following graph using kruskal's algorithm:



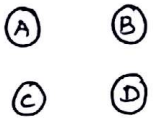
MST (Minimum-cost Spanning Tree) => Let $G=(V,E)$ be an undirected connected graph. A subgraph $t=(V,E')$ of G is a spanning tree of G iff t is a tree.

A minimum-cost spanning tree of a weighted graph G is a spanning tree T such that the sum of the weights of all its edges is minimum.

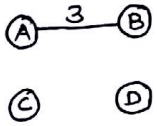
Mathematically, $$\text{weight}(T) = \min\left(\sum_{(u,v)\in T} \text{weight}(u,v)\right)$$

Problem solution:

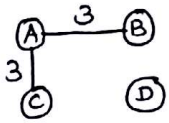
Step(1)



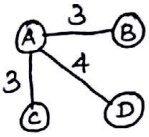
step(2)



step(3)



Step (4)



step(5)

Minimum (optimal cost)

Cost = 3+3+4

= 10

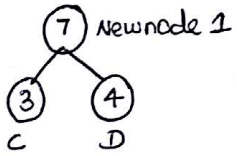
---

④ Compute huffman code for the set of symbols {A,B,C,D} with their frequencies {16,12,3,4} respectively and find its optimal cost.

solution: Step(1) Sort the symbol based on their frequencies in ascending order.

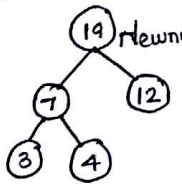
Symbol	C	D	B	A
Frequencies	3	4	12	16

Step(2) select 1st two least frequency



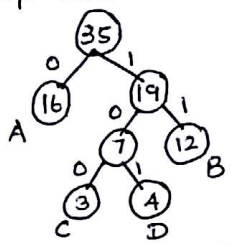
Step(3) select Newnode1 = 7

B = 12



Step(4) select Newnode2 = 19

A = 16



[Assign 0-leftchild 1-rightchild]

Symbol	Frequency	Code	Codelength	Cost
A	16	0	1	16x1=16
B	12	11	2	12x2=24
C	3	100	3	3x3=9
D	4	101	3	4x3=12

Total cost = 61

(5) Differentiate Kruskal's algorithm with prims algorithm.

Kruskal's algorithm	prim's algorithm.
(i) Starts with all the vertices of the graph as a forest and every addition of edge takes a forest one step further towards a Complete tree.	starts with any vertex of the graph arbitrarily
(ii) It is based on the acyclic nature of the graph	It is based on the connectedness.
(iii) More time is saved as the edges are sorted first.	Saves a lot of spaces instead, as no additional structures are used for storing the edges.
(iv) Adding of an edge is performed by selecting the sorted edge.	Always a new vertex is joined to an old vertex.
(v) Next edge is always determined by the edge list.	Addition of nodes is based on the concepts of shortest distance or weight.

(6) Discuss the components of Dynamic programming.

Dynamic programming is useful for solving multistage optimization problems.

Components:

1. Stages - The given problem can be divided into a number of subproblems called stages.
2. Decision - In every stage, there can be multiple decisions out of which the best decision should be taken.
3. State - There are many states in the problem. Every decision takes moves the problem from the current state to the next state of the next stage.
4. policy - It determines the decision at each stage (Rules).

(7) State principle of optimality.

Principle of optimality ⇒ It states that the optimal sequence of decisions in a multistage decision problem is feasible if and only if its sub-sequences are optimal.

In other words, the Bellman principle of optimality states that "an optimal policy has the property that whatever the initial state and decision are, the remaining decisions must be constitute an optimal policy with regard to the state resulting from the first decisions.

⑧ What are the advantages and disadvantages of dynamic programming paradigm?

Advantages:

(i) Idea of reuse - to reuse the solutions of the subproblems rather than recomputing them.

(ii) Its solution is globally optimum.

(iii) It is a valuable tool for computing both exact and approximate solutions.

Disadvantages (or) limitations:

(i) This approach works better on linear ordered objects than on an unordered set of objects.

(ii) There is no standard mathematical formulation.

(iii) Most of this problems have time constraints.

⑨ Compare dynamic programming with divide and conquer and Greedy approach.

