3 UNIT

Graph Algorithms

CONTENTS

PART-1

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

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.1. Write short note on divide and conquer. Write algorithm for merge sort and quick sort.

Answer

Divide and conquer :

1. Divide and conquer is an algorithm design paradigm based on multi-branched recursion.
2. A divide-and-conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly.
3. The solutions to the sub-problems are then combined to give a solution to the original problem.
4. Divide and conquer technique can be divided into the following three parts :
   a. **Divide :** This involves dividing the problem into some sub-problem.
   b. **Conquer :** Recursively calling sub-problem until it is solved.
   c. **Combine :** This involves combination of solved sub-problem so that we will get the solution of problem.

Merge Sort : Refer Q. 1.21, Page 1–20B, Unit-1.

Quick Sort : Refer Q. 1.19, Page 1–16B, Unit-1.

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

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.

Que 3.3. Describe in detail Strassen's matrix multiplication algorithms based on divide and conquer strategies with suitable example.

Answer

Strassen's matrix multiplication algorithm 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, ...A_7, B_7.
3. Recursively compute the seven matrix products.

$P_i = A_iB_i$ for $i = 1, 2, 3, ..., 7$

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.

Suppose, $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ $B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$ and $C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

In Strassen method first compute the 7 $n/2 \times n/2$ matrices.

P, Q, R, S, T, U, V as follows :

$$P = (A_{11} + A_{22})\ (B_{11} + B_{22})\ Q$$
$$= (A_{21} + A_{22})\ B_{11}$$
$$R = A_{11}\ (B_{12} - B_{22})\ S$$
$$= A_{22}\ (B_{21} - B_{11})\ T$$
$$= (A_{11} + A_{12})\ B_{22}$$
$$U = (A_{21} - A_{11})\ (B_{11} + B_{12})$$
$$V = (A_{12} - A_{22})\ (B_{21} + B_{22})$$

Then the c_{ij}'s are computed using the formulas

$$c_{11} = P + S - T + V$$
$$c_{12} = R + T$$
$$c_{21} = Q + S$$
$$c_{22} = P + R - Q + U$$

As can be seen P, Q, R, S, T, U and V can be computed using 7 matrix multiplication and 10 matrix addition or subtraction and c_{ij}'s require an additional 8 addition or subtraction.

For example :

Suppose,

$$A = \begin{bmatrix} 2 & 9 \\ 5 & 6 \end{bmatrix} \quad B = \begin{bmatrix} 4 & 11 \\ 8 & 7 \end{bmatrix} \quad C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$$AB = C$$

$$\begin{bmatrix} 2 & 9 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 4 & 11 \\ 8 & 7 \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$\lfloor \quad \rfloor\!\lfloor \quad \rfloor \; \lfloor \; {}_{21} \quad {}_{22} \rfloor$

So, $A_{11} = 2, B_{11} = 4$
$A_{12} = 9, B_{12} = 11$
$A_{21} = 5, B_{21} = 8$
$A_{22} = 6, B_{22} = 7$

Now calculate,

$S_1 = B_{12} - B_{22} = 5,$ $S_6 = B_{11} + B_{12} = 15$
$S_2 = A_{11} + A_{12} = 11,$ $S_7 = A_{12} - A_{22} = 3$

$S_3 = A_{21} + A_{22} = 11$, $S_8 = B_{21} + B_{22} = 15$
$S_4 = B_{21} - B_{11} = 4$, $S_9 = A_{12} - A_{22} = 3$
$S_5 = A_{11} + A_{22} = 8$, $S_{10} = B_{11} + B_{12} = 15$

Now, $R = A_{11} \times S_1 = 10$, $P = S_5 \times S_6 = 120$
$T = S_2 \times B_{22} = 77$, $U = S_7 \times S_8 = 45$
$Q = S_3 \times B_{11} = 44$, $V = S_9 \times S_{10} = 45$
$S = A_{22} \times S_4 = 24$

Now, $C_{11} = P + S - T + V = 88 + 24 - 77 + 45 = 80$
$C_{12} = R + T = 10 + 77 = 87$
$C_{21} = Q + S = 44 + 24 = 68$
$C_{22} = P + R - Q + U = 88 + 10 - 44 + 45 = 99$

Now matrix = $\begin{bmatrix} 80 & 71 \\ 68 & 99 \end{bmatrix}$

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

Answer

1. A graph G consists of a set of vertices V and a set of edges E.
2. Graphs are important because any binary relation is a graph, so graphs can be used to represent essentially any relationship.

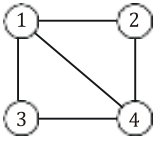


Fig. 3.4.1.

Various representations 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. Explain DFS. Also give DFS 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.

