PART-1

Divide and Conquer with Examples such as Sorting, Matrix Multiplication, Convex Hull, Searching.

CONCEPT OUTLINE : PART-1

- **Strassen's algorithm for matrix multiplication :** It is an application of divide and conquer technique. Suppose we wish to compute the product $C = AB$ where each A, B and C are $n \times n$ matrices. Assuming that n is an exact power of 2. We divide each of A, B and C into four $4/2 \times 4/2$ matrices.
- **Graph :** A graph is a collection of vertices V and edges E.
- **BFS (Breadth First Search) :** BFS is an algorithm for tree searching which works on directed or undirected graphs. It is a shortest path search algorithm.
- **DFS (Depth First Search) :** It is also tree searching algorithm. In this, edges are expanded out of the most recently discovered vertex V.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.1. Write algorithm for bubble sort and selection sort.

Answer

Bubble sort :

```
Bubble_Sort (A)
1.  for i → 1 to length [A]
2.  for j → length [A] down to i + 1
3.          if A[j] < A[j –1]
4.  Exchange (A[j], A[j – 1])
```

Selection_Sort :

```
Selection_Sort (A)
1.  n ← length [A]
2.    for j ← 1 to n –1
3.      smallest ← j
4.        for i ← j + 1 to n
5.          if A[i] < A[smallest]
```

6. then smallest $\leftarrow i$
7. exchange ($A[j]$), (A[smallest])

Que 3.2. What is matrix chain multiplication problem ? Describe a solution for matrix chain multiplication problem.

AKTU 2013-14, Marks 10

Answer

1. Matrix chain multiplication (or Matrix Chain Ordering Problem, MCOP) is an optimization problem that can be solved using dynamic programming.
2. MCOP helps to find the most efficient way to multiply given matrices.
3. Solution for matrix chain multiplication problem is Strassen's matrix multiplication.

Strassen's matrix multiplication :

1. It is an application of divide and conquer technique.
2. Suppose we wish to compute the product $C = AB$ where each A, B and C are $n \times n$ matrices.
3. Assuming that n is an exact power of 2. We divide each of A, B and C into four $n/2 \times n/2$ matrices.

   Rewriting the equation $C = AB$ as

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & g \\ f & h \end{pmatrix} \quad \ldots(3.2.1)$$

4. For convenience, the sub-matrices of A are labelled alphabetical from left to right, whereas those of B are labelled from top to bottom. So that matrix multiplication is performed.

   Equation (3.2.1) corresponds to the four equations :

$$r = ae + bf \quad \ldots(3.2.2)$$

$$s = ag + bh \quad \ldots(3.2.3)$$

$$t = ce + df \quad \ldots(3.2.4)$$

$$u = cg + dh \quad \ldots(3.2.5)$$

5. Each of these four equations specifies two multiplications of $n/2 \times n/2$ matrices and the addition of their $n/2 \times n/2$ products.
6. Using these equations to define a straight-forward divide and conquer strategy. We derive the following recurrence for the time $T(n)$ to multiply two $n \times n$ matrices :

$$T(n) = 8T(n/2) + \theta(n^2)$$

7. Unfortunately, this recurrence has the solution $T(n) = \Theta(n^3)$ and thus, this method is no faster than the ordinary one.

Strassen's method has four steps :

1. Divide the input matrices A and B into $n/2 \times n/2$ sub-matrices.
2. Using $\Theta(n^2)$ scalar additions and subtraction compute 14 $n/2 \times n/2$ matrices $A_1, B_1, A_2, B_2, \ldots A_7, B_7$.
3. Recursively compute the seven matrix products.
4. Compute the desired sub-matrices r, s, t, u of the result matrix C by adding and/or subtracting various combinations of the p_i matrices using only $\Theta(n^2)$ scalar additions and subtractions.

Que 3.3. Describe in detail Strassen's matrix multiplication algorithms based on divide and conquer strategies with suitable example. **AKTU 2014-15, Marks 10**

Answer

Strassen's matrix multiplication : Refer Q. 3.2, Page 3–3B, Unit-3.

Example :

Given matrices :

$$\begin{bmatrix} 2 & 9 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 4 & 11 \\ 8 & 7 \end{bmatrix}$$

So, $\text{def } a_{11} = 2, \text{def } b_{11} = 4$
$\text{def } a_{12} = 9, \text{def } b_{12} = 11$
$\text{def } a_{21} = 5, \text{def } b_{21} = 8$
$\text{def } a_{22} = 6, \text{def } b_{22} = 7$

Now calculate,

$S_1 = b_{11} - b_{22} = -3,$
$S_2 = a_{11} + a_{12} = 11,$
$S_3 = a_{21} + a_{22} = 11,$
$S_4 = a_{21} - b_{11} = 4,$
$S_5 = a_{11} + a_{22} = 8,$
$S_6 = b_{11} + b_{22} = 11$
$S_7 = a_{12} - a_{22} = 3$
$S_8 = b_{21} + b_{22} = 15$
$S_9 = a_{11} - a_{21} = -3$
$S_{10} = b_{11} + b_{12} = 15$

Now,

$P_1 = a_{11} \times S_1 = -6,$
$P_2 = S_2 \times b_{22} = 77,$
$P_3 = S_3 \times b_{11} = 44,$
$P_4 = a_{22} \times S_4 = 24$
$P_5 = S_5 \times S_6 = 88$
$P_6 = S_7 \times S_8 = 45$
$P_7 = S_9 \times S_{10} = -45$

Now,

$C_{11} = P_5 + P_4 - P_2 + P_6 = 80$
$C_{12} = P_1 + P_2 = 71$
$C_{21} = P_3 + P_4 = 68$
$C_{22} = P_5 + P_1 - P_3 - P_7 = 83$

Now matrix $= \begin{bmatrix} 80 & 71 \\ 68 & 83 \end{bmatrix}$

Que 3.4. What do you mean by graphs ? Discuss various representations of graphs.

Answer

A graph G consists of a set of vertices V together with a set E of vertex pairs of edges.

Graphs are important because any binary relation is a graph, so graphs can be used to represent essentially any relationship.

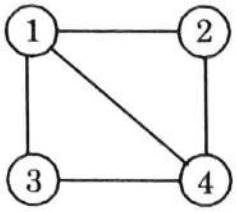


Fig. 3.4.1.

Example : A network of roads, with cities as vertices and roads between cities as edges.

Various representation of graphs :

1. **Matrix representation :** Matrices are commonly used to represent graphs for computer processing. Advantage of representing the graph in matrix lies in the fact that many results of matrix algebra can be readily applied to study the structural properties of graph from an algebraic point of view.

   a. Adjacency matrix :

   i. Representation of undirected graph :

   The adjacency matrix of a graph G with n vertices and no parallel edges is a $n \times n$ matrix $A = [a_{ij}]$ whose elements are given by

   $a_{ij} = 1$, if there is an edge between i^{th} and j^{th} vertices

   $= 0$, if there is no edge between them

   ii. Representation of directed graph :

   The adjacency matrix of a digraph D, with n vertices is the matrix

   $A = [a_{ij}]_{n \times n}$ in which

   $a_{ij} = 1$ if arc (v_i, v_j) is in D

   $= 0$ otherwise

   b. Incidence matrix :

   i. Representation of undirected graph :

   Consider an undirected graph $G = (V, E)$ which has n vertices and m edges all labelled. The incidence matrix $I(G) = [b_{ij}]$, is then $n \times m$ matrix, where

   $b_{ij} = 1$ when edge e_j is incident with v_i

   $= 0$ otherwise

ii. Representation of directed graph :

The incidence matrix $I(D) = [b_{ij}]$ of digraph D with n vertices and m edges is the $n \times m$ matrix in which.

$b_{ij} = 1$ if arc j is directed away from vertex v_i

$= -1$ if arc j is directed towards vertex v_i

$= 0$ otherwise.

2. Linked representation :

a. In linked representation, the two nodes structures are used :

i. For non-weighted graph,

INFO	Adj-list

ii. For weighted graph,

Weight	INFO	Adj-list

Where Adj-list is the adjacency list *i.e.*, the list of vertices which are adjacent for the corresponding node.

b. The header nodes in each list maintain a list of all adjacent vertices of that node for which the header node is meant.

Que 3.5. What is a bipartite graph ? How to check a graph is bipartite or not ?

Answer

A bipartite graph is an undirected graph $G = (V, E)$ in which V can be partitioned into two sets V_1 and V_2 such that $(u, v) \in E$ implies either $u \in V_1$ and $v \in V_2$ or $u \in V_2$ and $v \in V_1$.

To check graph is bipartite or not : The algorithm traverse the graph labeling the vertices 0, 1, or 2 corresponding to unvisited, partition 1 and partition 2 nodes. If an edge is detected between two vertices in the same partition, the algorithm returns.

Bipartite (*G*, *S*)

1. For each vertex $U \in V[G] - [s]$ do
2. Colour[u] ← WHITE
3. $d[u] \leftarrow \infty$
4. partition[u] ← 0
5. Colour[s] ← gray
6. partition[s] ← 1
7. $d[s] \leftarrow 1$
8. $Q \leftarrow [s]$
9. While Queue 'Q' is not empty do
10. $u \leftarrow$ head [Q]
11. for each v in Adj[u] do

12. if partition $[u]$ = partition $[v]$ then
13. return 0
14. else
15. if colour$[v]$ = WHITE then
16. colour$[v] \leftarrow$ gray
17. $d[v] \leftarrow d[u] + 1$
18. partition$[v] = 3 -$ partition$[u]$
19. ENQUEUE (Q, v)
20. DEQUEUE (Q)
21. Colour$[u] \leftarrow$ BLACK
22. Return 1

Que 3.6. **Explain DFS. Also give DFS algorithm.**

OR

Describe Depth First Search (DFS) strategy. How DFS can be used to solve the problem of unbounded trees ? Also write an algorithm.

