Graph Data Structures

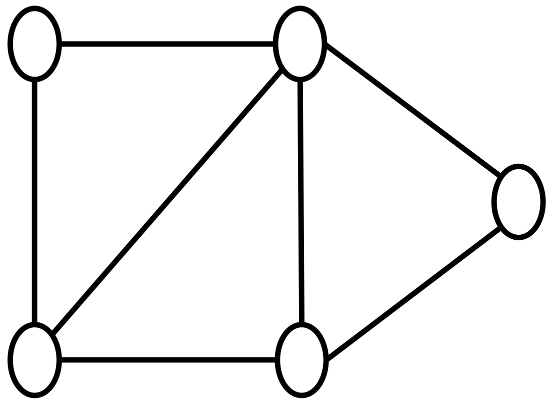
PROF. NAVRATI SAXENA

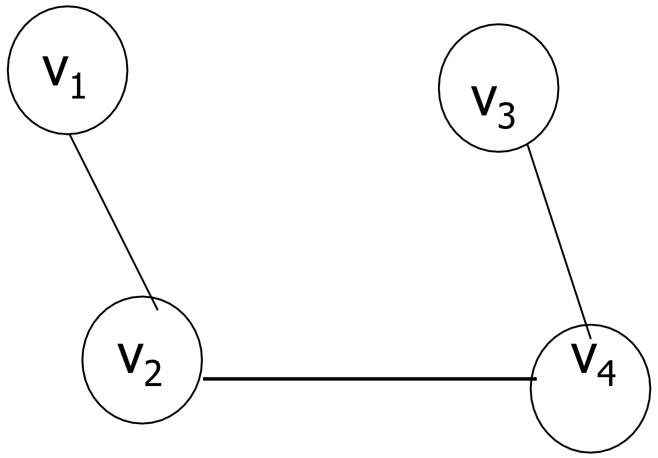
Graphs: Definition

- A mathematical non-linear data structure
- Capable of representing many kinds of physical structures
- A graph G(V, E) is defined as a set of vertices (V) and a set of edges (E)
  - Vertices: Collection of nodes, represented as points or circles
  - Edges:
    - Connect a pair of vertices and can have weights attached

Types of Graph

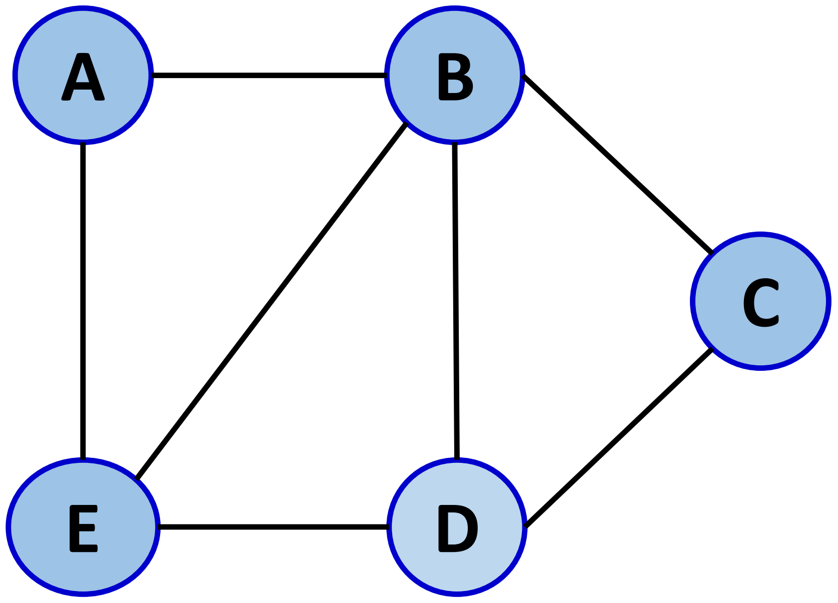
- Undirected graph
- Directed graph
- Simple graph
- Weighted graph
- Connected graph
- Non-connected graph





Adjacency Matrix Representation

1. Create a Matrix (e.g. use a 2-dimensional array)
2. Any index *i* represents a node
3. Any entry *(i, j)* in the matrix represents connectivity between two nodes *i, j*
    1. *entry (i, j) = 1 => an edge exists*
    2. *entry (i, j) = 0 => no edge exists*



$$\begin{array}{c|ccccc|}
 & A & B & C & D & E \\
\hline
A & 0 & 1 & 0 & 0 & 1 \\
B & 1 & 0 & 1 & 1 & 1 \\
C & 0 & 1 & 0 & 1 & 0 \\
D & 0 & 1 & 1 & 0 & 1 \\
E & 1 & 1 & 0 & 1 & 0 \\
\hline
\end{array}$$

Major Graph Operations

- Insert
- Delete
- Search (Traversal)
  - Breadth-First Search (BFS)
  - Depth-First Search (DFS)

Problems Specific with Graphs (1/2)

Minimum Spanning Tree

- Path Problems
  1. Simple Paths
  2. **Shortest Path Problem**
     - Single source shortest paths
     - All-pair shortest paths
  3. Find Cycles
  4. Euler Path and Circuit Problem
  5. Hamiltonian Path and Circuit Problem (or TSP)

Problems Specific with Graphs (2/2)

- Graph Coloring
- Connected Components
- Isomorphic graphs
- Search Graphs

Single Source Shortest Path Algorithms

Two famous algorithms for single source shortest paths

– **Dijkstra's Algorithm**

– Bellman-Ford's Algorithm

- Both use "edge relaxation" approach
- **Dijkstra's algorithm** (greedy approach) is faster
  - Can not be used with graphs having negative weight edges
- Bellman-Ford's algorithm (dynamic programming approach) can work with graphs having negative weight edges
  - But not with "negative cycles"

Greedy Algorithms

- Always makes local optimal choice at each stage with the intent of finding a global optimum.
- Might not always produce an optimal solution,
- Nonetheless yields local optimal solutions that approximate a global optimal solution in a reasonable amount of time.

Five components of Greedy Algorithms

1. **Candidate set**: From which a solution is created

2. **Selection function**: Chooses the best candidate to be added to the solution

3. **Feasibility function:** Determines if a candidate can be used to contribute to a solution

4. **Objective function:** Assigns a value to a solution, or a partial solution, and

5. **Solution function**: Indicates when we have discovered a complete solution

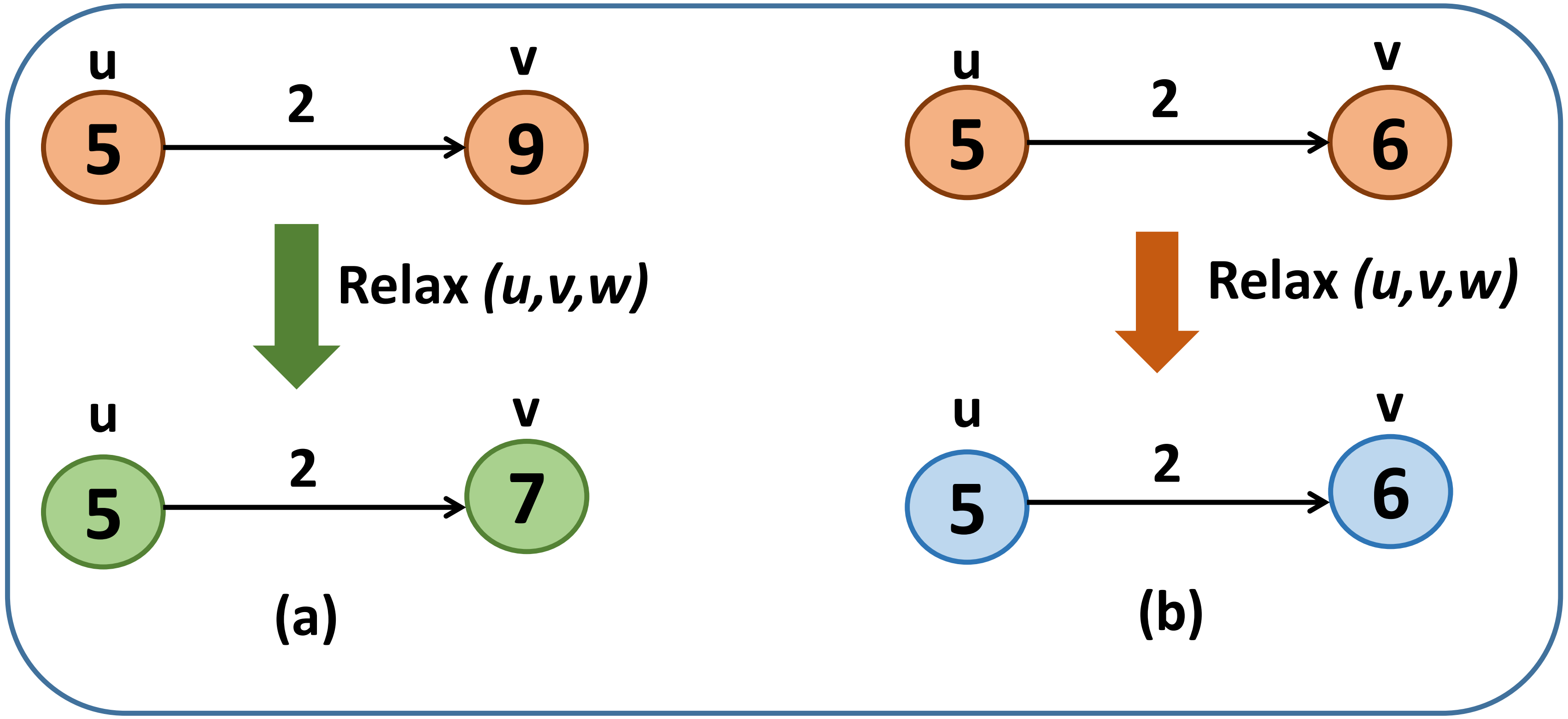
Shortest Path Algorithms: Major Components

```
Initialize-Single-Source (G, s)
1. for each vertex $v \in G$
2.         $v.dist \leftarrow \infty$
3.         $v.\pi \leftarrow NULL$
4. $s.dist \leftarrow 0$
```

```
Relax (u, v, w)
1. if $v.dist > (u.dist + w(u, v))$
2.         $v.dist \leftarrow u.dist + w(u, v)$
3.         $v.\pi \leftarrow u$
```

