Minimum Spanning Trees

Mohammad Asad Abbasi

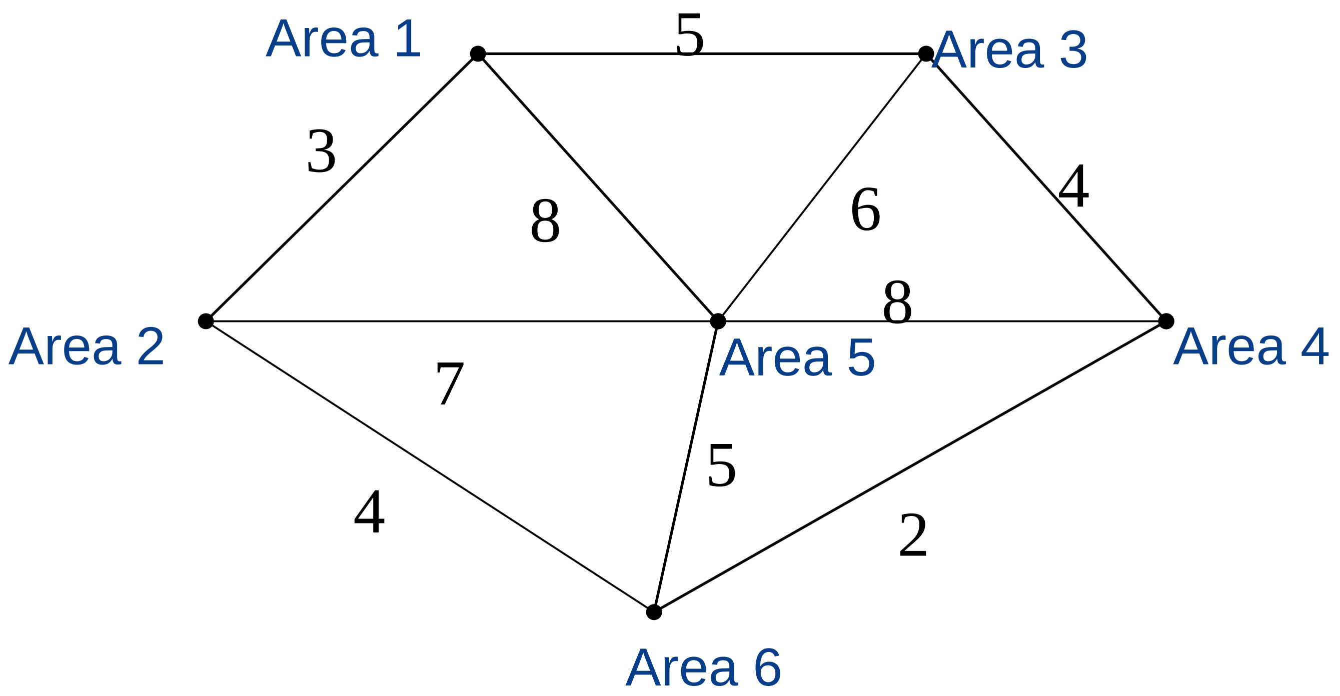
Lecture 15

Introduction

- Greedy algorithm is a technique for solving problems with the following properties:
  - The problem is an **optimization problem,** to find the solution that **minimizes** or **maximizes** some value (**cost/profit**)
  - The solution can be constructed in a **sequence** of **steps/choices**
  - For each choice point:
    - The choice must be feasible
    - The choice looks good or better than alternatives (Cost)
    - The choice cannot be revoked

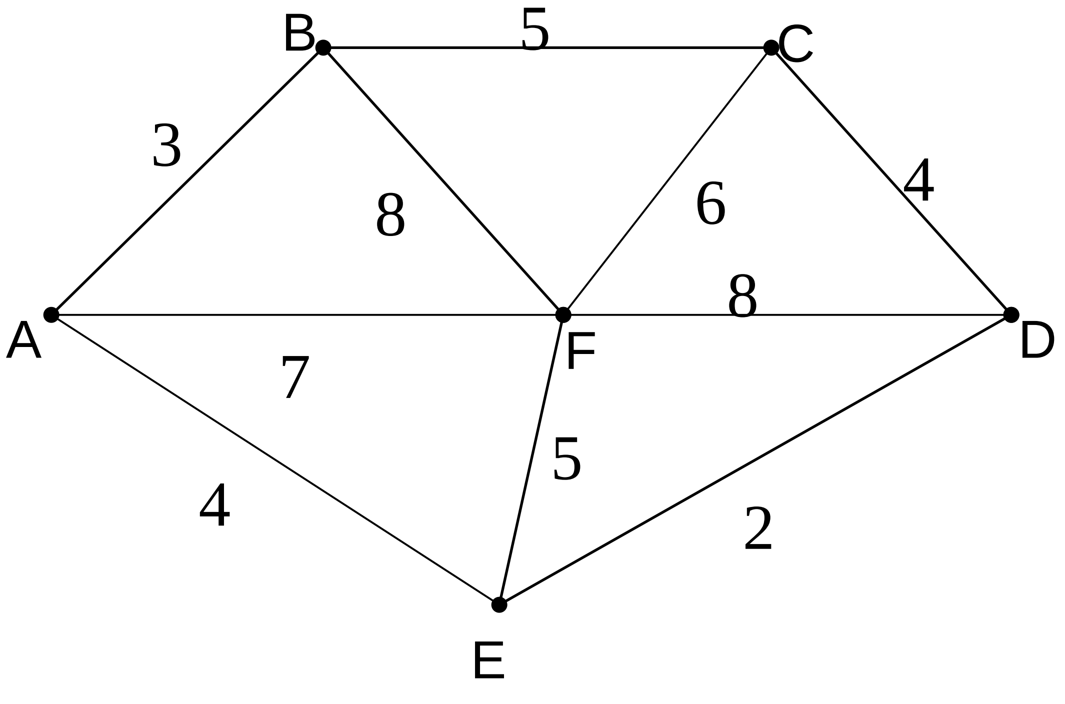
Example

- A cable company want to connect five villages to their network which currently extends to the market town. What is the minimum length of cable needed?



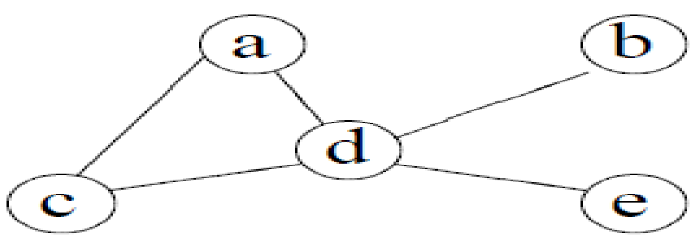
Example

- We model the situation as a network, then the problem is to find the minimum connector for the network

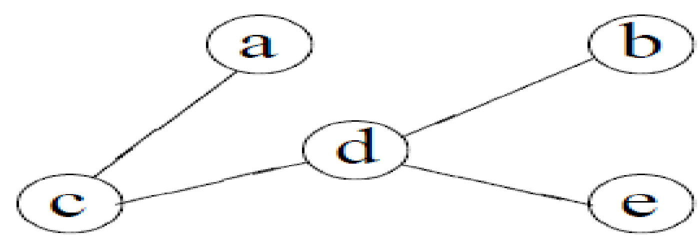


Spanning Trees

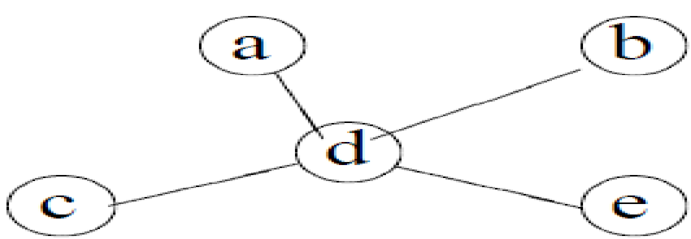
- **Spanning Trees:** A subgraph of a undirected graph is a spanning tree if it is a tree and contains every vertex of given graph