Answer

Depth First Search Algorithm :

1. Algorithm starts at a specific vertex S in G, which becomes current vertex.
2. Then algorithm traverse graph by any edge (u, v) incident to the current vertex u.
3. If the edge (u, v) leads to an already visited vertex v, then we backtrack to current vertex u.
4. If, on other hand, edge (u, v) leads to an unvisited vertex v, then we go to v and v becomes our current vertex.
5. We proceed in this manner until we reach to "dead end". At this point we start backtracking.
6. The process terminates when backtracking leads back to the start vertex.
7. Edges leads to new vertex are called discovery or tree edges and edges lead to already visited vertex are called back edges.

How to solve the problem of unbounded tree :

1. The problem of unbounded depth of trees can be overcome by limiting the depth first search to a pre-determined depth limit 'l'.
2. This means that nodes at depth 'l' are treated as if they have no successors. This is called depth-limited search. This solves the problem of infinite-path problem.
3. However, often we do not know the depth 'd' of the goal state. If we choose $l < d$, then we will never find the solution. If we choose $l > d$, we may end up with a non-optimal solution.

4. Depth first is a special case of depth-limited search, where $l = \infty$.

Algorithm :

```
depth limit = max depth to search to;
agenda = initial state;
    if initial state is goal state then return solution;
    else
        while agenda not empty do
        take node from front of agenda;
        if depth (node) < depth limit then
            {new nodes = apply operations to node;
            add new nodes to front of agenda;
            if goal state in new nodes then return solution;}
```

Que 3.7. Explain Breadth First Search (BFS). Give its algorithm.

Answer

Breadth first search :

1. The general idea behind a breadth first search beginning at a starting node A is as follows :
   a. First we examine the starting node A.
   b. Then, we examine all the neighbours of A, and so on.
2. Naturally, we need to keep track of the neighbours of a node, and we need to guarantee that no node is processed more than once.
3. This is accomplished by using a queue to hold nodes that are waiting to be processed, and by using a field STATUS which tells us the current status of any node.

Algorithm : This algorithm executes a breadth first search on a graph G beginning at a starting node A.

1. Initialize all nodes to ready state (STATUS=1).
2. Put the starting node A in queue and change its status to the waiting state (STATUS = 2).
3. Repeat steps 4 and 5 until queue is empty.
4. Remove the front node N of queue. Process N and change the status of N to the processed state (STATUS = 3).
5. Add to the rear of queue all the neighbours of N that are in the ready state (STATUS=1) and change their status to the waiting state (STATUS = 2).
   [End of loop]
6. End.

Que 3.8. Explain topological sort. Give its algorithm.

Answer

1. A topological sort of a Directed Acyclic Graph (DAG) $\overline{G}$ is an ordering of the vertices of $\overline{G}$ such that for every edge (e_i, e_j) of $\overline{G}$ we have $i < j$.
2. A topological sort is a linear ordering of all its vertices such that if DAG $\overline{G}$ contains an edge (e_i, e_j), then e_i appears before e_j in the ordering. If DAG is cyclic then no linear ordering is possible.
3. A topological ordering is an ordering such that any directed path in DAG, G traverses vertices in increasing order.

Topological_Sort(G)

i. For each vertex find the finish time by calling DFS(G).
ii. Insert each finished vertex into the front of a linked list.
iii. Return the linked list.

Que 3.9. Write an algorithm to test whether a given graph is connected or not.

Answer

Test-connected (G) :

1. Choose a vertex x
2. Make a list L of vertices reachable from x,
   and another list K of vertices to be explored.
3. Initially, $L = K = x$.
4. while K is non-empty
5. Find and remove some vertex y in K
6. for each edge (y, z)
7. if (z is not in L)
8. Add z to both L and K
9. if L has fewer than n items
10. return disconnected
11. else return connected.

Que 3.10. Discuss strongly connected components with its algorithm.

Answer

1. The Strongly Connected Components (SCC) of a directed graph G are its maximal strongly connected subgraphs.
2. If each strongly connected component is contracted to a single vertex, the resulting graph is a directed acyclic graph called as condensation of G.

Kosaraju's algorithm :

Kosaraju's algorithm is an algorithm to find the strongly connected components of a directed graph.

1. Let G be a directed graph and S be an empty stack.
2. While S does not contain all vertices :
   i. Choose an arbitrary vertex v not in S. Perform a depth first search starting at v.
   ii. Each time that depth first search finishes expanding a vertex u, push u onto S.
3. Reverse the direction of all arcs to obtain the transpose graph.
4. While S is non-empty :
   i. Pop the top vertex v from S. Perform a depth first search starting at v.
   ii. The set of visited vertices will give the strongly connected component containing v; record this and remove all these vertices from the graph G and the stack S.
   iii. Equivalently, breadth first search (BFS) can be used instead of depth first search.

Que 3.11. Explain convex hull problem.

AKTU 2017-18, Marks 10

OR

Discuss convex hull. Give Graham-Scan algorithm to compute convex hull.

Answer

1. The convex hull of a set S of points in the plane is defined as the smallest convex polygon containing all the points of S.
2. The vertices of the convex hull of a set S of points form a (not necessarily proper) subset of S.

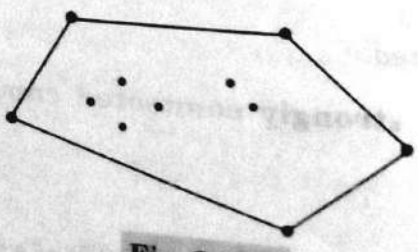


Fig. 3.11.1.

3. To check whether a particular point $p \in S$ is extreme, see each possible triplet of points and check whether p lies in the triangle formed by these three points.
4. If p lies in the triangle then it is not extreme, otherwise it is.

5. We denote the convex hull of S by CH(S). Convex hull is a convex set because the intersection of convex sets is convex and convex hull is also a convex closure.

Graham-Scan algorithm :

The procedure GRAHAM-SCAN takes as input a set Q of points, where $|Q| \geq 3$. It calls the functions Top(S), which return the point on top of stack S without changing S, and to NEXT-TO-TOP(S), which returns the point one entry below the top of stack S without changing S.

GRAHAM-SCAN(Q)

1. Let p_0 be the point in Q with the minimum y-coordinate, or the leftmost such point in case of a tie.
2. Let $<p_1, p_2, ..., p_m>$ be the remaining points in Q, sorted by polar angle in counter clockwise order around p_0 (if more than one point has the same angle remove all but the one that is farthest from p_0).
3. PUSH (p_0, S)
4. PUSH (p_1, S)
5. PUSH (p_2, S)
6. for $i \leftarrow 3$ to m
7. do while the angle formed by points NEXT-To-TOP(S), Top(S), and p_i makes a non left turn.
8. do POP(S)
9. PUSH (p_i, S)
10. return S

The worst case running time of GRAHAM-SCAN is

$$T(n) = O(n)\ O(n \log n) + O(1) + O(n) = O(n \log n)$$

where $n = |Q|$

Graham's scan running time depends only on the size of the input it is independent of the size of output.

Que 3.12. Give Jarvis's March algorithm to compute convex hull.

Answer

Jarvis's march computes the convex hull of a set Q of points by a technique known as package wrapping. The algorithm runs in time $O(nh)$ where n is the number of vertices of CH(Q).

Steps for Jarvis's March algorithm :

1. First, a base point p_0 is selected, this is the point with the minimum y-coordinate.
2. Select leftmost point in case of tie.

3. The next convex hull vertices p_1 has the least polar angle with respect to the positive horizontal ray from p_0.
4. Measure in counter clockwise direction.
5. If tie, choose the farthest such point.
6. Vertices $p_2, p_3, \ldots p_k$ are picked similarly until $y_k = y_{max}$.
7. p_{i+1} has least polar angle with respect to positive ray from p_0.
8. If tie, choose the farthest such point.

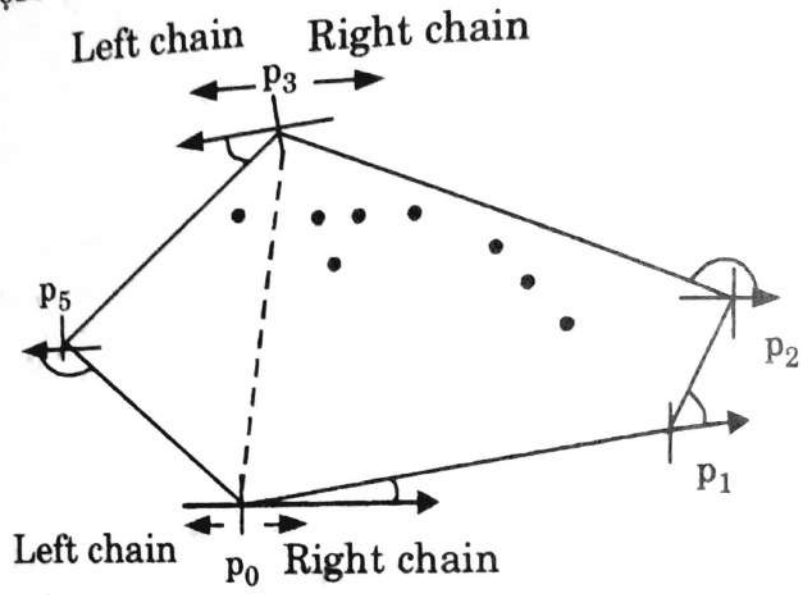


Fig. 3.12.1.

9. The sequence p_0, p_1, p_k is right chain of CH(Q).
10. To choose the left chain of CH(Q) start with p_k.
11. Choose p_{k+1} as the point with least polar angle with respect to the negative ray from p_k.
12. Again measure counterclockwise direction.
13. If tie occurs, choose the farthest such point.
14. Continue picking $p_{k+1}, p_{k+2}, \ldots p_t$ in same fashion unit obtain $p_t = p_0$.