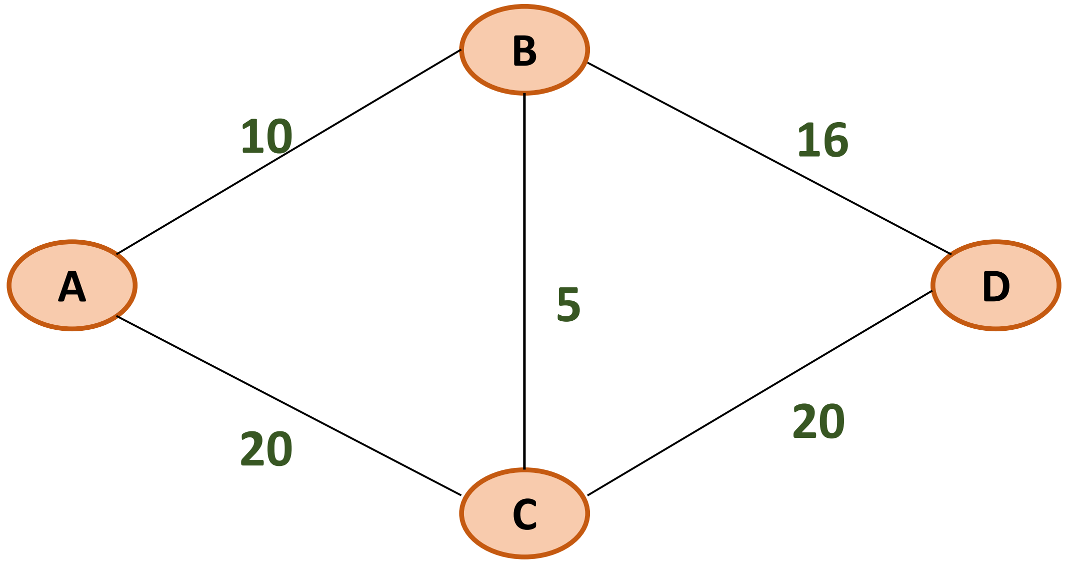
Shortest Paths

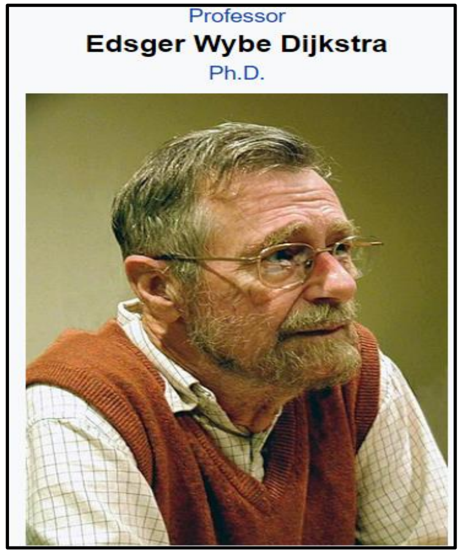
- **Main Idea**: Relaxing edges in the graph
  - Two cases



Dijkstra’s Shortest Path Algorithm

- Given a graph and a source vertex, find shortest paths from source to all vertices in the given graph

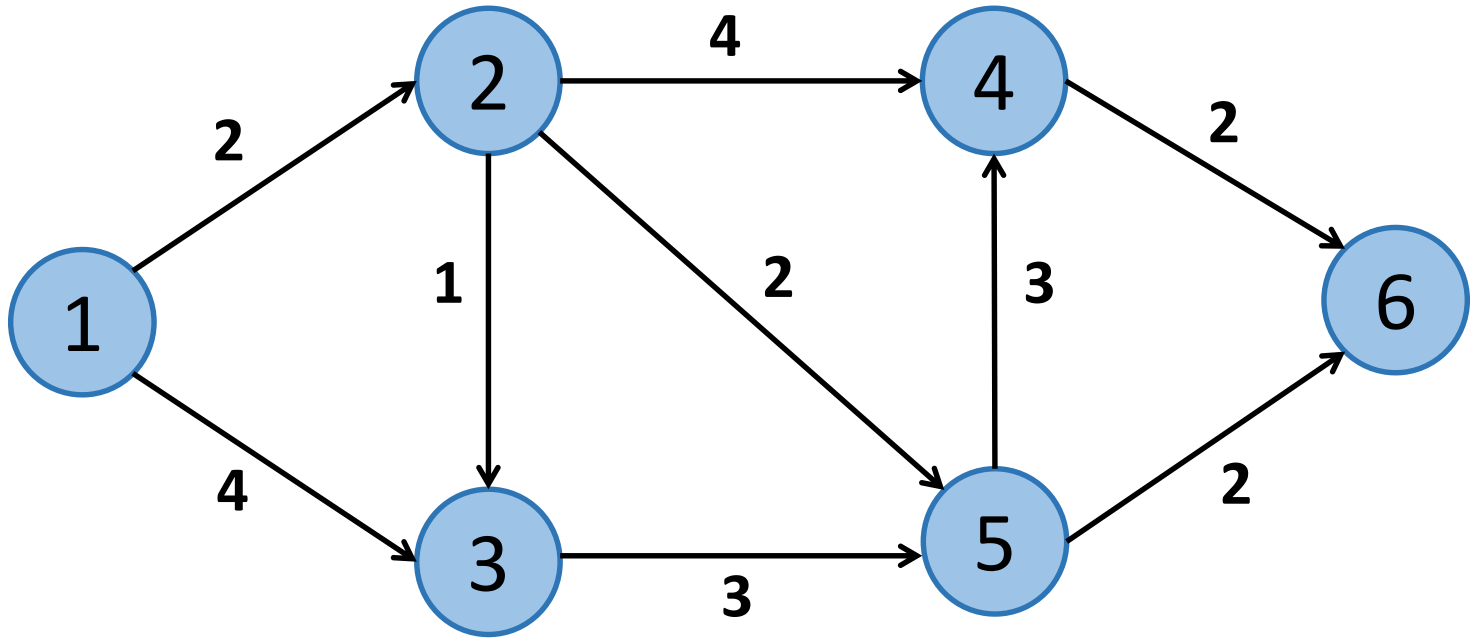




In 1959 Dijkstra published in a 3-page article 'A note on two problems in connexion with graphs' the algorithm to find the shortest path in a graph between any two given nodes, now called Dijkstra's algorithm. https://en.wikipedia.org/wiki/Edsger_W._Dijkstra

Dijkstra’s Algorithm

- Solution to the single source shortest path problem
  - For both directed as well as undirected graphs
  - All edges must have non-negative weights
  - Graphs must be connected