Algorithm :

In DFS, each vertex v has two timestamps : the first timestamp $d[v]$ *i.e.*, discovery time records when v is first discovered *i.e.*, grayed, and the second timestamp $f[v]$ *i.e.* finishing time records when the search finishes examining v's adjacency list *i.e.*, blacked. For every vertex $d[u] < f[u]$.

DFS(G) :

```
1.  for each vertex u ∈ V[G]
2.      do color[u] ← WHITE
3.          π[u] ← NIL
4.  time ← 0
5.  for each vertex u ∈ V[G]
6.      do if color[u] = WHITE
7.          then DFS-VISIT(u)
```

DFS-VISIT(u) :

```
1.  color[u] ← GRAY          // White vertex u has just been discovered.
2.  time ← time + 1
3.  d[u] ← time
4.  for each v ∈ Adj[u]      // Explore edge (u, v)
5.      do if color[v] = WHITE
6.          then π[v] ← u
7.              DFS-VISIT (v)
8.  color[u] ← BLACK         // Blacken u, it is finished.
9.  f[u] ← time ← time + 1
```

Que 3.6. Explain Breadth First Search (BFS). Give its algorithm.

Answer

Breadth first search :

1. The general idea behind a breadth first search 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.

Algorithm :

BFS (G, s) :

```
1.  for each vertex u ∈ V[G] – {s}
2.      do color[u] ← WHITE
```

```
3.              d[u] ← ∞
4.              π[u] ← NIL
5.      color[s] ← GRAY
6.      d[s] ← 0
7.      π[s] ← NIL
8.      Q ← ϕ
9.      ENQUEUE(Q, s)
10.     while Q ≠ ϕ
        do c


11.         do u ← DEQUEUE(Q)
12.             for each v ∈ Adj[u]
13.                 do if color [v] = WHITE
14.                     then color[v] ← GRAY
15.                         d[v] ← d[u] + 1
16.                         π[v] ← u
17.                         ENQUEUE(Q, v)
18.                 color[u] ← BLACK
```

Que 3.7. **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 3.8. **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.
3. Kosaraju's algorithm is an algorithm to find the strongly connected components of a directed graph.

Kosaraju's algorithm :

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.

Que 3.9. Explain convex hull problem.

AKTU 2017-18, Marks 10

OR

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

OR

What do you mean by convex hull ? Describe an algorithm that solves the convex hull problem. Find the time complexity of the algorithm.

AKTU 2019-20, Marks 07

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.
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.

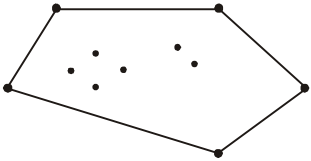


Fig. 3.9.1.

4. If p lies in any 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 $<p_1, p_2, \ldots\ldots, p_m>$ be the remaining points in Q, sorted by polar angle in counter clockwise order around p_o (if more than one point has the same angle remove all but the one that is farthest from p_o).
3. Top [S] $\leftarrow$ 0
4. PUSH (p_0, S)
5. PUSH (p_1, S)
6. PUSH (p_2, S)
7. for $i \leftarrow 3$ to m
8. do while the angle formed by points NEXT-To-TOP(S), Top(S), and p_i makes a non left turn.
9. do POP(S)
10. PUSH (p_i, S)
11. return S

Time complexity :

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.

PART-2

Greedy Methods with Examples such as Optimal Reliability Allocation.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.10. **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 3.11. **What are the four functions included in greedy algorithm ? Write structure of greedy algorithm.**

Answer

The greedy algorithm consists of four functions :

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 3.12. **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 $S = \{1, 2, ..., n\}$ 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.

i	s_i	f_i
1	1	4
2	3	5
3	0	6
4	5	7
4	3	8
6	5	9
7	6	10
8	8	11
9	8	12
10	2	13
11	12	14
12	∞	-

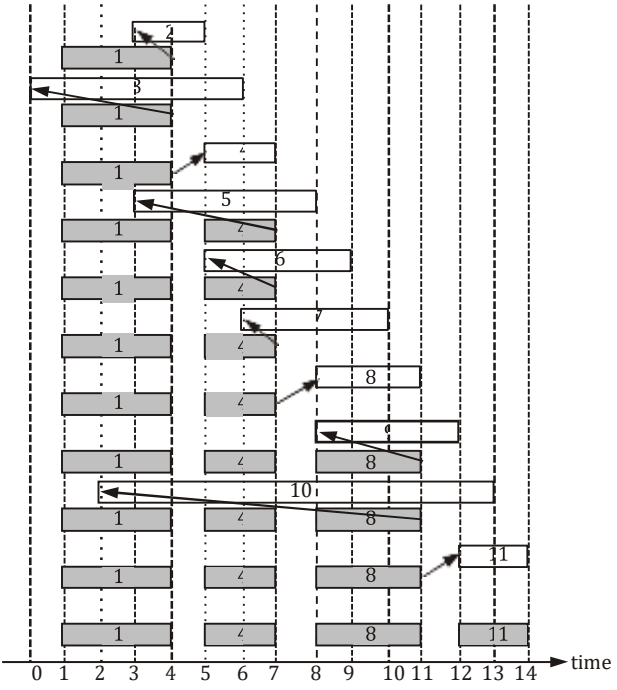


Fig. 3.12.1.

