Unit-2

The Greedy Method

Que- Explain knapsack Problem with example.
⇒

Knapsack Problem :-
- It involves selecting items with given weights and values to include in a knapsack (a bag) so that the total value is maximized without exceeding the weight capacity of the knapsack.
- You are allowed to take fractions of items of rather than having to choose between taking an item entirely or not taking it at all.

Problem statement :-
Given :-
- n items, each with :-
  - Value (v) :- The worth or profit of the item.
  - weight (w) :- The weight or size of the item.
- A knapsack with capacity W :- The maximum total weight that can be carried.

objective :-
select fractions of items to include in the knapsack such that :-
- The total value of the items in the knapsack is maximized.
- The total weight does not exceed the capacity W.

- There are 3 ways to solve the Problem :-
1) select the item based on maximum profit
2) select the item based on minimum weight

3) calculate ratio of Profit by weight.

- Example :-

max capacity = 15

object	1	2	3	4	5	6	7
Profit	10	15	7	8	9	4	6
weight	1	3	5	4	1	3	2

- By 1st approach :- select the item based on maximum Profit.

object	Profit	weight	Remaining weight
2	15	3	15 - 3 = 12
1	10	1	12 - 1 = 11
5	9	1	11 - 1 = 10
4	8	4	10 - 4 = 6
3	7	5	6 - 5 = 1
7	$= 6 \times \frac{1}{2}$ $= 3$	$= 2 \times \frac{1}{2}$ $= 1$	1 - 1 = 0

max profit = 15 + 10 + 9 + 8 + 7 + 3

= 52

- By second approach :- select the item based on minimum weight.

object	Profit	weight	Remaining weight
1	10	1	15 - 1 = 14
5	9	1	14 - 1 = 13
7	6	2	13 - 2 = 11
2	15	3	11 - 3 = 8
6	4	3	8 - 3 = 5
4	8	4	5 - 4 = 1
3	$= 7 \times \frac{1}{5}$ $= 1.4$	$= 5 \times \frac{1}{5}$ $= 1$	1 - 1 = 0

Max Profit :- 10 + 9 + 6 + 15 + 4 + 8 + 1.4
= 53.4

- By 3rd approach :- calculate ratio of Profit by weight.

$$\text{Ratio} = \frac{\text{Profit}}{\text{weight}} = \frac{P}{W}$$

object	1	2	3	4	5	6	7
Profit	10	15	7	8	9	4	6
weight	1	3	5	4	1	3	2
P/W	10	5	1.4	2	9	1.3	3

object	Profit	weight	Remaining weight
1	10	1	15 - 1 = 14
5	9	1	14 - 1 = 13
2	15	3	13 - 3 = 10
7	6	2	10 - 2 = 8
4	8	4	8 - 4 = 4
3	$= 7 \times \frac{4}{5}$ $= 5.6$	$= 5 \times \frac{4}{5}$ $= 4$	4 - 4 = 0

Max Profit = 10 + 9 + 15 + 6 + 8 + 5.6
= 53.6

- Time complexity :-
  - sorting items :- O(nlogn)
  - selecting items :- O(n)

- Advantages :-
  1) versatility in Application
  2) simple to understand
  3) optimization

- Disadvantages :-
  1) computational complexity
  2) scalability issues
  3) simplistic Assumptions.

- Application :-
  1) Resource allocation
  2) Manufacturing
  3) Network design
  4) Project selection.

que- Explain Job sequencing algorithm with example.

=>

-The Job sequencing problem is a classic optimization problem where the goal is to schedule jobs in such a way that maximizes the total profit, given certain constraints.
-Each job has a deadline by which it needs to be completed and it offers a profit if completed on or before its deadline.

- Problem Definition :-

1] Jobs : You have a set of n jobs, each with :-
- A profit P[i] associated with completing the job
- A deadline d[i] which is the latest time by which the job must be completed.

- objective :-

-maximize the total profit by selecting a subset of jobs and scheduling them in such a way that no two jobs overlap and each selected job is completed by its deadline.

- Greedy method for Job sequencing :-

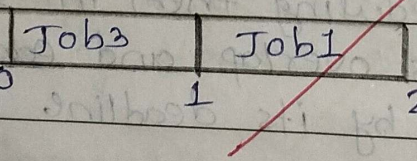
-Greedily choose the jobs with maximum profit first, by sorting the jobs in decreasing order of their profit.
-This would help to maximize the total profit as choosing the job with maximum profit for every time slot will eventually maximize the total profit.

Steps :-
1) Sort all jobs in decreasing order of Profit.
2) Iterate on jobs in decreasing order of profit. For each job, do the following :-
- Find a time slot i, such that slot is empty and i < deadline and i is greatest. Put the job in this slot and mark this slot filled
- If no such i exists, then ignore the job.