Dijkstra’s Shortest Path Algorithm

- Dijkstra’s algorithm is very similar to minimum spanning tree

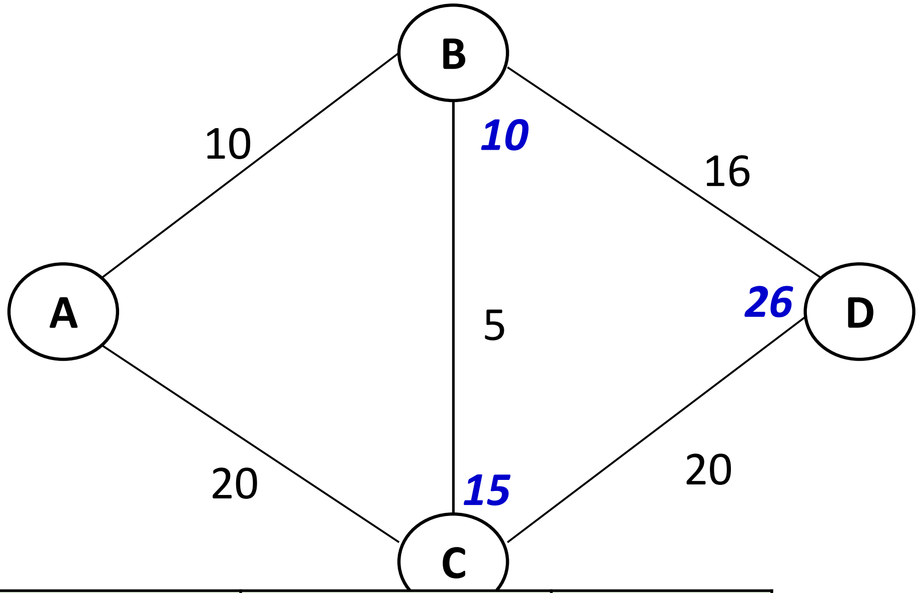
- Like Minimum Spanning Tree, in Dijkstra’s too:
  - Generate a *shortest path tree* with given source as root
  - **Maintain two sets**
    - One set contains vertices included in shortest path tree: **Set 1**
    - Other set includes vertices not yet included in shortest path tree: **Set 2**
  - **At every step**:
    - We find a vertex which is in Set 2 and
    - Has a minimum distance from the source

Output of Dijkstra’s Algorithm

- Original algorithm outputs value of shortest path
  - **Not the path itself**
- With slight modification, we can obtain the path too

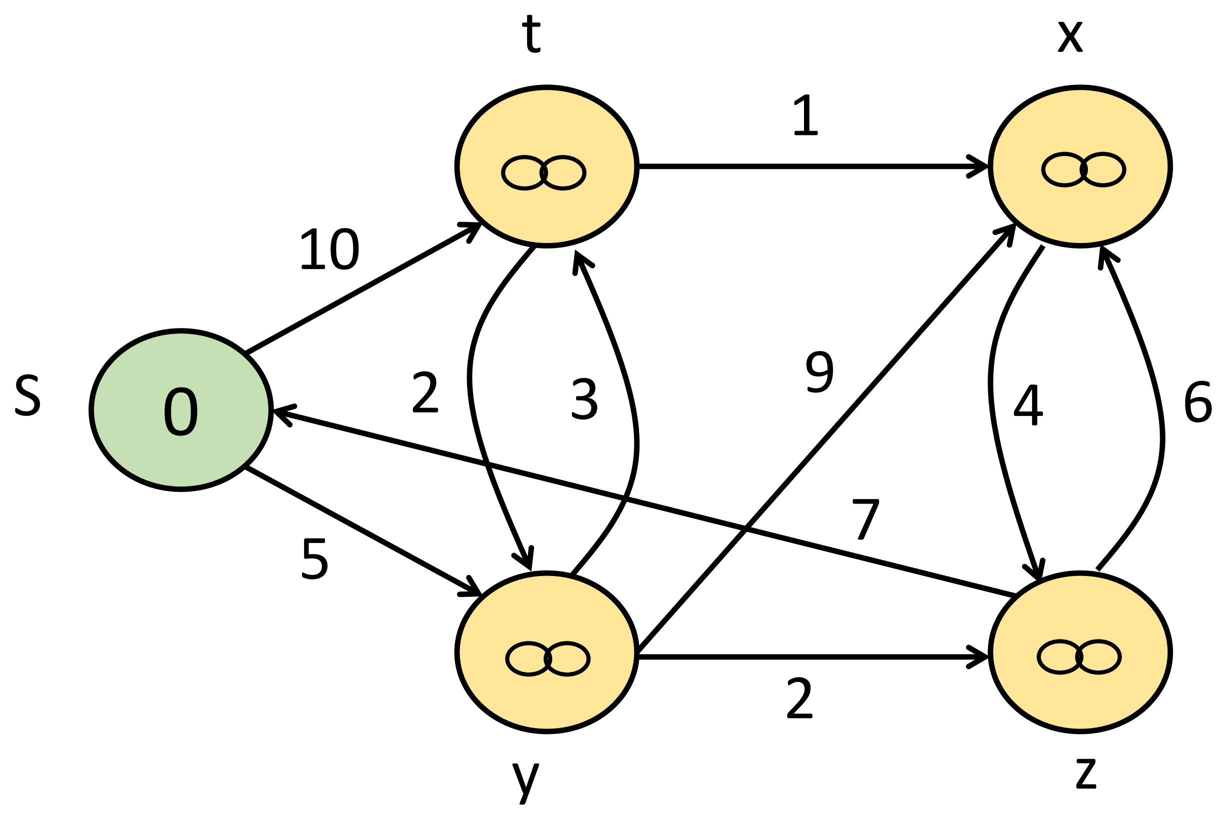
Dijkstra’s Shortest Path: Pseudo-Code

```
1. Set 1: S = Ø
2. Set 2: Q = V[G]
3. While Q != Ø
       u = extractMin(Q)
       S = S U {u}
       for each $v \in Adj[u]$ & Not in Q
           if (d[v] > d[u] + w)
               d[v] = d[u] + w
               Parent[v] = u
```



Vertex	Distance ***d***	Parent
A	O	NIL
B	∞	NIL
C	∞	NIL
D	∞	NIL

An Example: Dijkstra’s Algorithm



Edsger Dijkstra, in an interview with Philip L. Frana, Communications of the ACM 53 (8), 2001

“What is the shortest way to travel from Rotterdam to Groningen, in general: from given city to given city? It is the algorithm for the shortest path, which I designed in about twenty minutes. One morning I was shopping in Amsterdam with my young fiancée, and tired, we sat down on the café terrace to drink a cup of coffee and I was just thinking about whether I could do this, and I then designed the algorithm for the shortest path. As I said, it was a twenty-minute invention. In fact, it was published in ’59, three years late. The publication is still readable, it is, in fact, quite nice. One of the reasons that it is so nice was that I designed it without pencil and paper. I learned later that one of the advantages of designing without pencil and paper is that you are almost forced to avoid all avoidable complexities. Eventually that algorithm became, to my great amazement, one of the cornerstones of my fame.”