Greedy algorithm :

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

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 $A \leftarrow A \cup \{(m)\}$
7. $i \leftarrow m$
8. return A

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 3.13. **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. 3.10, Page 3–11B, 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. Activity selection problem uses greedy approach.

Algorithm for greedy activity selection : Refer Q. 3.12, Page 3–12B, 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_7 is not selected

Check for activity a_8

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

$\therefore$ a_8 is not selected

Check for activity a_9

Starting time of $a_9 \geq$ 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_9

$\therefore$ 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 3.14. What is "Greedy Algorithm" ? Write its pseudocode for recursive and iteration process.

Answer

Greedy algorithm : Refer Q. 3.10, Page 3–11B, 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 3.15. What is an optimization problem ? How greedy method can be used to solve the optimization problem ?

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.

PART-3

Knapsack.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.16. 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.

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... I_n having weight w_1, w_2...w_n in such a manner that the total weight of items cannot exceed the capacity of knapsack and maximum possible value (v) 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 in which left branch implies inclusion and right implies exclusion.
4. Upper bound of node can be calculated as :
   $ub = v + (W - w_{i+1}) (v_{i+1} / w_{i+1})$

Que 3.17. Write greedy algorithm for discrete knapsack problem with example.

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.

For 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, $W = 10 - 4 = 6$
2. Skip item of weight 7
3. Add item of weight 5, $W = 6 - 5 = 1$
4. Skip item of weight 3

Maximum value = 10.5 + 6 = 16.5

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

Que 3.18. 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 :

$$w_1x_1 + w_2x_2 + w_3x_3 + ... + w_nx_n = \sum_{i=1}^{n} w_ix_i \leq c$$

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

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 3.19. 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**	**0.4**
B	**50**	**35**	**0.7**
C	**40**	**20**	**0.5**
D	**20**	**4**	**0.2**
E	**10**	**10**	**1**
F	**10**	**6**	**0.6**

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.

First arrange in non-increasing order of value/weight :

Item ID	Weight	Value	Value/Weight
E	10	10	1
B	50	35	0.7
F	10	6	0.6
C	40	20	0.5
A	100	40	0.4
D	20	4	0.2

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

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

Weight of $C = 30/40 = 0.75$

So, $W = 0.75 \times 40 = 30$

$W = 70 + 30 = 100$

Total value $= 10 + 35 + 6 + 0.75\ (20)$

$= 10 + 35 + 6 + 15 = 66$

Que 3.20. **Consider the weight and values of item listed below. Note that there is only one unit of each item. The task is to pick a subset of these items such that their total weight is no more than 11 kgs and their total value is maximized. Moreover, no item may be split. The total value of items picked by an optimal algorithm is denoted by V_{opt}. A greedy algorithm sorts the items by their value-to-weight rations in descending order and packs them greedily, starting from the first item in the ordered list. The total value of items picked by the greedy algorithm is denoted by V_{greedy}. Find the value of $V_{opt} - V_{greedy}$.**

Item	I_1	I_2	I_3	I_4
W	10	7	4	2
V	60	28	20	24

AKTU 2018-19, Marks 07

Answer

For V_{greedy} :

Item	W	V
I_1	10	60
I_2	7	28
I_3	4	20
I_4	2	24

Arrange the items by V/W ratio in descending order :

Item	W	V	V/W
I_4	2	24	12
I_1	10	60	6
I_3	4	20	5
I_2	7	28	4

Total weight $W = 11$ kg

I_4 is picked so $W = 11 - 2 = 9$ kg

I_1 cannot picked $10 > 9$

I_3 is picked, $W = 9 - 4 = 5$ kg

I_2 cannot be picked $7 > 5$

I_4 and I_3 are picked so

$$V_{greedy} = V(I_4) + V(I_3) = 24 + 20 = 44$$

For V_{opt} : For calculating V_{opt} we use 0/1 knapsack problem, so only item 1 is picked. Hence, $V_{opt} = 60$

So, $V_{opt} - V_{greedy} = 60 - 44 = 16$

Que 3.21. **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}**

***M* = 110**

AKTU 2016-17, Marks 7.5

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.