- Example :-

	Job1	Job2	Job3	Job4
Profit	100	19	27	25
deadline	2	1	2	1

Initialize slots :-
- maximum deadline is 2, so we have 2 slots.



- choose maximum profit jobs first and so on and fill the slots.

max Profit = 100 + 27
= 127

- Assign Jobs :-
1) Job1 (Profit 100) → slot 2 is free, so assign Job1 to slot 2.
2) Job3 (Profit 27) → slot 2 is filled, but slot 1 is free, so assign Job3 to slot 1.

3] Job 4 (profit 25) → Both slots 1 and 2 are filled so skip Job 4.
4] Job 2 (Profit 19) → Both slots 1 and 2 are filled so, skip Job 2.

- Time complexity :-
  $O(N^2)$

- Advantages :-
1] optimized Resource utilization
2] Improved Profitability.
3] Time management
4] Flexibility.

- Disadvantages :-
1] Complexity
2] Assumption of Determinstic Environment.
3] Approximation in solutions

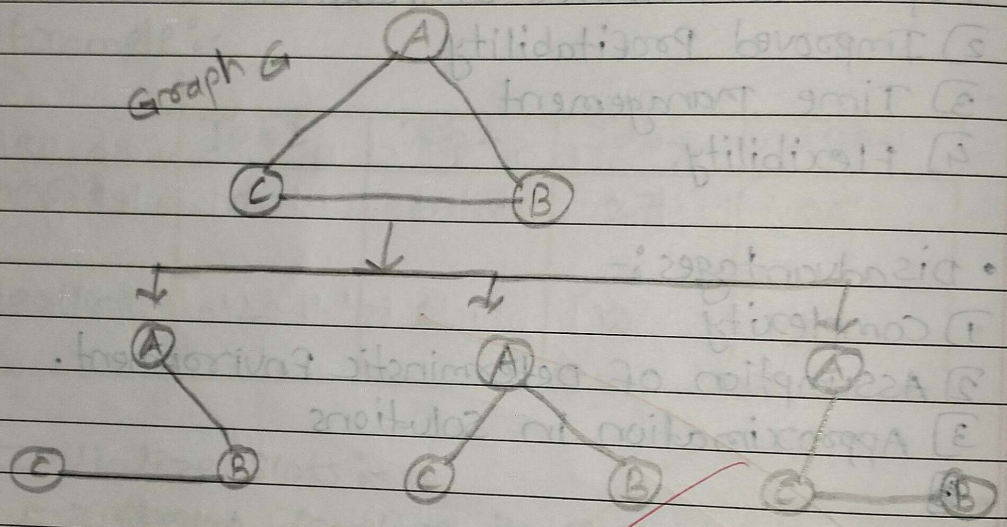
- Applications :-
1] Manufacturing
2] Job shops
3] Data Processing
4] Cloud computing.

Que- Explain minimum - cost spanning trees - Prim's and Kruskal's Algorithms.

⇒

Spanning tree :-

- A spanning tree is a subset of Graph G, which has all the vertices covered with minimum possible number of edges.
- Hence, a spanning tree does not have cycles and it cannot be disconnected.



- A complete undirected graph can have maximum n^{n-2} number of spanning trees, where 'n' is the number of nodes.

- General Properties of spanning Tree :-

1) A connected graph G can have more than one spanning tree.
2) All possible spanning trees of graph G, have the same number of edges and vertices.
3) The spanning tree does not have any cycle (loops).
4) Removing one edge from the spanning tree

will make the graph disconnected.
5) Adding one edge to the spanning tree will create a circuit or loop.

• Minimum Spanning Tree (MST):-
- In a weighted graph, a minimum spanning tree is a spanning tree that has minimum weight than all other spanning trees of the same graph.
- In real-world situations, this weight can be measured as distance, congestion, traffic load or any arbitrary value denoted to the edges.

