DAA– Question Bank

Unit - 3

Q.No Questions Mark 1. Define 'Greedy algorithm'? Explain the general method of Greedy Method. 7

Answer:

**Greedy algorithm:**

The greedy method is one of the strategies like Divide and conquer used to solve the problems. This method is used for solving optimization problems. An optimization problem is a problem that demands either maximum or minimum results..

**General method**

The Greedy method is the simplest and straightforward approach. It is not an algorithm, but it is a technique. The main function of this approach is that the decision is taken on the basis of the currently available information. Whatever the current information is present, the decision is made without worrying about the effect of the current decision in future..

This technique is basically used to determine the feasible solution that may or may not be optimal.

Characteristics of Greedy method

To construct the solution in an optimal way, this algorithm creates two sets where:

1)one set contains all the chosen items, and another set contains the rejected items.

2)A Greedy algorithm makes good local choices in the hope that the solution should be

either feasible or optimal.

**The components that can be used in the greedy algorithm are:**

* **Candidate set:** A solution that is created from the set is known as a candidate set.
* **Selection function:** This function is used to choose the candidate or subset which can be added in the solution.
* **Feasibility function:** A function that is used to determine whether the candidate or subset can be used to contribute to the solution or not.

**ALGORITHM:**

|  |
| --- |
| 1. Algorithm Greedy (a, n) 2. { 3. Solution : = 0; 4. for i = 0 to n do 5. { 6. x: = select(a); 7. if feasible(solution, x) 8. { 9. Solution: = union(solution , x) 10. } 11. return solution; 12. } } |

DAA– Question Banks

2. Write an algorithm knapsack problem .Give example 14

The fractional knapsack problem is also one of the techniques which are used to solve the knapsack problem. In fractional knapsack, the items are broken in order to maximize the profit. The problem in which we break the item is known as a Fractional knapsack problem.

**This problem can be solved with the help of using two techniques:**

* Brute-force approach: The brute-force approach tries all the possible solutions with all the different fractions but it is a time-consuming approach.
* Greedy approach: In Greedy approach, we calculate the ratio of profit/weight, and accordingly, we will select the item. The item with the highest ratio would be selected first.

**There are basically three approaches to solve the problem:**

* The first approach is to select the item based on the maximum profit.
* The second approach is to select the item based on the minimum weight.
* The third approach is to calculate the ratio of profit/weight.

Let us go with third approach;

Third approach:

**In the third approach, we will calculate the ratio of profit/weight.**

Objects:         1     2     3     4     5     6     7

Profit (P):         5     10     15     7     8     9     4

Weight(w):       1     3     5    4     1     3     2

In this case, we first calculate the profit/weight ratio.

Object 1: 5/1 = 5

Object 2: 10/3 = 3. 33

Object 3: 15/5 = 3

Object 4: 7/4 = 1.7

Object 5: 8/1 = 8

Object 6: 9/3 = 3

**Object 7: 4/2 = 2**

Objects:         1     2     3     4     5     6     7

Profit (P):         5    10     15     7     8     9     4

Weight(w):       1     3     5    4     1     3     2

**P:w:         5     3.3     3     1.7     8     3     2**

In this approach, we will select the objects based on the maximum profit/weight ratio. Here 8 is the max p:w ration .Since the P/W of object 5 is maximum so we select object 5.

|  |  |  |  |
| --- | --- | --- | --- |
| Object | Profit | Weight | Remaining weight |
| 5 | 8 | 1 | 15 - 8 = 7 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

After object 5, object 1 has the maximum profit/weight ratio, i.e., 5. So, we select object 1 shown in the below table:

|  |  |  |  |
| --- | --- | --- | --- |
| Object | Profit | Weight | Remaining weight |
| 5 | 8 | 1 | 15 - 1 = 14 |
| 1 | 5 | 1 | 14 - 1 = 13 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

After object 1, object 2 has the maximum profit/weight ratio, i.e., 3.3. So, we select object 2 having profit/weight ratio as 3.3.

|  |  |  |  |
| --- | --- | --- | --- |
| Object | Profit | Weight | Remaining weight |
| 5 | 8 | 1 | 15 - 1 = 14 |
| 1 | 5 | 1 | 14 - 1 = 13 |
| 2 | 10 | 3 | 13 - 3 = 10 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

After object 2, object 3 has the maximum profit/weight ratio, i.e., 3. So, we select object 3 having profit/weight ratio as 3.

|  |  |  |  |
| --- | --- | --- | --- |
| Object | Profit | Weight | Remaining weight |
| 5 | 8 | 1 | 15 - 1 = 14 |
| 1 | 5 | 1 | 14 - 1 = 13 |
| 2 | 10 | 3 | 13 - 3 = 10 |
| 3 | 15 | 5 | 10 - 5 = 5 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Object | Profit | Weight | Remaining weight |
| 5 | 8 | 1 | 15 - 1 = 14 |
| 1 | 5 | 1 | 14 - 1 = 13 |
| 2 | 10 | 3 | 13 - 3 = 10 |
| 3 | 15 | 5 | 10 - 5 = 5 |
| 6 | 9 | 3 | 5 - 3 = 2 |
|  |  |  |  |
|  |  |  |  |

After object 3, object 6 has the maximum profit/weight ratio, i.e., 3. So we select object 6 having profit/weight ratio as 3.

After object 6, object 7 has the maximum profit/weight ratio, i.e., 2. So we select object 7 having profit/weight ratio as 2.

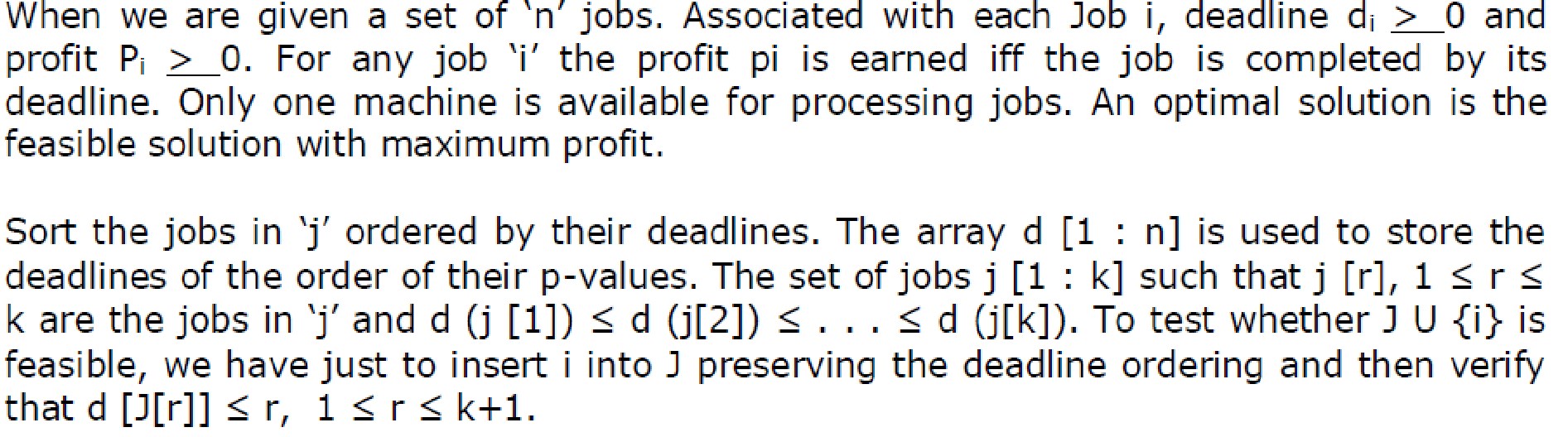
|  |  |  |  |
| --- | --- | --- | --- |
| Object | Profit | Weight | Remaining weight |
| 5 | 8 | 1 | 15 - 1 = 14 |
| 1 | 5 | 1 | 14 - 1 = 13 |
| 2 | 10 | 3 | 13 - 3 = 10 |
| 3 | 15 | 5 | 10 - 5 = 5 |
| 6 | 9 | 3 | 5 - 3 = 2 |
| 7 | 4 | 2 | 2 - 2 = 0 |
|  |  |  |  |