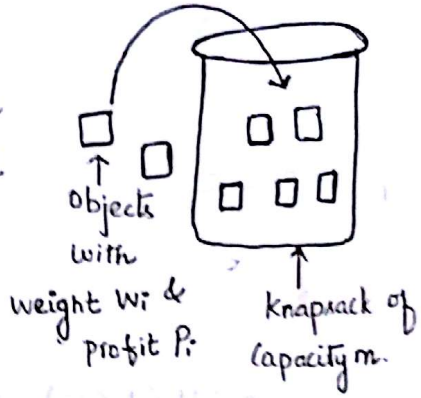
Dynamic programming	Divide and conquer.
(i) It divides a problem into multiple sub problems and uses either the top-down or the bottom-up strategy for solving problems.	It uses the top-down approach for solving problems.
(ii) Subproblems are overlapping problems.	subproblems are independent of each other.
(iii) It is suitable for solving optimization problems.	It is suitable for solving non-optimization problems also.

Dynamic Programming	Greedy approach
(i) It is useful for solving optimization problems. (Multistage)	It is useful in solving optimization problems.
(ii) It can generate multiple sequences of solutions for the given problem.	Generates only one solution sequence.

① Explain in detail about greedy knapsack problem. Find an optimal solution to the knapsack instance $n=7$, $m=15$, $(P_1, P_2, \ldots P_7) = (10,5,15,7,6,18,3)$ and $(w_1, w_2, \ldots, w_7) = (2,3,5,7,1,4,1)$

Knapsack problem - Greedy approach

Let consider given 'n' objects and a knapsack or bag, object 'i' has a weight W_i and the knapsack has a capacity m.



If a fraction x_i, $0 \le x_i \le 1$ of object i is placed into the knapsack then a profit of $P_i x_i$ is earned.

Objective ⇒ Filling of the knapsack that maximizes the total profit earned.

Constraints ⇒ Total weight of all chosen objects to be atmost m.

Formally, the problem can be stated as,

$$\text{maximize} \sum_{1 \le i \le n} P_i x_i \Rightarrow \text{(optimal solution)}$$

$$\text{subject to} \sum_{1 \le i \le n} w_i x_i \le m \quad \text{and} \quad 0 \le x_i \le 1,\ 1 \le i \le n \Rightarrow \text{(Feasible solution for any set } (x_1, x_2, \ldots x_n))$$

```
Algorithm GreedyKnapsack(m,n)
// P[1:n] and W[1:n] contain the profits and weights respectively,
// of the n objects ordered such that P[i]/w[i] ≥ P[i+1]/w[i+1].
// m is the knapsack size and x[1:n] is the solution vector.
{
   for i=1 to n do
        x[i] = 0.0;
          U = m;
   for i=1 to n do
   {  if (w[i] > U) Then break;
      x[i] = 1.0;
        U = U - w[i];
   }
   if (i ≤ n) then x[i] = U/w[i];
}
```

Pbm-solution: Given $n=7$, $m=15$

Profit $(P_1, P_2, \ldots P_7) = (10, 5, 15, 7, 6, 18, 3)$

Weight $(W_1, W_2, \ldots W_7) = (2, 3, 5, 7, 1, 4, 1)$

Step(1): Find the profit and weight ratio.

$P_1/W_1 = 10/2 = 5$ | $P_2/W_2 = 5/3 = 1.66$ | $P_3/W_3 = 15/5 = 3$ | $P_4/W_4 = 7/7 = 1$ | $P_5/W_5 = 6/1 = 6$ | $P_6/W_6 = 18/4 = 4.5$ | $P_7/W_7 = 6/1 = 6$

Step(2): Arrange the object in the non-increasing order of profit and weight ratios.

We get the order as $x_5, x_1, x_6, x_3, x_7, x_2, x_4$.

Consider $U=m=15$, check the condition

Condition $W[i] < U$	Probability (x)	Capacity $U = U - W[i]$
$W_5 < U$ $1 < 15 \Rightarrow$ True	$x_5 = 1$	$U = 15 - W_5 = 15 - 1 = 14$
$W_1 < U$ $2 < 14 \Rightarrow$ True	$x_1 = 1$	$U = 14 - W_1 = 14 - 2 = 12$
$W_6 < U$ $4 < 12 \Rightarrow$ True	$x_6 = 1$	$U = 12 - W_6 = 12 - 4 = 8$
$W_3 < U$ $5 < 8 \Rightarrow$ True	$x_3 = 1$	$U = 8 - W_3 = 8 - 5 = 3$
$W_7 < U$ $1 < 3 \Rightarrow$ True	$x_7 = 1$	$U = 3 - W_7 = 3 - 1 = 2$
$W_2 < U$ $3 < 2 \Rightarrow$ False	$x_2 = U/W_2$ $x_2 = 2/3$	$U = 2 - W_2 = 2 - 2 = 0$ $U = 0$ (stop)