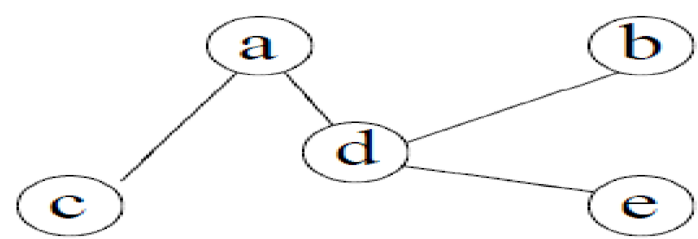
Graph



spanning tree 1



spanning tree 2



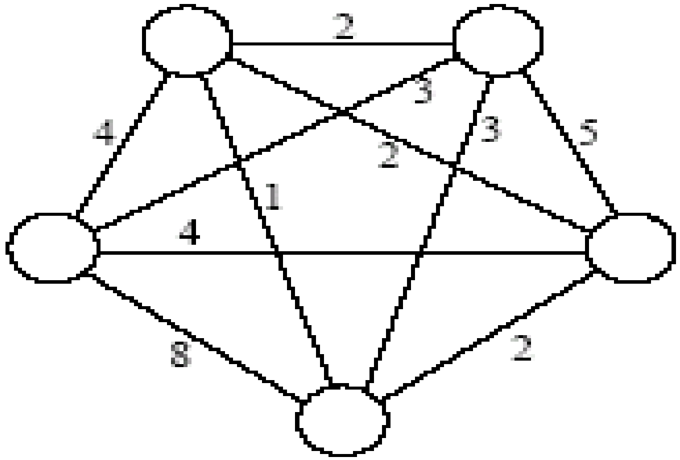
spanning tree 3

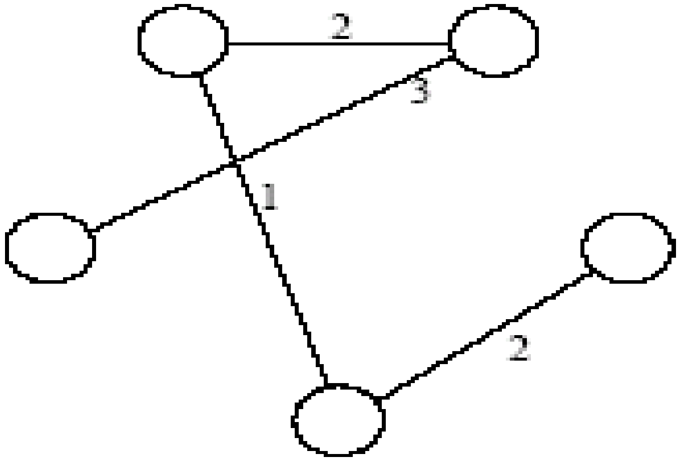
Minimum Spanning Trees

- A minimum spanning tree (MST) for a graph G = (V, E) is a **sub graph** G1 = (V1, E1) of G contains **all the vertices** of G
- If a graph G is **not** a **connected** graph, then it cannot have any spanning tree
- A minimum spanning tree (MST) for a weighted graph is a spanning tree with **minimum weight.** All the vertices in the weighted graph will be connected with minimum edge with **minimum weights**

What Makes A Spanning Tree The Minimum?

- **MST Criterion:**
  - When the sum of the edge weights in a spanning tree is the minimum over all spanning trees of a graph





Applications of Minimum-Cost Spanning Trees

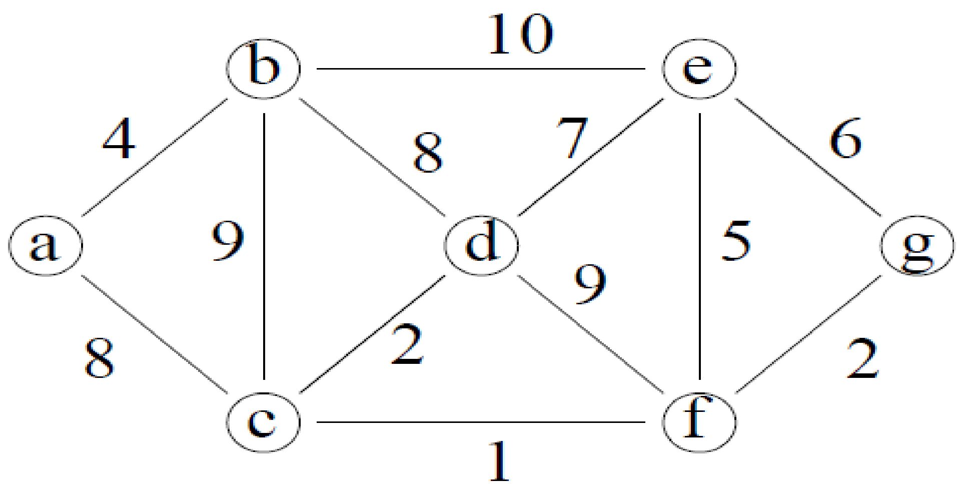
- Building **cable networks** that join n locations with minimum cost
- Building a **road network** that joins n cities with minimum cost
- Obtaining an independent set of circuit equations for an electrical network

Prim’s Algorithm

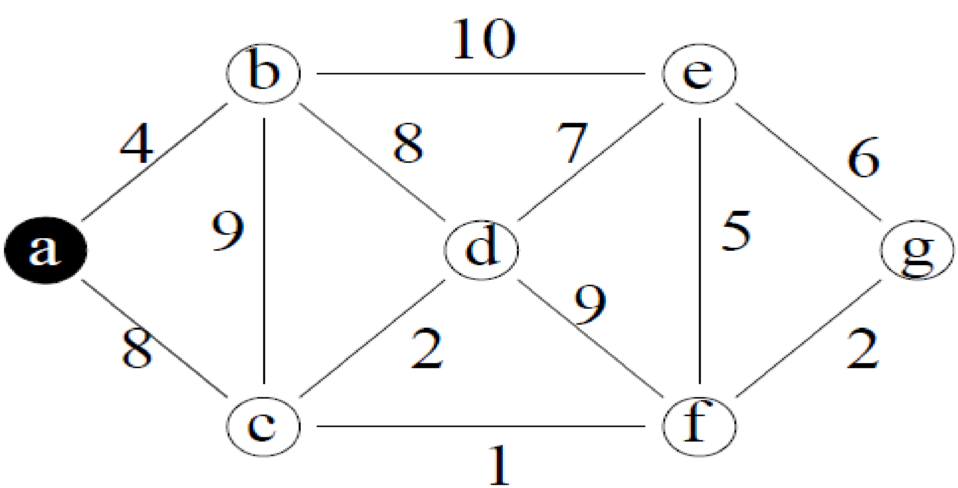
- Finds a minimum spanning tree for a connected weighted undirected graph
- It finds a subset of the edges that forms a tree that includes **every vertex,** where the total weight of all the edges in the tree is **minimized**

1) Select **any vertex**
2) Select the **shortest** edge connected to that vertex
3) Select the shortest edge connected to any vertex **already connected**
4) Repeat step 3 until all vertices have been connected