PART-2

Greedy Methods with Examples such as Optimal Reliability Allocation, Knapsack.

CONCEPT OUTLINE : PART-2

- **Greedy algorithms :** Greedy algorithms are simple and straightforward. They are shortsighted in their approach in the sense that they take decisions on the basis of information at hand without worrying about the effect these decisions may have in the future.

- **Activity selection problem :** In this problem, we first choose the activity with minimum duration $(f_i - s_i)$ and schedule it. Then we left all the activities that are not compatible to this one, which means we have to select an activity which is compatible having minimum duration.
- **Knapsack problem :**
  We want to pack m items such that :

i. The item i^{th} is worth x_i dollars and weights w_i pound.
ii. Take as valuable a load as possible, but cannot exceed w pounds.
iii. x_i, w_i, w are integers.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.13. Write note on the greedy algorithm.

Answer

1. Greedy algorithms are simple and straight forward.
2. Greedy algorithms are shortsighted in their approach in the sense that they take decisions on the basis of information at hand without worrying about the effect these decisions may have in the future.
3. Greedy algorithms are easy to invent, easy to implement and most of the time quite efficient.
4. Many problems cannot be solved correctly by greedy approach.
5. Greedy algorithms are used to solve optimization problems.

Que 3.14. What are the four functions included in greedy algorithm ? Write structure of greedy algorithm.

Answer

The greedy algorithm consists of four function :

i. A function that checks whether chosen set of items provide a solution.
ii. A function that checks the feasibility of a set.
iii. The selection function tells which of the candidates are most promising.
iv. An objective function, which does not appear explicitly, gives the value of a solution.

Structure of greedy algorithm :

i. Initially the set of chosen items is empty *i.e.*, solution set.

ii. At each step

a. Item will be added in a solution set by using selection function.

b. If the set would no longer be feasible

Reject items under consideration (and is never consider again).

c. Else if set is still feasible

add the current item.

Que 3.15. Define activity selection problem and give its solution by using greedy approach with its correctness.

Answer

1. An activity selection is the problem of scheduling a resource among several competing activity. Given a set $S = \{1, 2, \ldots, n\}$ of n activities.
2. Each activity has s_i a start time, and f_i a finish time.
3. If activity i is selected, the resource is occupied in the intervals (s_i, f_i). We say i and j are compatible activities if their start and finish time does not overlap *i.e.*, i and j compatible if $s_i \geq f_j$ and $s_j \geq f_i$
4. The activity selection problem is, to select a maximal sized subset of mutually compatible activities.

Here we maximize the number of activities selected, but if the profit were proportional to $s_i - f_i$, this will not maximize the profit.

Greedy algorithm :

Assume that $f_1 \leq f_2 \leq \ldots \leq f_n$

Greedy-Activity-Selector (s, f)

1. $n \leftarrow$ length $[s]$
2. $A \leftarrow \{a_1\}$
3. $i \leftarrow 1$
4. for $m \leftarrow 2$ to n
5. do if $s_m \geq f_i$
6. then $A \leftarrow A \cup \{(a_m)\}$
7. $i \leftarrow m$
8. return A

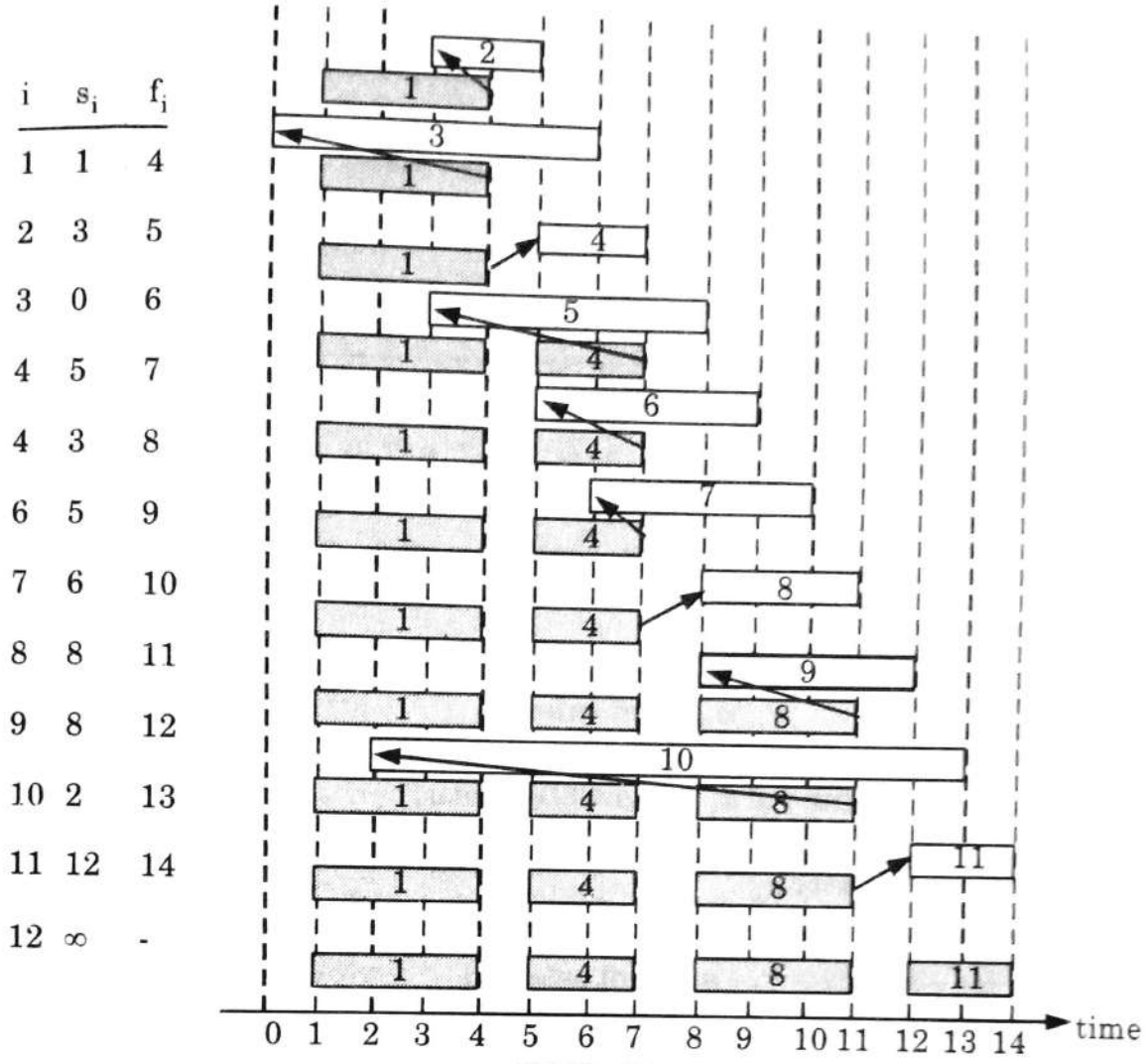


Fig. 3.15.1.

The algorithm starts with {1} and checks to see which can be added after 1, updating the global "finishing time" and comparing with each start time. The activity picked is always the first that is compatible. Greedy algorithms do not always produce optimal solutions.

Correctness : Greedy algorithm does not always produce optimal solutions but GREEDY-ACTIVITY-SELECTOR does.

Que 3.16. **What are greedy algorithms ? Find a solution to the following activity selection problem using greedy technique. The starting and finishing times of 11 activities are given as follows : (2, 3) (8, 12) (12, 14) (3, 5) (0, 6) (1, 4) (6, 10) (5, 7) (3, 8) (5, 9) (8, 11)**

OR

What is greedy approach ? Write an algorithm which uses this approach.

Answer

Greedy algorithm : Refer Q. 3.13, Page 3–13B, Unit-3.

Greedy approach : Greedy approach works by making the decision that seems most promising at any moment it never reconsiders this decision, whatever situation may arise later. For example consider the problem of "Activity selection".

Algorithm for greedy activity selection : Refer Q. 3.15, Page 3–14B, Unit-3.

Numerical :

Sorted activities	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}
Starting time	2	1	3	0	5	3	5	6	8	8	12
Finish time	3	4	5	6	7	8	9	10	11	12	14

We select first activity a_1

(2, 3)

Check for activity a_2

Starting time of $a_2 \not\geq$ time of a_1

$\therefore$ a_2 is not selected

Check for activity a_3

Starting time of $a_3 \geq$ finish time of a_1

$\therefore$ a_3 is selected

Check for activity a_4

Starting time of $a_4 \not\geq$ finish time of a_3

$\therefore$ a_4 is not selected

Check for activity a_5

Starting time of $a_5 \geq$ finish time of a_3

$\therefore$ a_5 is selected

Check for activity a_6

Starting time of $a_6 \not\geq$ finish time of a_5

$\therefore$ a_6 is not selected

Check for activity a_7

Starting time of $a_7 \not\geq$ finish time of a_5

$\therefore$ a_5 is not selected

Check for activity a_8

Starting time of $a_8 \not\geq$ finish time of a_5

$\therefore$ a_9 is selected

Check for activity a_{10}

Starting time of $a_{10} \not\geq$ finish time of a_9

$\therefore$ a_{10} is not selected

Check for activity a_{11}

Starting time of $a_{11} \geq$ finish time of a_4

$\therefore$ a_{11} is selected.

$\therefore$ Therefore selected activities are :

a_1 : (2, 3)
a_3 : (3, 5)
a_5 : (5, 7)
a_9 : (8, 11)
a_{11} : (12, 14)

Que 3.17. What is "Greedy Algorithm" ? Write its pseudo code for recursive and iteration process.

Answer

Greedy algorithm : Refer Q. 3.13, Page 3–13B, Unit-3.