https://en.wikipedia.org/wiki/Edsger_W._Dijkstra

Problems Specific with Graphs

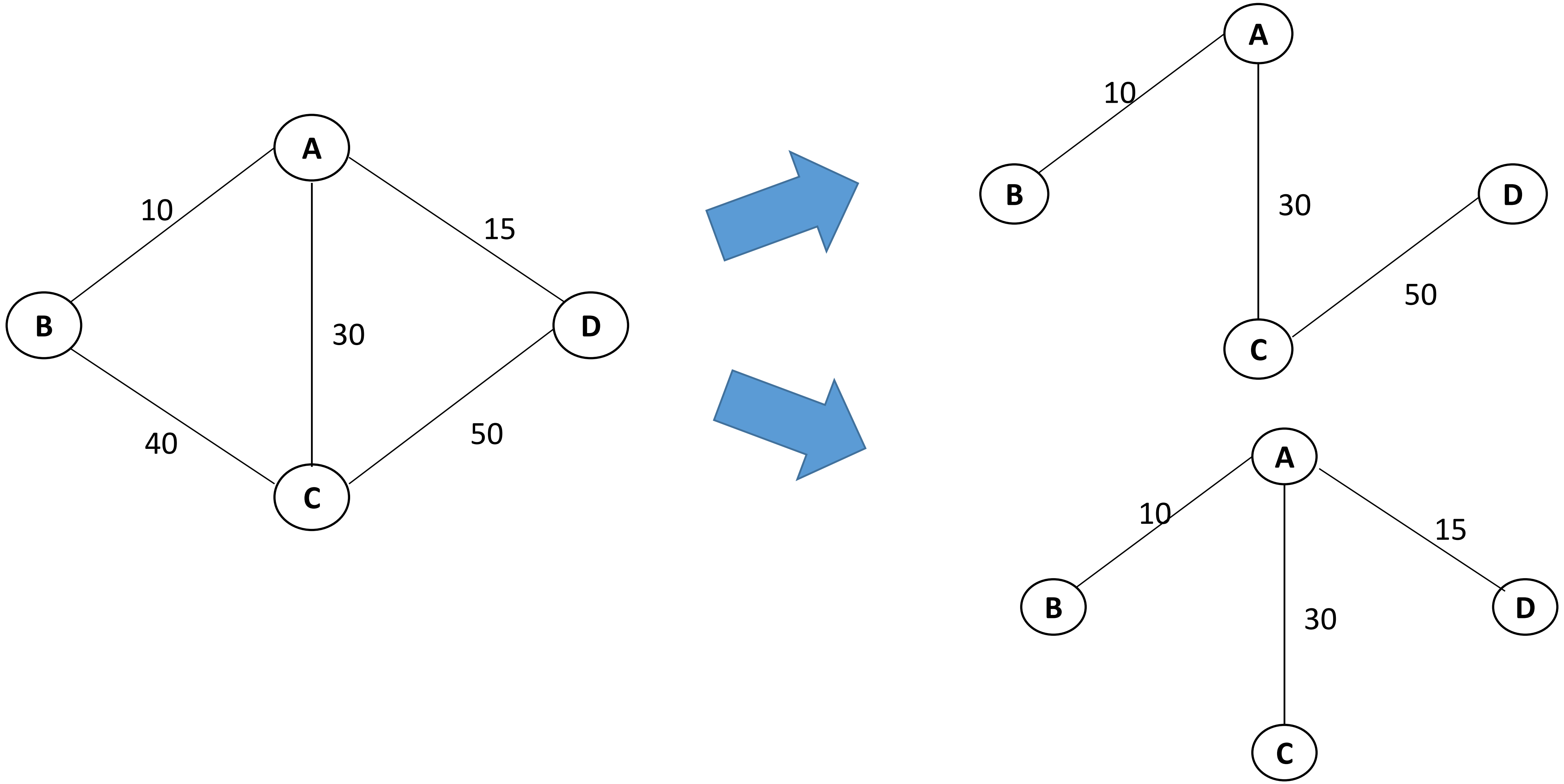
Minimum Spanning Tree

- Path Problems
  1. Simple Paths
  2. **Shortest Path Problem**
     - Single source shortest paths
     - All-pair shortest paths
  3. Find Cycles
  4. Euler Path and Circuit Problem
  5. Hamiltonian Path and Circuit Problem (or TSP)

Spanning Tree

- For a weighted undirected graph, Spanning Tree is a sub-graph that connects all the vertices together using the minimum number of edges required
- The graph should be connected: There should be no cycles
- For ***n*** number of *vertices*, *(n-1)* edges are needed
- Hence for four vertices, we need three edges without any cycles

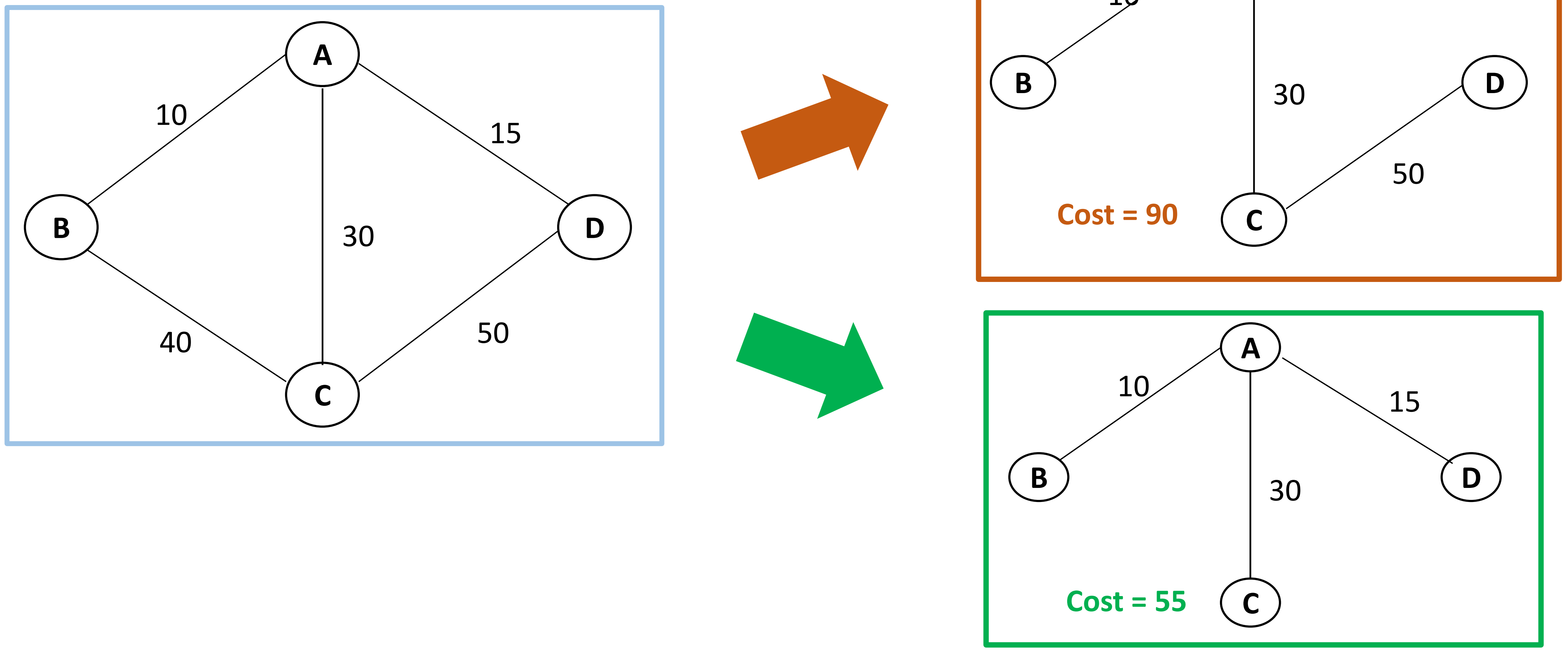
Many Spanning Trees for a Single Graph



Minimum Spanning Tree (MST)

- **Cost of any spanning tree**: Add weights of all the edges
- Many spanning tress for a single graph
- Calculate cost of all spanning trees and pick the minimum cost spanning tree
  - A tree with minimum weights is MST
- **Time complexity**: ***Exponential***
- Two Algorithms
  - Prim’s Algorithm
  - Kruskal’s Algorithm

Minimum Spanning Trees



Prim’s Algorithm: At a Glance

- Maintains two sets
- **MST Set**: vertices that are included in the spanning tree
- **Set 2**: vertices that are not yet included in the spanning tree

Prim’s MST Algorithm

- **Key Idea**: Find the local optimum in the hopes of finding a global optimum.
- Start from one vertex and keep adding edges with the lowest weight until all vertices are reached.

__Steps__:

1. Initialize the minimum spanning tree with a vertex chosen at random.
2. Find all the edges that connect the tree to new vertices, select the minimum and add it to the tree
3. Keep repeating step 2 until we get a minimum spanning tree

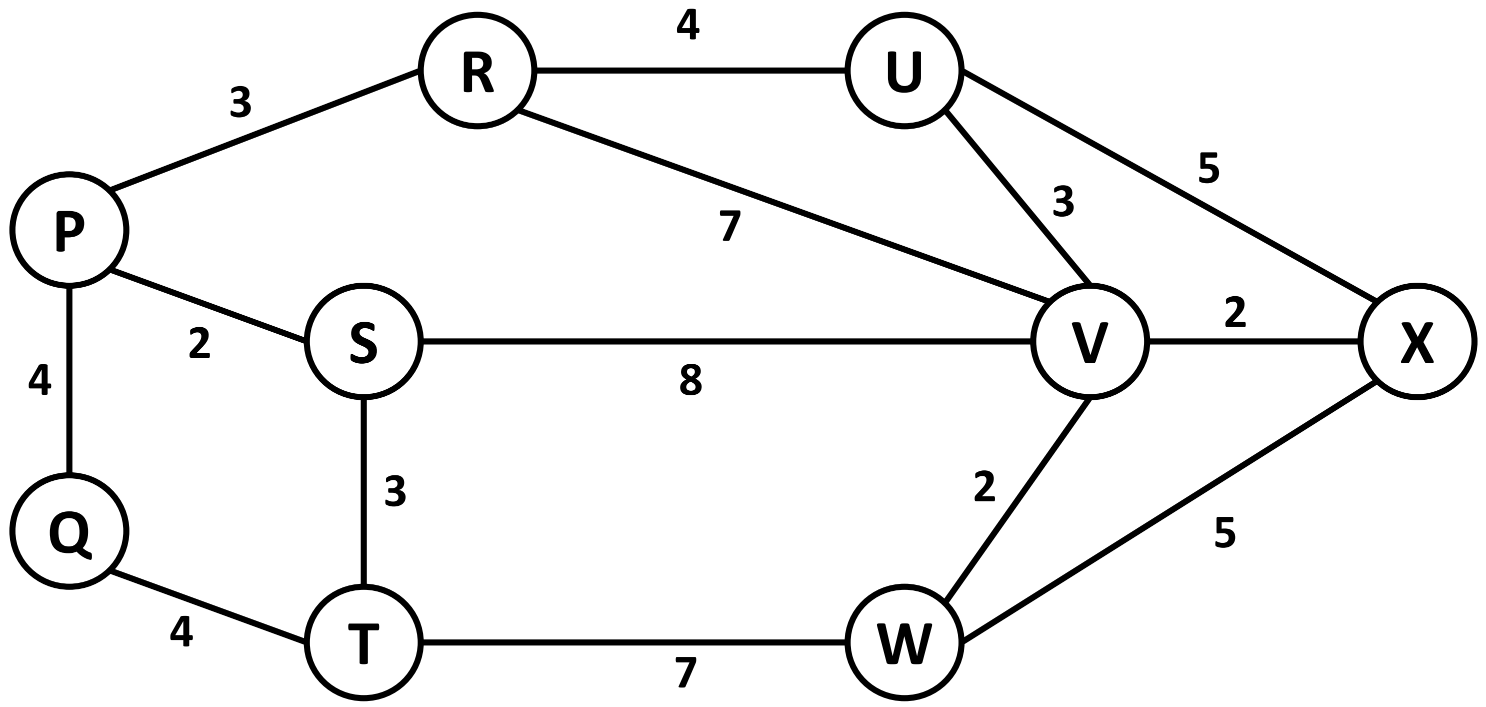
1. Create an array of size ***V*** and initialize it with NIL
2. Create a Priority Queue (min Heap) of size ***V***. Let the **min Heap be *Q***
3. Insert all vertices into **Q.** Assign **cost** of starting vertex to **0** and the **cost** of other vertices to infinite.