• Minimum cost spanning tree :-

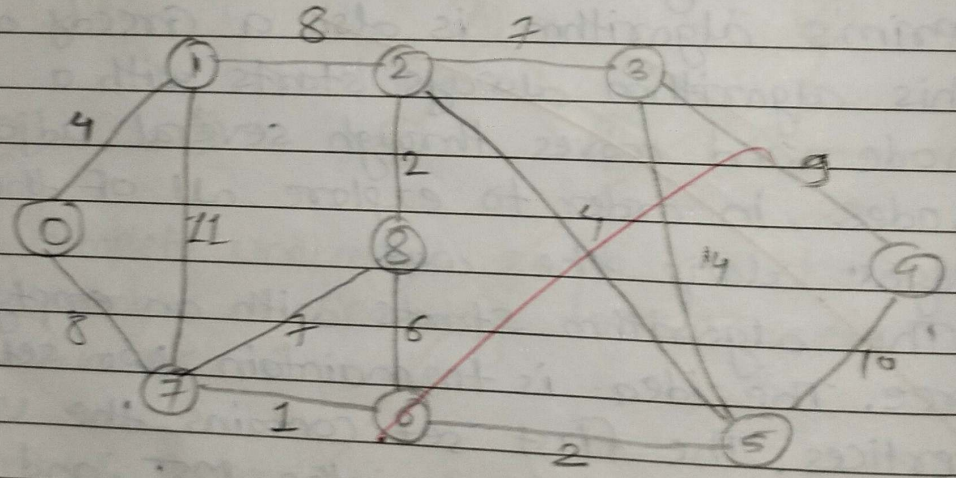
1) Prims Algorithm :-
- Prim's algorithm is also a Greedy algorithm. This algorithm always starts with a single node and moves through several adjacent nodes, in order to explore all of the connected edges along the way.
- The algorithm starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, and the other set contains the vertices already included in the MST, and the other set contains the vertices not yet included.
- At every step, it considers all the edges that connect the two sets and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.

How does Prim's Algorithm work?

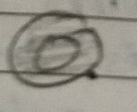
- The working of Prim's algorithm can be described by using the following steps:-

1) Determine an arbitrary vertex as the starting vertex of the MST.
2) Follow steps 3 to 5 till there are vertices that are not included in the MST (known as fringe vertex).
3) Find edges connecting any tree vertex with the fringe vertices.
4) Find the minimum among these edges.
5) Add the chosen edge to the MST if it does not form any cycle.
6) Return the MST and exit.

- Example:-

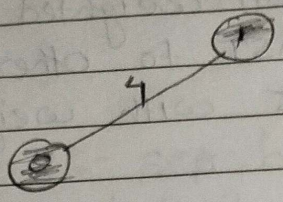


Step 1:- Firstly, we select an arbitrary vertex that acts as the starting vertex of the minimum spanning Tree. Here we have selected vertex 0 as starting vertex.



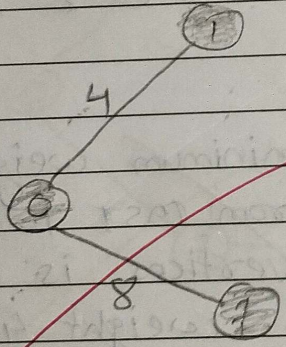
Select an arbitrary starting vertex. Here we have selected 0.

Step 2 :- All the edges connecting the incomplete MST and other vertices are the edges {0,1} and {0,7}. Between these two the edges with minimum weight is {0,1}. So include the edge and vertex 1 in the MST.



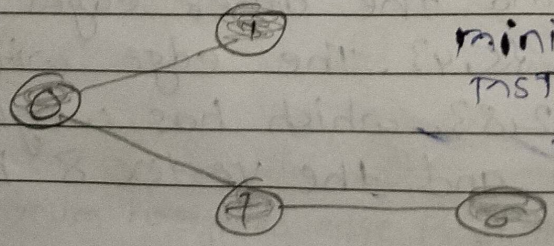
minimum weighted edge from MST to other vertices is 0-1 with weight 4.

Step 3 :- The edges connecting the incomplete MST to other vertices are {0,7}, {1,7} and {1,2}. Among these edges the minimum weight is 8. Which is of the edges {0,7} and {1,2}. Let us here include the edge {0,7} and the vertex 7 in the MST.



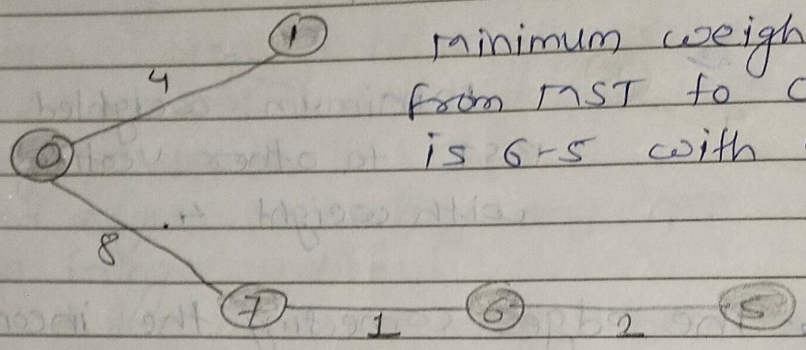
minimum weighted edge from MST to other vertices is 0-7 with weight 8.