Greedy algorithm defined in two different forms :

i. Pseudo code for recursive greedy algorithm :

R_A_S (s, f, i, j)

1. $m \leftarrow i + 1$
2. while $m < j$ and $s_m < f_i$
3. do $m \leftarrow m + 1$
4. if $m < j$
5. then return $\{a_m\} \cup R\_A\_S(s, f, m, j)$
6. else return ϕ

ii. Pseudo code for iterative greedy algorithm :

G_A_S (s, f)

1. $n \leftarrow$ length $[s]$
2. $A \leftarrow [a_1]$
3. $i \leftarrow 1$
4. $m \leftarrow 2$ to n
5. do if $s_m \geq f_i$
6. then $A \leftarrow A \cup \{a_m\}$
7. $i \leftarrow m$
8. return A

Que 3.18. What is an optimization problem ? How greedy method can be used to solve the optimization problem ?

AKTU 2013-14, Marks 10

Answer

1. An optimization problem is the problem of finding the best solution from all feasible solutions.
2. Optimization problems can be divided into two categories depending on whether the variables are continuous or discrete.

3. There is no way in general that one can specify if a greedy algorithm will solve a particular optimization problem.
4. However if the following properties can be demonstrated, then it is probable to use greedy algorithm :
   a. **Greedy choice property :** A globally optimal solution can be arrived at by making a locally optimal greedy choice. That is, when we are considering which choice to make, we make the choice that looks best in the current problem, without considering results from sub-problems.
   b. **Optimal substructure :** A problem exhibits optimal substructure if an optimal solution to the problem contains within it optimal solutions to sub-problems.

Que 3.19. What is knapsack problem ? Describe an approach used to solve the problem.

Answer

1. The knapsack problem is a problem in combinatorial optimization.
2. Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.
3. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items.

Approach use to solve the problem :

1. In knapsack problem, we have to fill the knapsack of capacity w, with a given set of items $I_1, I_2 \ldots I_n$ having weight $w_1, w_2 \ldots w_n$ in such a manner that the total weight of items cannot exceed the capacity of knapsack and maximum possible value can be obtained.
2. Using branch and bound approach, we have a bound that none of the items can have total sum more than the capacity of knapsack and must give maximum possible value.
3. The implicit tree for this problem is a binary tree which left branch implies inclusion and right exclusion.
4. Upper bound of node can be calculated as :
   $ub = v + (W - w)\,(v_{i+1} / w_{i+1})$

weight	value
ub	
node	

Que 3.20. **Write greedy algorithm for discrete knapsack problem.**

Answer

Greedy algorithm for the discrete knapsack problem :

1. Compute value/weight ratio v_i / w_i for all items.
2. Sort the items in non-increasing order of the ratios v_i / w_i.
3. Repeat until no item is left in sorted list using following steps :
   a. If current item fits, use it.
   b. Otherwise skip this item, and proceed to next item.

Example : Knapsack problem for the following instance using greedy approach. The item can be selected or skipped completely.

Item	Weight	Value
1	7	₹49
2	3	₹12
3	4	₹42
4	5	₹30

Consider $W = 10$.

Solution : This is also called 0-1 knapsack. Either we can completely select an item or skip it. First of all we will compute value-to-weight ratio and arrange them in non-increasing order of the ratio.

Item	Weight	Value	Value/Weight	
3	4	₹42	10.5	→
1	7	₹49	7	
4	5	₹30	6	→
2	3	₹12	4	

To fulfill the capacity $W = 10$, we will have

1. add item of weight 4
2. skip item of weight 7
3. add item of weight 5
4. skip item of weight 3

Maximum value = 10.5 + 6 = 16.5

This is the solution for given instance of knapsack problem.

But the greedy algorithm does not give optimal solution always rather there is no upper bound on the accuracy of approximate solution.

Que 3.21. **Given the six items in the table below and a knapsack with weight 100, what is the solution to the knapsack problem in all concepts. *i.e.*, explain greedy all approaches and find the optimal solution.**

Item ID	Weight	Value	Value/Weight
A	100	40	.4
B	50	35	.7
C	40	20	.5
D	20	4	.2
E	10	10	1
F	10	6	.6

AKTU 2017-18, Marks 10

Answer

We can use 0 – 1 knapsack problem when the items cannot be divided into parts and fractional knapsack problem when the items can be divided into fractions.

According to 0 – 1 knapsack problem, either we select an item or reject. So the item will be selected according to value per weight.

E is selected $W = 10 < 100$

B is selected $W = 10 + 50$

$= 60 < 100$

F is selected $W = 60 + 10$

$= 70 < 100$

C cannot be selected because

$W = 70 + 40 = 110 > 100$

Hence we select *D*

$W = 70 + 20 = 90 < 100$

Total value $= 10 + 35 + 6 + 4 = 55$

According to fractional knapsack problem, we can select fraction of any item.

E is selected $W = 10 < 100$

B is selected $W = 10 + 50$

$= 60 < 100$

F is selected $W = 60 + 10$

$= 70 < 100$

If we select *C* $W = 70 + 40$

$= 110 > 100$

Hence we select the fraction of item *C* as

$$\frac{100 - W}{\text{Weight of } C} = \frac{100 - 70}{40}$$

$$= \frac{30}{40} = 0.75$$

So,
$$W = 0.75 \times 40 = 30$$
$$W = 70 + 30 = 100$$
$$\text{Total value} = 10 + 35 + 6 + 0.75\,(20)$$
$$= 10 + 35 + 6 + 15 = 66$$

Que 3.22. What is 0/1-knapsack problem ? Does greedy method effective to solve the 0/1-knapsack problem ?

Answer

The 0/1-knapsack problem is defined as follows :

1. Given, a knapsack of capacity c and n items of weights $\{w_1, w_2, ..., w_n\}$ and profits $\{p_1, p_2, ..., p_n\}$, the objective is to choose a subset of n objects that fits into the knapsack and that maximizes the total profit.
2. Consider a knapsack (bag) with a capacity of c.
3. We select items from a list of n items.
4. Each item has both a weight of w_i and profit of p_i.
5. In a feasible solution, the sum of the weights must not exceed the knapsack capacity (c) and an optimal solution is both feasible and reaches the maximum profit.
6. An optimal packing is a feasible solution one with a maximum profit :

$$p_1x_1 + p_2x_2 + p_3x_3 + ... + p_nx_n = \sum_{i=1}^{n} p_ix_i$$

which is subjected to constraints :

$$p_1x_1 + w_2x_2 + w_3x_3 + ... + w_nx_n = \sum_{i=1}^{n} w_ix_i \le c$$

and
$$x_i = 1 \text{ or } 0, \quad 1 \le i \le n$$

7. We have to find the values of x_i where $x_i = 1$ if i^{th} item is packed into the knapsack and $x_i = 0$ if i^{th} item is not packed.

Greedy strategies for the knapsack problem are :

i. From the remaining items, select the item with maximum profit that fits into the knapsack.

ii. From the remaining items, select the item that has minimum weight and also fits into the knapsack.

iii. From the remaining items, select the one with maximum p_i/w_i that fits into the knapsack.

Greedy method is not effective to solve the 0/1-knapsack problem. By using greedy method we do not get optimal solution.

Que 3.23. **What is 0/1-knapsack problem? Solve the following instance using greedy approach, also write the algorithm. Knapsack capacity = 10, $P = <1, 6, 18, 22, 28>$ and $w = <1, 2, 5, 6, 7>$.**

Answer

0/1 knapsack problem : Refer Q. 3.22, Page 3–21B, Unit-3.

Numerical :

Knapsack capacity = 10

$$P = < 1, 6, 18, 22, 28 >$$
$$W = < 1, 2, 5, 6, 7 >$$

$\therefore$ $$P_i = \frac{V_i}{W_i}$$

$\therefore$ $$V_i = P_i \times W_i$$

Item	Weight	Value	Value/Weight
I_1	7	196	28
I_2	6	132	22
I_3	5	90	18
I_4	2	12	6
I_5	1	1	1

To fulfill the capacity $W = 10$

Add item of weight 7. (knapsack capacity = 10 – 7 = 3)

Skip item of weight 6 since available knapsack capacity is less than 6.

Skip item of weight 5 since available knapsack capacity is 3.

Add item of weight 2. (knapsack capacity = 3 – 2 = 1)

Add item of weight 1.

Maximum Value = 196 + 12 + 1
= 209

Que 3.24. **Consider following instance for simple knapsack problem. Find the solution using greedy method.**

$N = 8$

$P = \{11, 21, 31, 33, 43, 53, 55, 65\}$

$W = \{1, 11, 21, 23, 33, 43, 45, 55\}$

AKTU 2016-17, Marks 7.5

$M = 110$

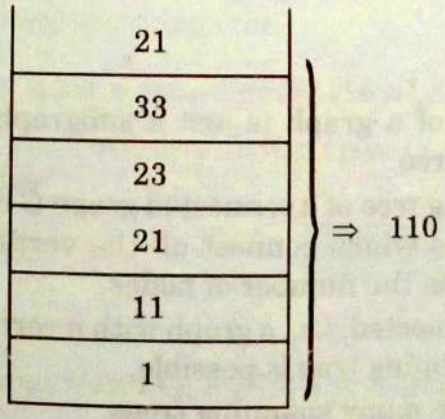
Answer

$N = 8$
$W = \{1, 11, 21, 23, 33, 43, 45, 55\}$
$P = \{11, 21, 31, 33, 43, 53, 55, 65\}$
$M = 110$
Now, arrange the value of P_i in decreasing order

N	W_i	P_i	$V_i = W_i \times P_i$
1	1	11	11
2	11	21	231
3	21	31	651
4	23	33	759
5	33	43	1419
6	43	53	2279
7	45	55	2475
8	55	65	3575

Now, fill the knapsack according to decreasing value of P_i. First we choose item $N = 1$ whose weight is 1.
Then choose item $N = 2$ whose weight is 11.
Then choose item $N = 3$ whose weight is 21.
Now, choose item $N = 4$ whose weight is 23.
Then choose item $N = 5$ whose weight is 33.
Total weight in knapsack is $= 1 + 11 + 21 + 23 + 33 = 89$
Now, the next item is $N = 6$ and its weight is 43, but we want only 21 because $M = 110$.
So, we choose fractional part of it, *i.e.*,
The value of fractional part of $N = 6$ is,

$$\frac{2279}{43} \times 21 = 1113$$



Thus, the maximum value is,

$$= 11 + 231 + 651 + 759 + 1419 + 1113$$
$$= 4184$$

PART-3

Minimum Spanning Trees-Prim's and Kruskal's Algorithm, Single Source Shortest Paths-Dijkstra's and Bellman-Ford Algorithm.