Prim's Algorithm: Pseudo-code

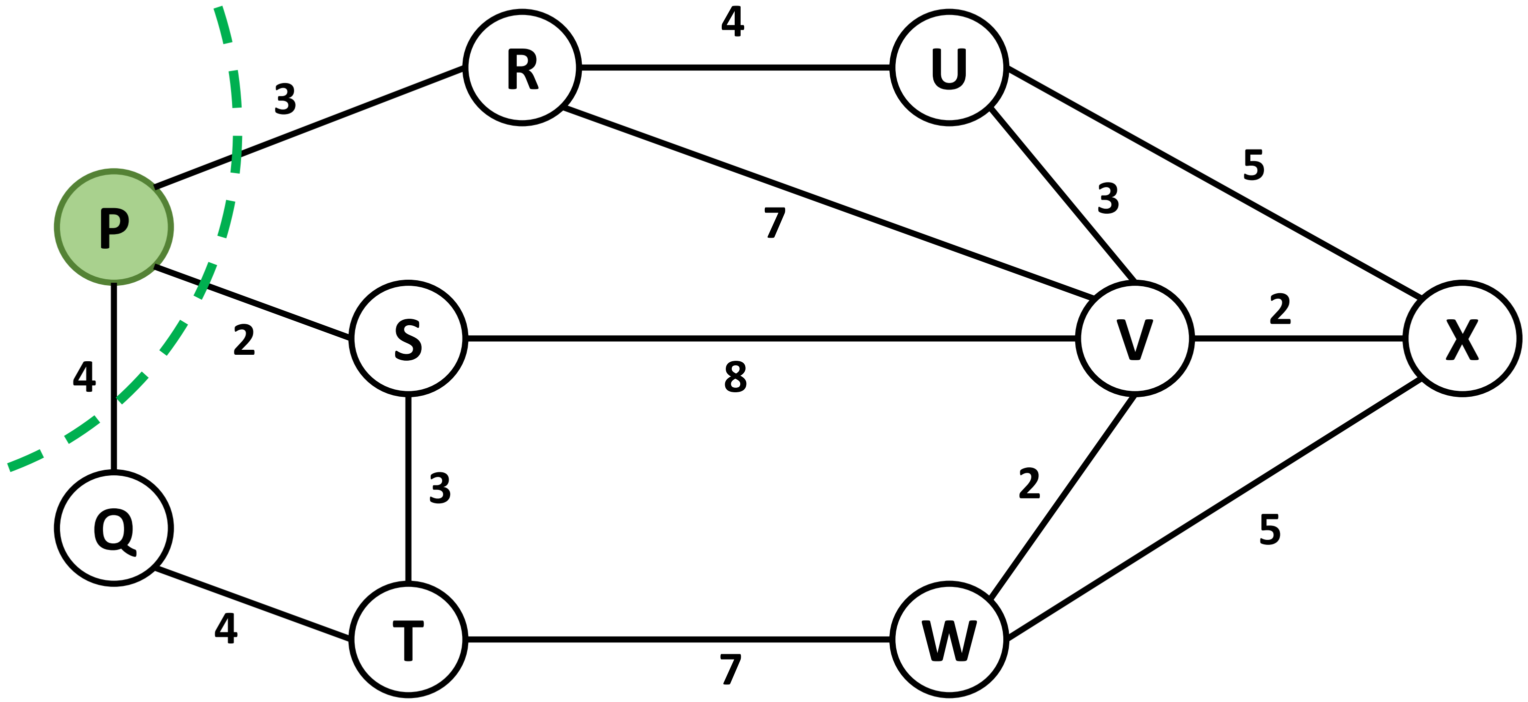
```
MST-PRIM(G, w, source)

1.  for each u ∈ V [G] do // for all vertices
2.      cost[u] ← ∞
3.      π[u] ← NULL
4.  cost[source] ← 0
5.  Q ← V [G] // set 2
6.  while Q ≠ Ø do
7.      u ← EXTRACT-MIN(Q)
8.      for each v ∈ Adj[u] do
9.          if v ∈ Q and w(u, v) < cost[v]
10.             π[v] ← u
11.             cost[v] ← w(u, v)
```