$\therefore$ Probability of object selected $(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = (1, 2/3, 1, 0, 1, 1, 1)$

Step(3): To find optimum solution (profit)

$$\sum P_i x_i = (1 \times 10) + (2/3 \times 5) + (1 \times 15) + (0 \times 7) + (1 \times 6) + (1 \times 18) + (1 \times 3)$$

$$= 10 + 3.33 + 15 + 0 + 6 + 18 + 3$$

Maximum Profit $= 55.33$

② Explain in detail about Huffman code algorithm. Let A={a/5, b/9, c/12, d/13, e/16, f/45} be the letters and its frequency distribution in a text file. compute a suitable huffman coding to compress the data effectively and also compute optimal cost.

Answer:

* Huffman coding ⇒ Huffman codes are used in the communication theory and in the field of data compression extensively.
* Encoder ⇒ the related symbols or message must be encoded using any coding technique. The encoder assigns a unique address to all the symbols and then the message is transmitted across the channel as a series of 0s and 1s.
* Decoder ⇒ the message is received across the channel and decoded using decoder. The decoded message is received by the receiver.

Informally, Huffman coding algorithm is written as,

Step1: List the symbols and sort them according to their frequencies in an ascending order.

Step2: pick two symbols having the least frequencies.

Step3: Create a new node. Add the probabilities of the symbols selected in step2 and label the new node in it.

Step4: Repeat Step 2 and 3 till only one node remains.

Step5: For every internal node, start assigning code 0 for left child and 1 for right child.

Step6: For every leaf node, construct the code word by tracing the code from the root to the leaf node that represents each label.

```
Algorithm  Huffman (A[1....n])
I/p: A set of n symbols with frequencies
o/p: optimal code for symbols.
Begin
      Let Q be the priority queue
      Q= { Initialize priority queue with frequencies of all symbols }
      Repeat n-1 times
            create a new node z
            x = extract_min (Q)
            y = extract_min (Q)
            left(z) = x ; right(z) = y ;
            frequency(z) = frequency(x) + frequency(y)
            insert (Q, z)
      End repeat
            return (extract_min (Q))
End
```

Problem-solution: $A = \{ a/5, b/9, c/12, d/13, e/16, f/45 \}$

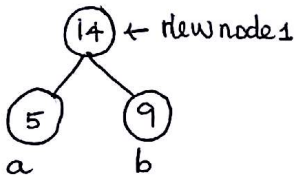
Symbol	a	b	c	d	e	f
Frequency	5	9	12	13	16	45

Step(1): Arrange the symbol based on their frequency (Ascending order)

$\{a, b, c, d, e, f\} = \{5, 9, 12, 13, 16, 45\}$

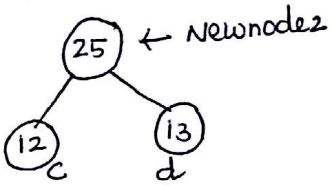
Step(2): Construct the tree with least two frequencies.

a=5, b=9



Newnode1 = 14 > c=12 and d=13

Select c=12, d=13

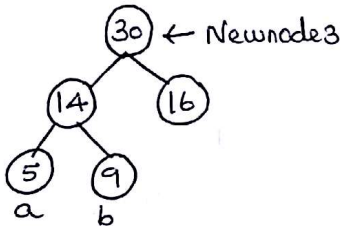


Newnode2 = 25

Newnode1 = 14

e = 16

Select Newnode1=14 and e=16

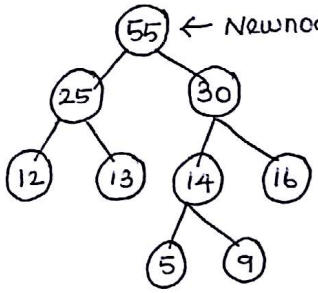


Newnode3 = 30

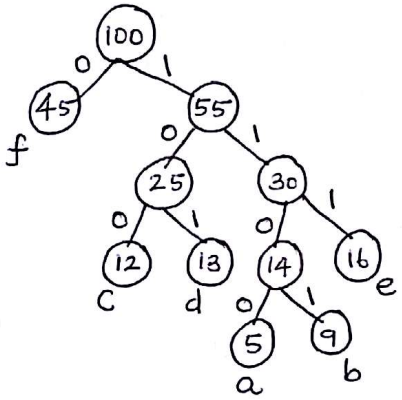
Newnode2 = 25

f = 45

Select newnode2=25 and Newnode3=30



Newnode4 = 55
f = 45 } ⇒



Assign 0's – leftchild

1's – rightchild

Symbol	Frequency	code	Code length	Cost
a	5	1100	4	5x4=20
b	9	1101	4	9x4=36
c	12	100	3	12x3=36
d	13	101	3	13x3=39
e	16	111	3	16x3=48
f	45	0	1	45x1=45