CONCEPT OUTLINE : PART-3

- **Minimum spanning tree :** A spanning tree of a graph is just a subgraph that contains all the vertices and does not contain any cycle.
- There are two techniques of finding minimum spanning tree which are as follows :
  a. Kruskal's algorithm
  b. Prim's algorithm
- **Single source shortest paths problem :** Given a graph $G = (V, E)$, we want to find a shortest path from a given vertex (source) $S \in V$ to every vertex $v \in V$.
- **Algorithms to single source shortest path problem :**
  i. Dijkstra's algorithm
  ii. Bellman-Ford algorithm

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.25. What do you mean by spanning tree and minimum spanning tree ?

Answer

Spanning tree :

1. A spanning tree of a graph is just a subgraph that contains all the vertices and is a tree.
2. That is, a spanning tree of a connected graph G contains all the vertices and has the edges which connect all the vertices. So, the number of edges will be 1 less the number of nodes.
3. If graph is not connected, *i.e.*, a graph with n vertices has edges less than $n - 1$ then no spanning tree is possible.
4. A graph may have many spanning trees.

Minimum spanning tree :

1. Given a connected weighted graph G, it is often desired to create a spanning tree T for G such that the sum of the weights of the tree edges in T is as small as possible.

2. Such a tree is called a minimum spanning tree and represents the "cheapest" way of connecting all the nodes in G.

3. There are number of techniques for creating a minimum spanning tree for a weighted graph but the most famous methods are Prim's and Kruskal's algorithm.

Que 3.26. Write Kruskal's algorithm to find minimum spanning tree.

Answer

i. In this algorithm, we choose an edge of G which has smallest weight among the edges of G which are not loops.

ii. This algorithm gives an acyclic subgraph T of G and the theorem given below proves that T is minimal spanning tree of G. Following steps are required :

Step 1 : Choose e_1, an edge of G, such that weight of e_1, $w(e_1)$ is as small as possible and e_1 is not a loop.

Step 2 : If edges $e_1, e_2, \ldots\ldots, e_i$ have been selected then choose an edge e_{i+1} not already chosen such that

i. the induced subgraph

$$G[\{e_1 \ldots\ldots e_{i+1}\}] \text{ is acyclic and}$$

ii. $w(e_{i+1})$ is as small as possible

Step 3 : If G has n vertices, stop after $n - 1$ edges have been chosen. Otherwise repeat step 2.

If G be a weighted connected graph in which the weight of the edges are all non-negative numbers, let T be a subgraph of G obtained by Kruskal's algorithm then, T is minimal spanning tree.

Que 3.27. Describe and compare following algorithms to determine the minimum cost spanning tree :

i. Kruskal's algorithm

ii. Prim's algorithm

AKTU 2013-14, Marks 10

Answer

i. **Kruskal's algorithm :** Refer Q. 3.26, Page 3–25B, Unit-3.

ii. **Prim's algorithm :**

First it chooses a vertex and then chooses an edge with smallest weight incident on that vertex. The algorithm involves following steps :

Step 1 : Choose any vertex V_1 of G.

Step 2 : Choose an edge $e_1 = V_1V_2$ of G such that $V_2 \neq V_1$ and e_1 has smallest weight among the edge e of G incident with V_1.

Step 3 : If edges $e_1, e_2, \ldots\ldots, e_i$ have been chosen involving end points $V_1, V_2, \ldots\ldots\ldots, V_{i+1}$, choose an edge $e_{i+1} = V_jV_k$ with $V_j = \{V_1 \ldots\ldots V_{i+1}\}$ and $V_k \notin \{V_1 \ldots\ldots\ldots V_{i+1}\}$ such that e_{i+1} has smallest weight among the edges of G with precisely one end in $\{V_1 \ldots\ldots\ldots V_{i+1}\}$.

Step 4 : Stop after $n - 1$ edges have been chosen. Otherwise goto step 3.

Comparison :

S. No.	Kruskal's algorithm	Prim's algorithm
1.	Kruskal's algorithm initiates with an edge.	Prim's algorithm initializes with a node.
2.	Kruskal's algorithm selects the edges in a way that the position of the edge is not based on the last step.	Prim's algorithms span from one node to another.
3.	Kruskal's can function on disconnected graphs too.	In prim's algorithm, graph must be a connected graph.
4.	Kruskal's time complexity in worst case is $O(E \log E)$.	Prim's algorithm has a time complexity in worst case of $O(E \log V)$.

Que 3.28. What do you mean by minimum spanning tree ? Write an algorithm for minimum spanning tree that may generate multiple forest trees and also explain with suitable example.

AKTU 2014-15, Marks 10

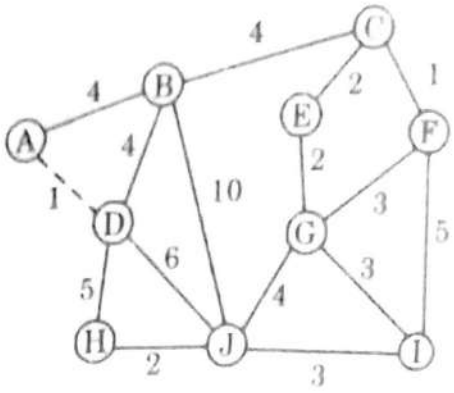
Answer

Minimum spanning tree : Refer Q. 3.25, Page 3–24B, Unit-3.

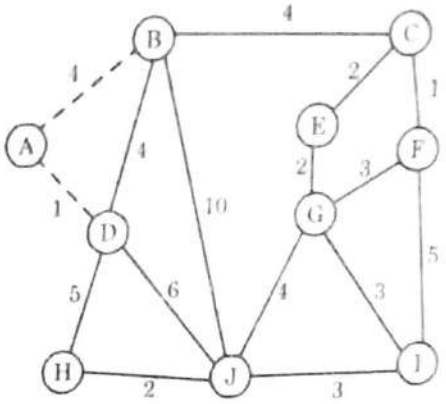
Prim's algorithm : Refer Q. 3.27, Page 3–25B, Unit-3.

Example :

According to algorithm we choose vertex A from the set $\{A, B, C, D, E, F, G, H, I, J\}$.



Now edge with smallest weight incident on A is $e = (AD)$



Now we look on weight

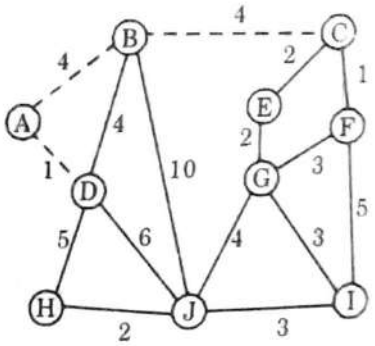
$W(A, B) = 4$

$W(D, B) = 4 \quad W(D, H) = 5$

$W(D, J) = 6$

We choose $e = AB$ since it is minimum.

$W(D, B)$ can also be chosen because it has same value.



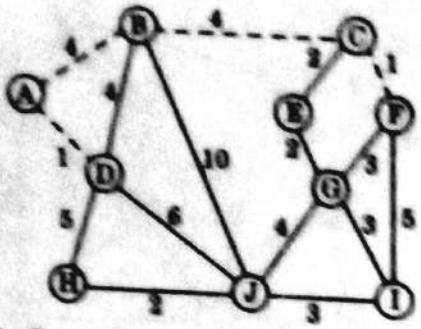
Again, $W(B, C) = 4$

$W(B, J) = 10$

$W(D, H) = 5$

$W(D, J) = 6$

We choose $e = BC$ since it has minimum value.

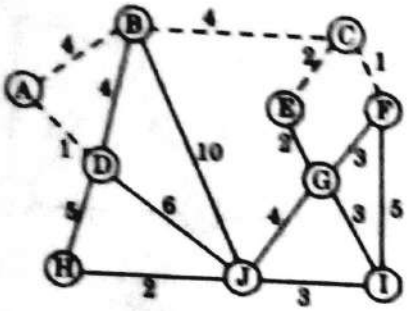


Now, $W(B, J) = 10$

$W(C, E) = 2$

$W(C, F) = 1$

We choose $e = CF$ because $W(C, F)$ has minimum value.

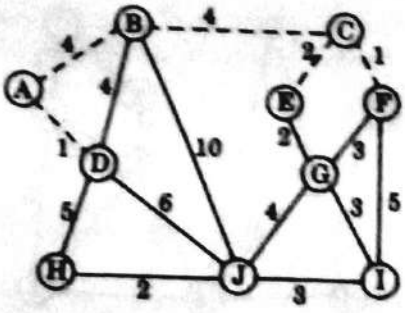


Now, $W(C, E) = 2$

$W(F, G) = 3$

$W(F, I) = 5$

We choose $e = CE$, since $W(C, E)$ is minimum.