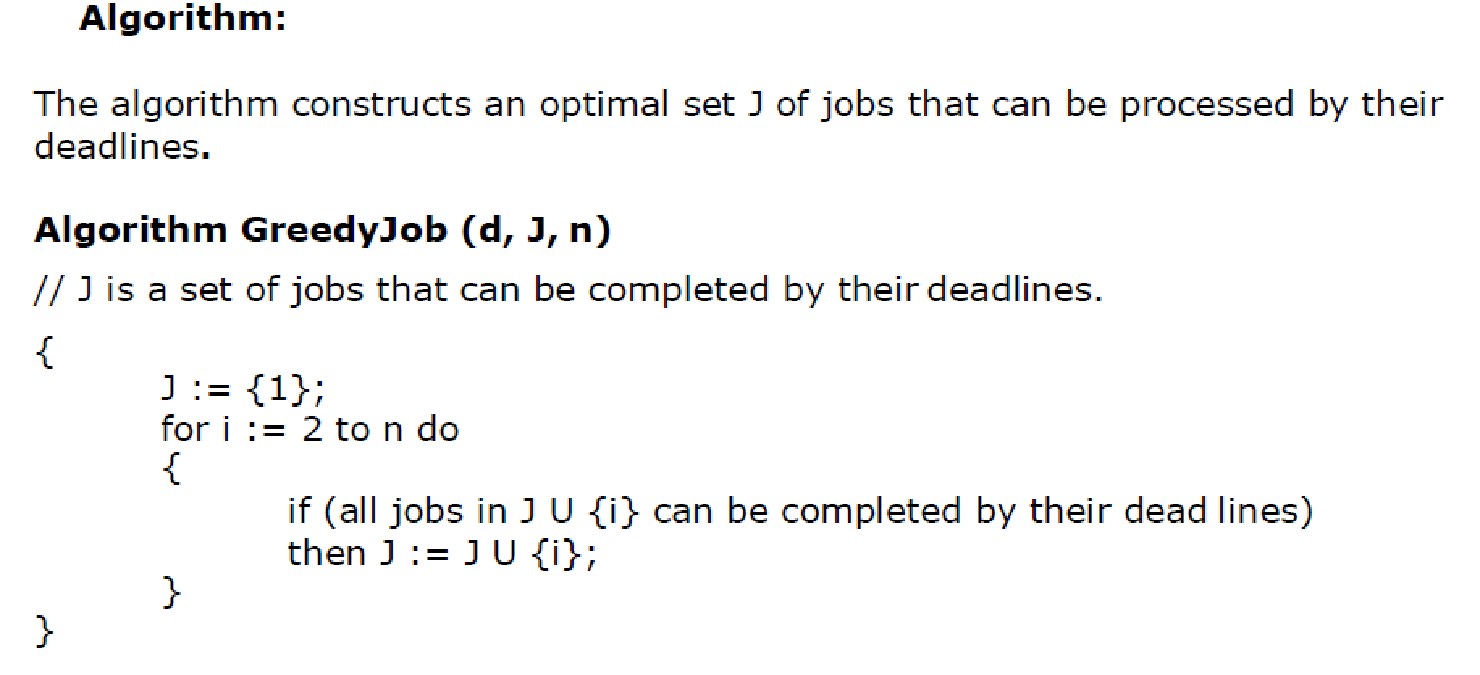
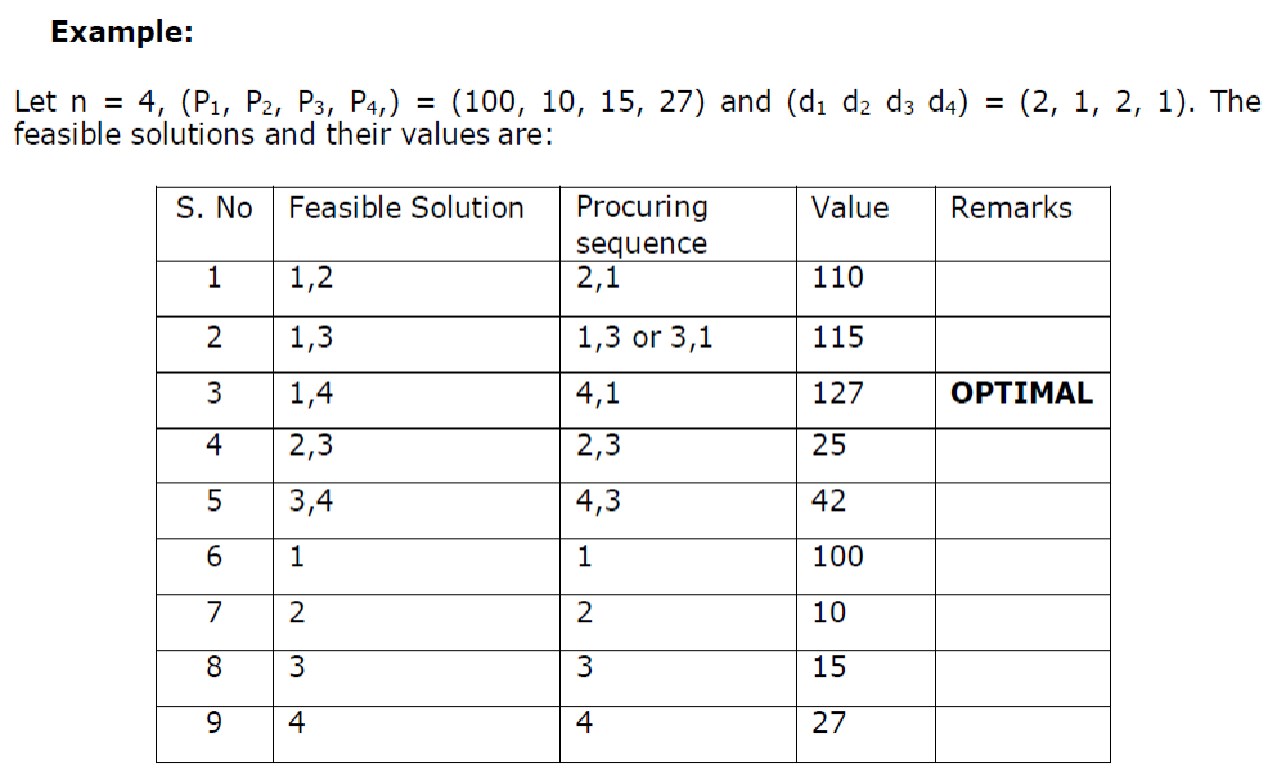
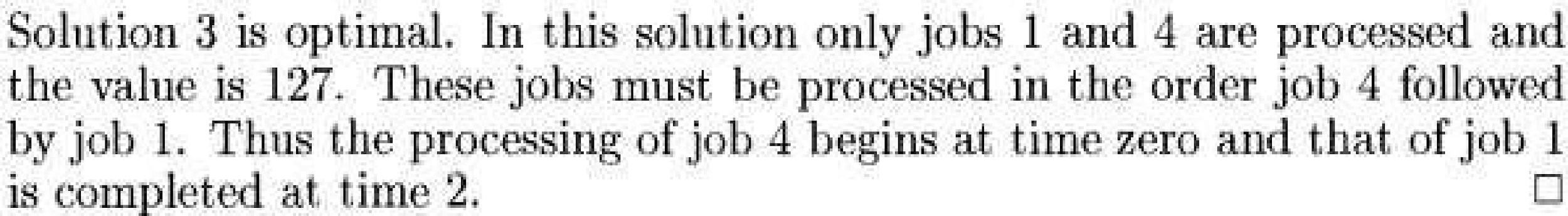
As we can observe in the above table that the remaining weight is zero which means that the knapsack is full. We cannot add more objects in the knapsack. Therefore, the total profit would be equal to (8 + 5 + 10 + 15 + 9 + 4), i.e., 51.

In the first approach, the maximum profit is 47.25. The maximum profit in the second approach is 46. The maximum profit in the third approach is 51. Therefore, we can say that the third approach, i.e., maximum profit/weight ratio is the best approach among all the approaches.

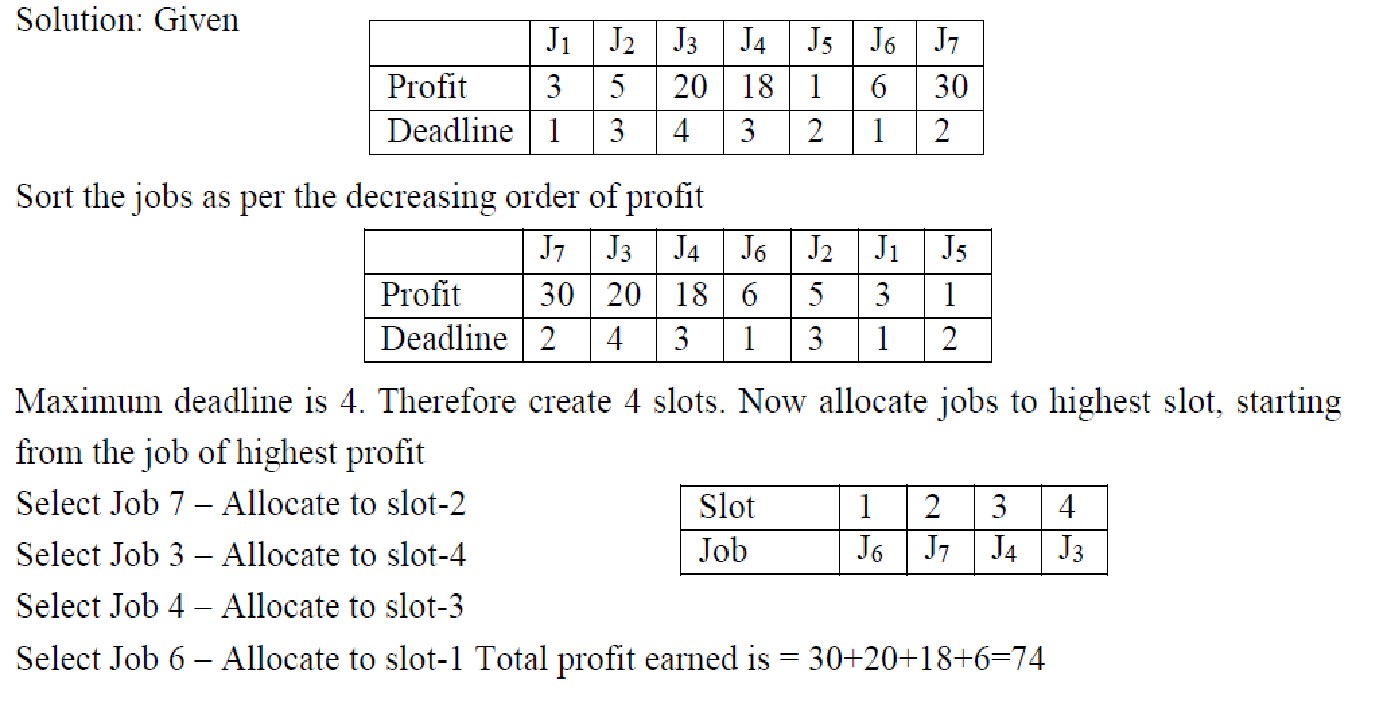
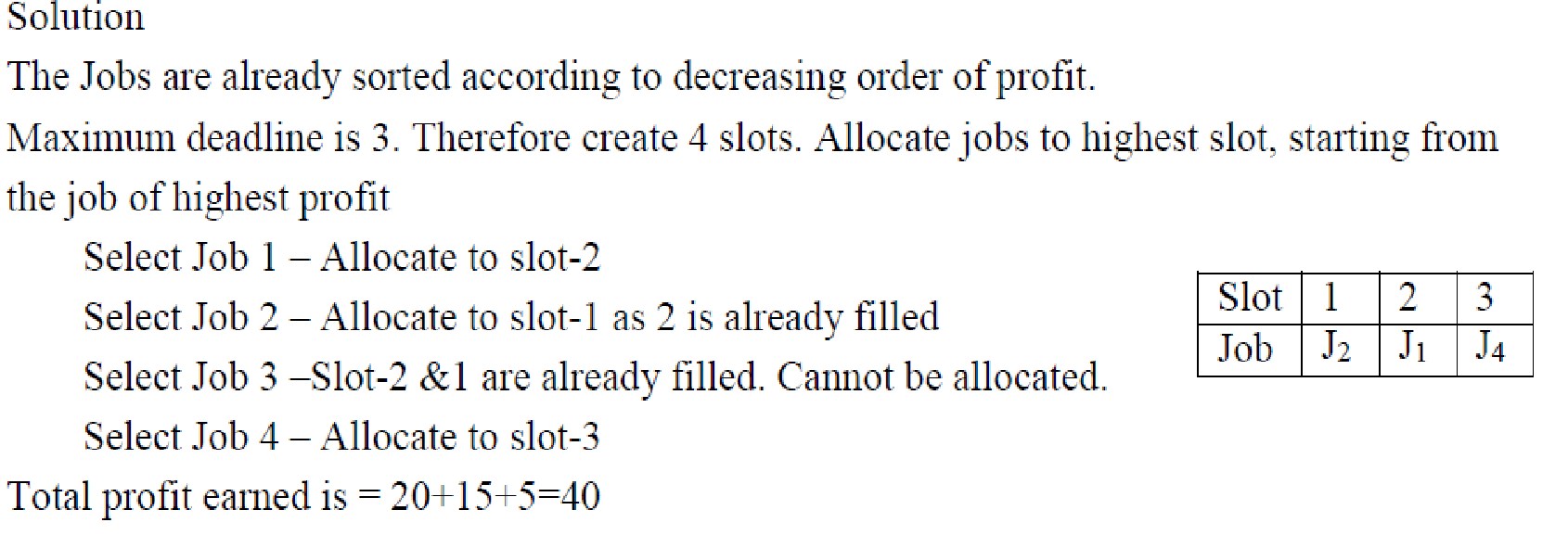
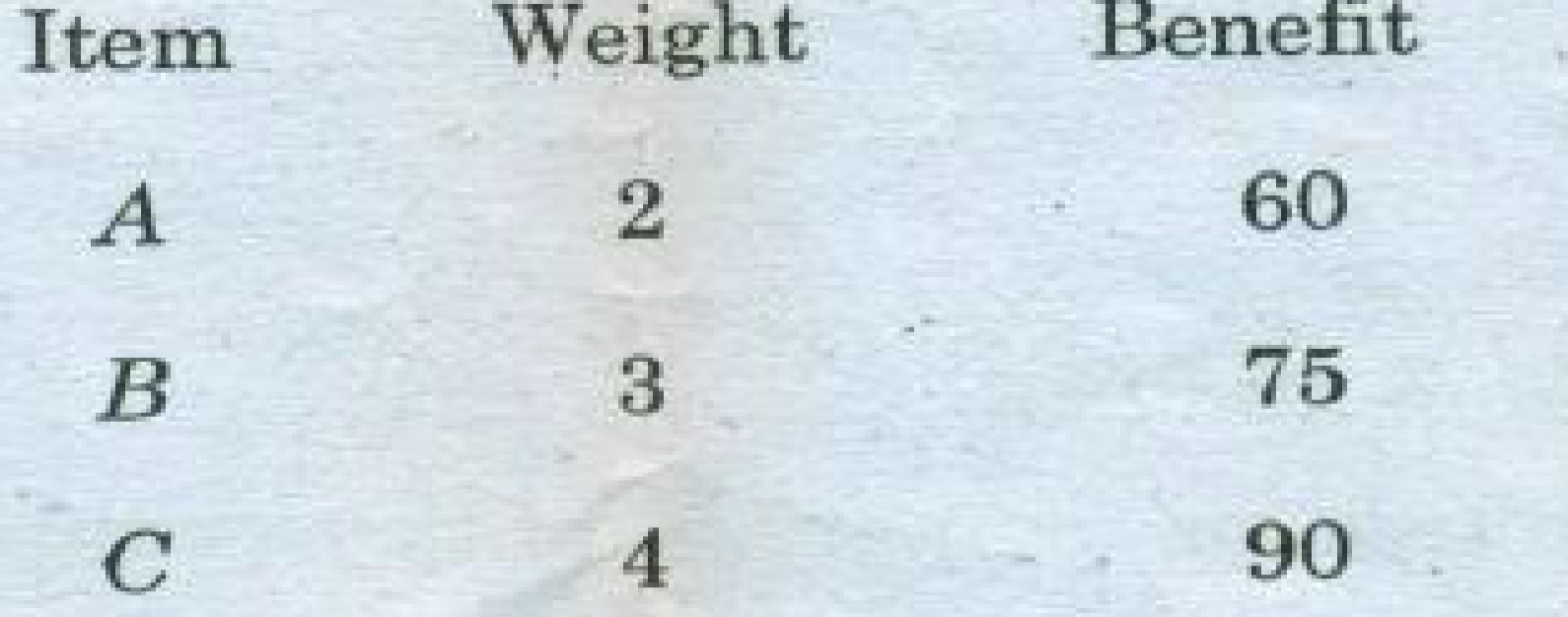
.