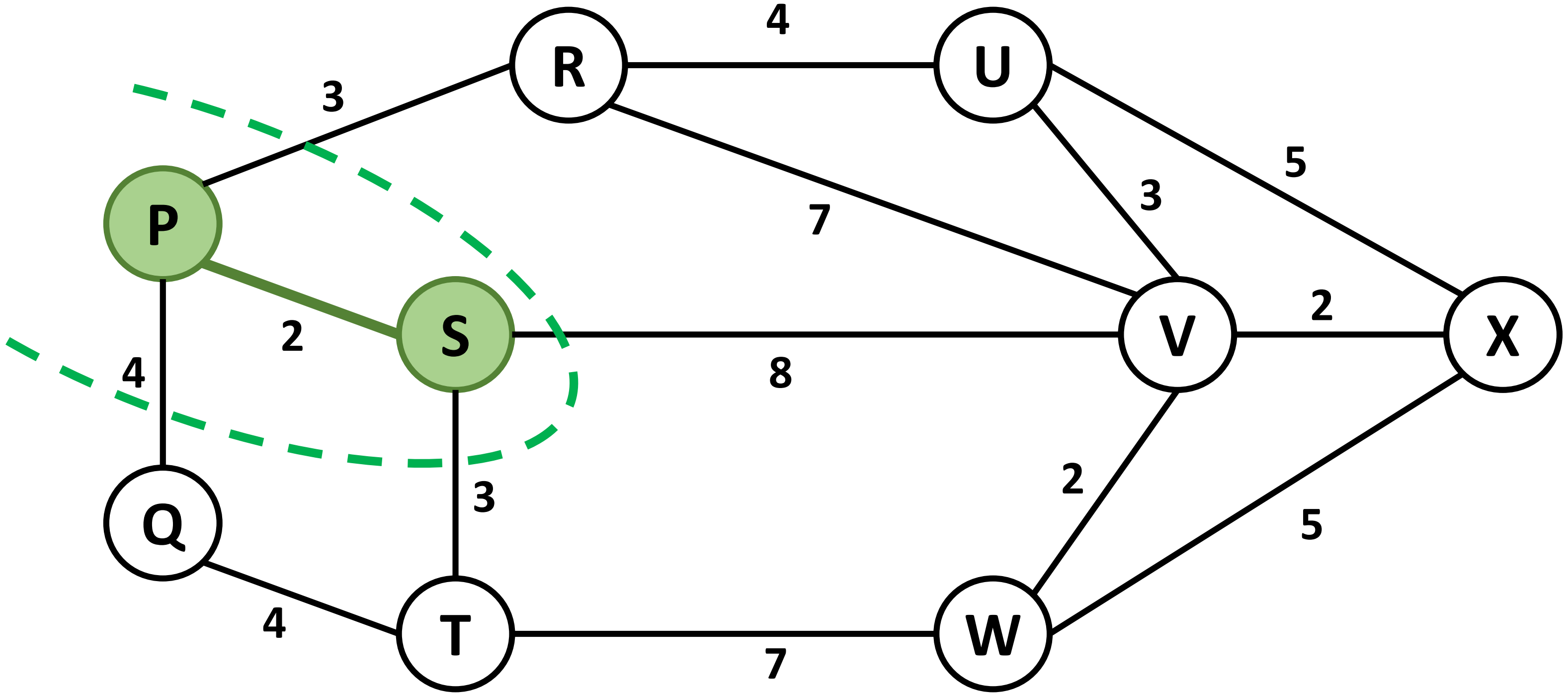
Prim’s Algorithm: An Example



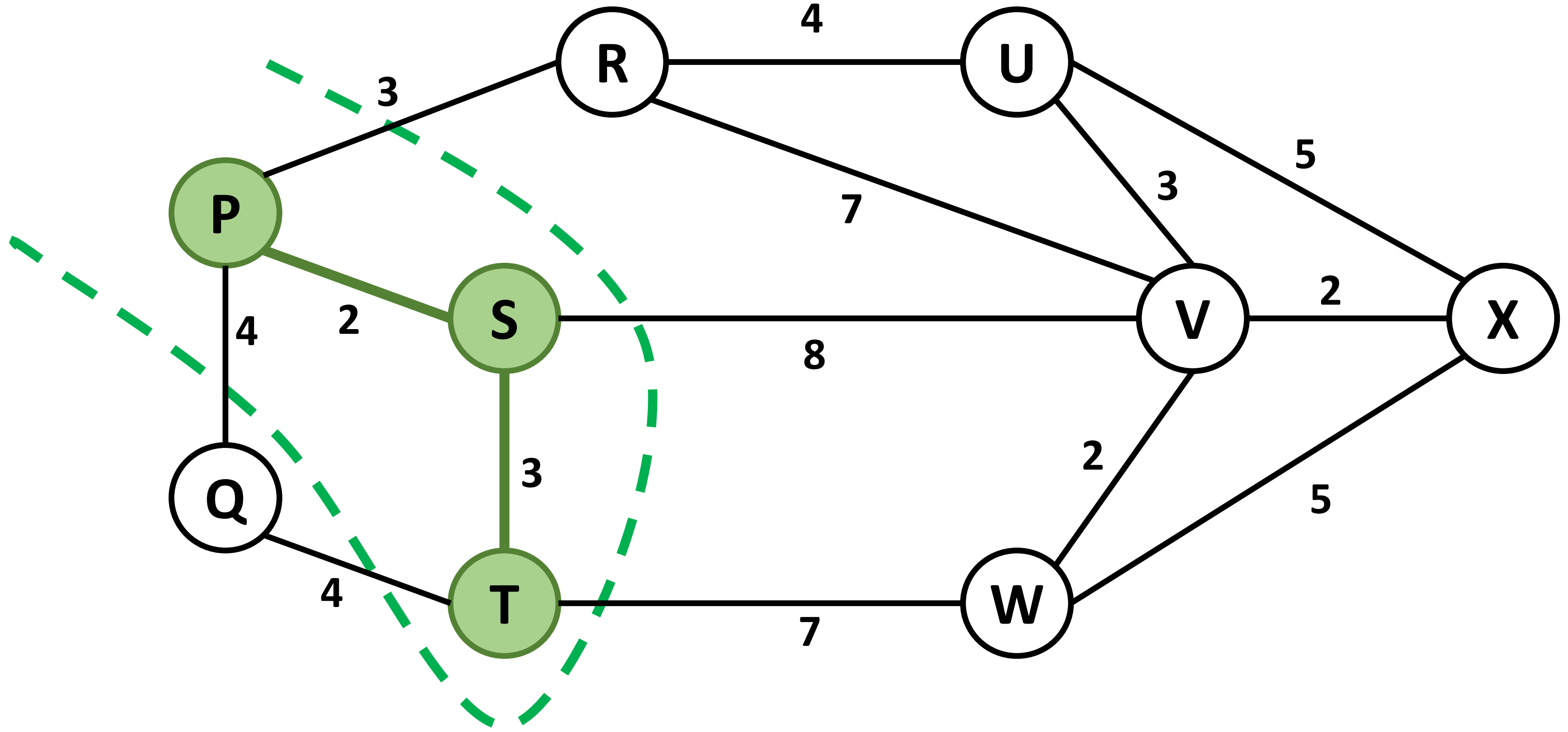
Prim’s Algorithm – An Example: Step 1



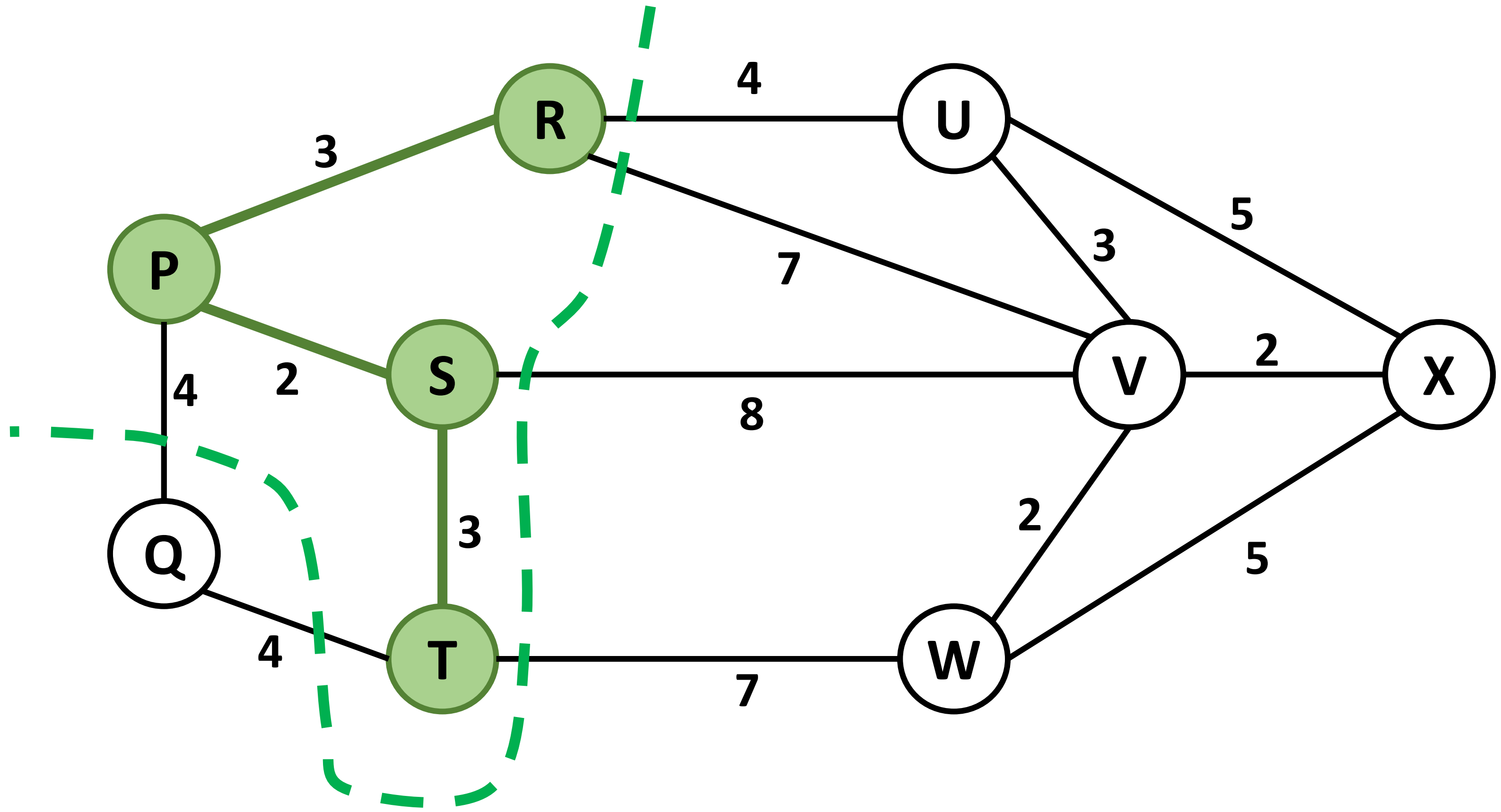
Prim’s Algorithm – An Example: Step 2



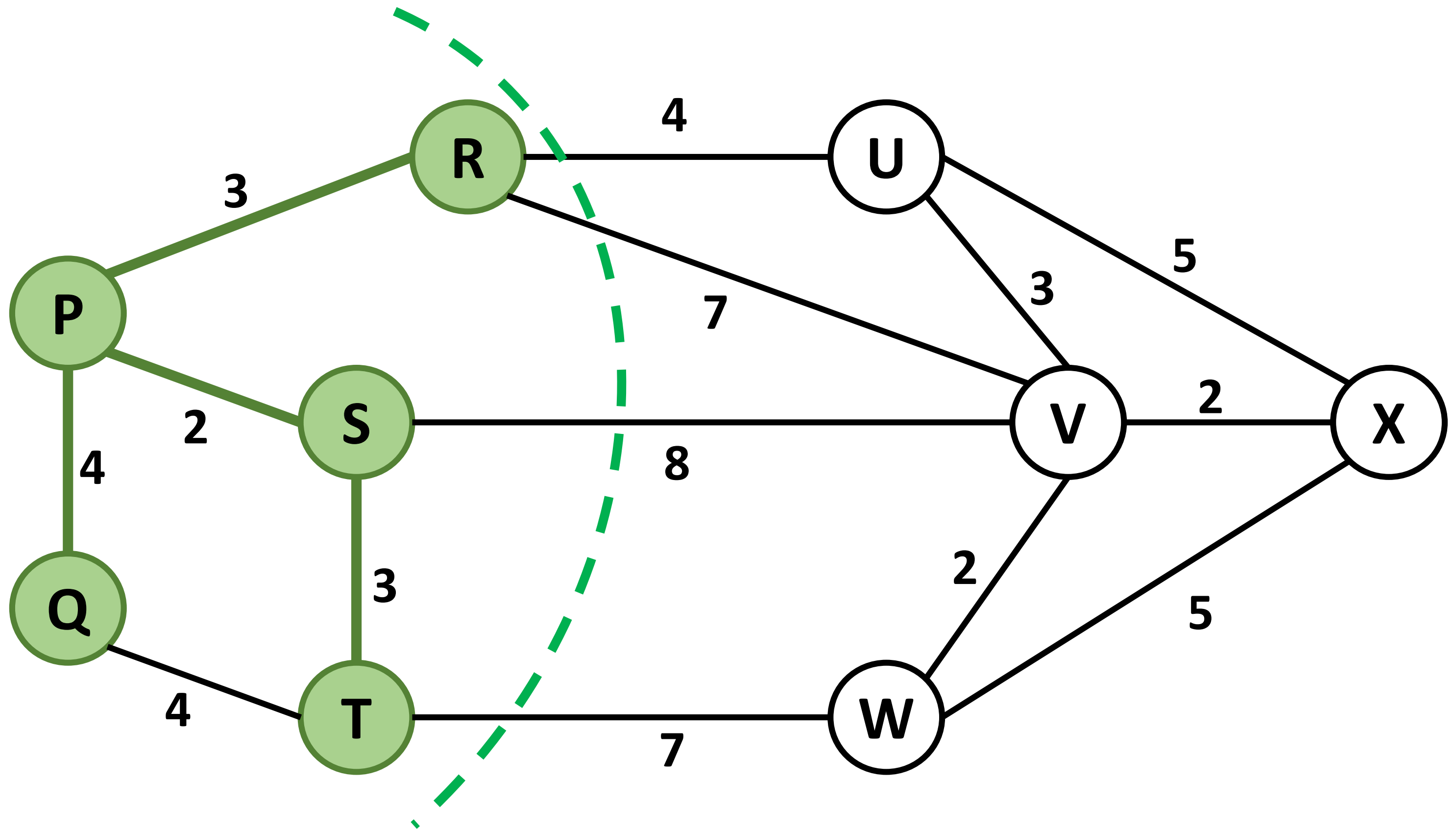
Prim’s Algorithm – An Example: Step 3



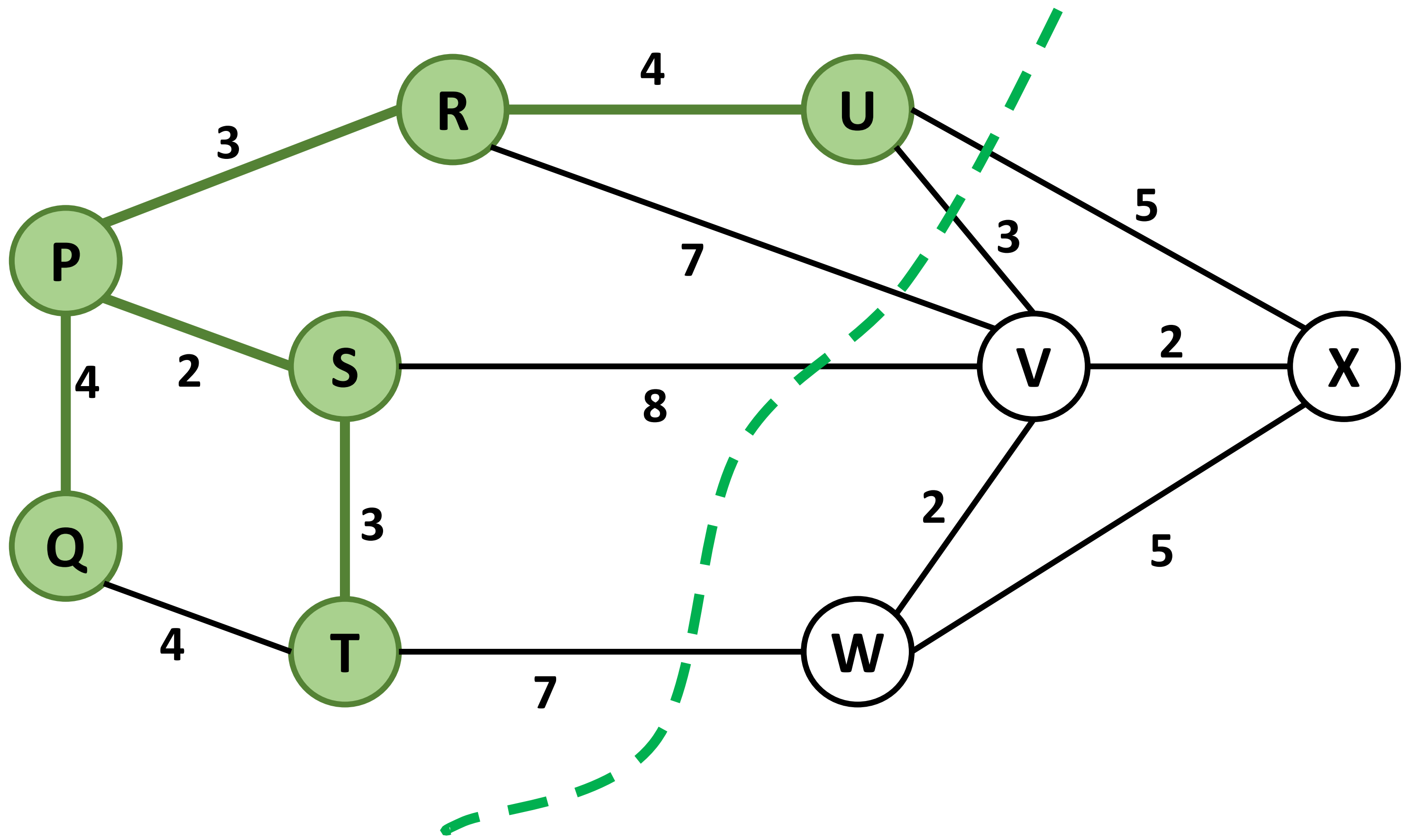
Prim’s Algorithm – An Example: Step 4



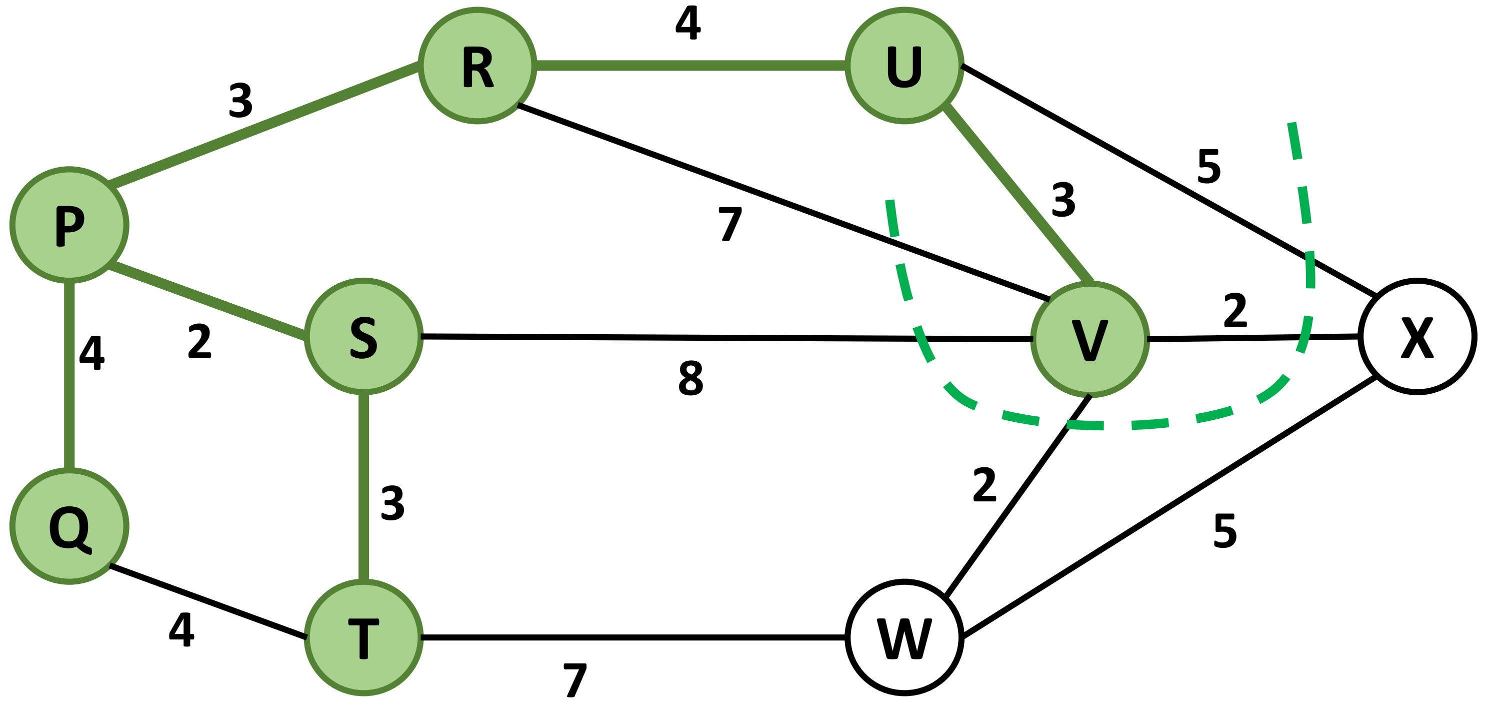
Prim’s Algorithm – An Example: Step 5