⇒ ∴ Total cost = 20+36+36+39+48+45

= 224

∴ optimum Cost = 224

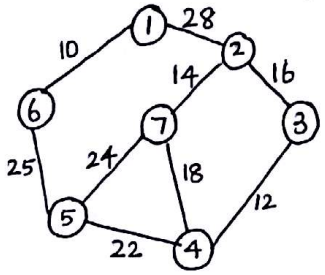
Complexity Analysis

Algorithm based on min-heap data structure

∴ Removing the root and adjusting take $\Theta(\log m)$ times and loop takes at most (m-1) steps.

∴ The complexity = $\Theta(m \log m)$

③ Explain in detail about kruskal's algorithm. Apply Kruskal's algorithm to the following graph and construct corresponding MST's. What is the minimum cost obtained.



Answer:

⇒ Minimum Spanning Tree (MST): A minimum cost spanning tree (MST) of a weighted undirected graph G is a spanning tree T such that the sum of the weight of all its edges in minimum.

⇒ Mathematically, $\text{Weight}(T) = \min \left[\sum_{(u,v) \in T} \text{weight}(u,v) \right]$

Properties of MST:

1) An MST has the minimal cost.
2) It has all the vertices, which are connected indirectly.
3) It has no cycles.

Kruskal's algorithm: Informally, a kruskal's algorithm can be written as follows,

Step1: Arrange all the edges in an increasing order of weights

Step2: Repeat the following step until all the vertices are connected

2a: Add an edge if the added edge does not form a cycle.

Step3: Add the costs of all edges in an MST to get the minimum cost.

```
Algorithm kruskal(G, cost, n, t)
{
  Sort the edges and form the edge list
    E = {e1, e2, .... en} such that W(e1) ≤ W(e2) ≤ ... ≤ W(en)
  for all v ∈ V of G do
        make_set(v)
  End for
   i=0; mincost=0.0;
  while ((i<n-1) and heap
  {
      choose an edge(u,v)
      Delete the edge(u,v) from the edge list E
      j = Find(u); k = Find(v)
```

```
    if (j≠k) then
    {
      i = i+1;
      t[i,1] = u; t[i,2] = v;
      mincost = mincost + cost[u,v];
      Union(j,k);
    }
  }
  if (i≠n-1) then write ("No spanning tree");
  else return mincost;
}
```

⇒ It can be observed that the algorithm sort the edges based on its weight.

⇒ Then the algorithm greedily selects the edges and adds each of them to the spanning tree if it does not form a cycle.

⇒ Cycle detection is done using the disjoint set structure. otherwise, the edge is discarded.

⇒ If a spanning tree has n-1 edges, then the algorithm terminates. otherwise, the algorithm reports an error message.

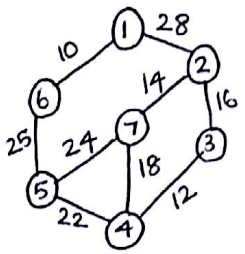
Complexity: Since the disjoint set data structure is used, initialization takes at most (|V|) time.

⇒ The time complexity of an algorithm depends on the number of edges. As there are(|E|) edges, O(|E| |E|) time is required to sort these edges.

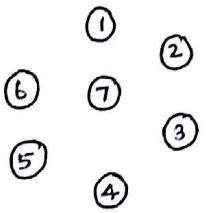
⇒ The disjoint set takes at most 2|E| find operations and |V|-1 operations.

∴ The total complexity of kruskal's algorithm is at most O(|E|log|E|) time.

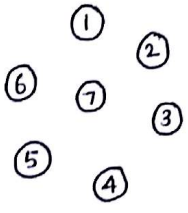
Problem Solution:



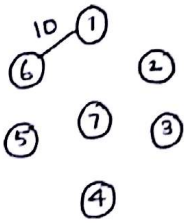
Step(1):



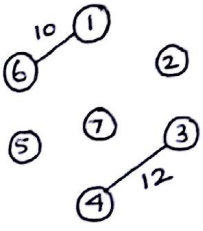
Step(2)



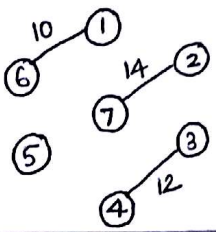
Step(3)