3. Explain in detail job sequencing with deadlines problem with an example

Answer;

DAA– Question Bank

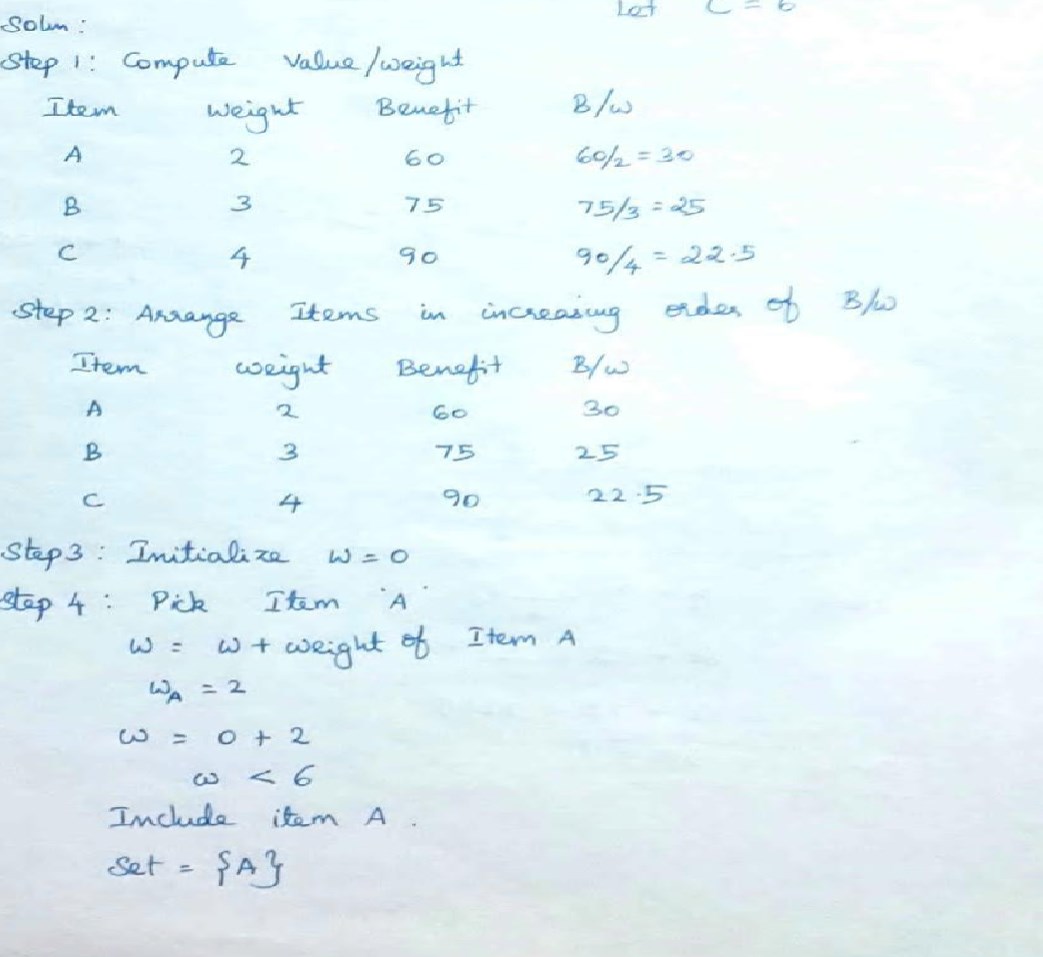
4. Find solution generated by job sequencing problem with deadlines for 7 jobs given profits 3, 7 5, 20, 18, 1, 6, 30 and deadlines 1, 3, 4, 3, 2, 1, 2 respectively.

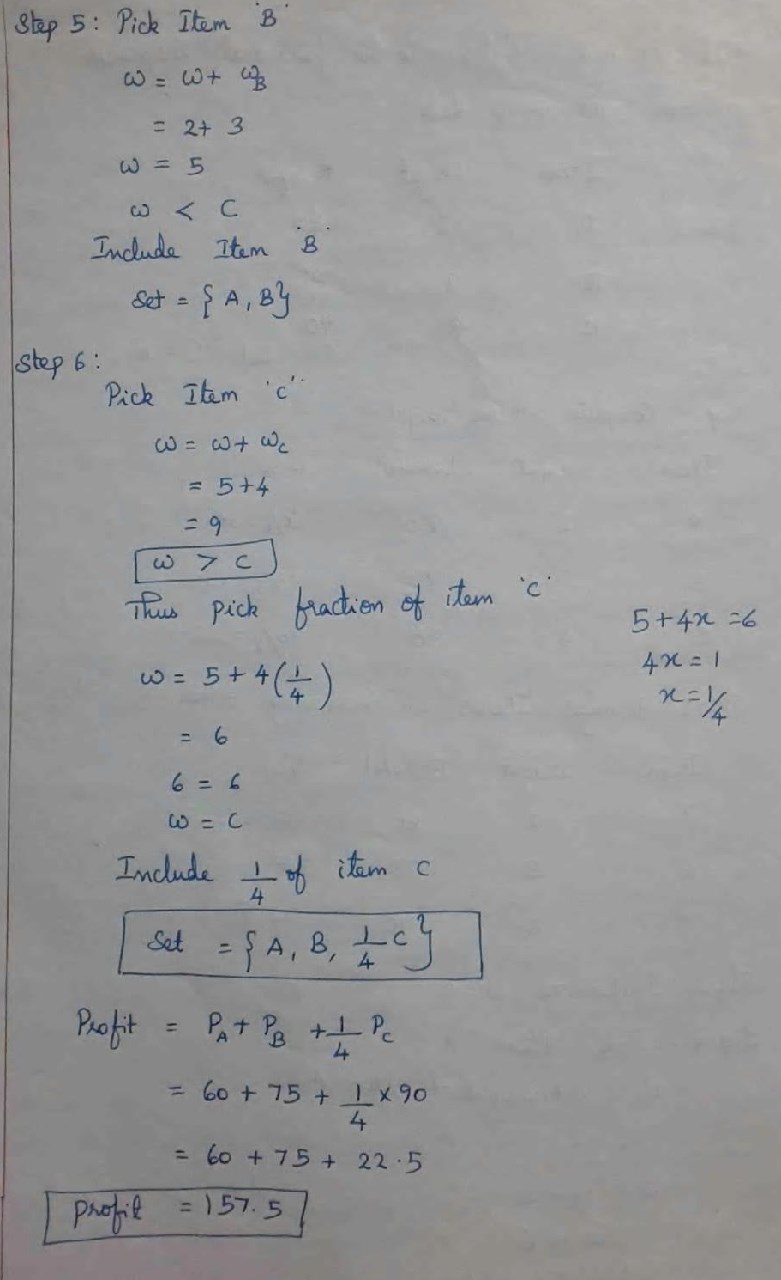
DAA– Question Bank

5. Find the solution generated by job sequencing when n = 5, (P1, P2, P3, P4, P5) 7 = (20, 15, 10, 5, 1), (d1, d2, d3, d4, d5) = (2, 2, 1, 3, 3)