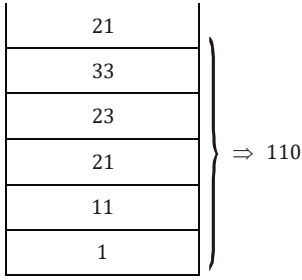
Total weight in knapsack is = 1 + 11 + 21 + 23 + 33 = 89

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,

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

Que 3.22. **Solve the following 0/1-knapsack problem using dynamic programming $P = \{11, 21, 31, 33\}$ $w = \{2, 11, 22, 15\}$ $c = 40$, $n = 4$.**

AKTU 2019-20, Marks 07

Answer

Numerical :

$w = \{2, 11, 22, 15\}$
$c = 40$
$p = \{11, 21, 31, 33\}$

Initially,

Item	w_i	p_i
I_1	2	11
I_2	11	21
I_3	22	31
I_4	15	33

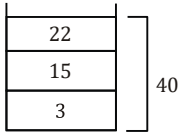
Taking value per weight ratio, *i.e.*, p_i / w_i

Item	w_i	v_i / w_i	p_i
I_1	2	11	22
I_2	11	21	232
I_3	22	31	682
I_4	15	33	495

Now, arrange the value of p_i in decreasing order.

Item	w_i	p_i	p_i
I_3	22	31	682
I_4	15	33	495
I_2	11	21	232
I_1	2	11	22

Now, fill the knapsack according to decreasing value of p_i.
First we choose item I_3 whose weight is 22, then choose item I_4 whose weight is 15. Now the total weight in knapsack is 22 + 15 = 37. Now, next item is I_2 and its weight is 11 and then again I_1. So, we choose fractional part of it, *i.e.*,



The value of fractional part of I_1 is,

$$= \frac{232}{11} \times 3 = 63$$

Thus, the maximum value is,

$$= 682 + 495 + 63 = 1190$$

PART-4

Minimum Spanning Trees-Prim's and Kruskal's Algorithm, Single.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.23. **What do you mean by spanning tree and minimum spanning tree ?**

Answer

Spanning tree :

1. A spanning tree of a graph is a subgraph that contains all the vertices and is a tree.
2. 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 3.24. 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 3.25. Describe and compare following algorithms to determine the minimum cost spanning tree :

i. Kruskal's algorithm

ii. Prim's algorithm

OR

Define spanning tree. Write Kruskal's algorithm or finding minimum cost spanning tree. Describe how Kruskal's algorithm is different from Prim's algorithm for finding minimum cost spanning tree.

AKTU 2019-20, Marks 07

Answer

Spanning tree : Refer Q. 3.23, Page 3–23B, Unit-3.

i. Kruskal's algorithm : Refer Q. 3.24, Page 3–24B, 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, ..., e_i have been chosen involving end points V_1, V_2, ..., V_{i+1}, choose an edge $e_{i+1} = V_jV_k$ with $V_j = \{V_1 \ldots\ldots V_{i+1}\}$ and $V_k \notin \{V_1 \ldots\ldots V_{i+1}\}$ such that e_{i+1} has smallest weight among the edges of G with precisely one end in $\{V_1 \ldots\ldots V_{i+1}\}$.

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 be used on disconnected graphs.	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 3.26. 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.

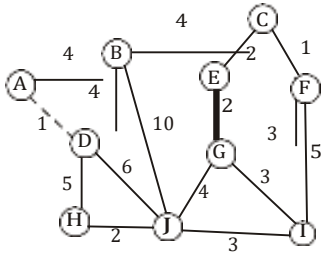
Answer

Minimum spanning tree : Refer Q. 3.23, Page 3–23B, Unit-3.

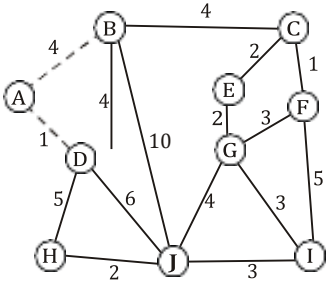
Prim's algorithm : Refer Q. 3.25, Page 3–24B, Unit-3.

For 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

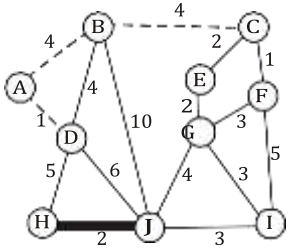
$W(A, B) = 4$

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

$W(D, J) = 6$

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

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



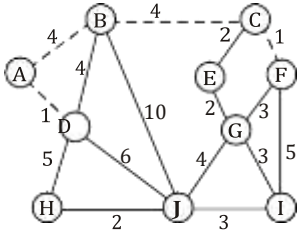
Again, $W(B, C) = 4$

$W(B, J) = 10$

$W(D, H) = 5$

$W(D, J) = 6$

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

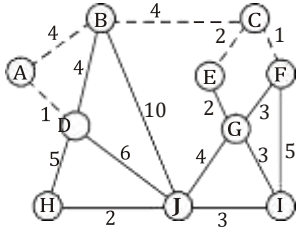


Now, $W(B, J) = 10$

$W(C, E) = 2$

$W(C, F) = 1$

We choose $e = CF$, since it has minimum value.