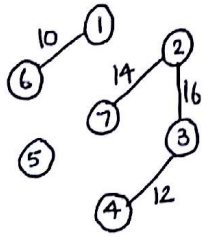
Step(4)



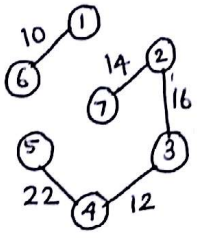
Step(5)



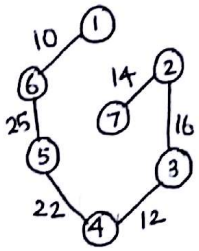
Step(6)



Step(7)



Step(8)



∴ Minimum cost

= 10+12+14+16+22+25

= 99

④ Explain about dynamic 0/1 - knapsack problem with suitable algorithm. Solve the 0/1 knapsack instance $n=3$, $(w_1, w_2, w_3) = (1, 2, 4)$ and $(P_1, P_2, P_3) = (1, 6, 4)$ with capacity $m=3$.

Answer:

⇒ Dynamic programming is useful in solving 0/1 knapsack problem where either the item is selected or rejected subject to the constraints of knapsack capacity.

⇒ Let us assume there are 'n' items with weights w_i and knapsack of capacity W. The knapsack problem can be divided into a number of subproblems using the items.

⇒ At every stage, the state represents the remaining capacity of the knapsack that can be loaded. the decision represents how many times the given item can be loaded.

⇒ The dynamic programming approach constructs a table $V[0 \cdots n, 0 \cdots w]$ for $1 \le i \le n$ and $0 \le j \le w$. As soon as the optimal decisions are taken at every stage, the decisions are updated in the table.

⇒ Thus, each entry of the table $V[i, j]$ stores the maximum value of items that can be filled into the knapsack of weight j.

⇒ Index i represents the items and j the weights of item i. value of the items are subjected to the constraints of knapsack capacity.

⇒ Finally, after filling the table, $V[n, w]$ is the maximum value of all the items that fit into the knapsack of capacity W.

Two choices:

1. If $x_i = 0$, then the item i is left out. In this case, the knapsack has items $(x_1, x_2, \cdots x_{i-1})$. the profit would be the profit of these items - $V[i-1, j]$.
2. If $x_i = 1$, then put the object into the knapsack. This implies that the number of items in the knapsack is in the range $(x_1, x_2, \cdots x_i)$ with profit $v_i + V[i-1, j-w_i]$.

With all these conditions, the recursive formulation of knapsack Problem is given as,

$$V[i,j] = \begin{cases} -\infty & \text{if } j < 0 \\ 0 & \text{if } j \ge 0 \\ V[i-1, j] & \text{if } w_i > j \\ \text{Max}\{V[i-1, j], v_i + V[i-1, j-w_i]\} & \text{if } w_i < j \end{cases}$$

Complexity analysis: Let n - no of items, W - Capacity of knapsack.

∴ Complexity of the algorithm is $O(nW)$.

Algorithm 0/1-knapsack (n, w, v, W)

I/P: n items with knapsack weight w, value v, and capacity of knapsack W

O/P: Maximum value of items that fit into the knapsack.

```
Begin
    for w = 0 to m do
        V[0,w] = 0
    End for
    for i = 0 to n
        V[i,0] = 0
    End for
```

→

```
for w = 0 to m do
```

$$V[i,j] = \begin{cases} -\infty & \text{if } j<0 \\ 0 & \text{if } j \geq 0 \\ V[i-1,j] & \text{if } w_i > j \\ \max\{V[i-1,j], v_i + V[i-1, j-w_i]\} & \text{if } w_i \leq j \end{cases}$$

```
End for
Return (V[n,W])
End.
```

Problem-solution: Given, Maximum knapsack capacity m = 3

Initial knapsack table

Item	weight	value
1	1	1
2	2	6
3	4	4

Step(1) Initial table V[i, j] where i - items, j - weight of items

i \ j	0	1	2	3
0	0	0	0	0
1	0			
2	0			
3	0			

Base condition: V[0, j] = 0 for j ≥ 0

V[i, 0] = 0 for i ≥ 0

Step(2) i = 1 and j = 1, 2, 3

$V[1,1] = \max\{V[1-1,1], 1+V[0,0]\}$

$= \max\{V[0,1], 1+0\}$

$= \max\{0,1\}$

$V[1,1] = 1$

$V[1,2] = \max\{V[0,1], 1+V[0,1]\} = 1$

$V[1,3] = \max\{V[0,1], 1+V[0,2]\} = 1$

After first item is loaded.