Prim's Algorithm Example



Connected graph



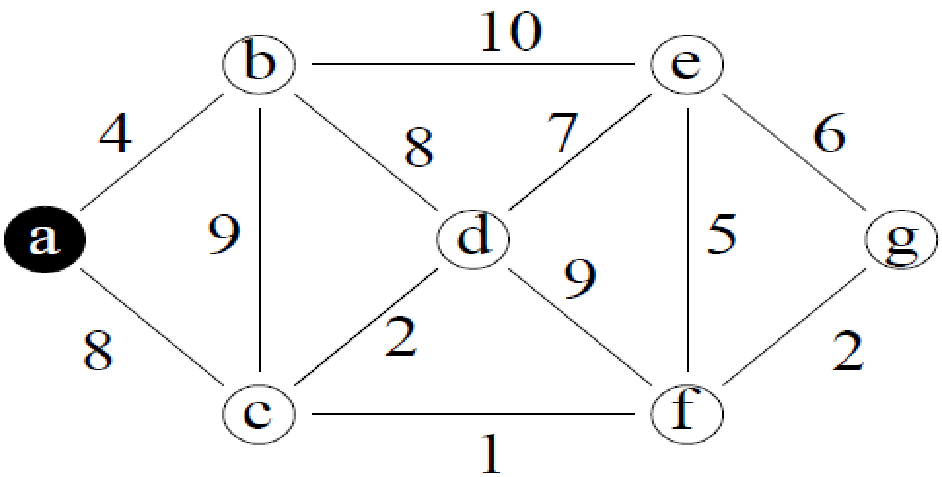
Step 0

S={a}

V \ S = {b,c,d,e,f,g}

lightest edge = {a,b}

Prim's Algorithm Example



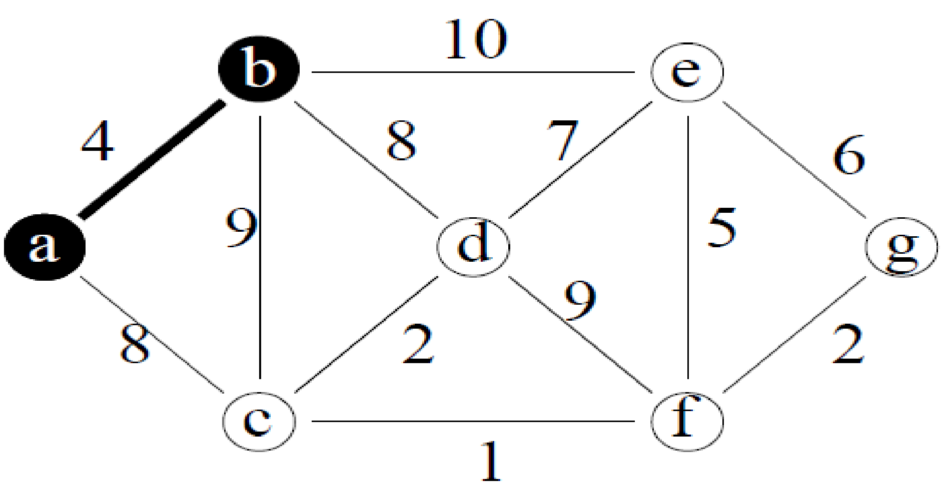
Step 1.1 before

S={a}

V \ S = {b,c,d,e,f,g}

A={}

lightest edge = {a,b}



Step 1.1 after

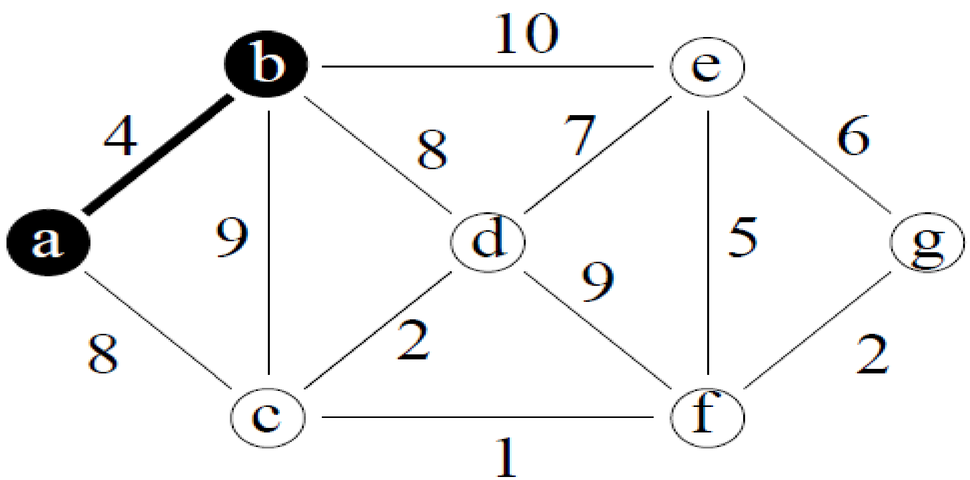
S={a,b}

V \ S = {c,d,e,f,g}

A={{a,b}}

lightest edge = {b,d}, {a,c}

Prim's Algorithm Example



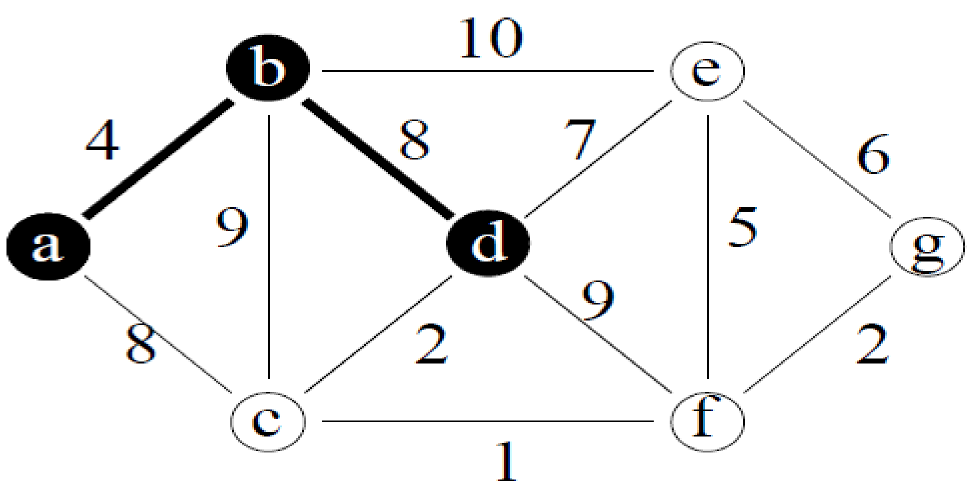
Step 1.2 before

S={a,b}

V \ S = {c,d,e,f,g}

A={{a,b}}

lightest edge = {b,d}, {a,c}



Step 1.2 after

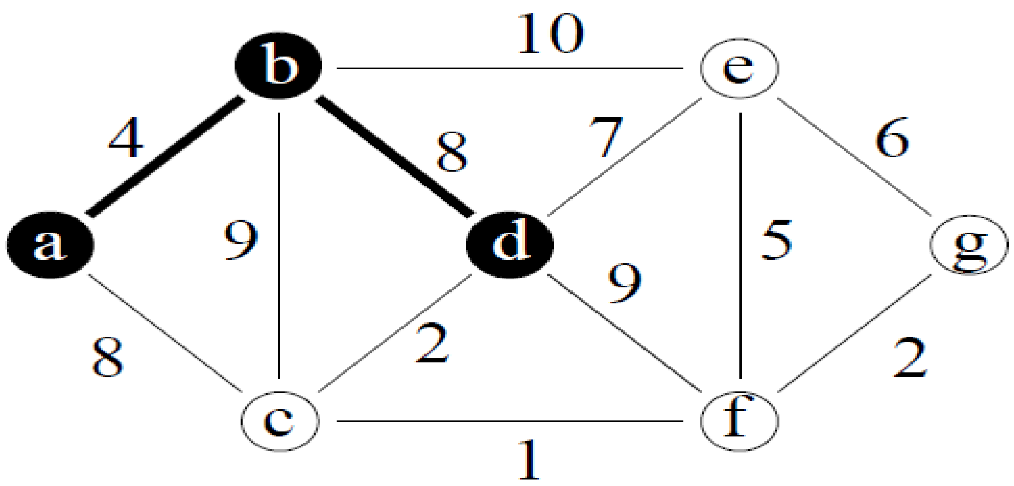
S={a,b,d}

V \ S = {c,e,f,g}

A={{a,b},{b,d}}

lightest edge = {d,c}

Prim's Algorithm Example



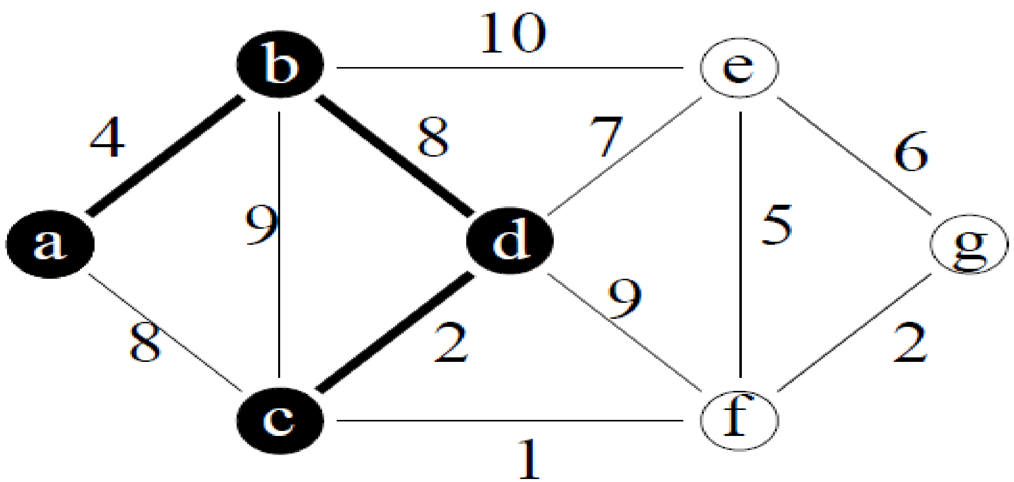
Step 1.3 before

S={a,b,d}

V \ S = {c,e,f,g}

A={{a,b},{b,d}}

lightest edge = {d,c}



Step 1.3 after

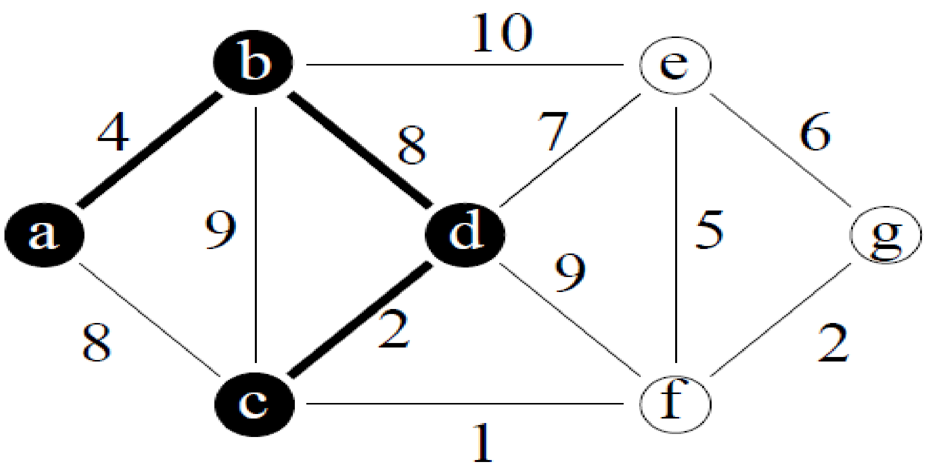
S={a,b,c,d}

V \ S = {e,f,g}

A={{a,b},{b,d},{c,d}}

lightest edge = {c,f}

Prim's Algorithm Example



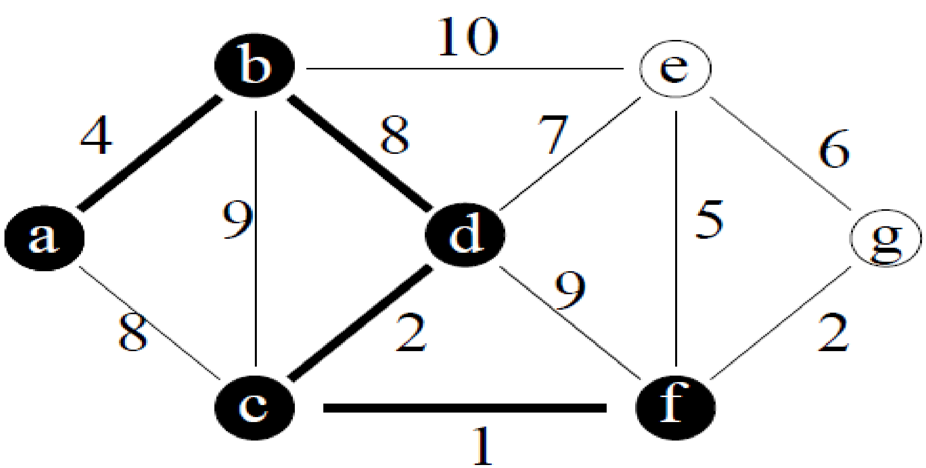
Step 1.4 before

S={a,b,c,d}

