CHAPTER THREE

Greedy Algorithms

Greedy Algorithm

- **A greedy algorithm is an algorithmic paradigm that follows the problem-solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum.**
  - **It doesn't worry whether the current best result will bring the overall optimal result.**
- **This approach never reconsiders the choices taken previously even if the choice is wrong.**

Greedy Algorithm

- **It is used to solve optimization problems that involve finding the best solution among many possible solutions.**
  - **An optimization problem is a problem that demands either maximum or minimum results.**
- **Therefore, it works for cases where minimization or maximization leads to the required solution.**

Greedy Algorithm

- **The method is basically used to determine the feasible solution that may or may not be optimal.**
  - **The feasible solution is a subset that satisfies the given criteria.**
  - **The optimal solution is the solution which is the best and the most favorable solution in the subset.**

Greedy Algorithm: Example

- **Problem:** You have to make a change of an amount using the smallest possible number of coins.
- **Amount: $18**
- **Available coins are:**

  $5 coin

  $2 coin

  $1 coin

- **by hoping to reach the destination faster, we should start by selecting the largest value at each step.**
- **This concept is called greedy choice property.**

- **There is no limit to the number of each coin you can use.**

- **Solution-set = {5, 5, 5, 2, 1}.**

Greedy Algorithm: Example

Solution:

- Create an empty solution-set = { }. Available coins are {5, 2, 1}.
- We are supposed to find the sum = 18. Let's start with sum = 0.
- Always select the coin with the largest value (i.e. 5) until the sum > 18.
- In the first iteration, solution-set = {5} and sum = 5.
- In the second iteration, solution-set = {5, 5} and sum = 10.
- In the third iteration, solution-set = {5, 5, 5} and sum = 15.
- In the fourth iteration, solution-set = {5, 5, 5, 2} and sum = 17. (We cannot select 5 here because if we do so, sum = 20 which is greater than 18. So, we select the 2nd largest item which is 2.)
- Similarly, in the fifth iteration, select 1. Now sum = 18 and solution-set = {5, 5, 5, 2, 1}.

Drawback of Greedy Algorithm

- **The greedy algorithm doesn't always produce the optimal solution.**

- **For example, if the available coins are 1 cent, 3 cents, and 4 cents, and the amount is 6 cents, the greedy algorithm would select one 4-cent coin and two 1-cent coins { 4, 1, 1}, while the optimal solution is to use two 3-cent coins {3, 3}.**

Characteristics of Greedy Algorithm

- **The following are the characteristics of a greedy method:**
  - **To construct the solution in an optimal way, this algorithm creates two sets where one set contains all the chosen items, and another set contains the rejected items.**
  - **A greedy algorithm makes good local choices in the hope that the solution should be either feasible or optimal.**

Greedy Algorithm: Structure

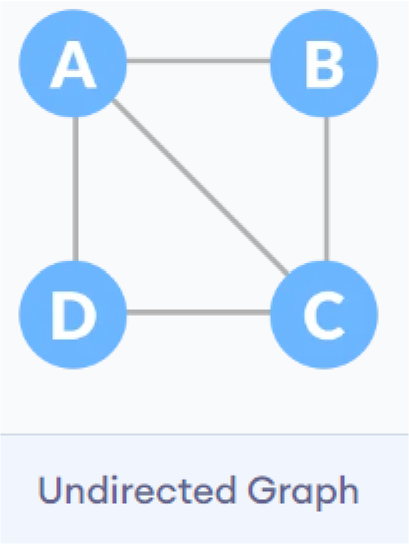
- **The general structure of a greedy algorithm can be summarized in the following steps:**
  1. **Identify the problem as an optimization problem where we need to find the best solution among a set of possible solutions.**
  2. **Determine the set of feasible solutions for the problem.**
  3. **Identify the optimal substructure of the problem, meaning that the optimal solution to the problem can be constructed from the optimal solutions of its subproblems.**
  4. **Develop a greedy strategy to construct a feasible solution step by step, making the locally optimal choice at each step.**
  5. **Prove the correctness of the algorithm by showing that the locally optimal choices at each step lead to a globally optimal solution.**

Applications of Greedy Algorithm

- **Greedy approach is used to solve many problems, such as:**
  - **It is used in finding the shortest path between two vertices using Dijkstra's algorithm.**
  - **It is used to find the minimum spanning tree using the Prim's algorithm or the Kruskal's algorithm.**
  - **It is used in a job sequencing with a deadline.**
  - **It is also used to solve the fractional knapsack problem.**

Minimum Spanning Tree (MST)

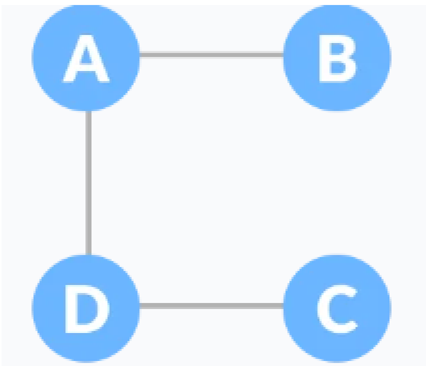
- A **spanning tree** is a sub-graph of an undirected graph that has all the vertices connected by **minimum number of edges**.

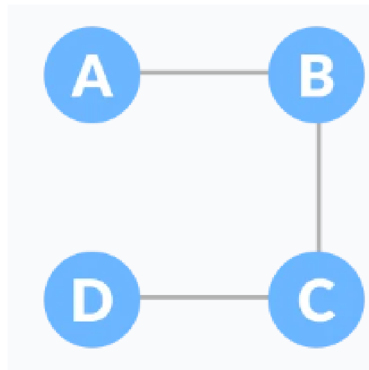


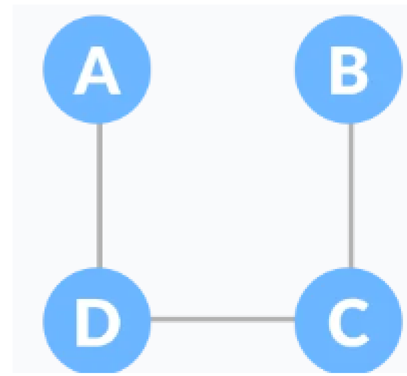
- The total number of spanning trees with **n** vertices that can be created from a complete graph is equal to n^{n-2}.
  - **For Example:** If we have **n = 4**, the maximum number of possible spanning trees is equal to $4^{4-2} = 16$.
- The maximum number of edges (e) that can be removed to construct a spanning tree equals to **e-n+1.**
  - **For Example:** for the above graph, **5 - 4 + 1 = 2** edges can be removed.