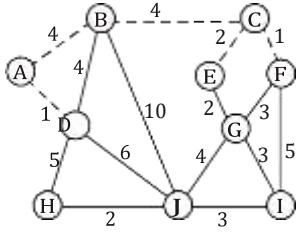


Now, $W(C, E) = 2$

$W(F, G) = 3$

$W(F, I) = 5$

We choose $e = CE$, since it has minimum value.

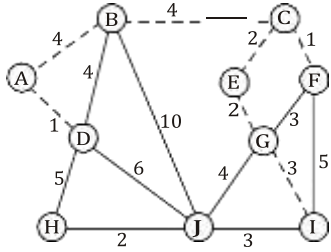


$W(E, G) = 2$

$W(F, G) = 3$

$W(F, I) = 5$

We choose $e = EG$, since it has minimum value.

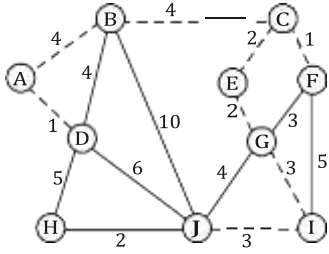


$W(G, J) = 4$

$W(G, I) = 3$

$W(F, I) = 5$

We choose $e = GI$, since it has minimum value.



$W(I, J) = 3$

$W(G, J) = 4$

We choose $e = IJ$, since it has minimum value.

$W(J, H) = 2$

Hence, $e = JH$ will be chosen.

The final minimal spanning tree is given as :

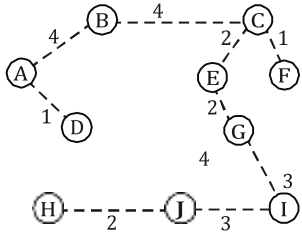


Fig. 3.26.1.

Que 3.27. **What is minimum cost spanning tree ? Explain Kruskal's algorithm and find MST of the graph. Also write its time complexity.**

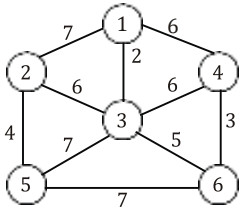


Fig. 3.27.1.

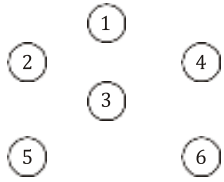
AKTU 2017-18, Marks 10

Answer

Minimum spanning tree : Refer Q. 3.23, Page 3–23B, Unit-3.
Kruskal's algorithm : Refer Q. 3.24, Page 3–24B, Unit-3.
Numerical :
Step 1 : Arrange the edge of graph according to weight in ascending order.

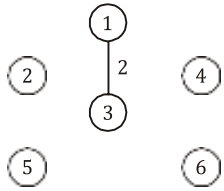
Edges	Weight	Edge	Weight
13	2	32	6
46	3	17	7
25	4	35	7
36	5	56	7
34	6		
41	6		

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

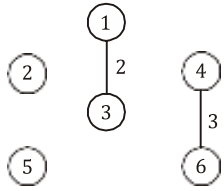


Now draw the edge according to the ascending order of weight. If any edge forms cycle, leave that edge.

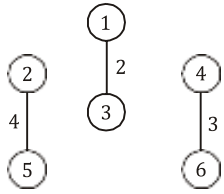
Step 3 : Select edge 13



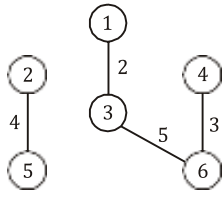
Step 4 : Select edge 46



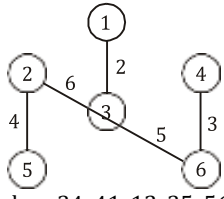
Step 5 : Select edge 25



Step 6 : Select edge 36

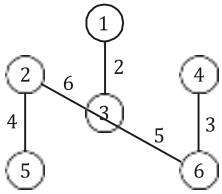


Step 7 : Select edge 23



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



Minimum cost = 2 + 3 + 4 + 5 + 6 = 20

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

Que 3.28. **What is minimum spanning tree ? Explain Prim's algorithm and find MST of graph Fig. 3.28.1.**

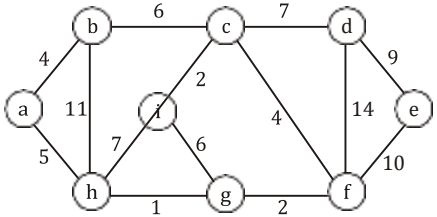


Fig. 3.28.1.

