MODULE 2-5 [CPE11202]
Graph

Assc. Prof. Dr. Natasha Dejdumrong

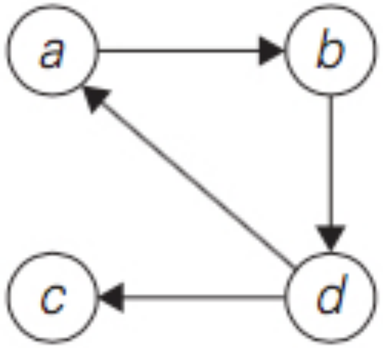
KING MONGKUT'S UNIVERSITY
OF TECHNOLOGY THONBURI

Outlines

- Transitive Closure of Directed Graph
  - Warshall's Algorithm
- All Pairs Shortest-Path Problem
  - Floyd's Algorithm
- Minimum Spanning Tree
  - Prim's Algorithm
  - Kruskal's Algorithm
- Single Source Shortest-Path Problem
  - Dijkstra's Algorithm

Transitive Closure of a Directed Graph

- A matrix that tells (in a constant time) if j^{th} vertex is reachable from i^{th} vertex (there exists a directed path from i to j)
  - Represent by a boolean matrix $T = \{t_{ij}\}$



$$A = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array} \qquad T = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

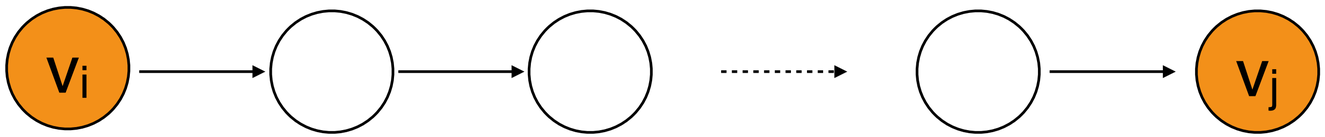
Warshall's Algorithm

- Label vertices from 1 to n
- Define matrix $R^{(k)}$ whose elements are

$$r_{ij}^{(k)} = \begin{cases} 1 & \text{there exists a directed path from } i \text{ to } j \text{ with} \\ & \text{intermediate vertices (if any) not higher than } k \\ 0 & \text{otherwise} \end{cases}$$

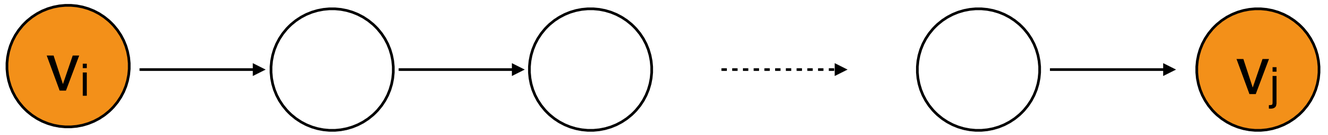
- Transitive closure is constructed through a series of n x n boolean matrices $R^{(0)}$, $R^{(1)}$, …, $R^{(k-1)}$, $R^{(k)}$, …, $R^{(n)}$.

- Examples
  - $R^{(0)}$ is just the adjacency matrix.
  - $R^{(1)}$: Paths only contains the first vertex as intermediate
  - $R^{(2)}$: Paths only contains the first two vertices as intermediate
  - $R^{(n)}$: Paths contains any vertices as intermediate
- $R^{(k)}$ with entry $r_{ij}^{(k)} = 1$ means that there exists a path between i^{th} vertex v_i and j^{th} vertex v_j such that



intermediate vertices ≤ k

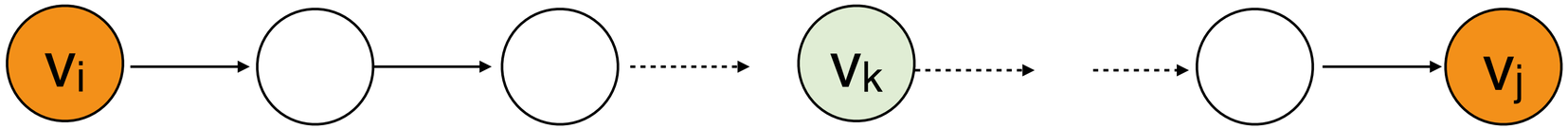
- Two situations regarding the path are possible.
  - list of intermediate vertices does not have k^{th} vertex.



intermediate vertices ≤ k-1

$$r_{ij}^{(k-1)} = 1$$

  - list of intermediate vertices have k^{th} vertex.