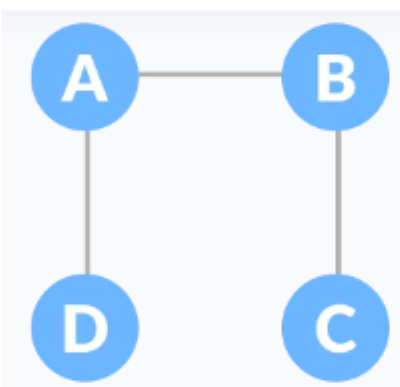
Minimum Spanning Tree (MST)

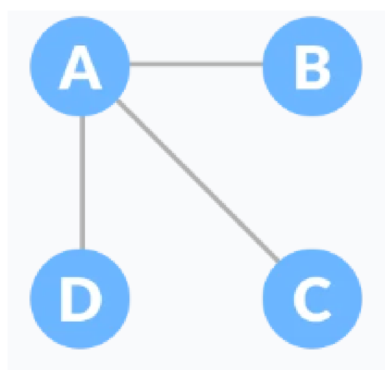
- **Some of the possible spanning trees that can be created from the previous graph are:**







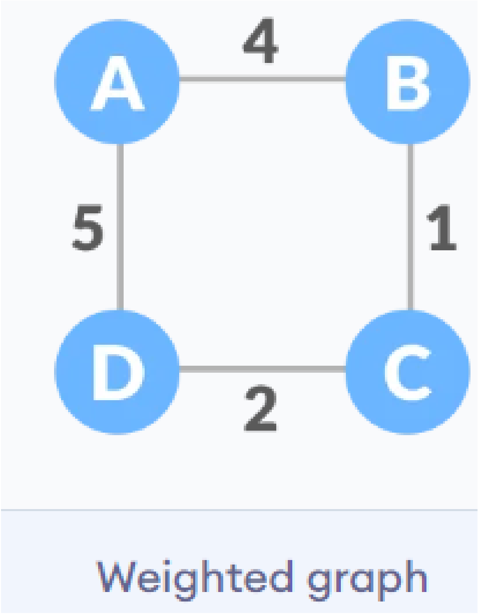




- **Note:** **If there are n number of vertices, the spanning tree should have n-1 number of edges.**

Minimum Spanning Tree (MST)

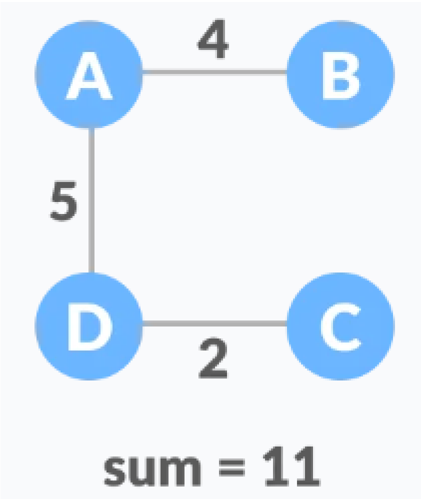
- A **Minimum Spanning Tree (MST)** is a spanning tree in which the **sum of the weight of the edges is minimum**.
- **For Example:** the initial graph is:

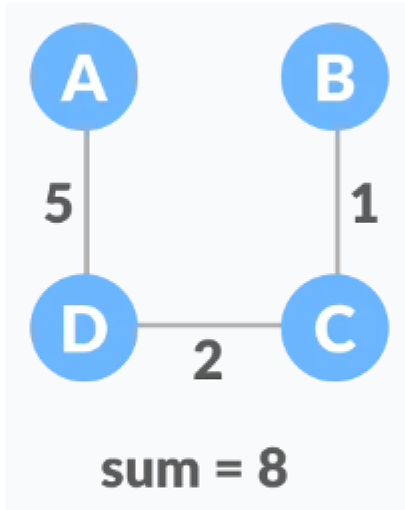


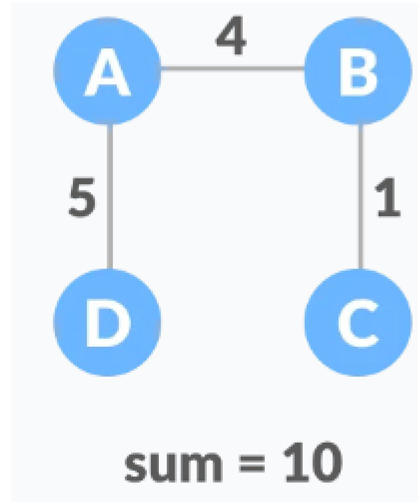
- **MST:**
  - It is a tree i.e no cycle
  - It covers all the vertices V
  - It contains |V| - 1 edges
  - The total cost associated with tree edges is the minimum among all possible spanning trees.

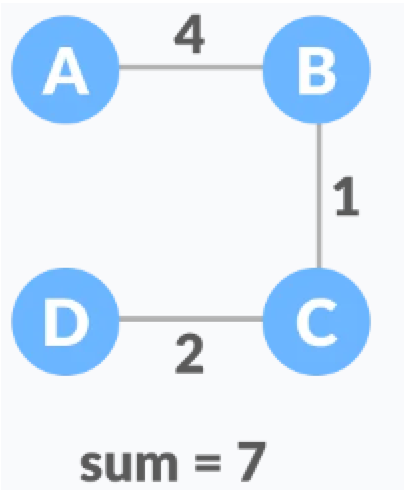
- The possible spanning trees from the above graph are:

Minimum Spanning Tree (MST)

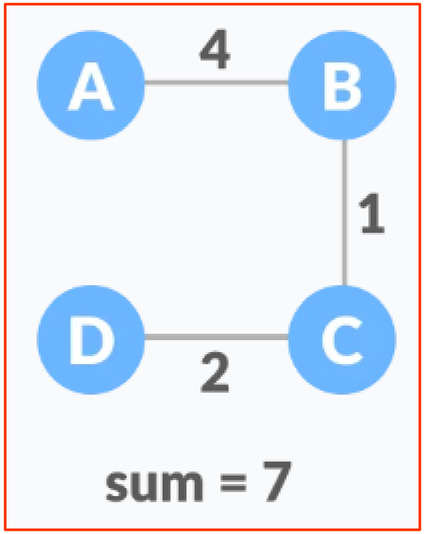








- **Therefore, the MST from the above spanning trees is:**



Minimum Spanning Tree (MST)