V \ S = {e,f,g}

A={{a,b},{b,d},{c,d}}

lightest edge = {c,f}



Step 1.4 after

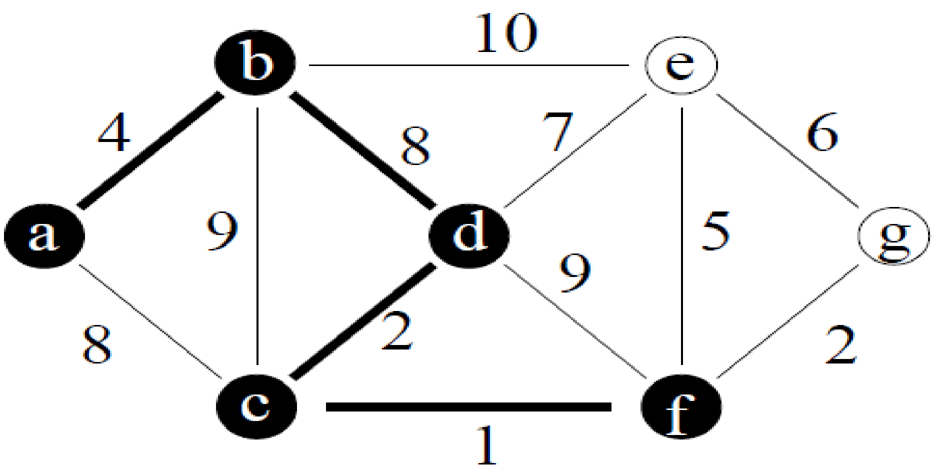
S={a,b,c,d,f}

V \ S = {e,g}

A={{a,b},{b,d},{c,d},{c,f}}

lightest edge = {f,g}

Prim’s Algorithm Example



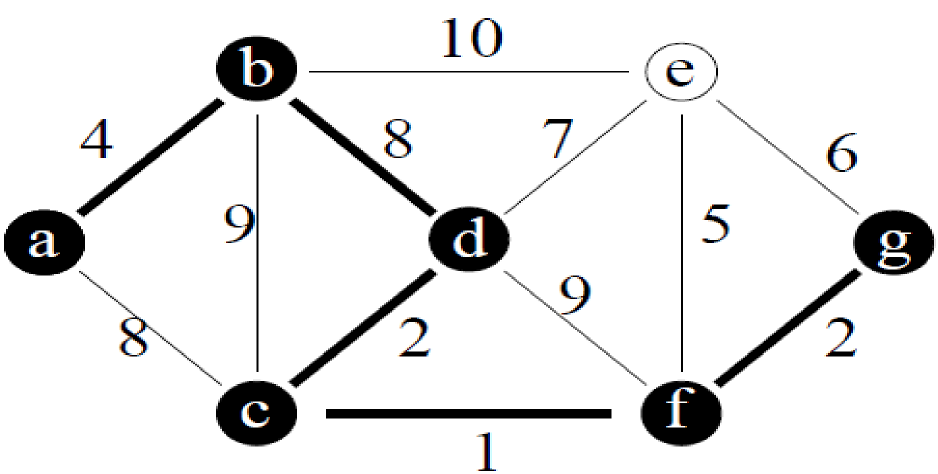
Step 1.5 before

S={a,b,c,d,f}

V \ S = {e,g}

A={{a,b},{b,d},{c,d},{c,f}}

lightest edge = {f,g}



Step 1.5 after

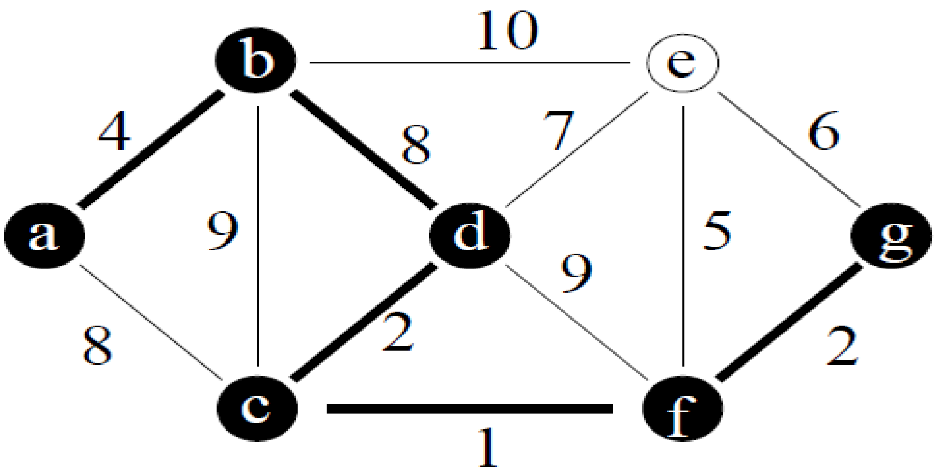
S={a,b,c,d,f,g}

V \ S = {e}

A={{a,b},{b,d},{c,d},{c,f},
{f,g}}

lightest edge = {f,e}

Prim’s Algorithm Example



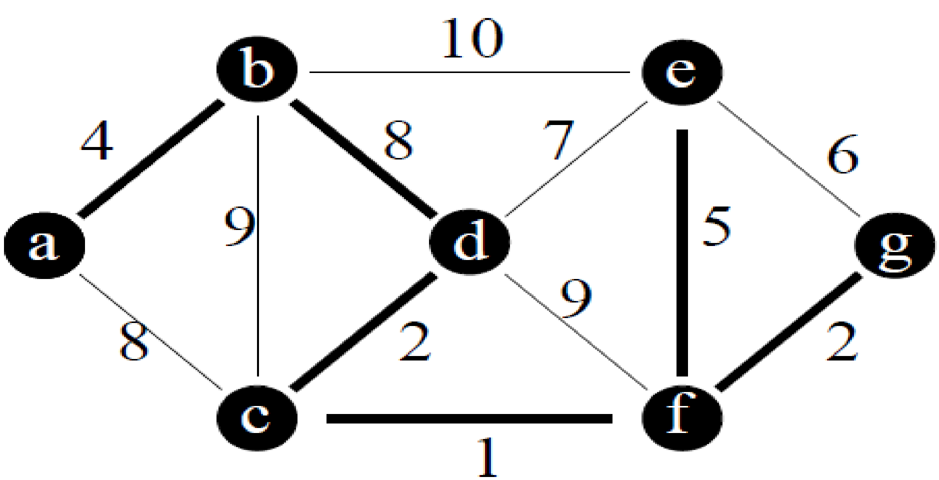
Step 1.6 before

S={a,b,c,d,f,g}

V \ S = {e}

A={ {a,b},{b,d},{c,d},{c,f},
{f,g} }

lightest edge = {f,e}



Step 1.6 after

S={a,b,c,d,e,f,g}

V \ S = {}

A={ {a,b},{b,d},{c,d},{c,f},
{f,g},{f,e} }

MST completed

Dijkstra’s Algorithm

- Finds shortest (minimum weight) path between a particular pair of vertices in a *weighted* directed graph with nonnegative edge weights
  - solves the "one vertex, shortest path" problem

- Works on both directed and undirected graphs. However, all edges must have nonnegative weights.