AKTU 2015-16, Marks 05

Answer

Minimum spanning tree : Refer Q. 3.23, Page 3–23B, Unit-3.

Prim's algorithm : Refer Q. 3.25, Page 3–24B, Unit-3.

Numerical :

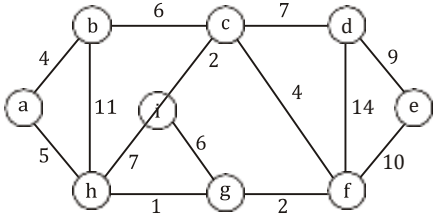
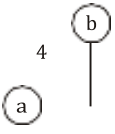
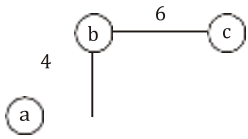


Fig. 3.28.2.

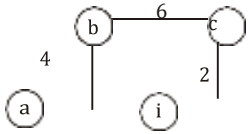
Let *a* be the source node. Select edge (*a*, *b*) as distance between edge (*a*, *b*) is minimum.



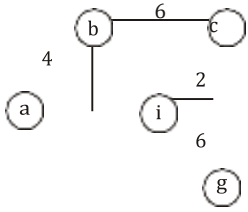
Now, select edge (*b*, *c*)



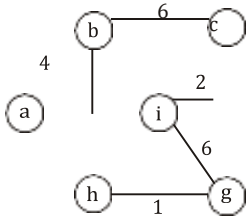
Now, select edge (*c*, *i*)



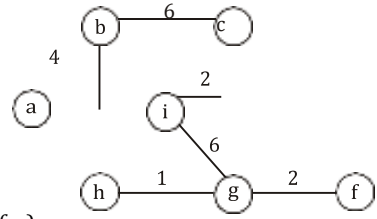
Now, select edge (*i*, *g*)



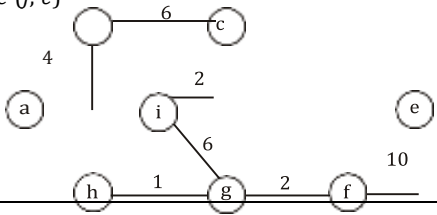
Now, select edge (*g*, *h*)



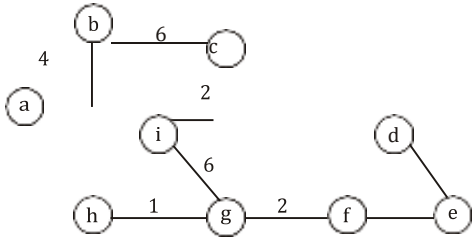
Now, select edge (*g*, *f*)



Now, select edge (*f*, *e*)



Now, select edge (e, d)



Thus, we obtained MST for Fig. 3.28.1.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 3.29. **Prove that the weights on the edge of the connected undirected graph are distinct then there is a unique minimum spanning tree. Give an example in this regard. Also discuss prim's minimum spanning tree algorithm in detail.**

AKTU 2018-19, Marks 07

Answer

Proof :

1. Let we have an algorithm that finds an MST (which we will call A) based on the structure of the graph and the order of the edges when ordered by weight.
2. Assume MST A is not unique.
3. There is another spanning tree with equal weight, say MST B.
4. Let $e1$ be an edge that is in A but not in B.
5. Then, B should include at least one edge e2 that is not in A.
6. Assume the weight of $e1$ is less than that of $e2$.
7. As B is a MST, $\{e1\} \cup B$ must contain a cycle.
8. Replace $e2$ with $e1$ in B yields the spanning tree $\{e1\} \cup B - \{e2\}$ which has a smaller weight compared to B.
9. This contradicts that B is not a MST.

So, MST of undirected graph with distinct edge is unique.

Example :

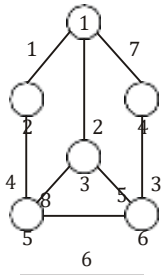
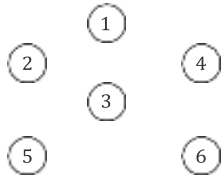


Fig. 3.29.1.

Step 1 : Arrange the edge of graph according to weight in ascending order.

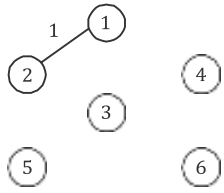
Edges	Weight	Edge	Weight
12	1	14	7
13	2	35	8
46	3		
25	4		
36	5		
56	6		

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

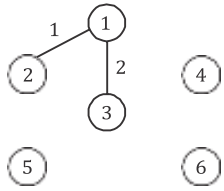


Now draw the edge according to the ascending order of weight. If any edge forms cycle, leave that edge.