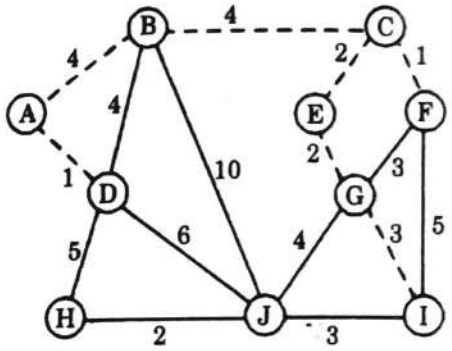


$W(E, G) = 2$

$W(F, G) = 3$

$W(F, I) = 5$

We choose $e = EG$, since $W(E, G)$ is minimum.

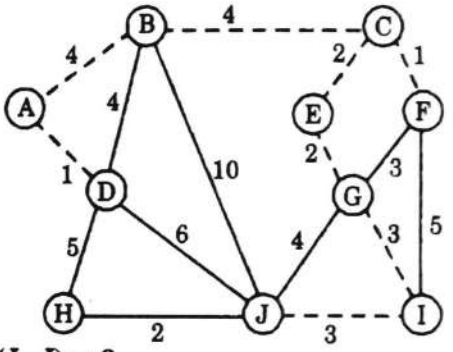


$W(G, J) = 4$

$W(G, I) = 3$

$W(F, I) = 5$

We choose $e = GI$, since $W(G, I)$ is minimum.



$W(I, J) = 3$

$W(G, J) = 4$

We choose $e = IJ$, since $W(I, J)$ is minimum

$W(J, H) = 2$

Hence, $e = JH$ will be chosen.

The final minimal spanning tree is given as :

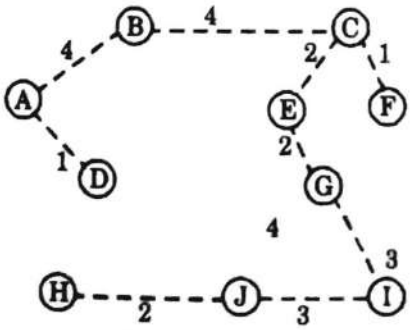


Fig. 3.28.1.

Que 3.29. What is minimum cost spanning tree ? Explain Kruskal's algorithm and find MST of the graph. Also write its time complexity.

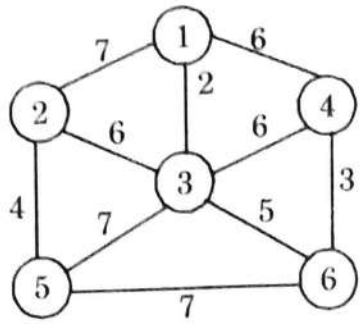


Fig. 3.29.1.

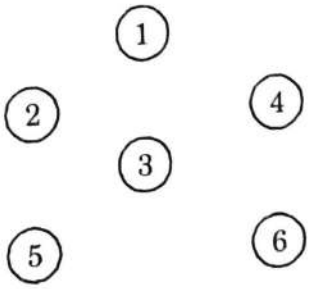
AKTU 2017-18, Marks 10

Answer

Minimum spanning tree : Refer Q. 3.25, Page 3–24B, Unit-3.
Kruskal's algorithm : Refer Q. 3.26, Page 3–25B, Unit-3.
Numerical :
Step 1 : Arrange the edge of graph according to weight in ascending order.

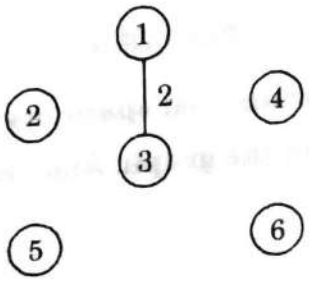
Edges	Weight	Edge	Weight
13	2	32	6
46	3	17	7
25	4	35	7
36	5	56	7
34	6		
41	6		

Step 2 : Now draw the vertices as given in graph,

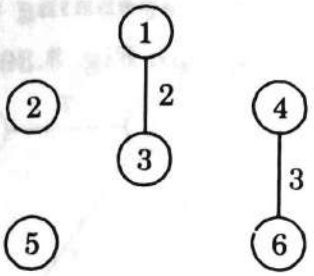


Now draw the edge according to the ascending order of weight. If any edge forms cycle, leave that edge.

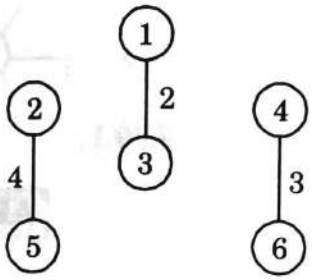
Step 3 :



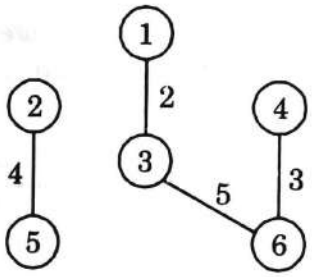
Step 4 :



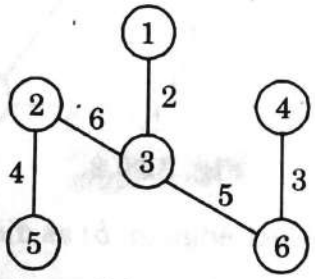
Step 5 :



Step 6 :

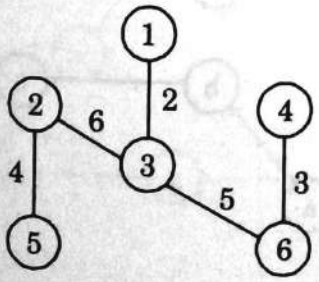


Step 7 :



All the remaining edges, such as : 34, 41, 12, 35, 56 are rejected because they form cycle.

All the vertices are covered in this tree. So, the final tree with minimum cost of given graph is



Time complexity : Time complexity is $O(|E| \log |E|)$.

Que 3.30. **What is minimum spanning tree ? Explain Prim's algorithm and find MST of graph Fig. 3.30.1.**

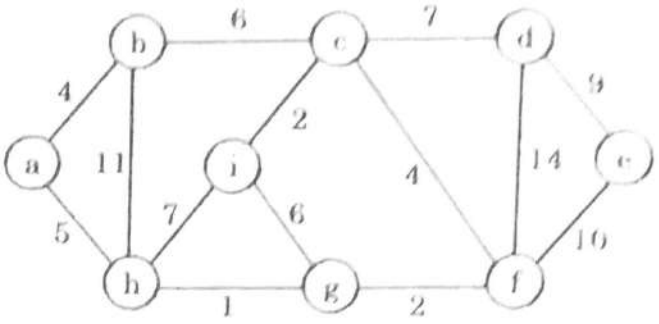


Fig. 3.30.1.

AKTU 2015-16, Marks 10

Answer

Minimum spanning tree : Refer Q. 3.25, Page 3–24B, Unit-3.
Prim's algorithm : Refer Q. 3.27, Page 3–25B, Unit-3.
Numerical :

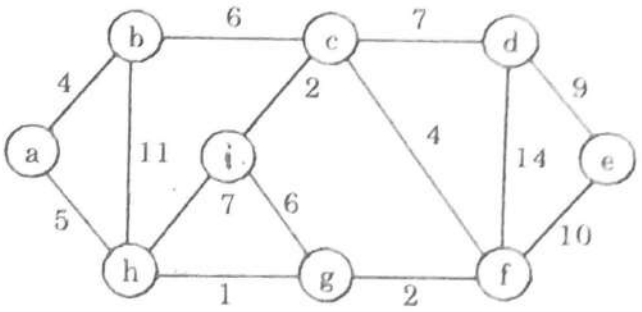
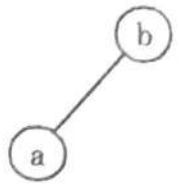
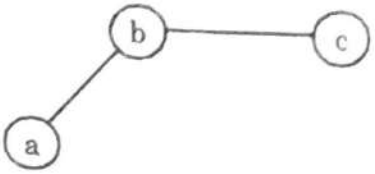


Fig. 3.30.2.

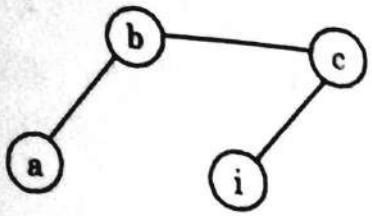
Let a be the source node. Select edge (a, b) as distance between edge (a, b) is minimum.



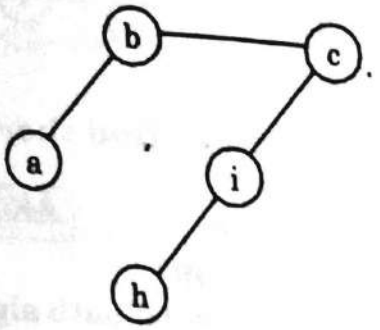
Now, select edge (b, c)



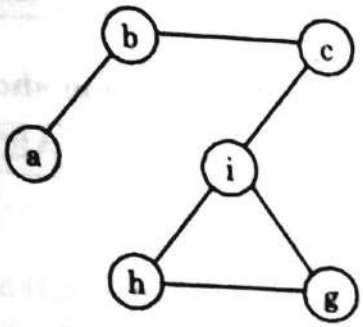
Now, select edge (c, i)



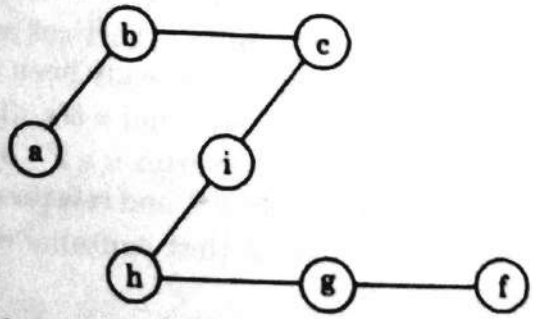
Now, select edge (i, h)