- **MST Applications:**
  - **To find paths in the map**
  - **To design networks like telecommunication networks, water supply networks, and electrical grids.**

- **The MST from a graph is found using the following algorithms:**

  1. **Kruskal's Algorithm**
  2. **Prim's Algorithm**

MST - Kruskal's Algorithm

- **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.**
  3. **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.**
  4. **Repeat step 2 until there are (V-1) edges in the spanning tree.**

MST - Kruskal's Algorithm

KRUSKAL(G):

A = ∅

For each vertex v ∈ V[G]:

MAKE-SET(v)

For each edge (u, v) ∈ E[G] ordered by increasing order by weight (u, v):

if FIND-SET(u) ≠ FIND-SET(v):

A = A ∪ {(u, v)}

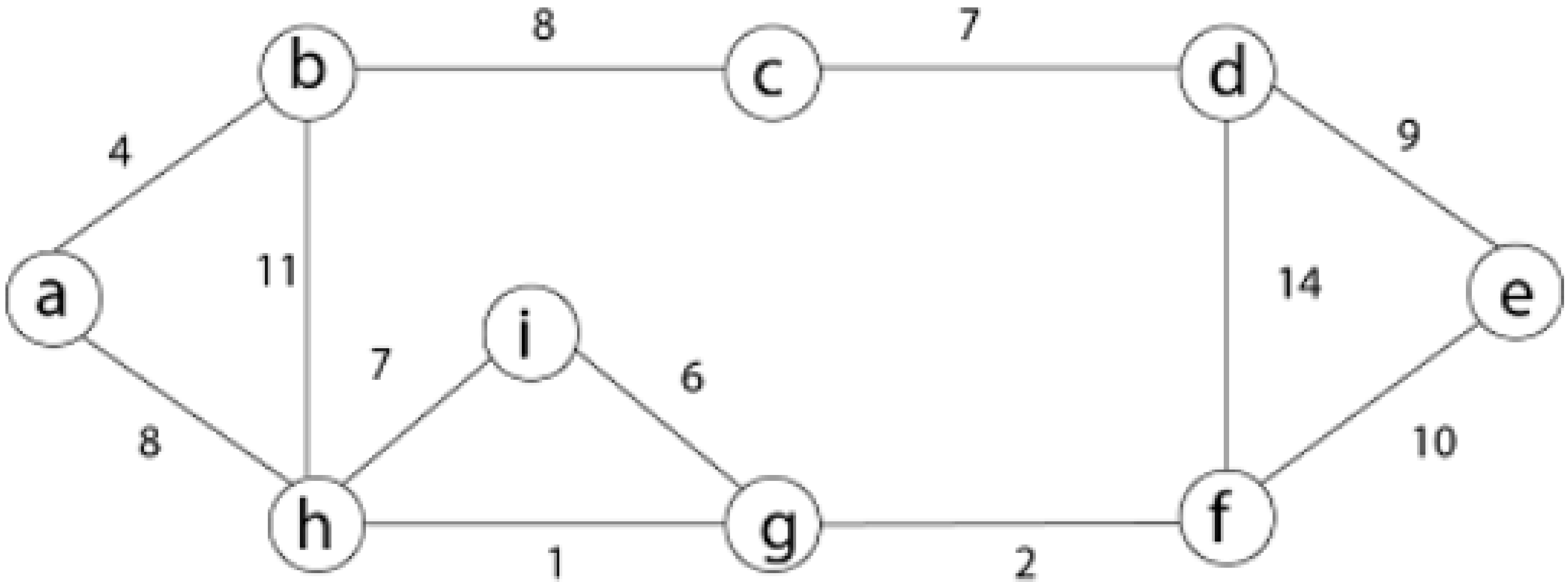
UNION(u, v)

return A

//where u is a vertex in the spanning tree and v is a vertex on the graph G.

MST - Kruskal's Algorithm: Example

- **Find the MST of the following graph using Kruskal's algorithm.**



MST - Kruskal's Algorithm: Example

Solution:

- First we initialize the set A to the empty set and create |v| trees, one containing each vertex with MAKE-SET procedure.
- Then sort the edges in E into order by non-decreasing weight.

Edges	(h, g)	(g, f)	(a, b)	(I, g)	(h, i)	(c, d)	(b, c)	(a, h)	(d, e)	(e, f)	(b, h)	(d, f)
Weight	1	2	4	6	7	7	8	8	9	10	11	14

- There are 9 vertices and 12 edges.
- So MST formed (9-1) = 8 edges

MST - Kruskal's Algorithm: Example

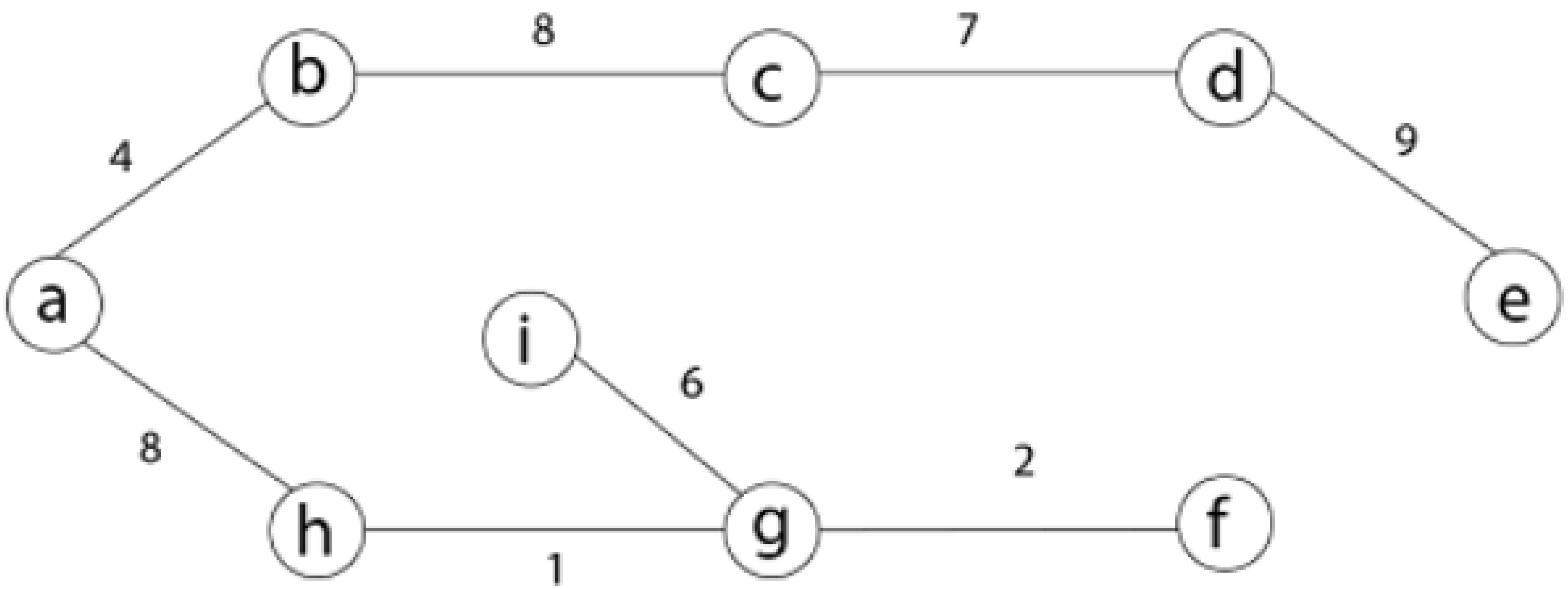
Solution:

- **Now, check for each edge (u, v) whether the endpoints u and v belong to the same tree.**
  - **If they do then the edge (u, v) cannot be supplementary.**
  - **Otherwise, the two vertices belong to different trees, and the edge (u, v) is added to A, and the vertices in two trees are merged in by union procedure.**

- **Therefore, edges (h, i), (e, f), (b, h), and (d, f) are discarded. This is because the vertices of each edges belongs to the same tree.**

MST - Kruskal's Algorithm: Example

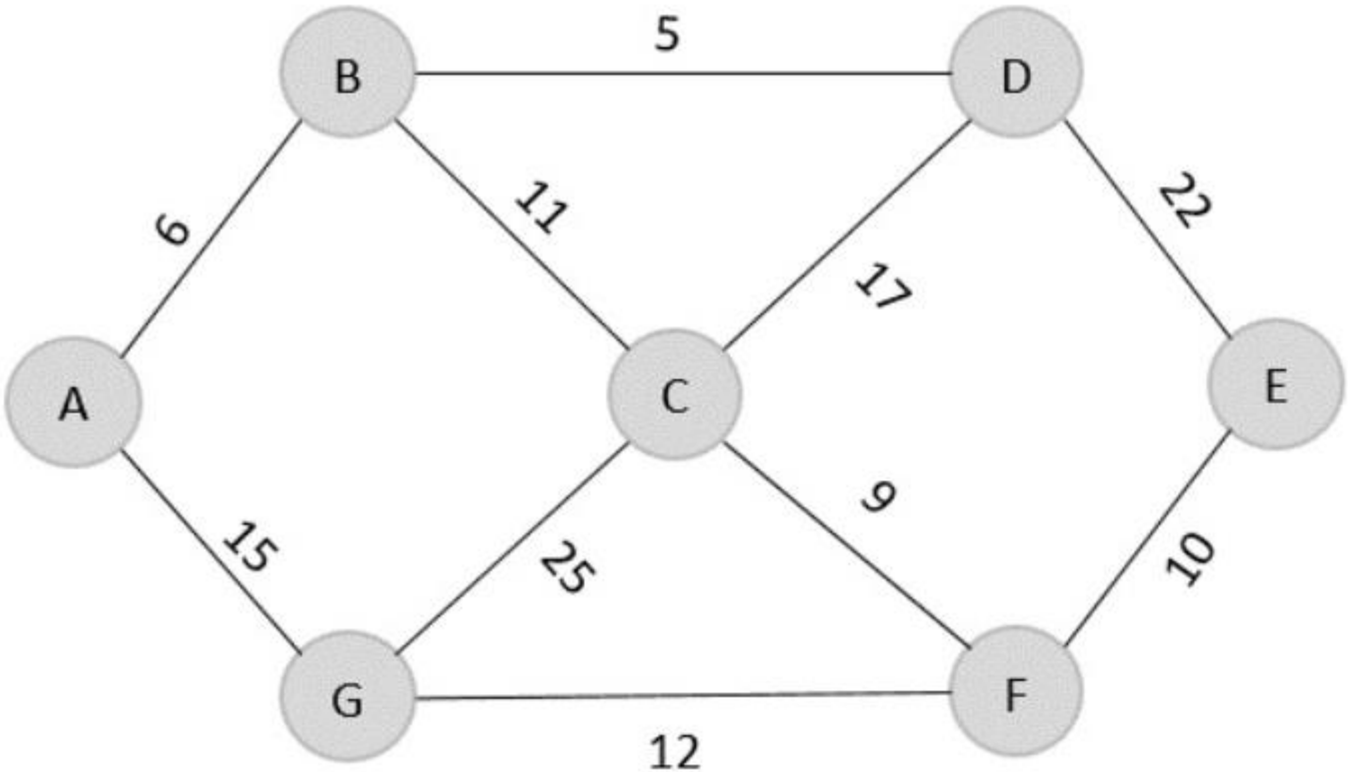
Solution:



- **The obtained result is the MST of the given graph with cost = 1+2+6+8+4+8+7+9 = 45.**

MST - Kruskal's Algorithm: Exercise

- **Find the MST using Kruskal's algorithm for the graph given below:**



MST - Kruskal's Algorithm: Analysis

- **For the given graph G(V, E): where E is the number of edges in the graph and V is the number of vertices, Kruskal's Algorithm can be shown to run in O (E*logE) time, or simply, O (E*logV) time.**
  - **Sorting of edges takes O(E*logE) time.**
  - **After sorting, we iterate through all edges and apply the find-union algorithm. The find and union operations can take at most O(logV) time.**

MST - Kruskal's Algorithm: Analysis

- **So overall complexity is O(E*logE + E*logV) time.**
  - **The value of E can be at most O(V^2), so O(logV) and O(logE) are the same.**
- **Therefore, the overall time complexity of Kruskal’s algorithm is O(E*logE) or O(E*logV).**
- **The space complexity of Kruskal’s algorithm is O(V+E), where V is the number of vertices and E is the number of edges in the graph.**