6. Find the optimal solution to the fractional knapsack problem with given data; 4M

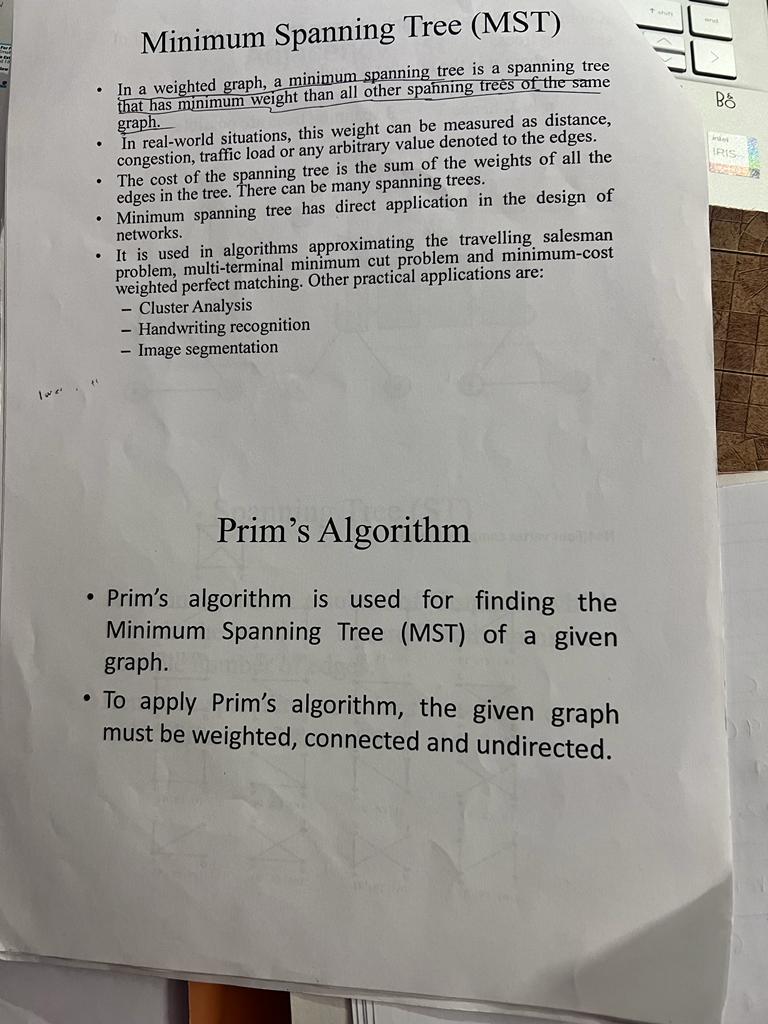
Capacity C=6

DAA– Question Bank

DAA– Question Bank

DAA– Question Bank

7. Write and Analyse Prim’s Algorithm 6 Answer:



## **Complexity of Prim's algorithm**

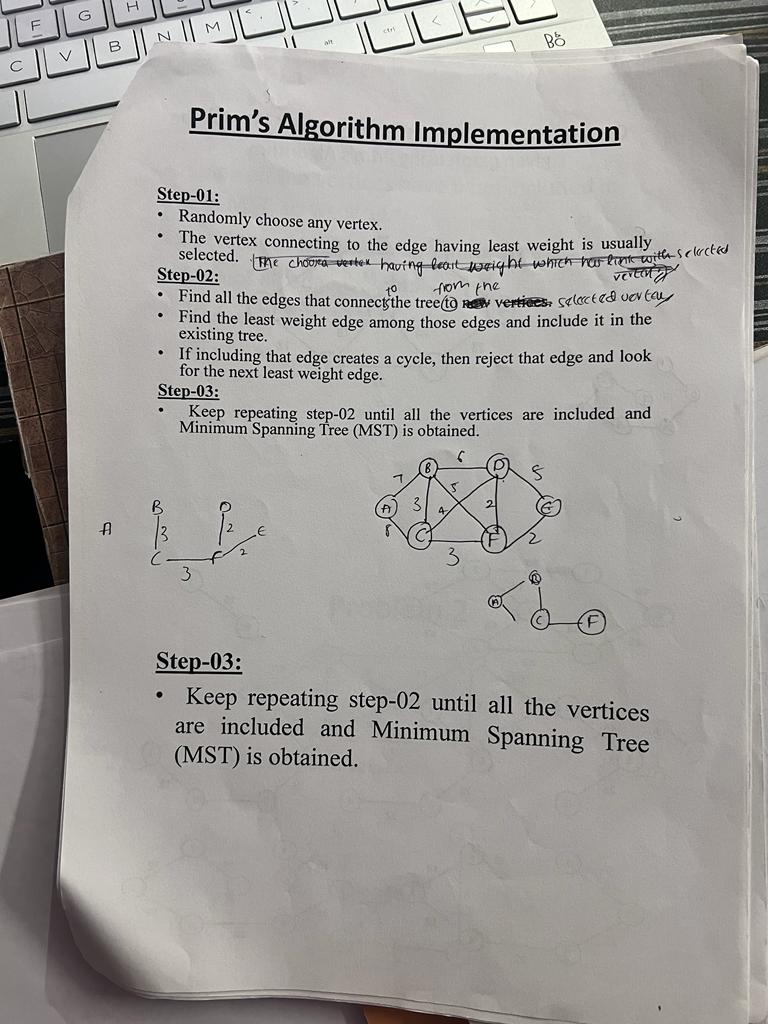
Now, let's see the time complexity of Prim's algorithm. The running time of the prim's algorithm depends upon using the data structure for the graph and the ordering of edges. Below table shows some choices -

* **Time Complexity**

|  |  |
| --- | --- |
| **Data structure used for the minimum edge weight** | **Time Complexity** |
| Adjacency matrix, linear searching | O(|V|2) |
| Adjacency list and binary heap | O(|E| log |V|) |
| Adjacency list and Fibonacci heap | O(|E|+ |V| log |V|) |

Prim's algorithm can be simply implemented by using the adjacency matrix or adjacency list graph representation, and to add the edge with the minimum weight requires the linearly searching of an array of weights. It requires O(|V|2) running time. It can be improved further by using the implementation of heap to find the minimum weight edges in the inner loop of the algorithm.

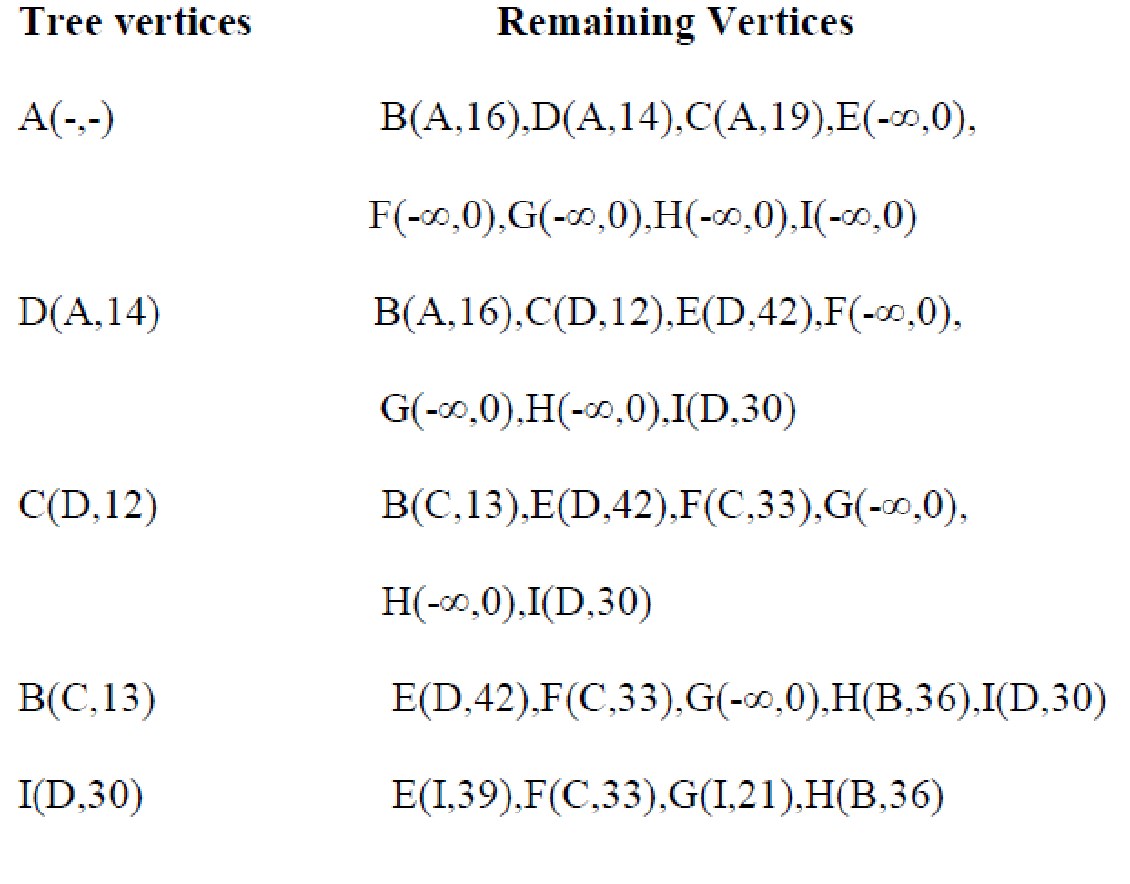
The time complexity of the prim's algorithm is O(E logV) or O(V logV), where E is the no. of edges, and V is the no. of vertices.