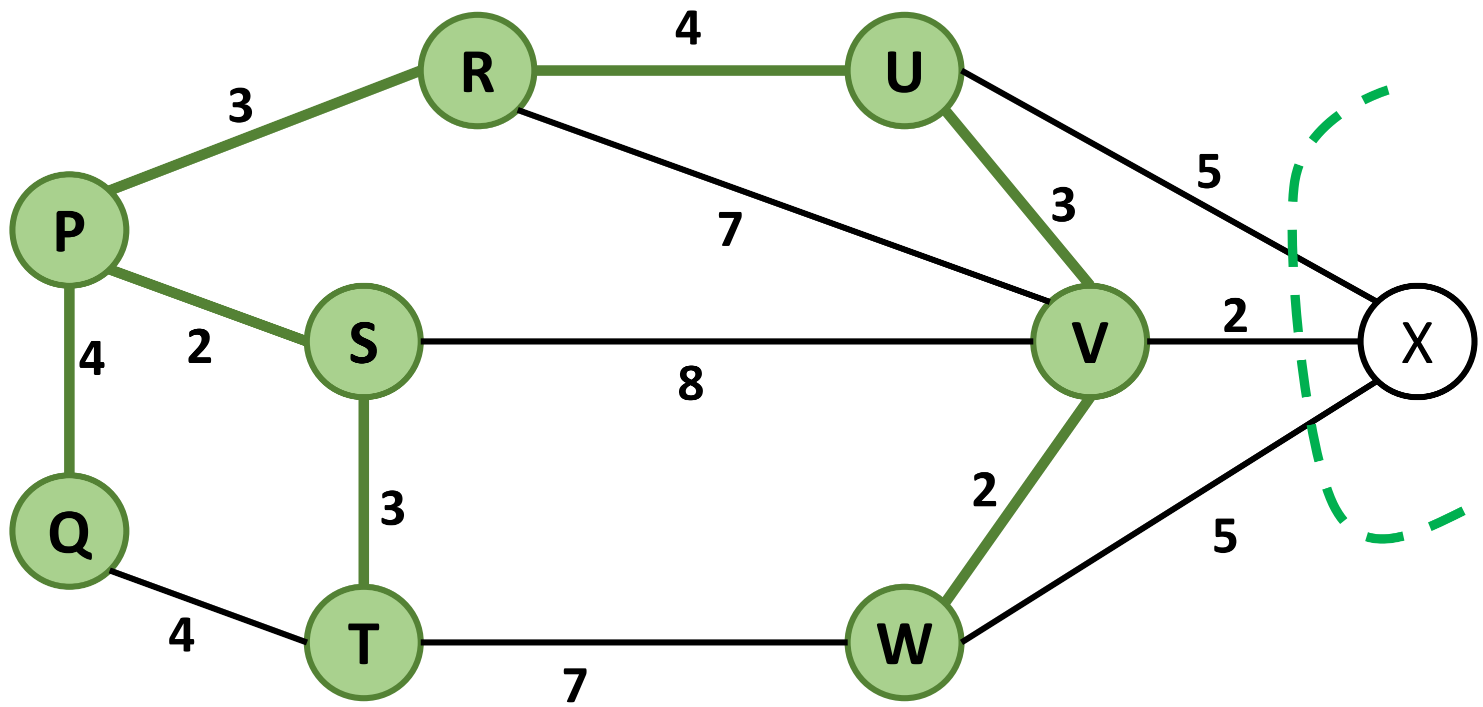
Prim’s Algorithm – An Example: Step 6



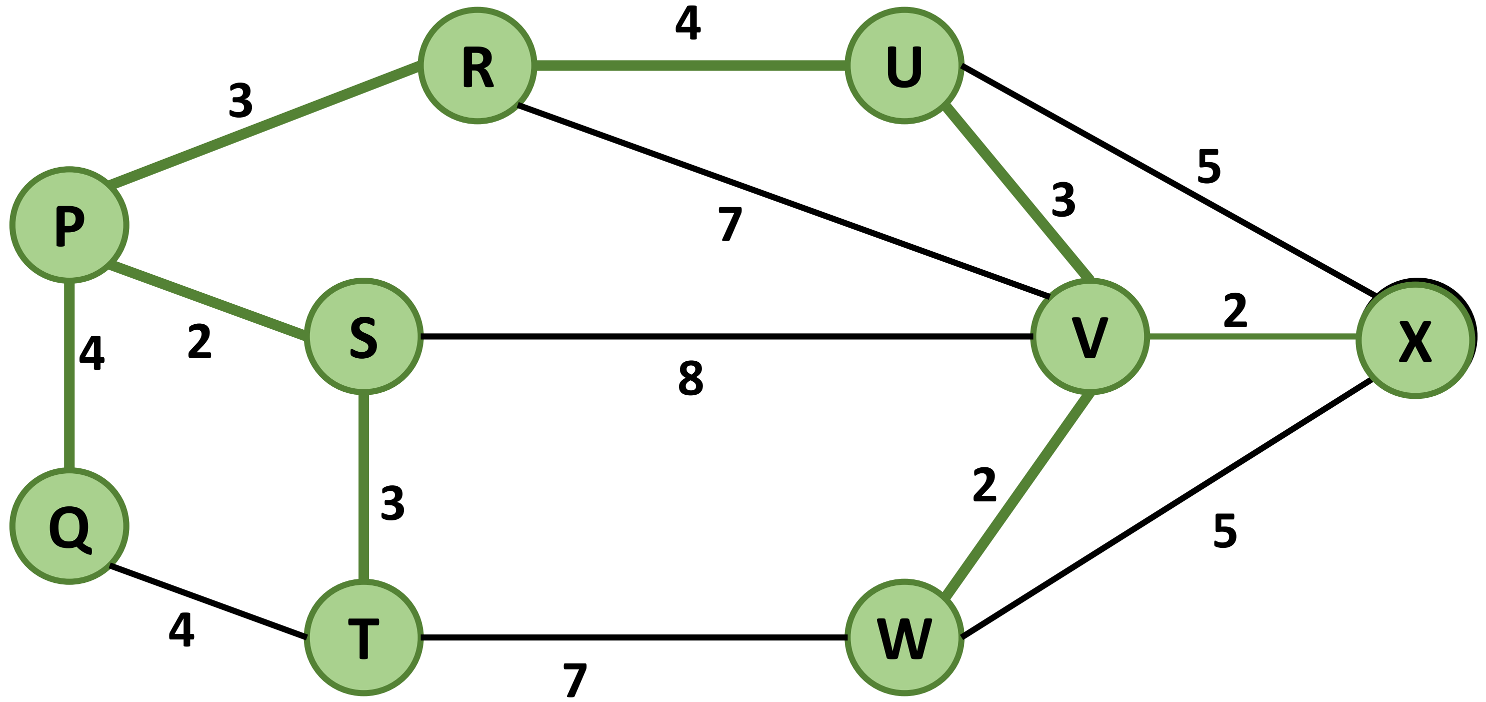
Prim's Algorithm – An Example: Step 7



Prim's Algorithm – An Example: Step 8



Prim’s Algorithm – An Example: Step 9



Prim’s Algorithm

- The algorithm was developed in 1930 by Czech mathematician Vojtěch Jarník
- Later rediscovered and republished by computer scientists Robert C. Prim in 1957 and Edsger W. Dijkstra in 1959.
- Therefore, it is also sometimes called the **Jarník's algorithm**, **Prim–Jarník algorithm**, **Prim–Dijkstra algorithm** or the **DJP algorithm**.

https://en.wikipedia.org/wiki/Prim%27s_algorithm

Minimum Spanning Trees

- A tree of the nodes of the graph with minimum total edge weight.
- Both Prim’s and Kruskal’s algorithms are examples of greedy algorithms
- **Applications**
  - Reducing copper to connect multiple nodes in a electrical/electronic circuit
  - Minimizing network length (cable cost) to connect multiple routers / computers etc.

Thank you!