i \ j	0	1	2	3
0	0	0	0	0
1	0	1	1	1
2	0			
3	0			

Step(3) i = 2 and j = 1, 2 and 3

$V[2,1] = 1$

$V[2,2] = \max\{V[1,1], 6+V[1,0]\} = 6$

$V[2,3] = \max\{V[1,1], 6+V[1,1]\} = 7$

i \ j	0	1	2	3
0	0	0	0	0
1	0	1	1	1
2	0	1	6	7
3	0			

Step(4) i = 3 and j = 1, 2 and 3

$V[3,1] = V[2,1] = 1$

$V[3,2] = V[2,2] = 6$

$V[3,3] = V[2,3] = 7$

i \ j	0	1	2	3
0	0	0	0	0
1	0	1	1	1
2	0	1	6	7
3	0	1	6	7

⇒ After third item is loaded.

It can be observed that the maximum value is 7.

Here, 1 ⇒ Item is included

0 ⇒ Item is not picked up.

Check the condition $[n-1, W-W_i]$

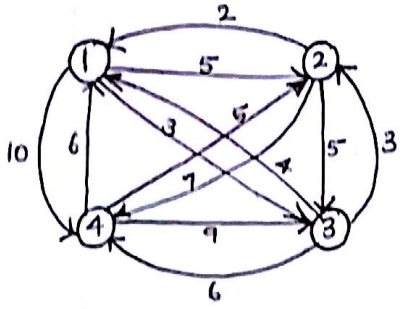
$V[3,3] = V[2,3]$ ⇒ Item 3 is not picked up.

$V[2,3] \neq V[1,2]$ ⇒ Item 2 is picked up.

$V[1,1] \neq V[0,1]$ ⇒ Item 1 is picked up.

Thus, the final result of this problem is the item 1 and 2 are picked up with a maximum profit of 3.

(5) Describe the travelling salesperson algorithm using dynamic programming approach and solve the TSP for the following graph.



Answer:

Dynamic programming is a popular approach for solving the TSP. TSP is given as graph $G=(V,E)$ with weighted adjacency matrix A.

Let us assume that Vertex 1 is the starting node of the TSP tour. The problem involves computation of the minimum-cost path that starting from Vertex 1 visits all other vertices exactly once.

Let us assume that the cost of such path is cost(i). The cost of the TSP cycle is cost(i) + d(i), where d(i) is the distance from vertex i to the starting node.

The informal travelling salesperson algorithm is given as follows,

Step 1: Read weighted graph $G=\langle V,E\rangle$

Step 2: Initialize d[i,j] as follows,

$$d[i,j]=\begin{cases}\infty \text{ if } edge(i,j)\notin E(G)\\ 0 \text{ if } i=j\\ w_{ij} \text{ if } edge(i,j)\in E(G)\end{cases}$$

Step 3: Compute a function g(i,s), a function that gives the length of the shortest path starting from a vertex i, travelling through all the vertices in set S, and terminating at vertex 1 as follows,

3a: $Cost(i,\phi) = d[i,1]$, $1\le i\le n$

3b: For $|S|=2$ to $n-1$, where $i\neq 1$, $1\notin S$, $i\notin S$, compute recursively

$$Cost(S,i)=\min_{j\in S}\{cost[i,j], S-\{j\}\}$$ and store the value.

Step 4: Compute the minimum cost of the travelling salesperson tour as follows,

Compute $cost(1, v-\{1\}) = \min_{2\le i\le k}\{d[1,k] + cost[k, v-\{1,k\}]\}$ using $g(i,S)$ computed in step 3.

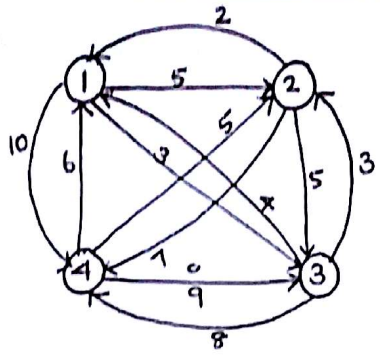
Step 5: Return the value of step 4.

Step 6: End.

Complexity Analysis.