MST - Prim's Algorithm

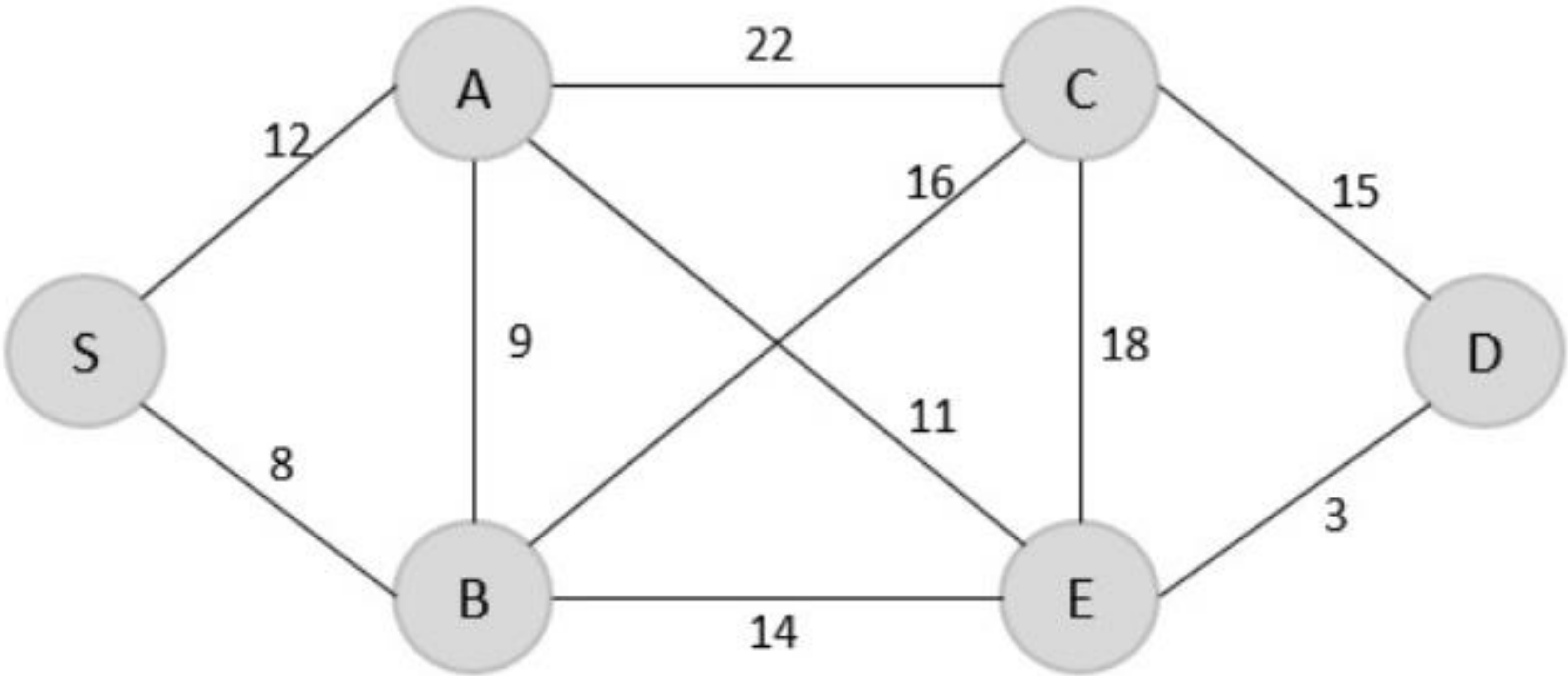
- **Steps to find MST using Prim’s algorithm:**
  1. **Initialize the MST with a vertex chosen at random.**
  2. **Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree**
  3. **Repeat step 2 until we get a MST.**

MST - Prim's Algorithm: Algorithm

1. Declare an array visited[] to store the visited vertices and firstly, add the arbitrary root, say S, to the visited array.
2. Check whether the adjacent vertices of the last visited vertex are present in the visited[] array or not.
3. If the vertices are not in the visited[] array, compare the cost of edges and add the least cost edge to the output spanning tree.
4. The adjacent unvisited vertex with the least cost edge is added into the visited[] array and the least cost edge is added to the minimum spanning tree output.
5. Steps 2 and 4 are repeated for all the unvisited vertices in the graph to obtain the full minimum spanning tree output for the given graph.
6. Calculate the cost of the minimum spanning tree obtained.

MST - Prim's Algorithm: Example

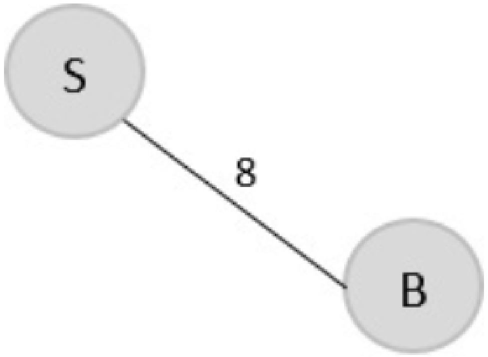
- **Find the MST using Prim's algorithm for the graph given below with S as the arbitrary root.**



MST - Prim's Algorithm: Example

Solution:

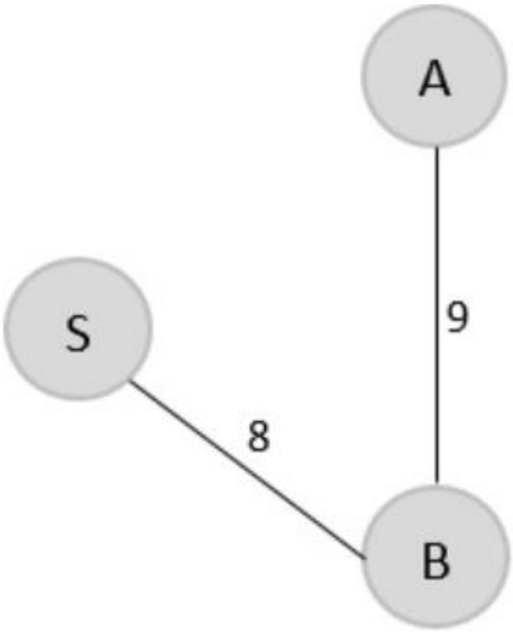
- **Step 1:**
  - **Create a visited array to store all the visited vertices into it, i.e. V = { }.**
  - **The arbitrary root is mentioned to be S, so among all the edges that are connected to S we need to find the least cost edge. i.e. S → B = 8 and V = {S, B}**