Step 4 :- The edges that connect the incomplete MST with the fringe vertices are {1,2}, {7,6} and {7,8}. Add the edge {7,6} and the vertex 6 in the MST as it has the least weight (i.e. 1).



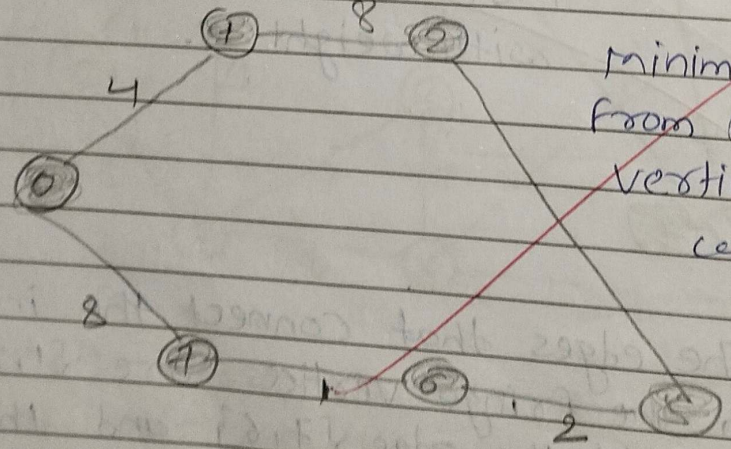
minimum weighted edge from MST to other vertices is 7-6 with weight 1.

step 5 :- The connecting edges now are {7,8} {1,2} {6,8} and {6,5}. Include edge {6,5} and vertex 5 in the MST as the edge has the minimum weight (i.e. 2) among them 6.



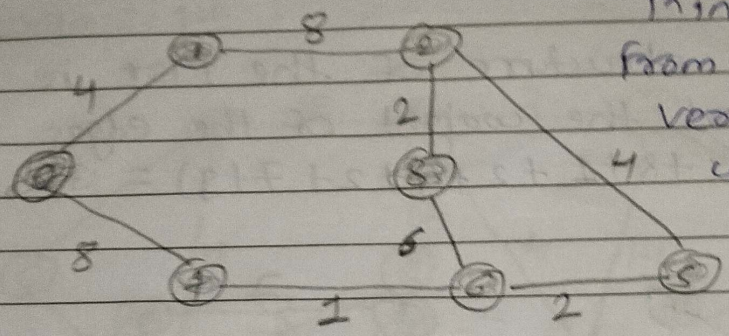
minimum weighted edge from MST to other vertices is 6-5 with weight 2.

step 6 :- Among the current connecting edges, the edge {5,2} has the minimum weight. so h
step 6 : Amonge the current connecting edges, the edge (5,2) has the minimum weight. so include that edge and the vertex 2 in the MST of edge.



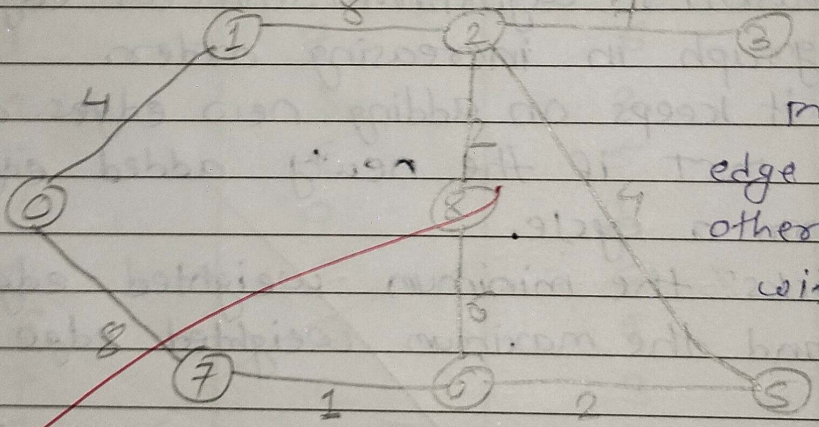
minimum weighted edge from MST to other vertices is 5-2 with weight 4

step 7 :- The connecting edges between the incomplete MST and the other edges are {2,8}, {2,3}, {5,3} and {5,4}. The edge with minimum weight is edge {2,8} which has weight 2. So include this edge and the vertex 8 in the MST.