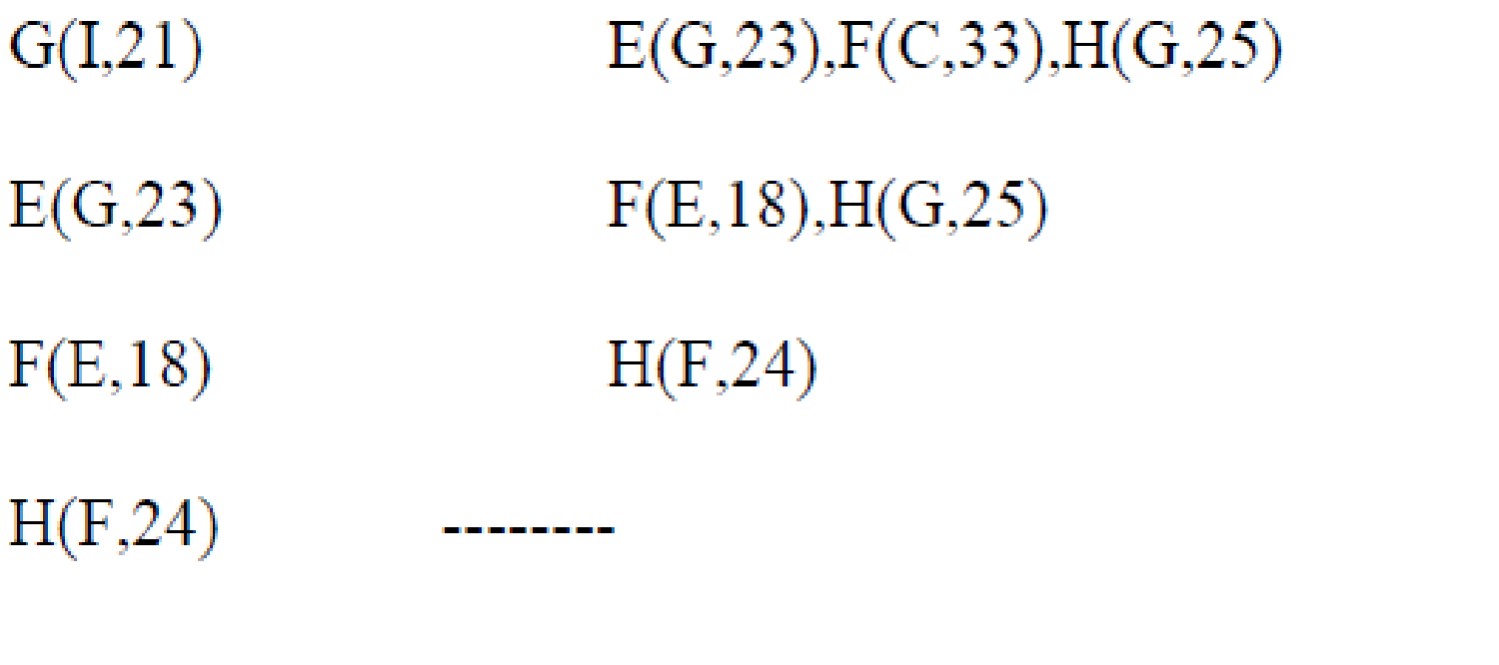
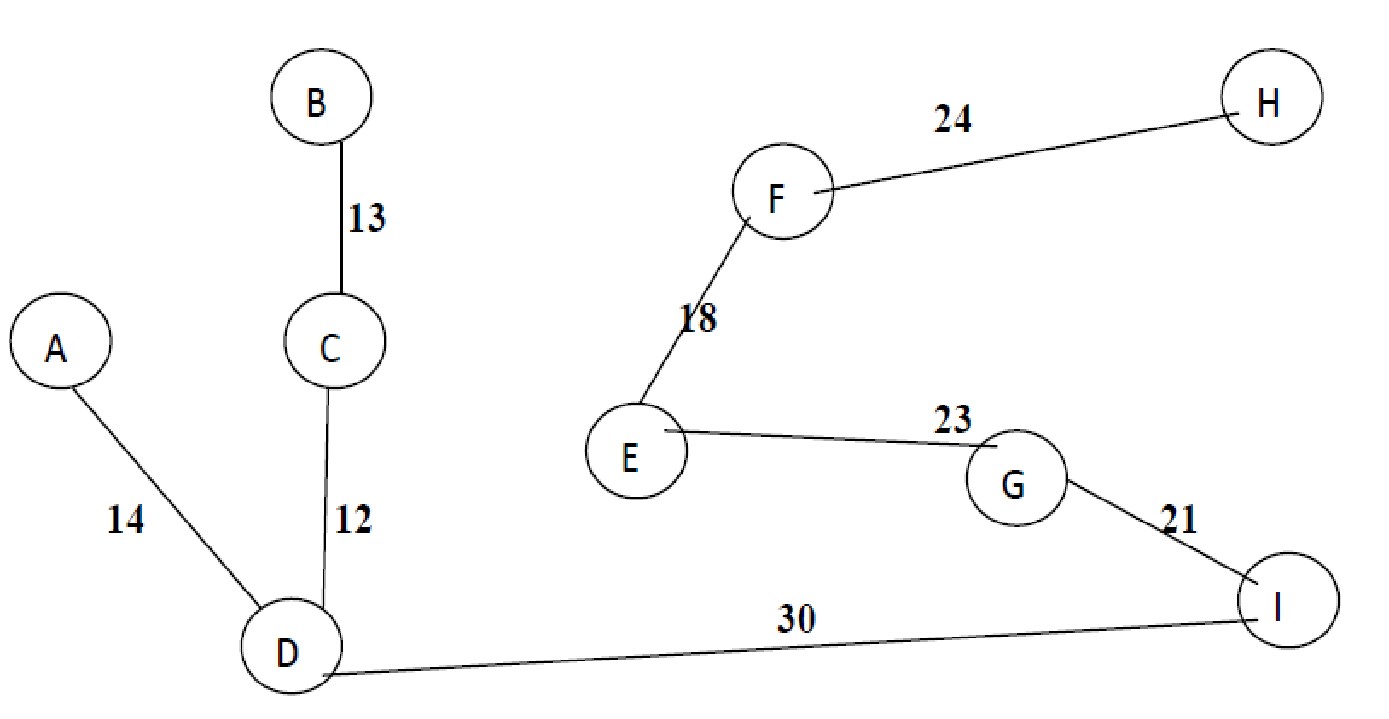
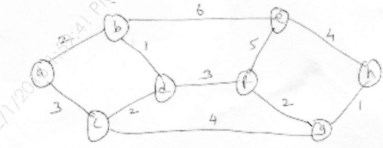


DAA– Question Bank

8. Consider the following weighted Graph: 8

Give the list of edges in the MST in the order that Prim’s Algorithm inserts them. Start Prim’s algorithm from Vertex A.

Answer;

DAA– Question Bank

**The minimum spanning tree for the given graph is:**

**Total cost of the minimum spanning tree=155**

9. Compare Prims and Kruskals method for finding Minimum spanning Tree find MST for 14 following using prims method

Answer:

**Prims** **Kruskalʼs**

This algorithm is for minimum spanning

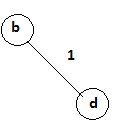
obtaining tree by

This algorithm is for obtaining minimum spanning tree but it is not necessary to



DAA– Question Bank

selecting the adjacent vertices of already selected vertices.



choose adjacent vertices of already selected vertices.

Primʼs algorithm initializes with a Kruskalʼs algorithm initiates with an edge node

Primʼs algorithms span from one node to another

Kruskalʼs algorithm will goes on w.r.t to least weight edges they might be adjacent nodes or not

In primʼs algorithm, graph must be a connected graph

Kruskalʼs can function on disconnected graphs too.

Primʼs algorithm has a time Kruskalʼs time complexity is O(logV). complexity of O(V2)

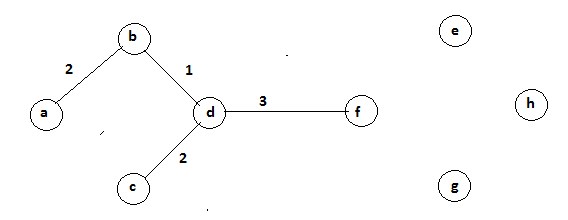
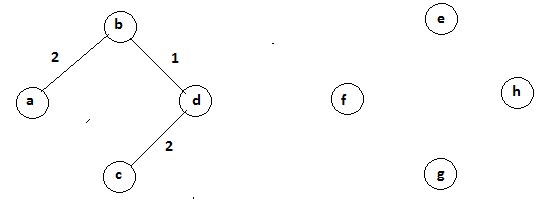
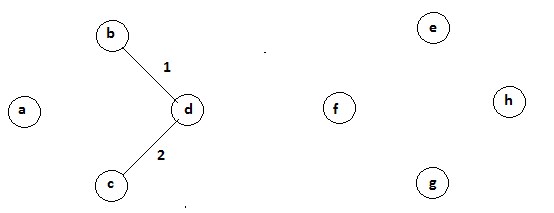
Problem: Step1:

We will first select a minimum distance edge from given graph.

V={b,d} Cost=1 Step 2:

The next minimum distance edge is d-c. This edge is adjacent to previously selected vertex d.

DAA– Question Bank



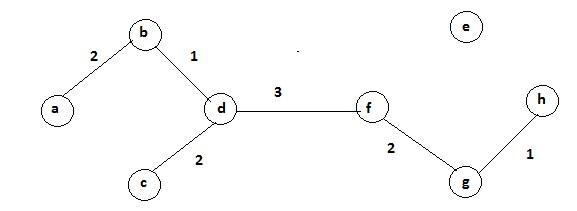
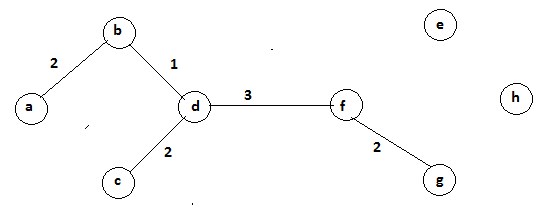
V={b,d,c} Cost=3 Step 3:

The next minimum distance edge is a-b. This edge is adjacent to previously selected vertex b.