MST - Prim's Algorithm: Example

Solution:

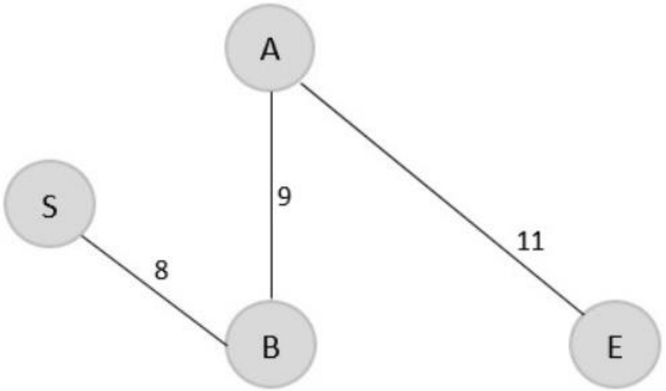
- **Step 2:**
  - **Since B is the last visited, check for the least cost edge that is connected to the vertex B.**
    - **B → A = 9**
    - **B → C = 16**
    - **B → E = 14**
  - **Hence, B → A is the edge added to the spanning tree.**
    - **V = {S, B, A}**



MST - Prim's Algorithm: Example

Solution:

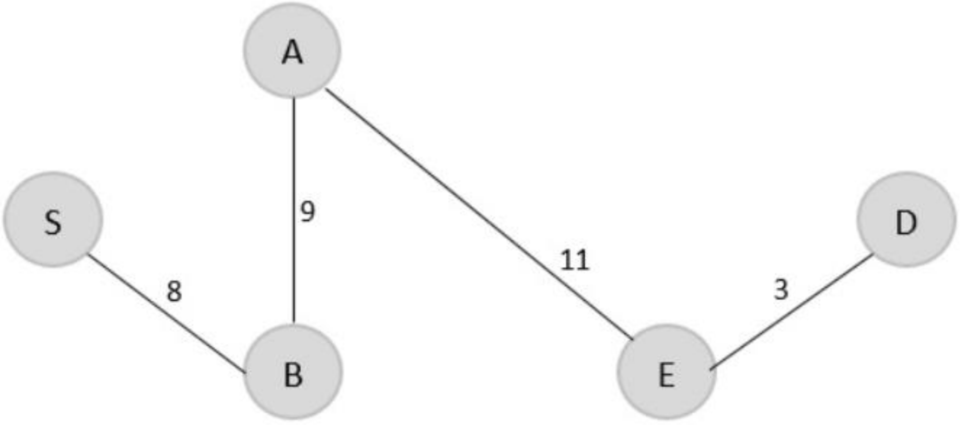
- **Step 3:**
  - Since A is the last visited, check for the least cost edge that is connected to the vertex A.
    - A → C = 22
    - A → B = 9
    - A → E = 11
  - But A → B is already in the spanning tree, check for the next least cost edge. Hence, A → E is added to the spanning tree.
    - V = {S, B, A, E}



MST - Prim's Algorithm: Example

Solution:

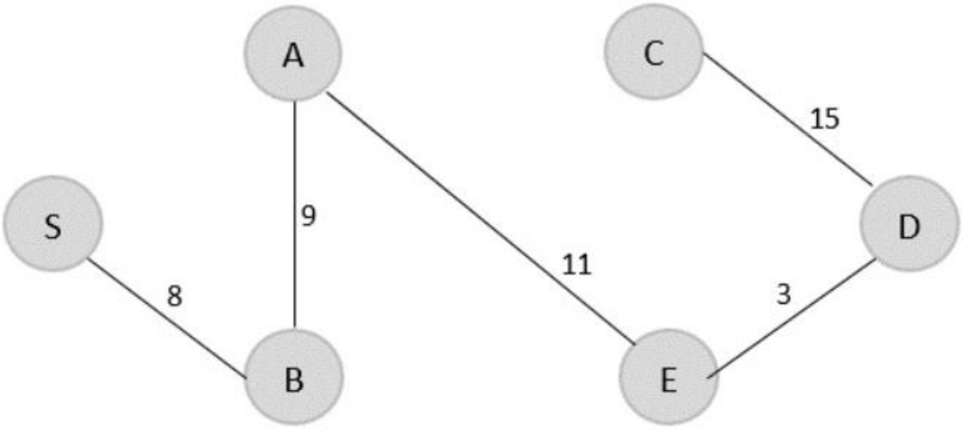
- **Step 4:**
  - **Since E is the last visited, check for the least cost edge that is connected to the vertex E.**
    - **E → C = 18**
    - **E → D = 3**
  - **Therefore, E → D is added to the spanning tree.**
    - **V = {S, B, A, E, D}**