Complexity analysis of travelling salesman algorithm is $\theta(n^2 2^n)$

Problem Solution:



⇒

Step(1) Find distance matrix d[i,j]

$$\begin{bmatrix} 0 & 5 & 3 & 10 \\ 2 & 0 & 5 & 7 \\ 4 & 3 & 0 & 8 \\ 6 & 5 & 9 & 0 \end{bmatrix}$$

Step(2): Compute cost (S,i) function recursively

2a: When $|S| = \phi$ vertices 2,3,4 would be computed directly using base condition.

$$cost(i, \phi) = d[i,1]$$

$cost(2,\phi) = d[2,1] = 2$; $cost(3,\phi) = d[3,1] = 4$; $cost(4,\phi) = d[4,1] = 6$.

2b: When $|S| = 1$, $cost(i, V-\{1\}) = \min_{2 \le i \le k} \{ d[i,k] + cost(k, V-\{1,k\}) \}$

$cost(2,\{3\}) = d[2,3] + cost(3,\phi)$
$= 5+4 = 9$

$cost(3,\{2\}) = d[3,2] + cost(2,\phi)$
$= 3+2 = 5$

$cost(4,\{2\}) = d[4,2] + cost(2,\phi)$
$= 5+2 = 7$

$cost(2,\{4\}) = d[2,4] + cost(4,\phi)$
$= 7+6 = 13$

$cost(3,\{4\}) = d[3,4] + cost(4,\phi)$
$= 8+6 = 14$

$cost(4,\{3\}) = d[4,3] + cost(3,\phi)$
$= 9+4 = 13$

2c: When $|S| = 2$, set involves two intermediate nodes.

$cost(2,\{3,4\}) = \min \{ d[2,3] + cost(3,\{4\}), d[2,4] + cost(4,\{3\}) \}$
$= \min \{5+14, 7+13\} = \min\{19, 20\} = 19$

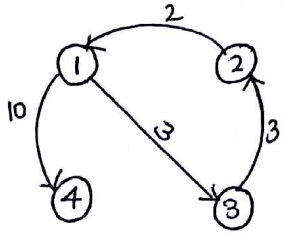
$cost(3,\{2,4\}) = \min \{ d[3,2] + cost(2,\{4\}), d[3,4] + cost(4,\{2\}) \}$
$= \min \{3+13, 8+7\} = \min\{16, 15\} = 15$

$cost(4,\{2,3\}) = \min \{ d[4,2] + cost(2,\{3\}), d[4,3] + cost(3,\{2\}) \}$
$= \min \{5+9, 9+5\} = \min\{14, 14\} = 14$

2d: When $|S| = 3$, set involves three intermediate nodes.

$cost(1,\{2,3,4\}) = \min \{ d[1,2] + cost(2,\{3,4\}),$
$d[1,3] + cost(3,\{2,4\}),$
$d[1,4] + cost(4,\{2,3\}) \}$
$= \min \{5+19, 3+15, 10+14\}$
$= \min \{24, 18, 24\}$
$= 18$

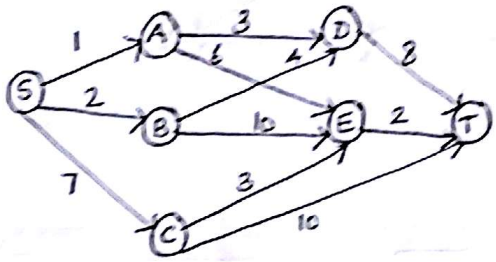
∴ Minimum cost of tour = 18



path of the tour is,

$1 \to 3 \to 2 \to 1$

(6) Explain about the multistage graph and find the shortest path between S and T using forward and backward approach.



Answer:

Multistage Graph

⇒ A multistage graph $G=(V,E)$ is a directed graph in which the vertices are partitioned into $k \geq 2$ disjoint sets V_i, $1 \leq i \leq k$.

⇒ If $\langle u,v \rangle$ is an edge in E, then $u \in V_i$ and $v \in V_{i+1}$ for some i, $1 \leq i \leq k$.

⇒ Let S and T respectively be the vertices in V_1 and V_k. Vertex S is source and T is sink.

⇒ Let $C[i,j]$ be the cost of edge $\langle i,j \rangle$. The cost of a path from S to T is the sum of the cost of the edges on the path.

⇒ The multistage graph problem is to find the minimum cost path from S to T. Each set V_i defines the stages in the graph.

⇒ Because of the constraints on E, every path from S to T, starts in stage 1, goes to stage 2, then to stage 3 and so on and eventually terminates in stage k.

Mathematical formulation:

Forward Approach ⇒ $Cost(i,j) = \min_{l \in V_{i+1}} \{ C(i,l) + cost(i+1,l) \}$, $\langle j,l \rangle \in E$

Backward Approach ⇒ $bcost(i,j) = \min_{l \in V_{r-1}} \{ bcost(i-1,l) + C(l,j) \}$, $\langle l,j \rangle \in E$.