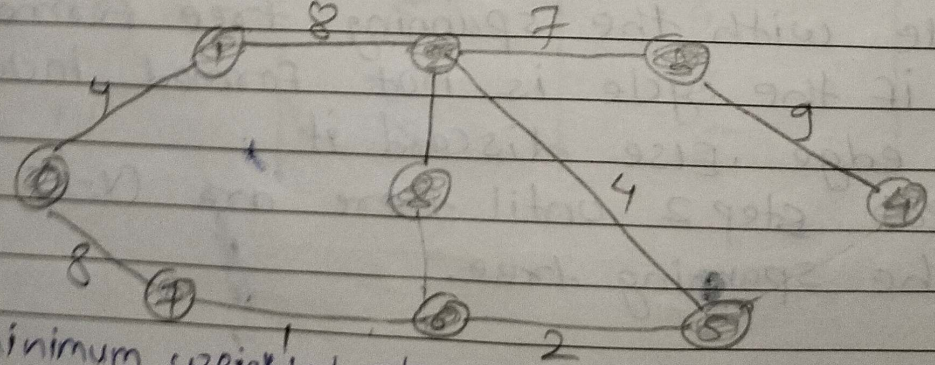
minimum weighted edge from MST to other vertices is 2-8 with weight 2.

Step 8:- see here that the edges {7,8} and {2,3} both have same weight which are minimum. But 7 is already part of MST. so we will consider the edge {2,3} and include that edge and vertex 3 in the MST.



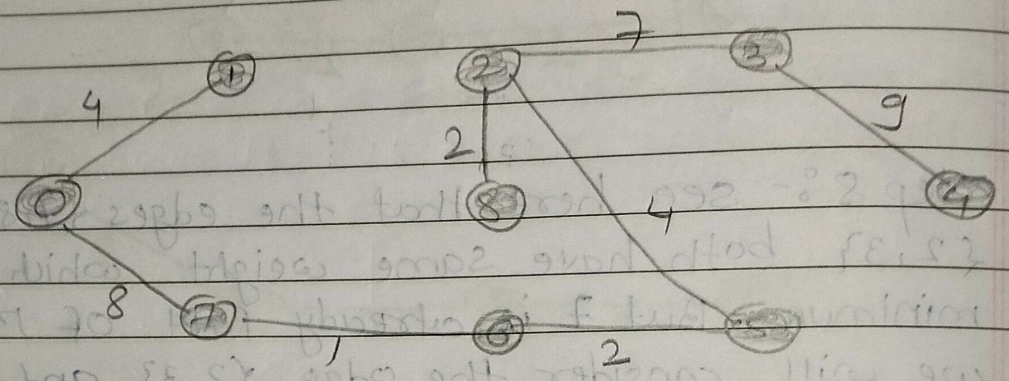
minimum weighted edge from MST to other vertices is 2-3 with weight 7.

Step 9:- only the vertex 4 remains to be included. The minimum weighted edge from the incomplete MST to 4 is {3,4} only.



minimum weighted edge from MST to other vertices is 3-4 with weight 9.

The final structure of the MST is as follows and the weight of the edges of the MST is (4+8+1+2+4+2+7+9) = 37.



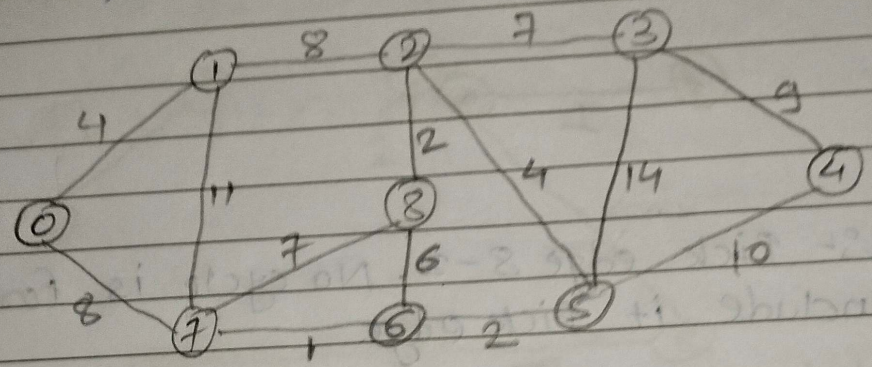
2] Kruskal's Algorithm :-

- In kruskal's algorithm, sort all edges of the given graph in increasing order.
- Then it keeps on adding new edges and nodes in the MST if the newly added edge does not form a cycle.
- It picks the minimum weighted edge at first and the maximum weighted edge at last.

Steps for finding MST using kruskal's algorithm

1) Sort all the edges in non-decreasing order of their weight.
2) Pick the smallest edge. check if it forms a cycle with the spanning tree formed so far. if the cycle is not formed, include this edge. Else discard it.
3) Repeat step 2 until there are (V-1) edges in the spanning tree.

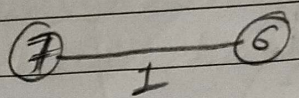
• Example :-



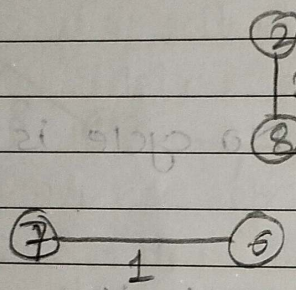
step 1 :- Pick edge 7-6. No cycle is formed, include it at graph.