MST - Prim's Algorithm: Example

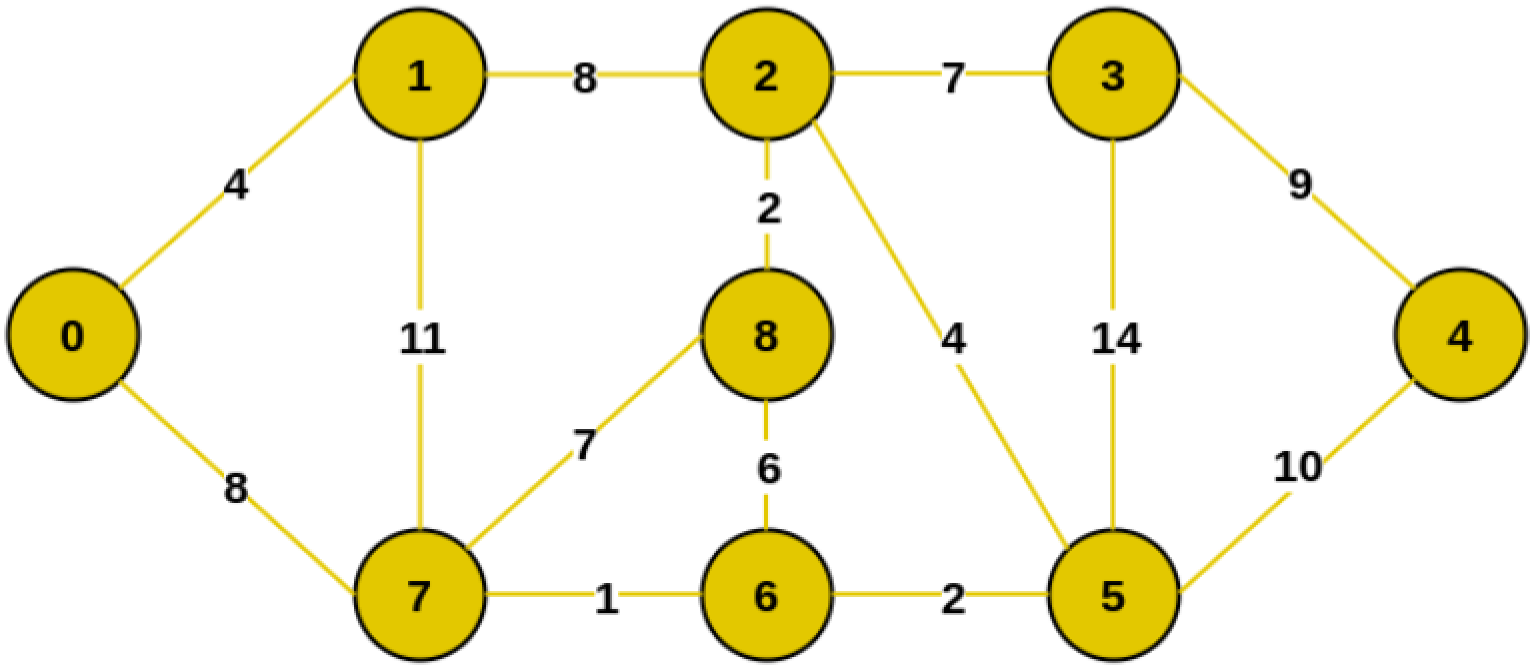
Solution:

- **Step 6:**
  - Since D is the last visited, check for the least cost edge that is connected to the vertex D.
    - D → C = 15
    - E → D = 3
  - Therefore, D → C is added to the spanning tree.
    - V = {S, B, A, E, D, C}
  - The MST is obtained with the minimum cost = 46.



MST - Prim's Algorithm: Exercise

- **Find the MST using Prim's algorithm for the graph given below with 0 as the arbitrary root.**



MST - Prim's Algorithm: Analysis

- **The time complexity of Prim's algorithm is:**
  - **$O(V^2)$ using an adjacency matrix and linear search**
  - **$O(ElogV)$/ $O(VlogV)$ using an adjacency list and binary heap**
    - **where V is the number of vertices and E is the number of edges in the graph.**
- **The space complexity is**
  - **$O(V+E)$ for the priority queue and**
  - **$O(V^2)$ for the adjacency matrix representation.**
- **The algorithm's time complexity depends on the data structure used for storing vertices and edges.**

MST - Kruskal's vs Prim's Algorithm

- **Kruskal's algorithm** sorts all the **edges** from low weight to high and keeps adding the lowest edges, ignoring those edges that create a cycle.

- **Prim's algorithm** starts from a **vertex** and keeps adding lowest-weight edges which aren't in the tree, until all vertices have been covered.

Dijkstra's Algorithm

- Dijkstra's algorithm allows us to find the shortest path between any two vertices of a graph.
- It is similar to that of Prim's algorithm as they both rely on finding the shortest path locally to achieve the global solution.
- However, it is designed to find the shortest path in the graph from one vertex to other remaining vertices in the graph, not to generate MST.
- The shortest distance between two vertices might not include all the vertices of the graph.

Dijkstra's Algorithm

- **Since the shortest path can be calculated from single source vertex to all the other vertices in the graph, Dijkstra's algorithm is also called single-source shortest path algorithm.**
- **The algorithm starts from the source (S).**
- **The inputs taken by the algorithm are the graph G {V, E}, where V is the set of vertices and E is the set of edges, and the source vertex S.**
- **And the output is the shortest path spanning tree.**

Dijkstra's Algorithm

```
function dijkstra(G, S)
    for each vertex V in G
        distance[V] = infinite //to store the distances from the source vertex to the other vertices in graph
        previous[V]  = NULL // to store the previously visited vertices.
        If V != S, add V to Priority Queue Q
    distance[S]  = 0
    while Q IS NOT EMPTY
        U = Extract MIN from Q
        for each unvisited neighbour V of U
            tempDistance = distance[U] + edge_weight(U, V)
            if tempDistance < distance[V]
                distance[V] = tempDistance
                previous[V] = U
    return distance[], previous[]
```

A minimum priority queue can be used to efficiently receive the vertex with least path distance.

where U is the visited or selected vertex, and V is the unvisited vertex on the graph