V={b,d,c,a} Cost=5 Step 4:

The next minimum distance edge is d-f. This edge is adjacent to previously selected vertex d.

DAA– Question Bank V={ b,d,c,a ,f} Cost=8



Step 5:

The next minimum distance edge is f-g. This edge is adjacent to previously selected vertex f.

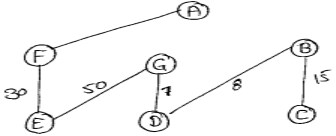
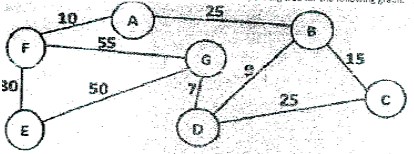
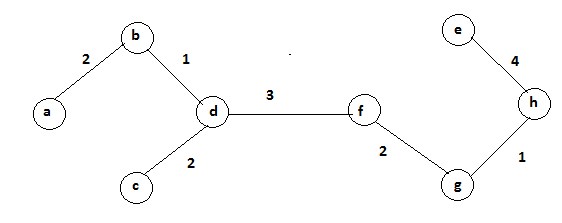
V={ b,d,c,a ,f ,g} Cost=10 Step 6:

The next minimum distance edge is g-h. This edge is adjacent to previously selected vertex g.

V={ b,d,c,a ,f ,g ,h} Cost=11 Step 7:

The next minimum distance edge is e-h. This edge is adjacent to previously selected vertex h.

DAA– Question Bank



V={ b,d,c,a ,f ,g ,h} Cost=15

Hence, the above is the minimum spanning tree with total cost 15.

10. Using Primʼs and Kruskalʼs Algorithm, find minimum spanning tree for following graph:

Answer;

**solving using Primʼs Algorithm:**

Step 1: draw the given graph.

DAA– Question Bank

Step2: remove all loops. Any edge that starts and ends at the same vertex is called a loop. In this case, there is no loop.

Step 3: remove all parallel edges two vertex except one with the least weight. In this case, there are no parallel edges.

Step 4: Create a table, where number of rows = number of columns = number of vertex in the graph.

Step 5: now, put 0 in cells having same row and column name.

Step 6: Now, Start filling other columns. Start with Vertex A. Find the edge that directly connects Vertex A and B. In this case, we have edge of weight 25 that directly connects A and B.

Step 7: put 25 in AB and BA.

Step 8: Repeat these steps for all the vertex that are directly connected.

Step 9: If any vertex is not connected directly, for eg: Vertex A and D; then put ∞(infinity symbol).

Here AD and DA will have ∞.

**-** **A** **B** **C** **D** **E** **F** **G**

A 0 25 ∞ ∞∞ ∞ 10 ∞

B 25 0 15 8 ∞ ∞ ∞

C ∞ 15 0 25 ∞ ∞ ∞

D ∞ 8 25 0 ∞ ∞ 7

E ∞ ∞ ∞ ∞ 0 30 50

F 10 ∞ ∞ ∞ 30 0 55

G ∞ ∞ ∞ 7 50 55 0

Table is now completely filled. Next Task is to find Minimum spanning Tree. Step 10: Start from vertex A. Find the smallest value in row A

DAA– Question Bank

Smallest Value in row A is 10. Mark AF and FA and draw the graph.

Smallest Value in row B is 8. Mark BD and DB and draw the graph. Smallest Value in row C is 15. Mark CB and BC and draw the graph. Similarly, do for the all the rows and you will get the following table.

**-** **A** **B** **C** **D** **E** **F** **G**

A 0 25 ∞ ∞∞ ∞ 10 ∞

B 25 0 15 8 ∞ ∞ ∞

C ∞ 15 0 25 ∞ ∞ ∞

D ∞ 8 25 0 ∞ ∞ 7

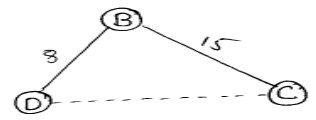
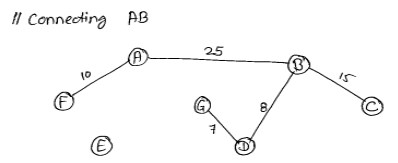
E ∞ ∞ ∞ ∞ 0 30 50

F 10 ∞ ∞ ∞ 30 0 55

G ∞ ∞ ∞ 7 50 55 0

(Note: we will not consider 0 as it will correspond to the same vertex.) So minimum spanning tree using primʼs algorithm is below:

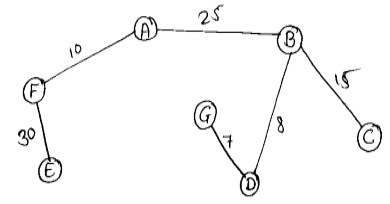
DAA– Question Bank



Now, if connect DC, so it will form a ckt like below:

Therefore, we will skip this edges and select next edge FE.

DAA– Question Bank



Since our minimum spanning tree should be of 6 edges, so we will stop here and this is our MST.

**solving using kruskalʼs algorithm:**

Step 1: Remove all the loops. Any edge that starts and ends at the same vertex is a loop. In our case, it does not exist.

Step 2: remove all parallel edges two vertex except one with the least weight. In this case, we donʼt have any parallel edges.

Step 3: Create the edge table. An edge table will have name of all the edges along with their weight in ascending order.

If you look at the graph, you will notice there are 9 edges in total, so our edge table will have 9 (nine) columns.

**Edge** **GD**

Weight 7

**BD** **AF** **BC** **AB** **DC** **FE** **EG** **FG**

8 10 15 25 25 30 50 55

In our case, AB and DC both edges have weight 25 so we will consider both. And you can write them in any order i.e AB first then DC or vice versa.

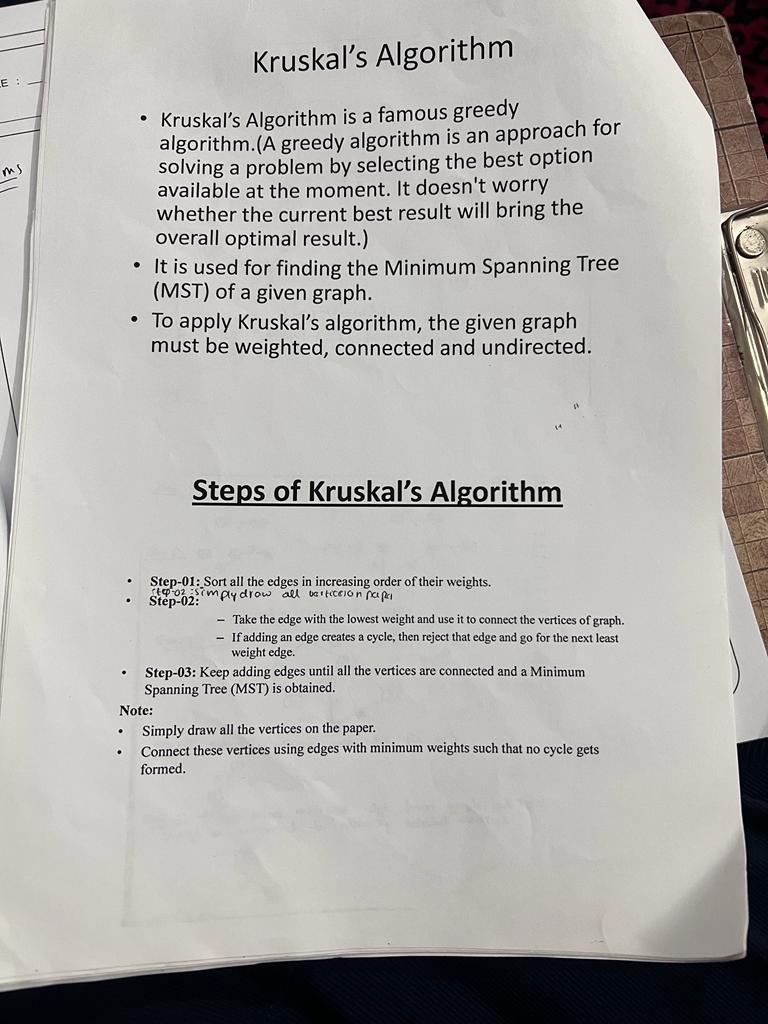
Step 4: To find minimum spanning tree, number of edges will be Number of edges=No. of vertices- 1

In our case, no. of vertices are 8, so our minimum spanning tree will have 7 edges.

Step 5: To find the MST, we will start with the smallest weight edge and keep selecting edges that does not form any circuit with the previously selected edges.

Since 7 is the smallest weight so we will select GD. Drawing this edge GD

11. **Write and analyse Kruskal’s Algorithm** 7 **Background:** Kruskal's algorithm is another greedy algorithm for the minimum spanning tree problem that also always yields an optimal solution