- Input: Weighted graph G={E,V} and source vertex *v*∈V, such that all edge weights are nonnegative

- Output: Lengths of shortest paths (or the shortest paths themselves) from a given source vertex *v*∈V to all other vertices

Dijkstra's Algorithm

- It starts by assigning some initial values for the distances from node s and to every other node in the network

- It operates in steps, where at each step the algorithm improves the distance values

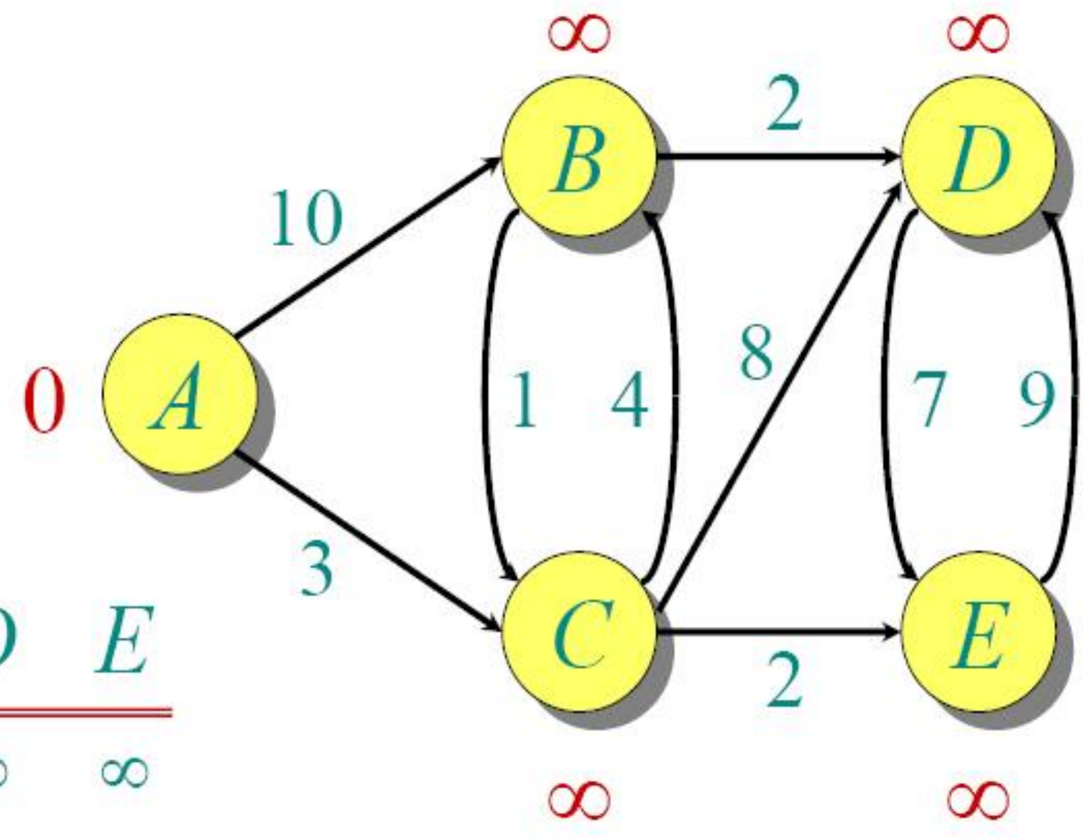
- At each step, the shortest distance from node s to another node is determined

Dijkstra's algorithm - Pseudocode

```
dist[s] ←0                                    (distance to source vertex is zero)
for  all v ∈ V–{s}
      do  dist[v] ←∞                          (set all other distances to infinity)
S←∅                                           (S, the set of visited vertices is initially empty)
Q←V                                           (Q, the queue initially contains all
vertices)
while Q ≠∅                                    (while the queue is not empty)
do   u ← mindistance(Q,dist)                  (select the element of Q with the min. distance)
     S←S∪{u}                                  (add u to list of visited vertices)
      for all v ∈ neighbors[u]
           do  if   dist[v] > dist[u] + w(u, v)                (if new shortest path found)
                    then     d[v] ←d[u] + w(u, v)     (set new value of shortest path)
                         (if desired, add traceback code)
return dist
```

Dijkstra's Algorithm Example

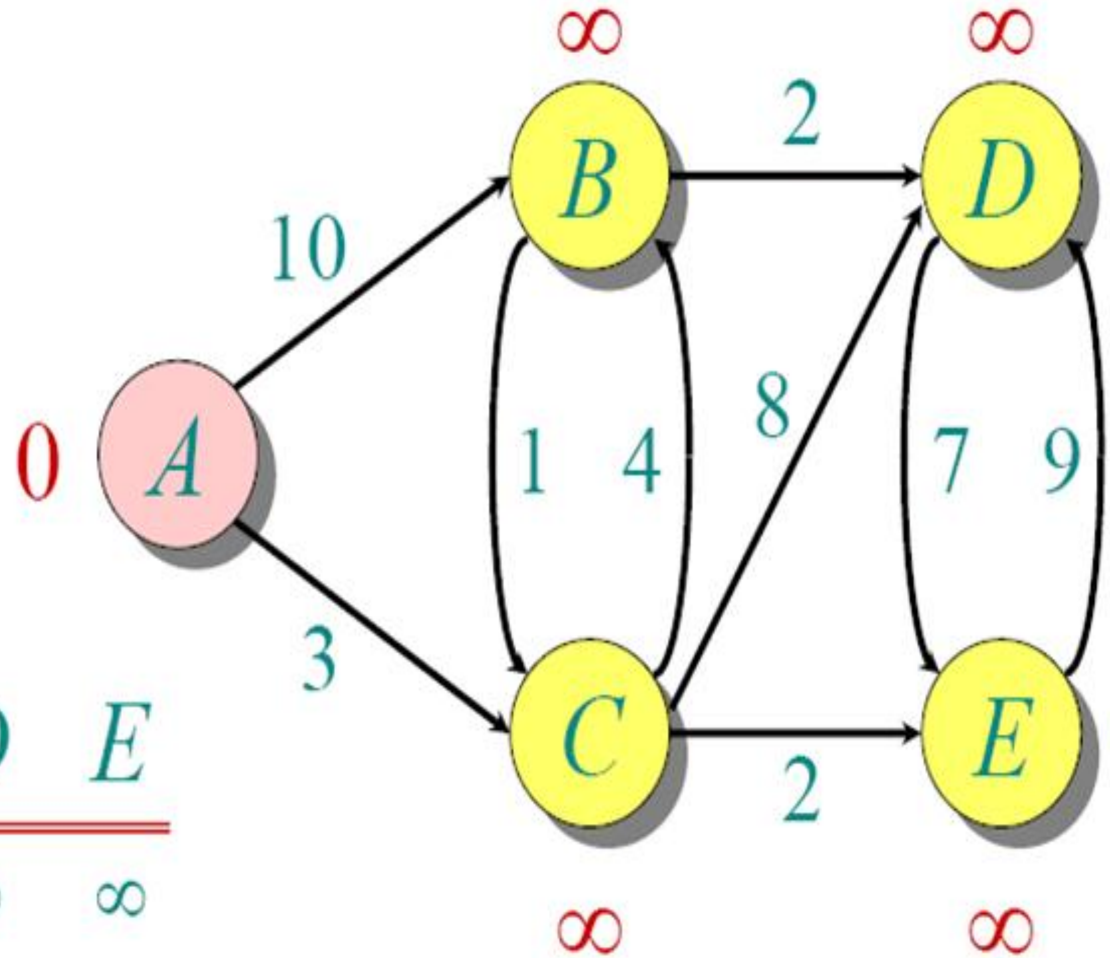
Initialize:



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

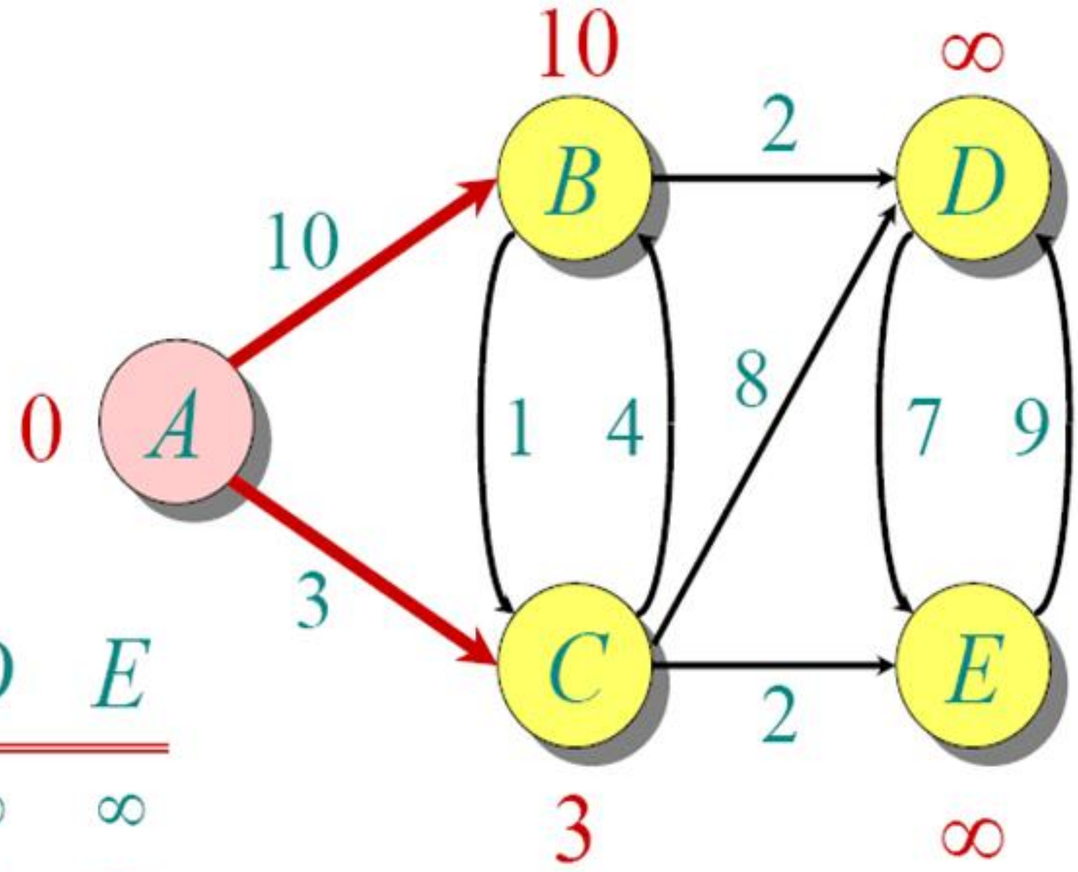
$S: \{\}$

Dijkstra's Algorithm Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

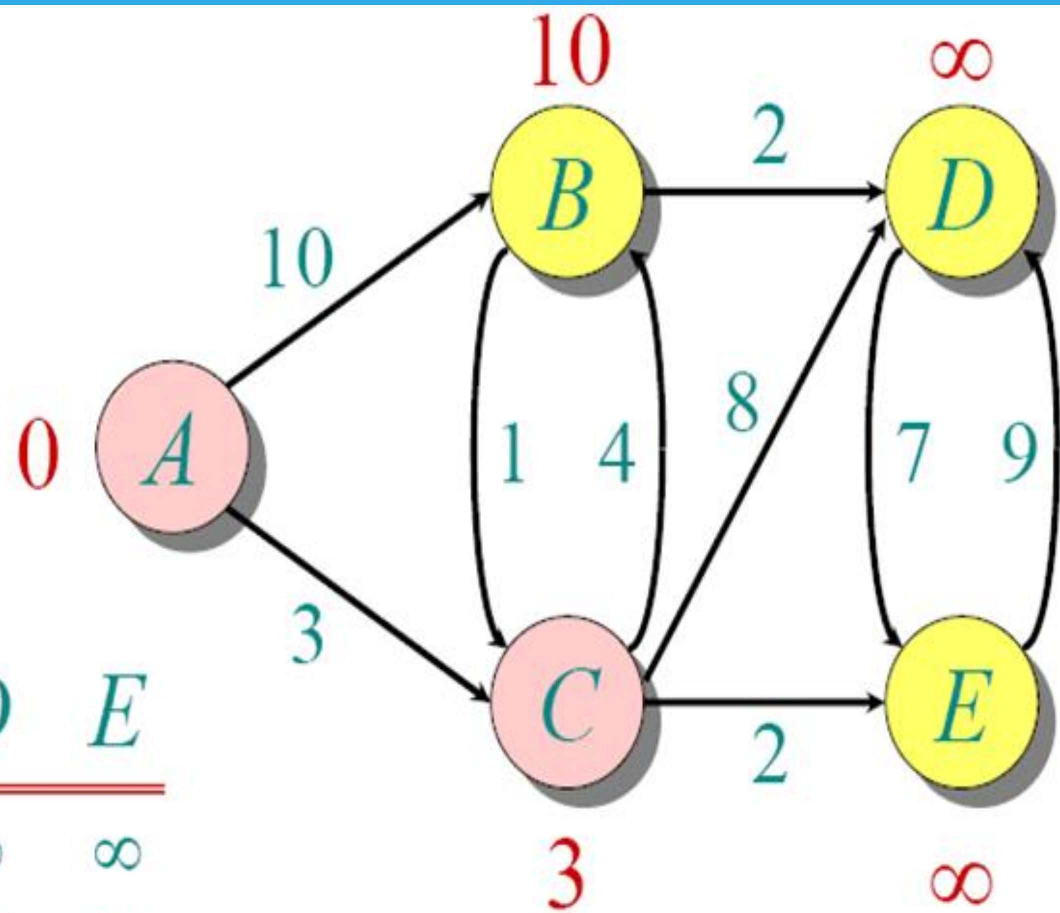
Dijkstra's Algorithm Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A }

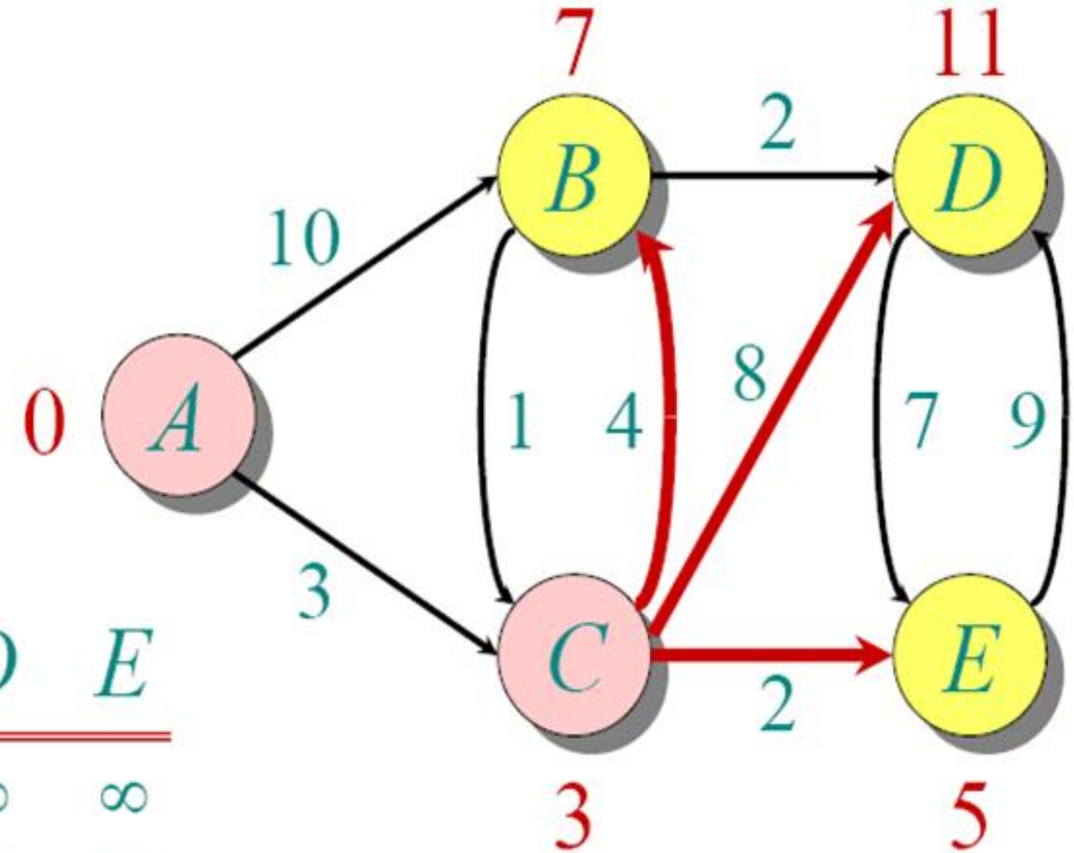
Dijkstra’s Algorithm Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A, C }