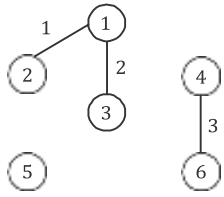
Step 3 :



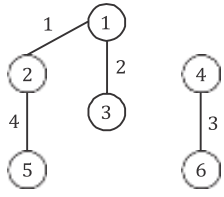
Step 4 :



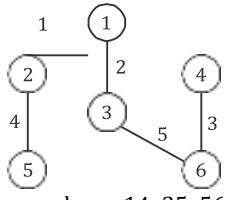
Step 5 :



Step 6 :

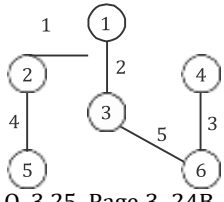


Step 7 :



All the remaining edges, such as : 14, 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



Prim's algorithm : Refer Q. 3.25, Page 3–24B, Unit-3.

Que 3.30. **Write an algorithm to find shortest path between all pairs of nodes in a given graph.**

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 :

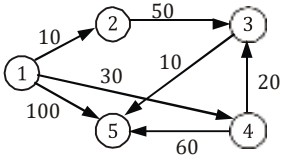
DIJKSTRA (G, w, s)

```
1.  INITIALIZE-SINGLE-SOURCE (G, s)
2.  s ← ϕ
3.  Q ← V[G]
4.  while Q ≠ ϕ
5.      do u ← EXTRACT-MIN (Q)
6.          S ← S ∪ {u}
7.          for each vertex v ∈ 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] ← d[u] + w(u, v)
3.          π[v] ← u
```

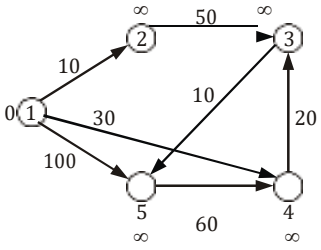
Que 3.31. **Find the shortest path in the below graph from the source vertex 1 to all other vertices by using Dijkstra's algorithm.**



AKTU 2017-18, Marks 10

Answer

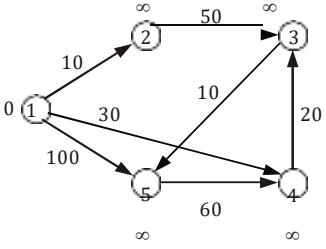
Initialize :



S : { }

Q :	1	2	3	4	5
	0	∞	∞	∞	∞

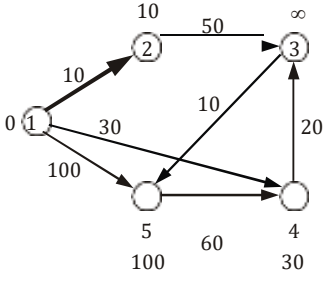
Extract min (1) :



S : {1}

Q :	1	2	3	4	5
	[0]	∞	∞	∞	∞

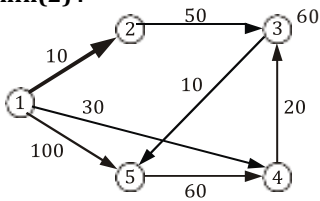
All edges leaving (1) :



S : {1}

Q :	1	2	3	4	5
	[0]	∞	∞	∞	∞
		10	∞	30	100

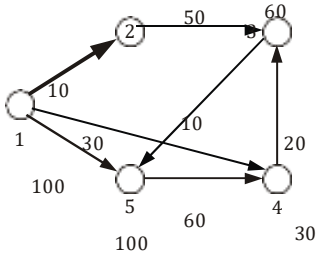
Extract min(2) :



S : {1, 2}

Q : 1	2	3	4	5
0	∞	∞	∞	∞
	10	60	30	100

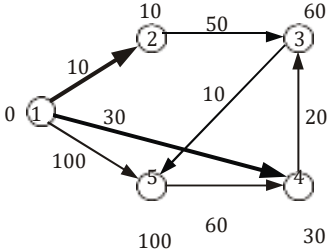
All edges leaving (2) :



S : {1, 2}

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

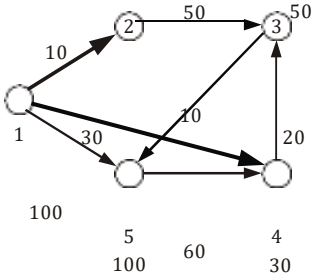
Extract min(4) :



S : {1, 2, 4}

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

All edges leaving (4) :



S : {1, 2, 4}

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

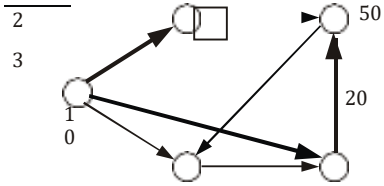
Extract min(3) :