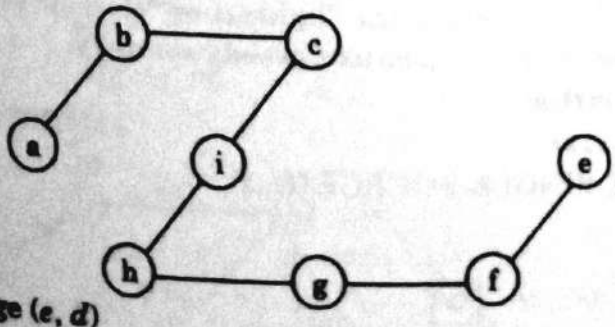
Now, select edge (h, g)



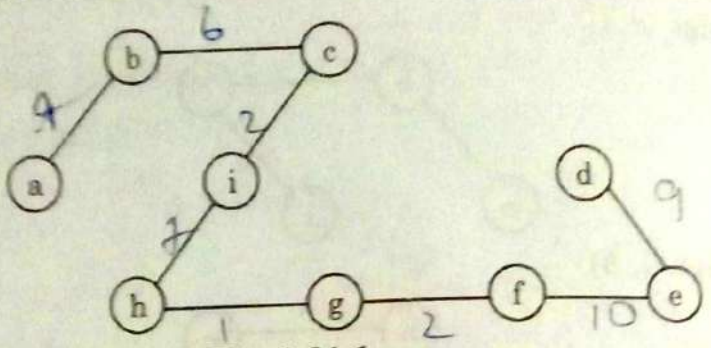
Now, select edge (g, f)



Now, select edge (f, e)



Now, select edge (e, d)



Thus, we obtained MST for Fig. 3.31.1.

Que 3.31. Write an algorithm to find shortest path between all pairs of nodes in a given graph. **AKTU 2013-14, Marks 10**

OR

Explain greedy single source shortest path algorithm with example. **AKTU 2015-16, Marks 10**

OR

Write short note on Dijkstra's algorithm shortest paths problems. **AKTU 2016-17, Marks 10**

Answer

1. Dijkstra's algorithm, is a greedy algorithm that solves the single source shortest path problem for a directed graph $G = (V, E)$ with non-negative edge weights, *i.e.*, we assume that $w(u, v) \geq 0$ each edge $(u, v) \in E$.
2. Dijkstra's algorithm maintains a set S of vertices whose final shortest path weights from the source s have already been determined.
3. That is, for all vertices $v \in S$, we have $d[v] = \delta(s, v)$.
4. The algorithm repeatedly selects the vertex $u \in V - S$ with the minimum shortest path estimate, inserts u into S, and relaxes all edges leaving u.
5. We maintain a priority queue Q that contains all the vertices in $v - s$, keyed by their d values.
6. Graph G is represented by adjacency list.
7. Dijkstra's always chooses the "lightest or "closest" vertex in $V - S$ to insert into set S that it uses as a greedy strategy.

Dijkstra's algorithm :

DIJKSTRA (G, w, s)

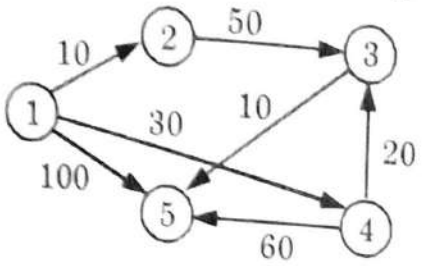
1. INITIALIZE-SINGLE-SOURCE (G, s)
2. $s \leftarrow \phi$
3. $Q \leftarrow V[G]$
4. while $Q \neq \phi$
5. do $u \leftarrow$ EXTRACT-MIN (Q)
6. $S \leftarrow S \cup \{u\}$
7. for each vertex $v \in$ Adj $[u]$

8. do RELAX (u, v, w)

RELAX (u, v, w) :

1. If $d[u] + w(u, v) < d[v]$
2. then $d[v] \leftarrow d[u] + w(u, v)$
3. $\pi[v] \leftarrow u$

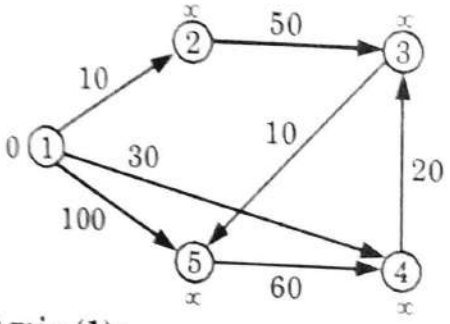
Que 3.32. Find the shortest path in the below graph from the source vertex 1 to all other vertices by using Dijkstra's algorithm.



AKTU 2017-18, Marks 10

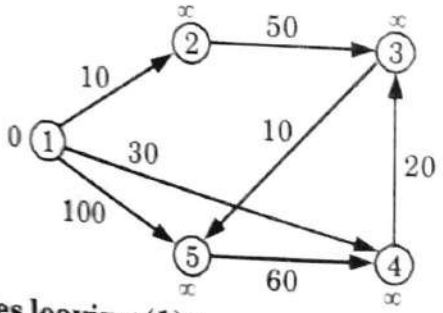
Answer

Initialize :



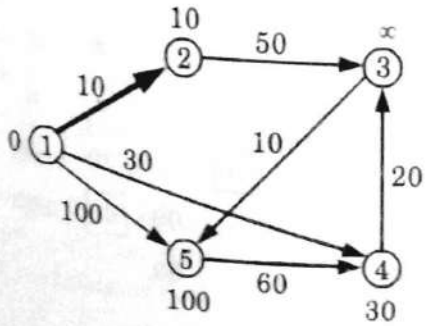
1	2	3	4	5
0	∞	∞	∞	∞

Extract min (1) :



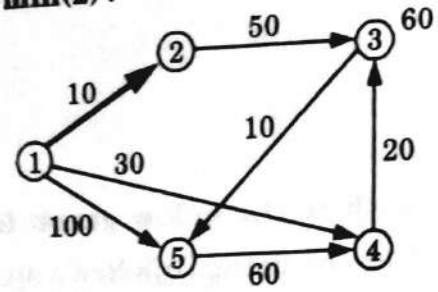
1	2	3	4	5
[0]	∞	∞	∞	∞

All edges leaving (1) :



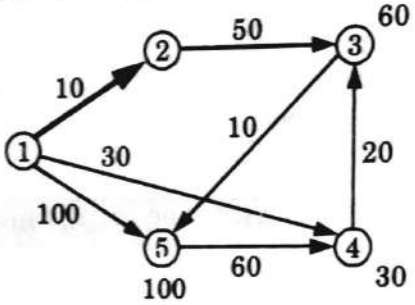
1	2	3	4	5
[0]	∞	∞	∞	∞
	10	∞	30	100

Extract min(2) :



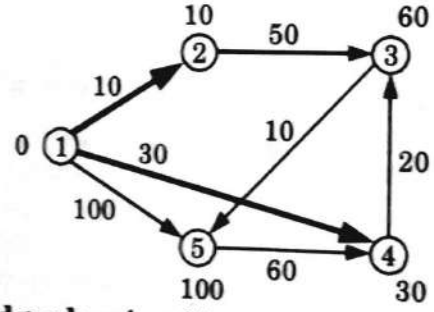
1	2	3	4	5
0	∞	∞	∞	∞
	10	60	30	100

All edges leaving (2) :



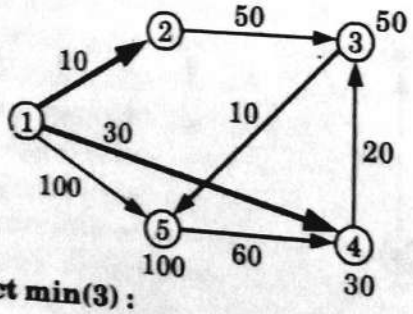
1	2	3	4	5
0	∞	∞	∞	∞
	10	60	30	100
		60	30	100

Extract min(4) :



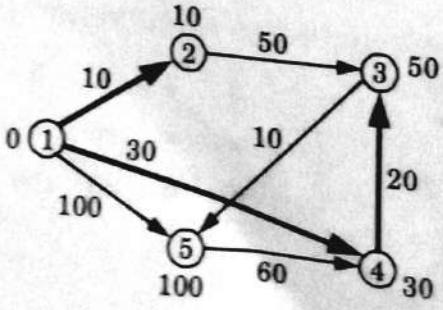
1	2	3	4	5
0	∞	∞	∞	∞
	10	60	30	100
		60	30	100

All edges leaving (4) :



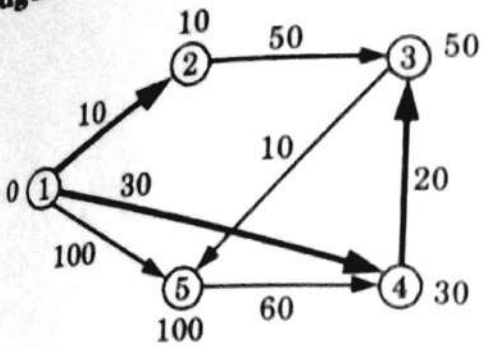
1	2	3	4	5
0	∞	∞	∞	∞
	10	60	30	100
		60	30	100
		50		

Extract min(3) :



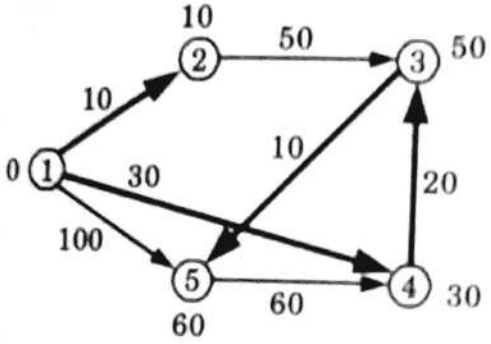
1	2	3	4	5
0	∞	∞	∞	∞
	10	∞	30	100
		60	30	100
		50		

All edges leaving (3) :



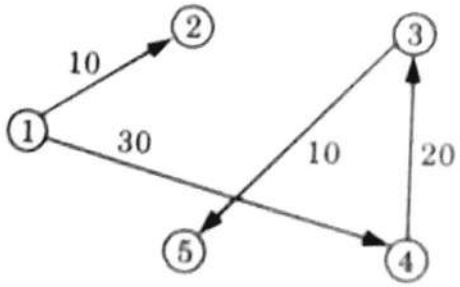
1	2	3	4	5
[0]	∞	∞	∞	∞
	[10]	∞	30	100
		60	[30]	100
		[50]		60

Extract min(5) :



1	2	3	4	5
[0]	∞	∞	∞	∞
	[10]	∞	30	100
		60	[30]	100
		[50]		
				[60]

Shortest path



Que 3.33. State Bellman-Ford algorithm.