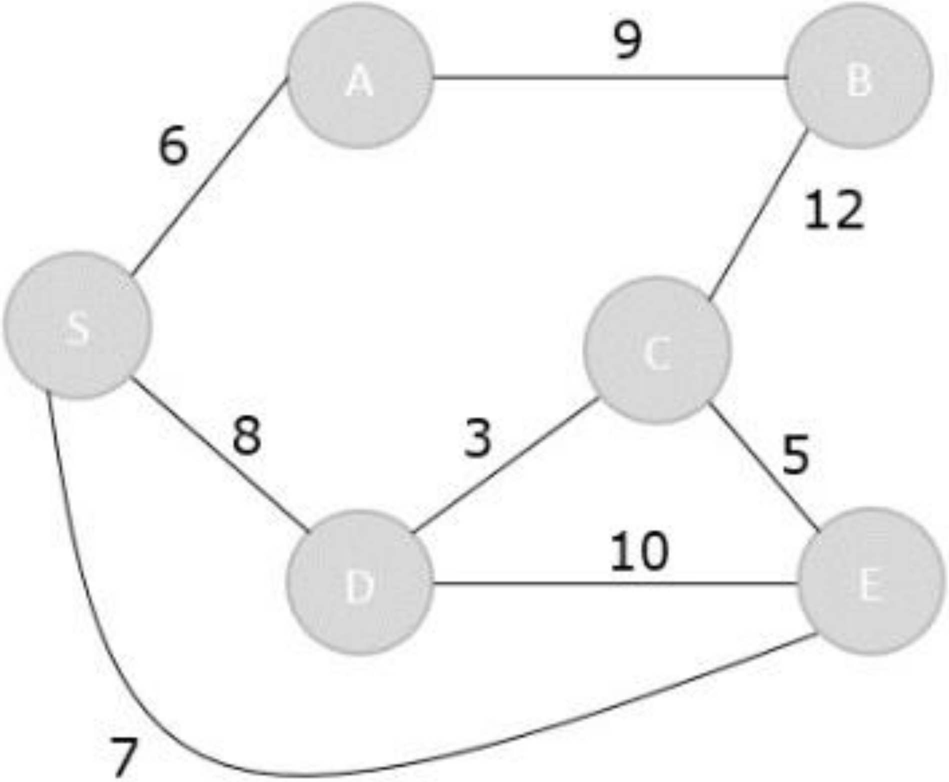
Dijkstra's Algorithm: Analysis

- **Time Complexity: O(ELogV), where, E is the number of edges and V is the number of vertices.**
- **Space Complexity: O(V), where V is the number of vertices.**

- **Application:**
  - **To find the shortest path**
  - **In social networking applications**
  - **In a telephone network**
  - **To find the locations in the map**

Dijkstra's Algorithm: Example

- **Find the shortest path using Dijkstra’s algorithm for the graph given below with S as the arbitrary source.**



Dijkstra's Algorithm: Example

Solution:

- **Step 1**
  - **Initialize the distances of all the vertices as ∞, except the source node S.**

Vertex	S	A	B	C	D	E
Distance	0	∞	∞	∞	∞	∞

  - **Now that the source vertex S is visited, add it into the visited array.**
    - **visited = {S}**

Dijkstra's Algorithm: Example

Solution:

- **Step 1**
  - **The formula for calculating the distance between the vertices:**

    if $(d(u) + d(u, v) < d(v))$ **then**

    $\text{(update) } d(v) = d(u) + c(u, v)$

    - **where u is the visited or selected vertex, and v is the unvisited vertex on the graph.**

Dijkstra's Algorithm: Example

Solution:

- Step 2
  - The vertex S has three adjacent vertices with various distances and the vertex with minimum distance among them all is A.
    - $S \rightarrow A = 6$
    - $S \rightarrow D = 8$
    - $S \rightarrow E = 7$

Vertex	S	A	B	C	D	E
Distance	0	6	∞	∞	8	7

  - Hence, A is visited and the distance[A] is changed from ∞ to 6.
    - Visited = {S, A}

Dijkstra's Algorithm: Example

Solution:

- **Step 3**
  - **There are two vertices visited in the visited array, therefore, the adjacent vertices must be checked for both the visited vertices.**
  - **Vertex S has two more adjacent vertices to be visited yet: D and E. Vertex A has one adjacent vertex B.**
  - **Calculate the distances from S to D, E, B and select the minimum distance:**
    - **$S \rightarrow D = 8$ and $S \rightarrow E = 7$.**
    - **$S \rightarrow B = S \rightarrow A + A \rightarrow B = 6 + 9 = 15$**

- **Visited = {S, A, E}**

Vertex	S	A	B	C	D	E
Distance	0	6	15	∞	8	7

Dijkstra's Algorithm: Example

Solution:

- **Step 4**
  - **Calculate the distances of the adjacent vertices: S, A, E of all the visited arrays and select the vertex with minimum distance.**
  - **S → D = 8**
  - **S → B = 15**
  - **S → C = S → E + E → C = 7 + 5 = 12**

- **Visited = {S, A, E, D}**

Vertex	S	A	B	C	D	E
Distance	0	6	15	12	8	7

Dijkstra's Algorithm: Example

Solution:

Vertex	S	A	B	C	D	E
Distance	0	6	15	11	8	7

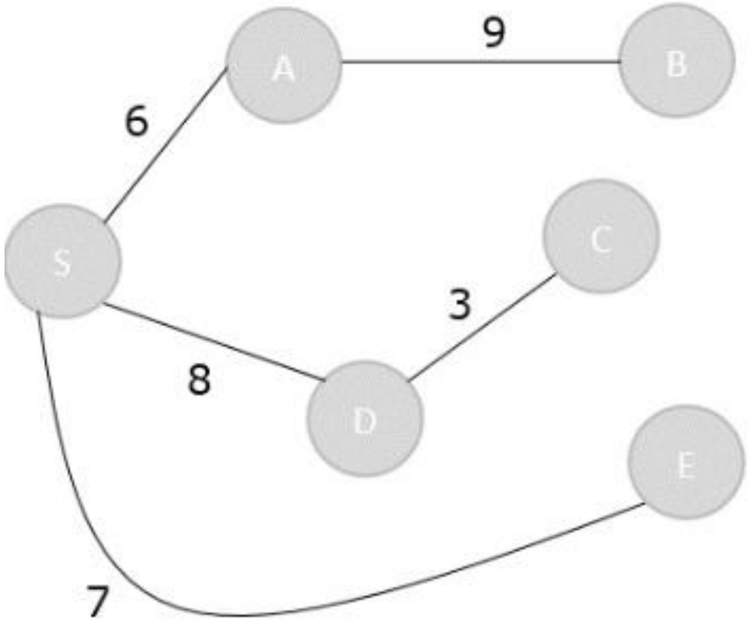
- **Step 5**
  - **Recalculate the distances of unvisited vertices and if the distances minimum than existing distance is found, replace the value in the distance array.**
    - **S → C = S → E + E → C = 7 + 5 = 12**
    - **S → C = S → D + D → C = 8 + 3 = 11**
    - **distance[C] = minimum (12, 11) = 11**
    - **S → B = S → A + A → B = 6 + 9 = 15**
    - **S → B = S → D + D → C + C → B = 8 + 3 + 12 = 23**
    - **distance[B] = minimum (15, 23) = 15**
- **Visited = {S, A, E, D, C}**

Dijkstra's Algorithm: Example

Solution:

- Step 6
  - The remaining unvisited vertex in the graph is B with the minimum distance 15, is added to the output spanning tree.

- Visited = {S, A, E, D, C, B}



- The shortest path spanning tree is obtained as an output using the Dijkstra’s algorithm (with a cost = 33).

Dijkstra's Algorithm: Exercise

- **Find the shortest path using Dijkstra's algorithm for the graph given below with 0 as the arbitrary source.**