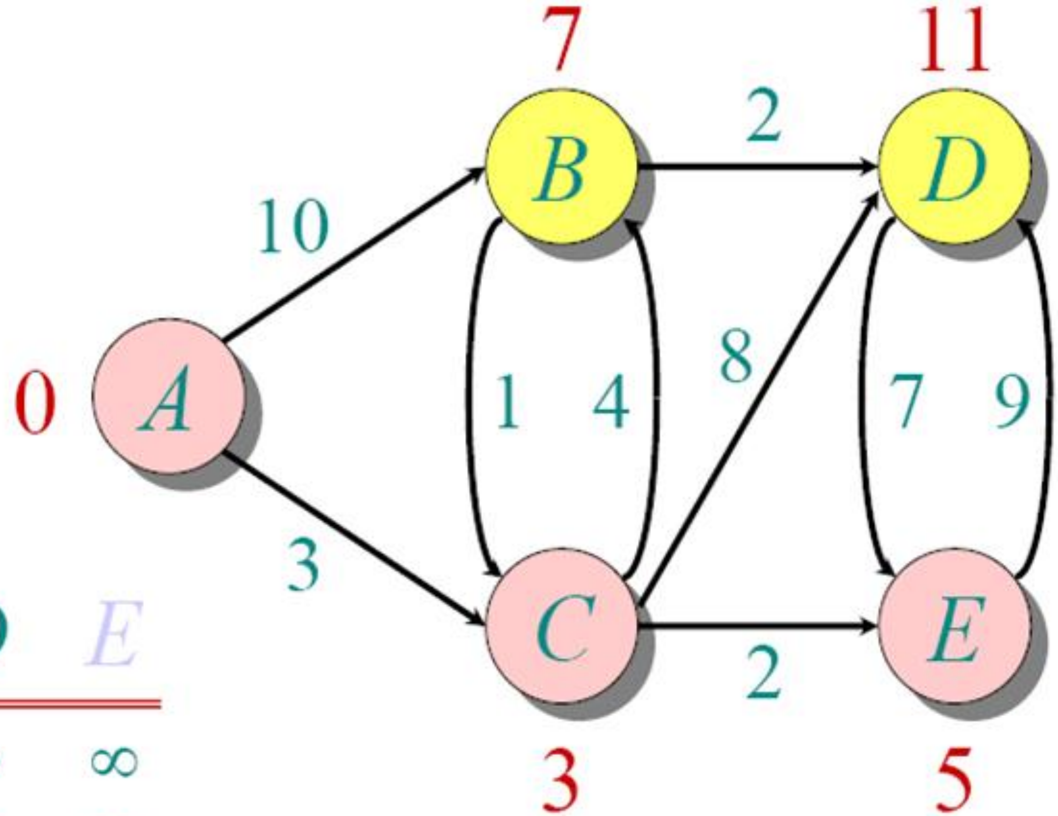
Dijkstra’s Algorithm Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { A, C }

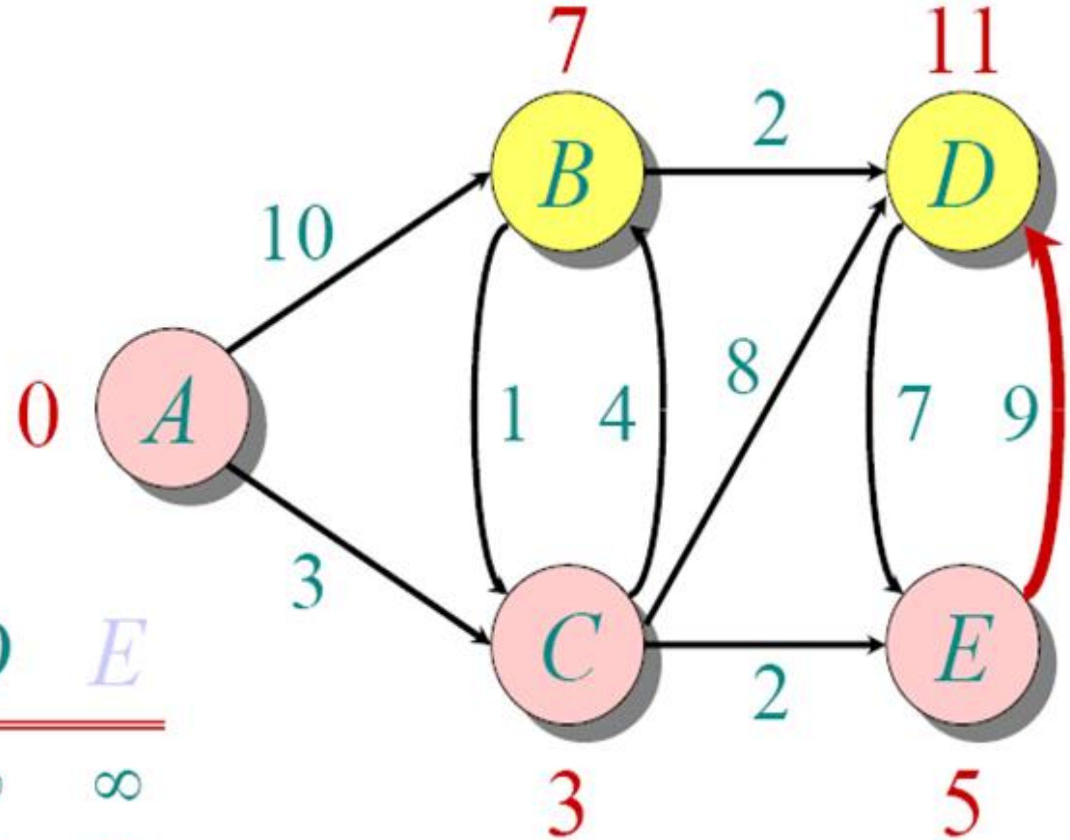
Dijkstra's Algorithm Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { A, C, E }

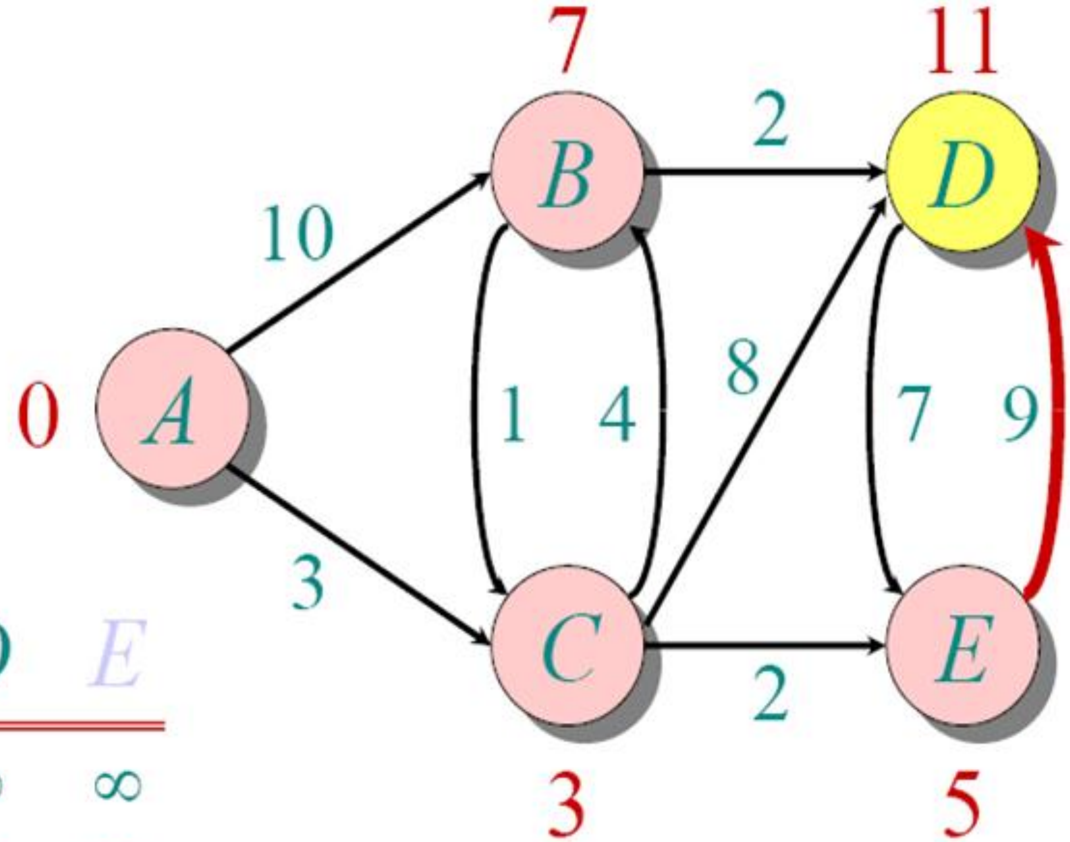
Dijkstra's Algorithm Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E }