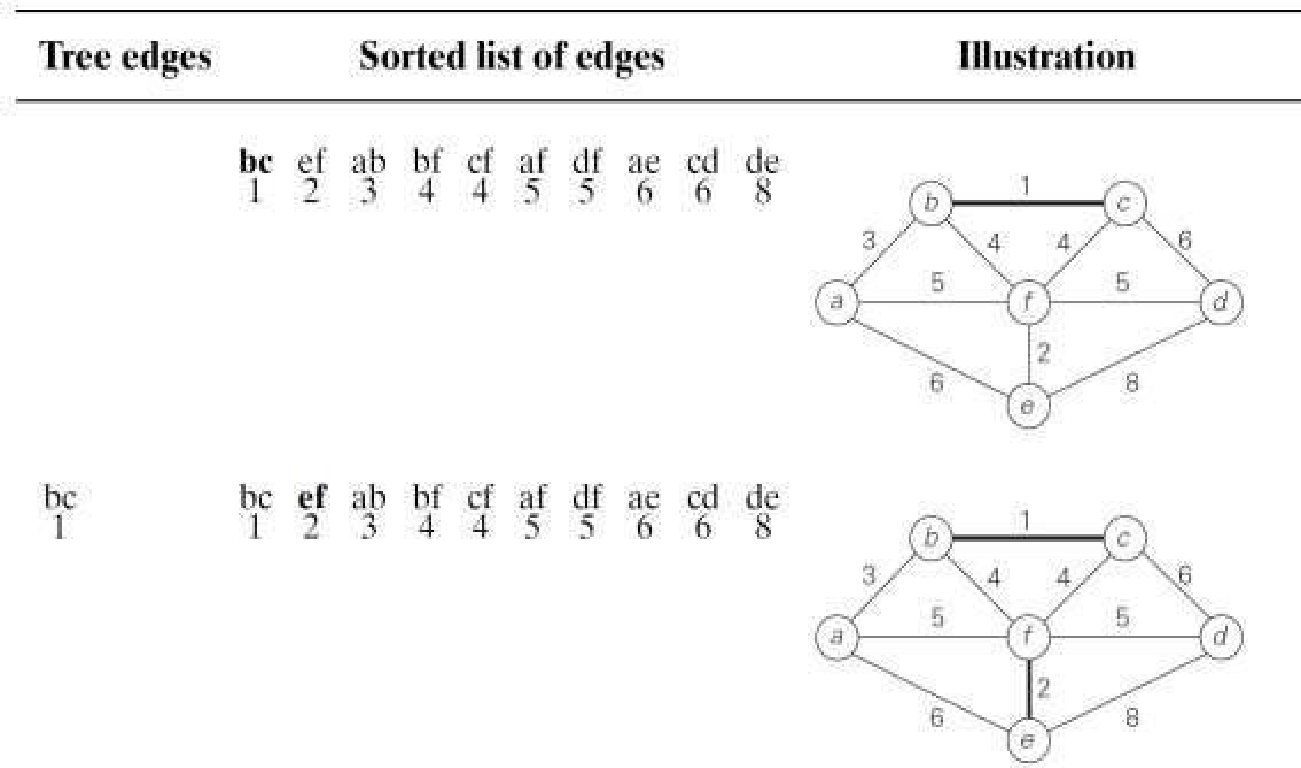
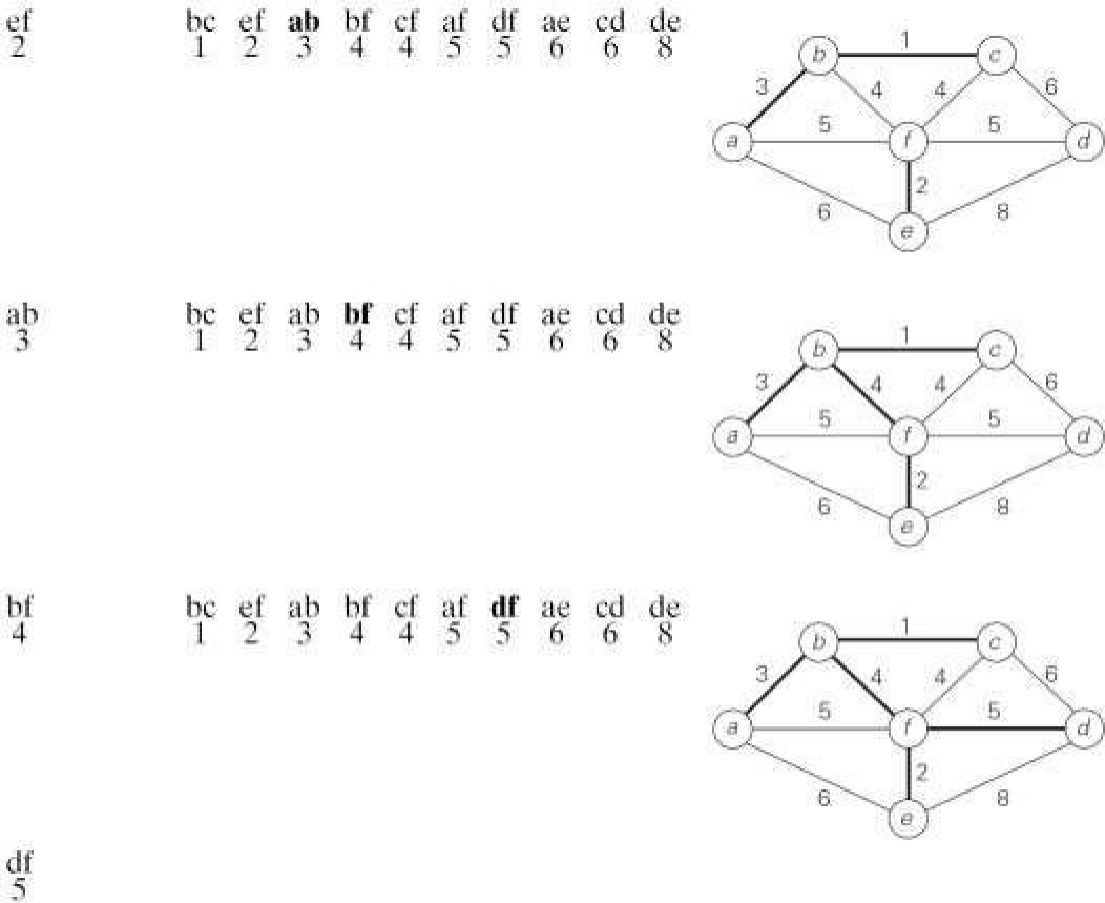


## **complexity of Kruskal's algorithm**

Now, let's see the time complexity of Kruskal's algorithm.

* **TimeComplexity**  
  The time complexity of Kruskal's algorithm is O(E logE) or O(V logV), where E is the no. of edges, and V is the no. of vertices.

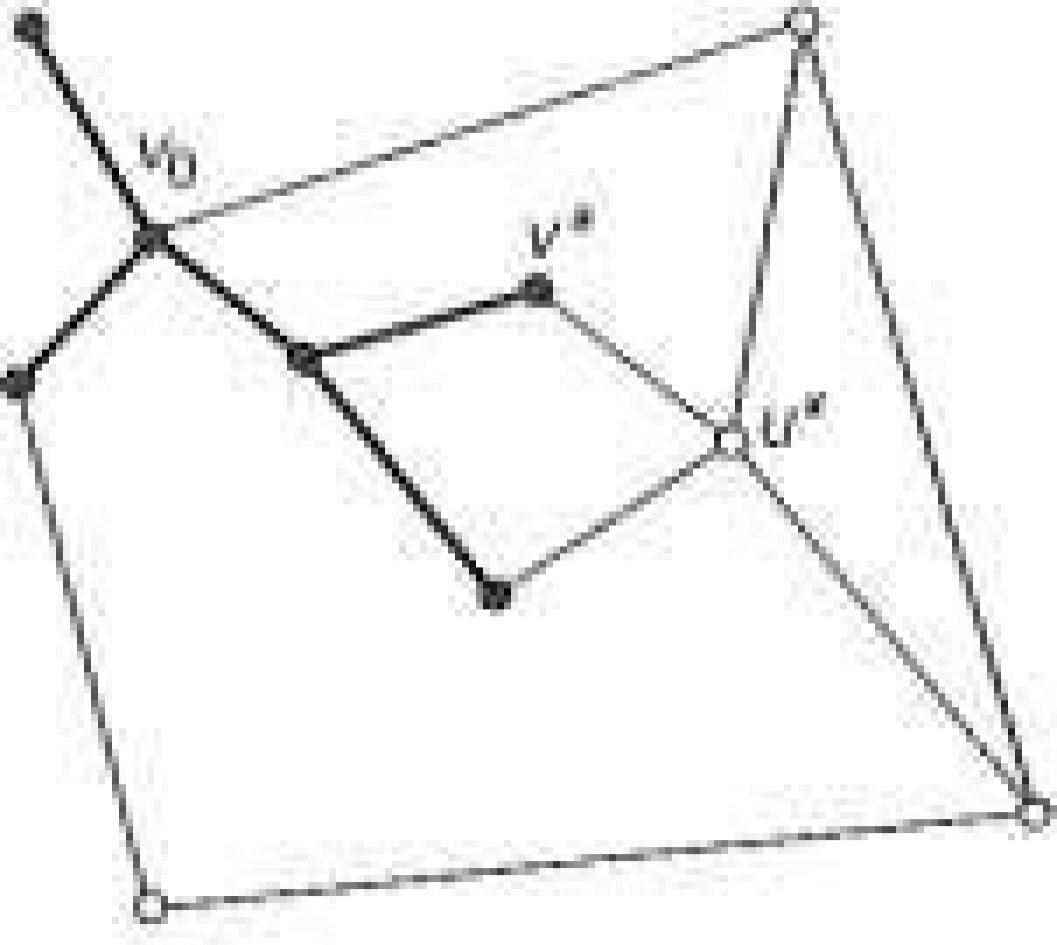
12. Apply Kruskal’s Algorithm to find the minimum spanning tree of the following graph: 7

DAA– Question Bank

13. Explain about single source shortest path problem using Dijksta’s Algorithm with example 14

Answer;

***Single-source shortest-paths problem*** is defined as follows. For a given vertex called the *source* in a weighted connected graph, the problem is to find shortest paths to all its other vertices. The single-source shortest-paths problem asks for a family of paths, each leading from the source to a different vertex in the graph, though some paths may, of course, have edges in common. **Dijkstra'sAlgorithm**

DAA– Question Bank

**Dijkstra's** Algorithm is the best-known algorithm for the single-source shortest-paths problem. This algorithm is applicable to undirected and directed graphs with nonnegative weights only. **Working -** Dijkstra's algorithm finds the shortest paths to a graph's vertices in order of their distance from a given source.

▪ First, it finds the shortest path from the source to a vertex nearest to it, then to a second nearest, and so on.

▪ In general, before its ith iteration commences, the algorithm has already identified the shortest

paths to *i*-1 other vertices nearest to the source. These vertices, the source, and the edges of the shortest paths leading to them from the source form a subtree *Ti* of the given graph shown in the figure.

▪ Since all the edge weights are nonnegative, the next vertex nearest to the source can be found

among the vertices adjacent to the vertices of *Ti.* The set of vertices adjacent to the vertices in *Ti* can be referred to as "fringe vertices"; they are the candidates from which Dijkstra's algorithm selects the next vertex nearest to the source.

To identify the ith nearest vertex, the algorithm computes, for every fringe vertex *u,* the sum of the distance to the nearest tree vertex *v* (given by the weight of the edge (*v,* u)) and the length *d.,* of the shortest path from the source to *v* (previously determined by the algorithm) and then selects the vertex with the smallest such sum. The fact that it suffices to compare the lengths of such special paths is the central insight of Dijkstra's algorithm.