Weight	source	Destination
1	7	6
2	8	2
2	6	5
4	0	1
4	2	5
6	8	6
7	2	3
7	7	8
8	0	7
8	1	2
9	3	4
10	5	4
11	1	7
14	3	5

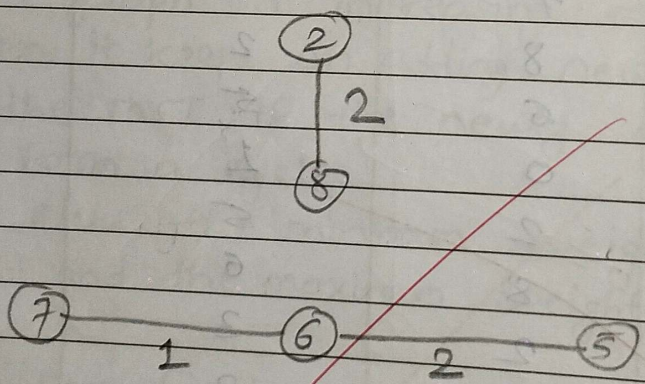
Step 2:- Add edge 7-6 in the MST



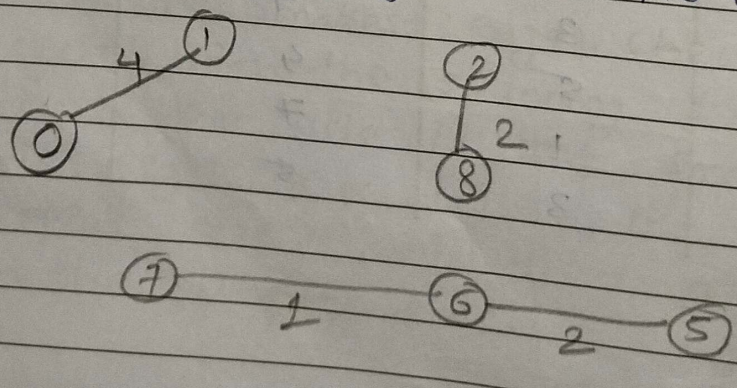
step3:- Pick edge 8-2. No cycle is formed, include it Pick edge.



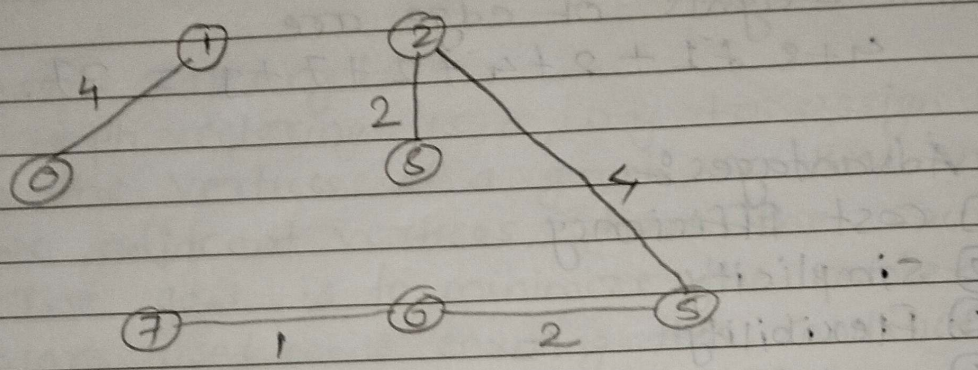
step4:- Pick edge 6-5. No cycle is formed.



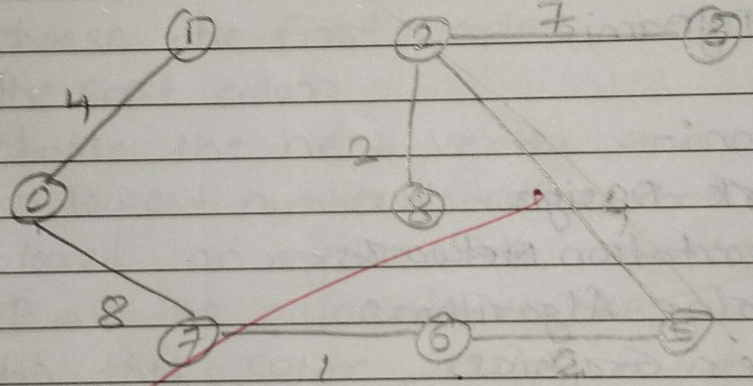
step 5:- Pick edge 0-1. No cycle is formed.