10 0 1

10

50

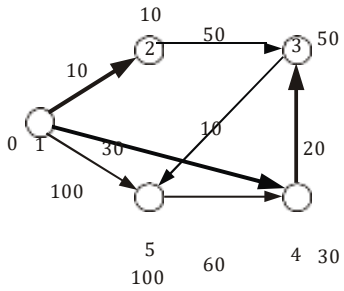
S : {1, 2, 4, 3}



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

50

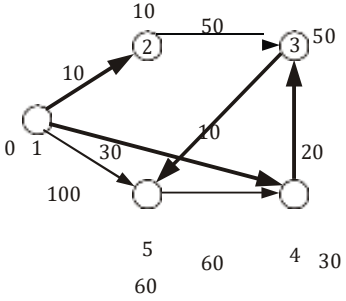
All edges leaving (3) :



S : {1, 2, 4, 3, }

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

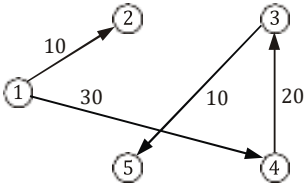
Extract min(5) :



S : {1, 2, 4, 3, 5}

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

Shortest path



Que 3.32. 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 ← 1 to V[G] – 1
3.      do for each edge (u, v) in E[G]
4.          do RELAX (u, v, w)
5.  for each edge (u, v) in E[G] do
6.      do if d[u] + w(u, v) < d[v] then
7.          then 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 3.33. **When do Dijkstra and the Bellman Ford algorithm both fail to find a shortest path ? Can Bellman Ford detect all negative weight cycles in a graph ? Apply Bellman Ford algorithm on the following graph :**

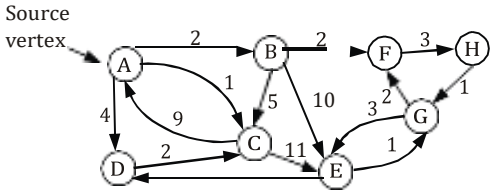


Fig. 3.33.1.

AKTU 2018-19, Marks 07

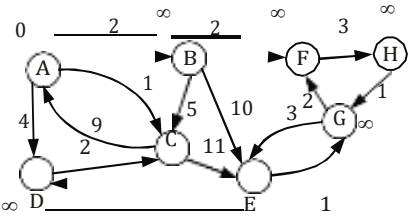
Answer

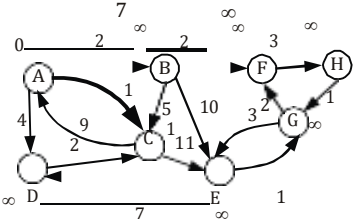
Dijkstra algorithm fails to find a shortest path when the graph contains negative edges.

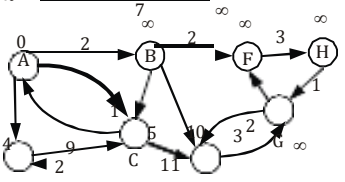
Bellman Ford algorithm fails to find a shortest path when the graph contain negative weight cycle.

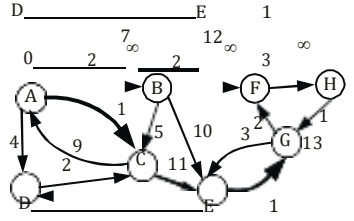
No, Bellman Ford cannot detect all negative weight cycle in a graph.

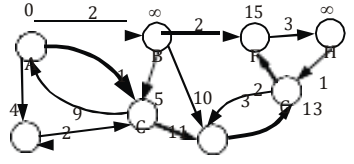
Numerical :

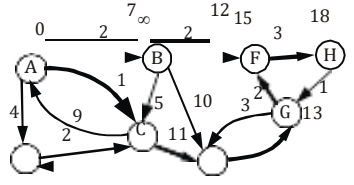












D E 1

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. 3.9.

Q. 4. What is greedy algorithm ? Write its pseudocode forrecursive and iterative process.

Ans. Refer Q. 3.14.

Q. 5. Discuss 0/1-knapsack problem.

Ans. Refer Q. 3.18.

Q. 6. Write short note on the following :

a. Minimum spanning tree

b. Kruskal's algorithm

c. Prim's algorithm

Ans.

a. Refer Q. 3.23.

b. Refer Q. 3.24.

c. Refer Q. 3.25.

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

Ans. Refer Q. 3.30.

Q. 8. State Bellman Ford algorithm.

Ans. Refer Q. 3.32.

Q. 9. Solve the following 0/1-knapsack problem using dynamic programming $P = \{11, 21, 31, 33\}$ $w = \{2, 11, 22, 15\}$ $c = 40$, $n = 4$.

Ans. Refer Q. 3.22.

☺☺☺