▪ To facilitate the algorithm's operations, we label each vertex with two labels.

o The numeric label **d** indicates the length of the shortest path from the source to this vertex found by the algorithm so far; when a vertex is added to the tree, d indicates the length of the shortest path from the source to that vertex.

o The other label indicates the name of the next-to-last vertex on such a path, i.e., the parent of the vertex in the tree being constructed. (It can be left unspecified for the sources and vertices that are adjacent to none of the current tree vertices.)

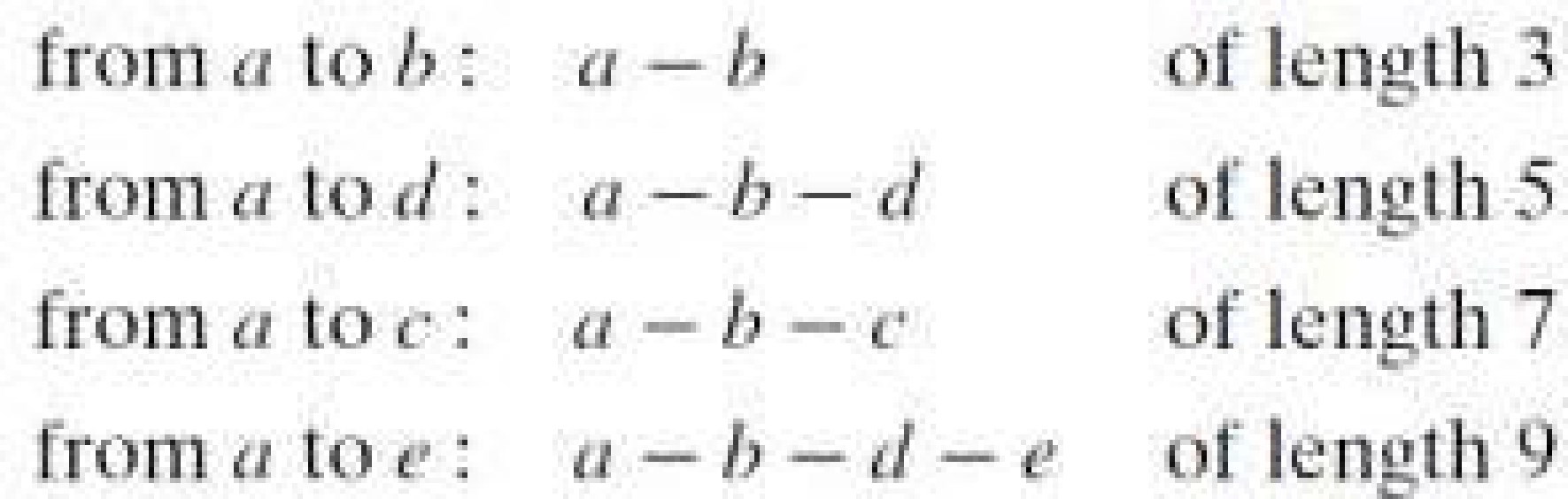
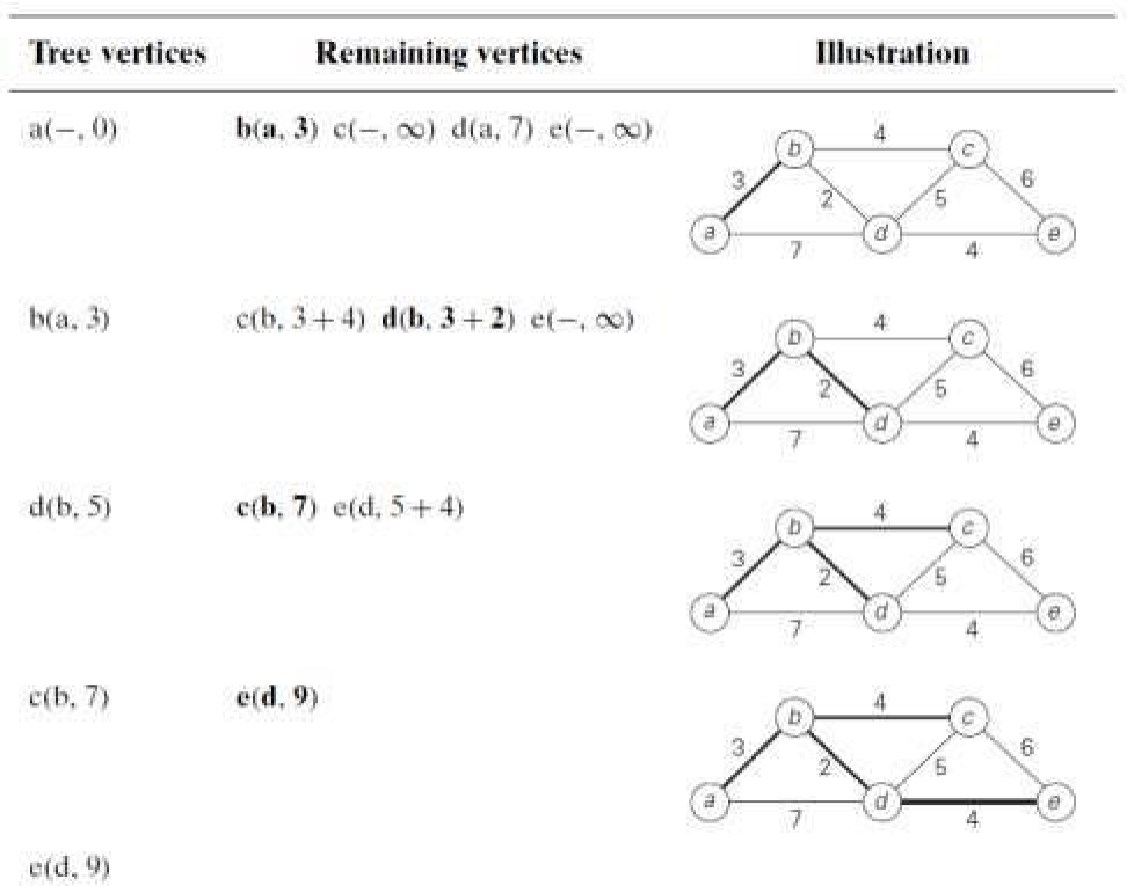
With such labeling, finding the next nearest vertex u\* becomes a simple task of finding a fringe vertex with the smallest d value. Ties can be broken arbitrarily.

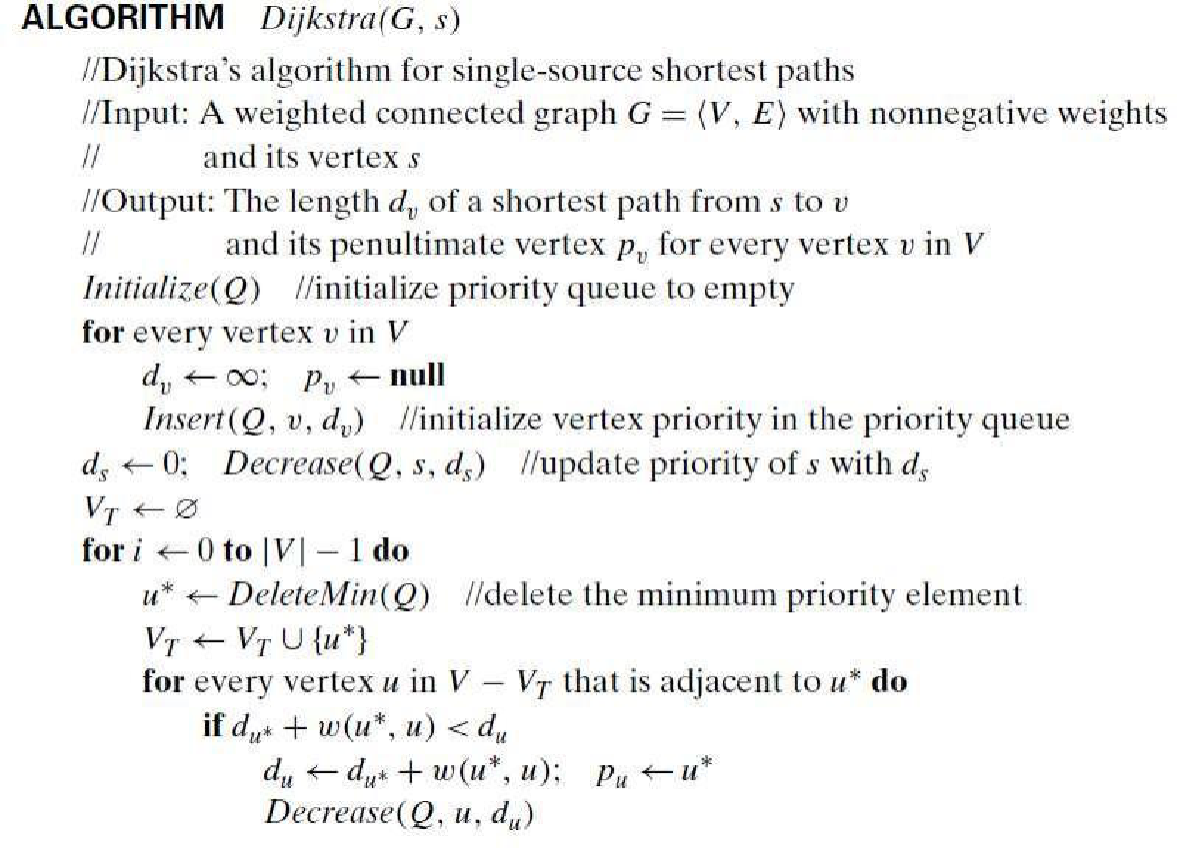
▪ After we have identified a vertex u\* to be added to the tree, we need to perform two operations: o Move *u\** from the fringe to the set of tree vertices.

o For each remaining fringe vertex *u* that is connected to *u\** by an edge of weight *w(u\*, u)* such that *du\*+ w(u\*, u) <du,* update the labels of *u* by *u\** and *du\**+ *w(u\*, u),* respectively.

Example;

DAA– Question Bank

The shortest paths (identified by following nonnumeric labels backward from a destination vertex in the left column to the source) and their lengths (given by numeric labels of the tree vertices) are as follows:

DAA– Question Bank

**Analysis:**

The time efficiency of Dijkstra’s algorithm depends on the data structures used for implementing the priority queue and for representing an input graph itself.

Efficiency is **Θ(|V|2)** for graphs represented by their ***weight matrix*** and the priority queue implemented as an ***unordered array***.

For graphs represented by their ***adjacency lists*** and the priority queue implemented as a ***min-heap***, it is in **O (|E| log |V| )**