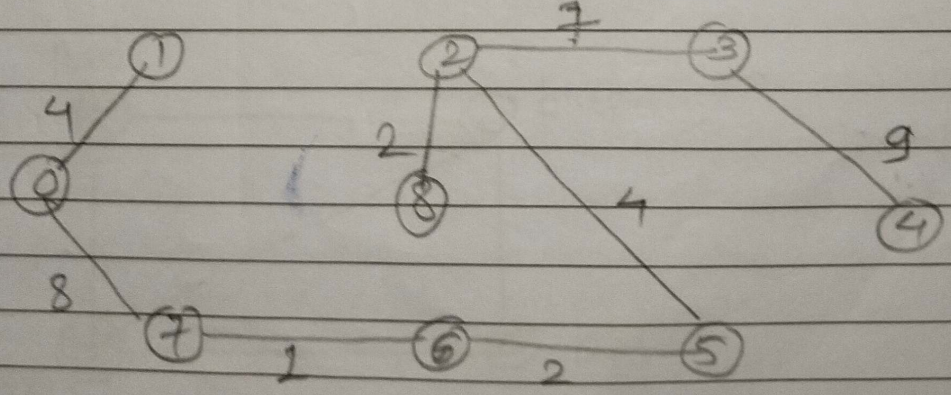
step 6:- Pick edge 2-5. No cycle is formed.



step 7:- Pick edge 7-8. since including this edge results in the cycle, discard it. Pick edge 0-7. No cycle is formed.



step 8:- Pick edge 1-2. since including this edge results in the cycle, discard it. Pick edge 3-4. No cycle is formed.



This is final structure of the MST and weights of edge are
$4+8+1+2+4+2+7+9 = 37$.

- Advantages:-
1) cost Efficiency
2) simplicity
3) Flexibility
4) scalability

- Disadvantages:-
1) single objective focus
2) No fault tolerance
3) Not Dynamic

- Application:-
1) Network Design
2) Transportation Networks
3) clustering Algorithms
4) Computer Graphics.

Que- Explain graph coloring with example.

⇒

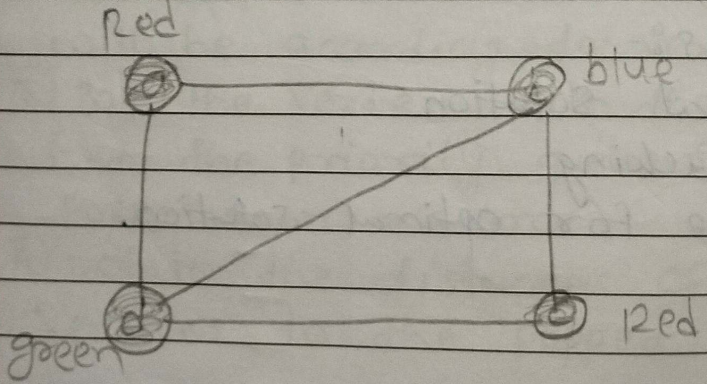
Graph coloring :-

- Graph coloring is a way to assign colors to the vertices of a graph so that no two adjacent vertices share the same color.
- The goal is to minimize the number of colors used while ensuring this condition holds.

• steps for coloring of graph :-

1) Arrange the vertices of the graph in some order.
2) choose the first vertex and color it with the First color.
3) choose the next vertex and color it with the lowest numbered color that has not been colored on any vertices adjacent to it.
- If all the adjacent vertices are colored with this color, assign a new color to it.
- Repeat this step until all the vertices are colored.

• Example :-



Consider a simple graph with vertices $V = \{A, B, C, D\}$ and edges $E = \{(A,B)(A,D)(B,C)(B,D)(C,D)\}$

- Vertex order :- Suppose we order the vertices as A, B, C, D.

- Color Assignment :-
  1) color A with Red
  2) color B with blue
  3) color C with Red
  4) color D with green

- Application :-
  1) Register Allocation
  2) Map coloring
  3) making time table
  4) Bipartite graph checking.

- Advantages :-
  1) simplicity
  2) Efficiency
  3) Flexibility
  4)

- Disadvantages :-
  1) NON-optimal solution
  2) No Backtracking
  3) Not suitable for optimal solution.

Que- Explain single source shortest path with example.

⇒

• single source shortest Path :-
- The single source shortest path (SSSP) problem involves finding the shortest paths from a given source vertex to all other vertices in a weighted graph.
- The goal is to determine the minimum possible distance from the source to each of the other vertices.
- It is a graph algorithm for finding the shortest path from a source node to all other nodes in a graph (single source shortest path).
- It is a type of greedy algorithm.
- It only works on weighted graphs with positive weights.