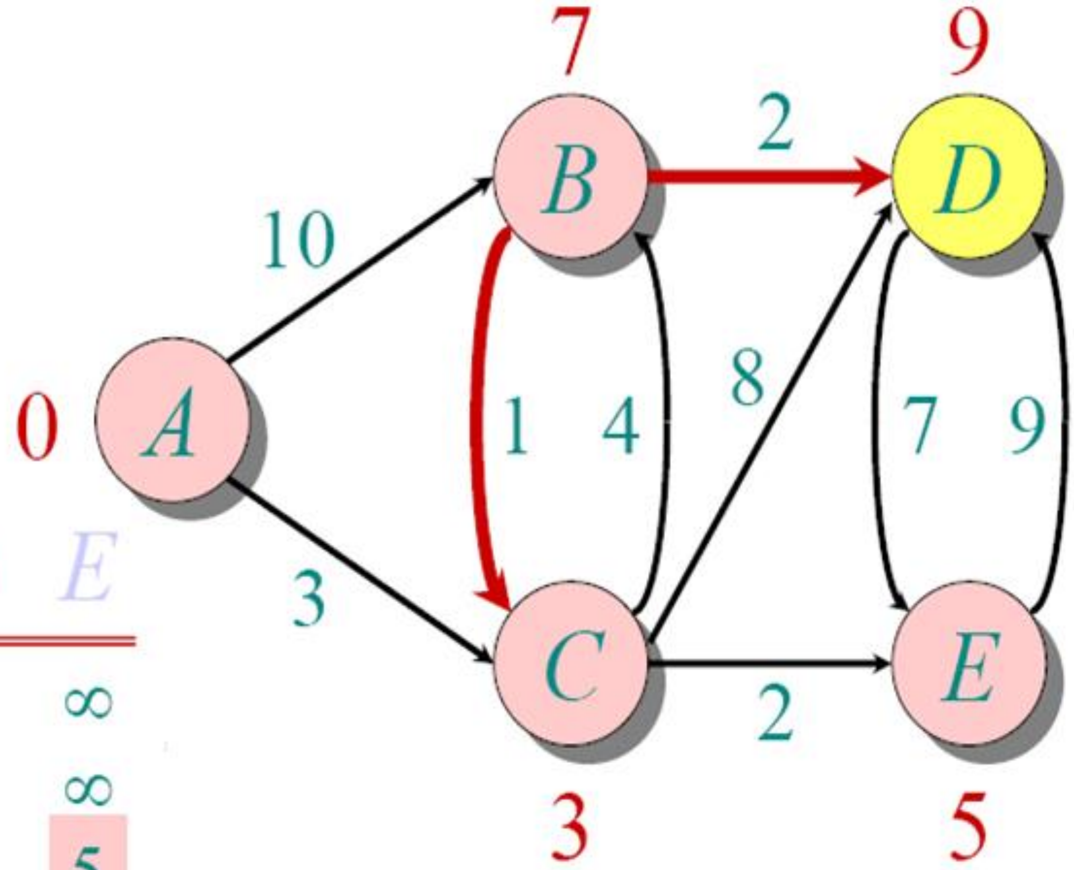
Dijkstra's Algorithm Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E, B }

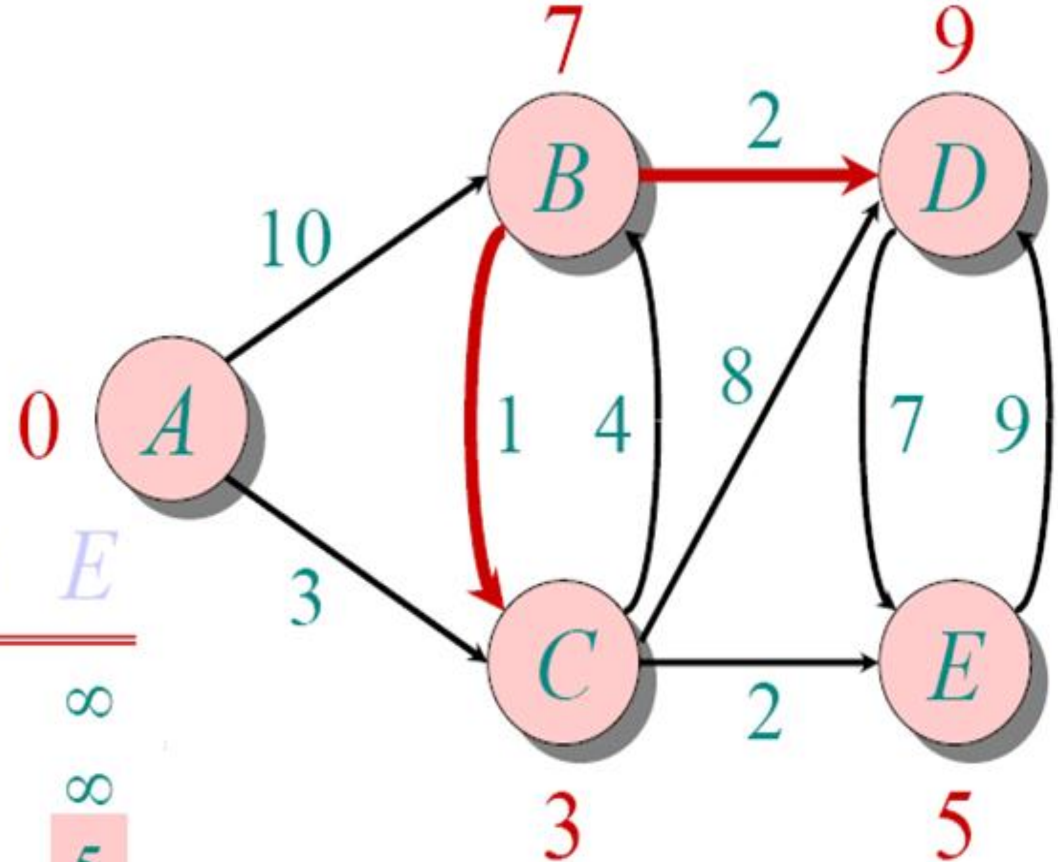
Dijkstra's Algorithm Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: { A, C, E, B }

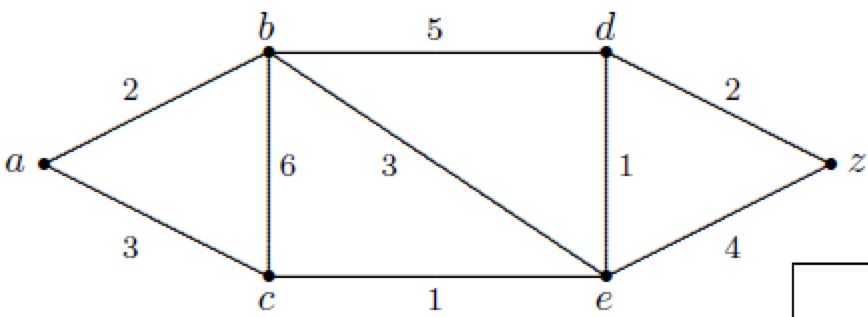
Dijkstra's Algorithm Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: { A, C, E, B, D }

Dijkstra's Algorithm Example 1



S	$L(a)$	$L(b)$	$L(c)$	$L(d)$	$L(e)$	$L(z)$
$\emptyset$	$\boxed{0}$	∞	∞	∞	∞	∞
$\{a\}$		$\boxed{2}$	3	∞	∞	∞
$\{a, b\}$			$\boxed{3}$	7	5	∞
$\{a, b, c\}$				7	$\boxed{4}$	∞
$\{a, b, c, e\}$				$\boxed{5}$		8
$\{a, b, c, e, d\}$						$\boxed{7}$
$\{a, b, c, e, d, z\}$						

At the last iteration, z 2 S and L(z) = 7.
We conclude that the cheapest path from a to z has a **cost of 7**.