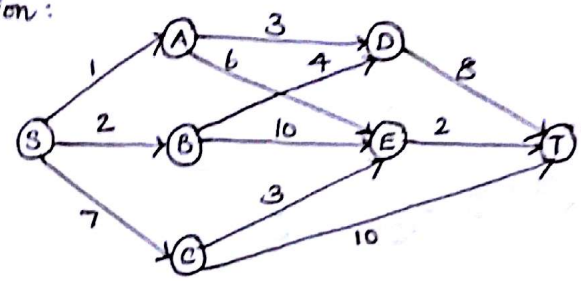
Algorithm: Forward Approach (Backward Reasoning)

```
Algorithm FGraph(G,k,n,P)
{
  cost[n]=0.0;
  for j=n-1 to 1 do
  { // compute cost[j]
    let r be the vertex such that <j,r> is an
    edge of G and C[j,r]+cost[r] is minimum;
    cost[j] = C[j,r] + cost[r];
      d[j]=r;
  }
  // Find a minimum-cost path
    P[1]=1; P[k]=n;
  for j=2 to k-1 do
      P[j]= d[P[j-1]];
}
```

Backward Approach (Forward Reasoning)

```
Algorithm BGraph(G,k,n,P)
{
   bcost[1]=0.0;
  for j=2 to n do
  { // compute bcost[j]
  Let r be the vertex such that <r,j> is an
  edge of G and bcost[r]+C[r,j] is minimum;
    bcost[j] = bcost[r]+C[r,j];
      d[j]=r;
  }
  // Find a minimum-cost path
    P[1]=1; P[k]=n;
  for j=k-1 to 2 do
      P[j]=d[P[j+1]];
}
```

Problem - Solution :



Forward approach (Backward Reasoning)

From the graph shortest path between S & T is,

$$cost(S,T) = \min\{(1+cost(A,T)),\ (2+cost(B,T)),\ (7+cost(C,T))\}$$

$$cost(A,T) = \min\{3+cost(D,T),\ 6+d(E,T)\}$$

$$= \min\{3+8,\ 6+2\}$$

$$= \min\{11, 8\}$$

$$\boxed{cost(A,T) = 8}$$

$$Cost(B,T) = \min\{4+cost(D,T),\ 10+cost(E,T)\}$$

$$= \min\{4+8,\ 10+2\}$$

$$= \min\{12, 12\}$$

$$\boxed{cost(B,T) = 12}$$

$$Cost(C,T) = \min\{3+cost(E,T),\ cost(C,T)\}$$

$$= \min\{3+2, 10\}$$

$$= \min\{5, 10\}$$

$$\boxed{cost(C,T) = 5}$$

$$\therefore Cost(S,T) = \min\{1+cost(A,T),\ 2+cost(B,T),\ 7+cost(C,T)\}$$

$$= \min\{1+8,\ 2+12,\ 7+5\}$$

$$= \min\{9, 14, 12\}$$

$$\boxed{cost(S,T) = 9}$$

∴ The shortest path between S and T is,

$$S \rightarrow A \rightarrow E \rightarrow T$$

Minimum cost = 9

Backward approach (Forward Reasoning)

Initially, From stage1 to stage2

bcost (S,A) = 1, bcost(S,B) = 2 and

bcost (S,C) = 7

From stage 1 to stage 3 through stage 2

$$bcost(S,D) = \min\{1+bcost(A,D),\ 2+bcost(B,D)\}$$

$$= \min\{1+3,\ 2+4\}$$

$$= \min\{4, 6\}$$

$$\boxed{bcost(S,D) = 4}$$

$$bcost(S,E) = \min\{1+bcost(A,E),\ 2+bcost(B,E),\ 7+bcost(C,E)\}$$

$$= \min\{1+6,\ 2+10,\ 7+3\}$$

$$= \min\{7, 12, 10\}$$

$$\boxed{bcost(S,E) = 7}$$

From stage1 to stag4 through stage 2&3

$$bcost(S,T) = \min\{bcost(S,D)+bcost(D,T),\ bcost(S,E)+bcost(E,T),\ bcost(S,C)+bcost(C,T)\}$$

$$= \min\{4+8,\ 7+2,\ 7+10\}$$

$$= \min\{12, 9, 17\}$$

$$\boxed{bcost(S,T) = 9}$$

∴ The shortest path between S and T is,

$$S \rightarrow A \rightarrow E \rightarrow T$$

Minimum Cost = 9