• Algorithm steps :-
1] set all vertices distances = infinity except for the source vertex, set the source distance = 0
2] Push the source vertex in a min-priority queue in the form (distance, vertex) as the comparison in the min-priority queue will be according to vertices distances.
3] Pop the vertex with the minimum distance from the priority queue (at first the popped vertex = source).
4] update the distances of the connected vertices to the popped vertex in case

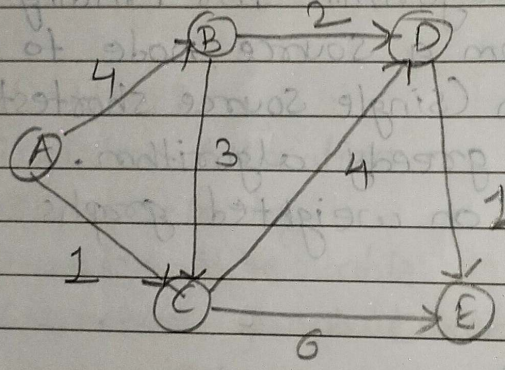
of "current vertex distance + edge weight < next vertex distance", then push the vertex with the new distance to the priority queue.

5) If the popped vertex is visited before, just continue without using it.

6) Apply the same algorithm again until the priority queue is empty.

- Example:- Dijkstra's algorithm.

Let consider a graph with 5 vertices (A, B, C, D, E) and the following weighted edges :-



- We want to find the shortest path from source vertex A to all other vertices.

Step-1 :- Initialization

- We start by initializing the distances from the source vertex (A) to all other vertices as infinity (∞), except for the source vertex itself, which has a distance of 0.

vertex	Distance of A	Previous vertex
A	0	
B	∞	—
C	∞	—
D	∞	—
E	∞	—

Step 2:- Visit vertex A
- current node: A (Distance = 0)
- update distance for neighbors B and C:-
  - Distance to B = 0 + 4 = 1
  - Distance to C = 0 + 1 = 1

Vertex	Distance from A	Previous vertex
A	0	—
B	4	A
C	1	A
D	∞	—
E	∞	—

Step 3:- Visit vertex C (shortest Distance = 1)
- current Node : C (Distance = 1)
- update distances for neighbors D and E:
  - Distance to D = 1 + 4 = 5
  - Distance to E = 1 + 6 = 7

vertex	Distance from A	Previous vertex
A	0	—
B	4	A
C	1	A
D	5	C
E	7	C

Step 4:- Visit vertex B (shortest Distance = 4)
- current Node : B (Distance = 4)
- update distance for neighbor D:
  - Distance to D = 4 + 2 = 6 (since 5 < 6), we do not update the distance for D).

Vertex	Distance from A	Previous vertex
A	0	—
B	4	A
C	1	A
D	5	C
E	7	C

steps :- Vist vertex D (shortest Distance = 5)
- current Node : D (Distance = 5)
- update distance for neighbor E :-
  - Distance to E = 5+1 = 6 (shorter than previous value 7)

vertex	Distance from A	Previous vertex
A	0	—
B	4	A
C	1	A
D	5	C
E	6	D

Step 6 :- Visit vertex E (shortest Distance = 6
- current Node : E (Distance = 6)
- All neighbors have already been processed.

Final shortest Distances From A :-
- A → A = 0
- A → B = 4
- A → C = 1
- A → D = 5
- A → E = 6

- shortest Paths :-
- A → B : A → B (Distance = 4)
- A → C : A → C (1)
- A → D : A → C → D (5)
- A → E : A → C → D → E (6).

Que- Difference between Greedy approach and divide and conquer.

⇒

Greedy Approach	Divide and conquer
1) Make a series of locally optimal choices to find a global solution.	1) Divides the problem into smaller sub-problems, solve them and combines their results.
2) Guarantees an optimal solution only for specific problems.	3) May or may not guarantee an optimal solution, but is often used in problems where subproblem combination is straightforward.
3) Incrementally builds a solution step by step, selecting the best choice at each step.	3) Recursively divides the problem into smaller instances, solves each instance and merges the results.
4) Problem Example :- Dijkstra's Algorithm, Kruskal's Algorithm	4) Problem Example :- merge sort, quick sort, Binary search.
5) Usually faster as it doesn't revisit solutions.	5) requires more time due to recursive divisions but can handle complex problems more easily.
6) No backtracking ; once a decision is made, its final	6) May involve merging results from recursive calls to solve the overall problem

7) Time complexity: usually O(n log n) or O(n) for most cases, depending on problem	7) Time complexity: O(n log n) for sorting; O(log n) for searching
8) Requires less memory	8) require more memory due to recursive calls.
39	