AKTU 2016-17, Marks 7.5

Answer

1. Bellman-Ford algorithm finds all shortest path length from a source $s \in V$ to all $v \in V$ or determines that a negative-weight cycle exists.
2. Bellman-Ford algorithm solves the single source shortest path problem in the general case in which edges of a given digraph G can have negative weight as long as G contains no negative cycles.
3. This algorithm, uses the notation of edge relaxation but does not use with greedy method.
4. The algorithm returns boolean TRUE if the given digraph contains no negative cycles that are reachable from source vertex otherwise it returns boolean FALSE.

Bellman-Ford (G, w, s) :

1. INITIALIZE-SINGLE-SOURCE (G, s)
2. for each vertex $i \leftarrow 1$ to $V[G] - 1$ do
3. for each edge (u, v) in $E[G]$ do
4. RELAX (u, v, w)
5. For each edge (u, v) *in* $E[G]$ do
6. if $d[u] + w(u, v) < d[v]$ then
7. return FALSE
8. return TRUE

RELAX (u, v, w) :

1. If $d[u] + w(u, v) < d[v]$
2. then $d[v] \leftarrow d[u] + w(u, v)$
3. $\pi[v] \leftarrow u$

If Bellman-Ford returns true, then G forms a shortest path tree, else there exists a negative weight cycle.

Que 3.34. Given a weighted directed graph $G = (V, E)$ with source and weight function $W : E \rightarrow R$ then write an algorithm to solve a single source shortest path problem whose complexity is O (VE). Apply the same on the following graph.

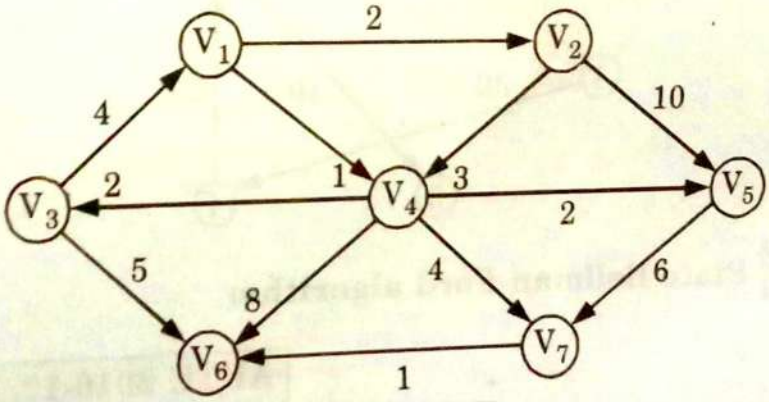


Fig. 3.34.1.

AKTU 2014-15, Marks 10

Answer

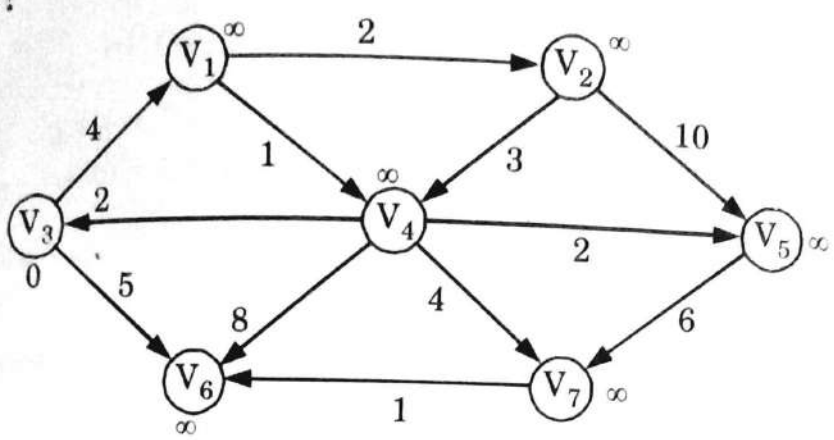
Bellman-Ford algorithm is used to solve single source shortest path problem whose complexity is O(VE).

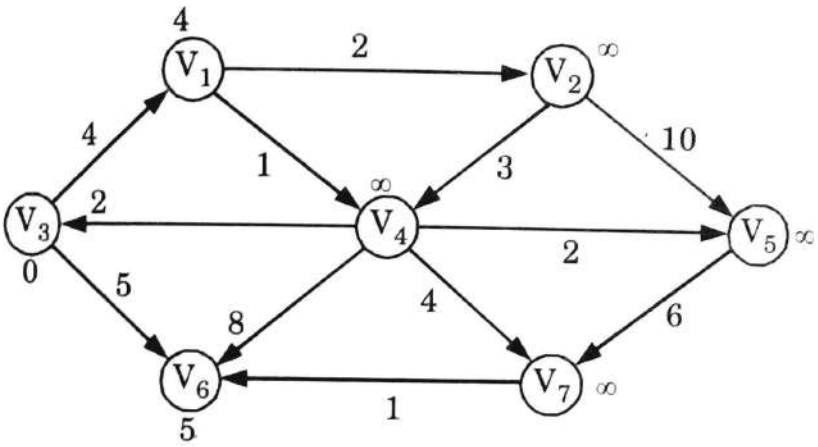
Bellman-Ford algorithm : Refer Q. 3.33, Page 3–37B, Unit-3.

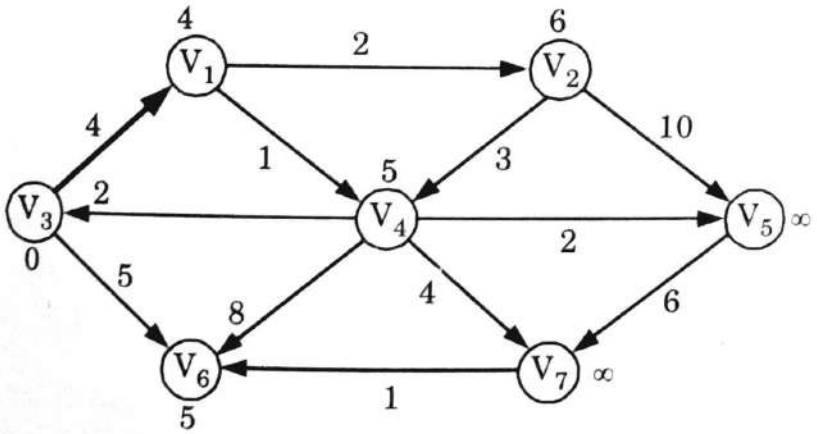
Numerical :

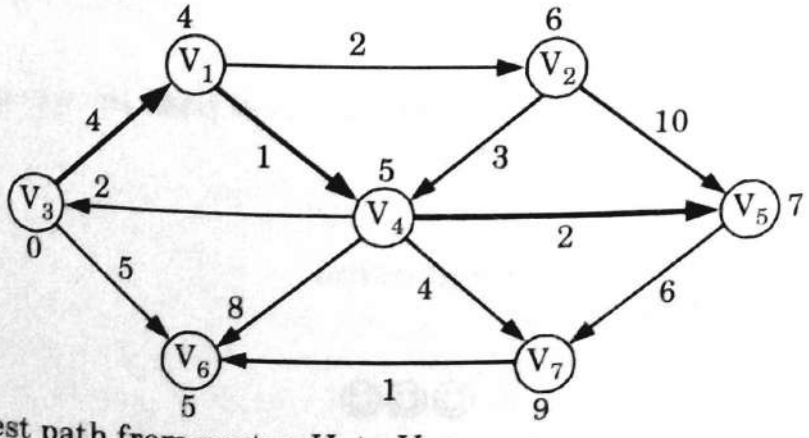
Let V_3 is source

Initialize :









Shortest path from vertex V_3 to V_5

$= d_{V_3V_1} + d_{V_1V_4} + d_{V_4V_5} = 4 + 1 + 2 = 7$

VERY IMPORTANT QUESTIONS

Following questions are very important. These questions may be asked in your SESSIONALS as well as UNIVERSITY EXAMINATION.

Q. 1. Discuss matrix chain multiplication problem and its solution.

Ans. Refer Q. 3.2.

Q. 2. Explain graphs with its representations.

Ans. Refer Q. 3.4.

Q. 3. Write short note on convex null problem.

Ans. Refer Q. 3.11.

Q. 4. What is greedy algorithm ? Write its pseudo code for recursive and iterative process.

Ans. Refer Q. 3.17.

Q. 5. Discuss knapsack problem.

Ans. Refer Q. 3.19.

Q. 6. Write short note on the following :

a. Minimum spanning tree

b. Kruskal's algorithm

c. Prim's algorithm

Ans.

a. Refer Q. 3.25.

b. Refer Q. 3.26.

c. Refer Q. 3.27.

Q. 7. Write an algorithm to find shortest path between all pairs of nodes in a given graph.

Ans. Refer Q. 3.31.

Q. 8. State Bellman Ford algorithm.

Ans. Refer Q. 3.33.