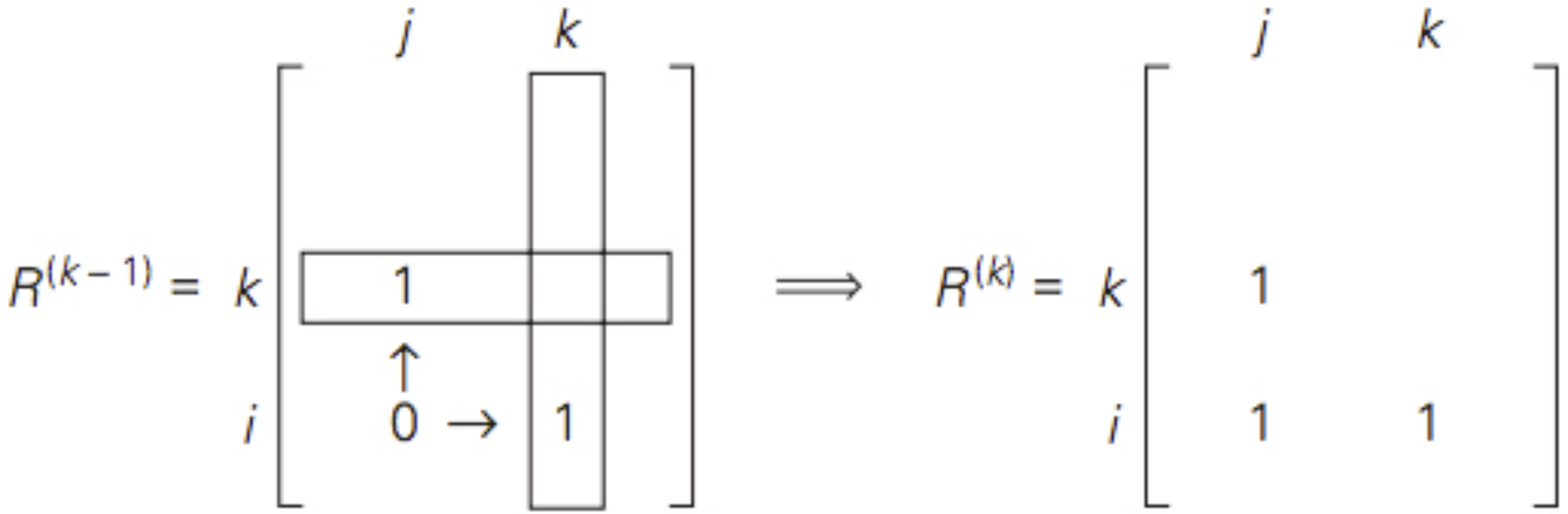
intermediate vertices ≤ k-1

intermediate vertices ≤ k-1

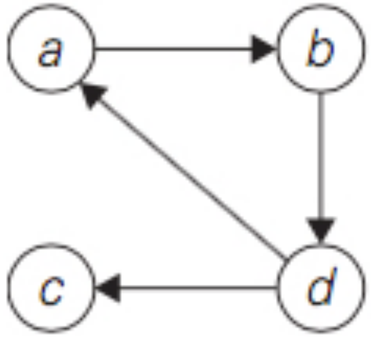
$$r_{ik}^{(k-1)} = r_{kj}^{(k-1)} = 1$$

- Therefore, if $r_{ij}^{(k)} = 1$, then either

$$r_{ij}^{(k)} = r_{ij}^{(k-1)} \quad \text{or} \quad \left(r_{ik}^{(k-1)} \text{ and } r_{kj}^{(k-1)}\right)$$



- Rules for applying Warshall's algorithm by hand:
  - If r_{ij} is 1 in $R^{(k-1)}$, it remains 1 in $R^{(k)}$
  - If r_{ij} is 0 in $R^{(k-1)}$, it has to be changed to 1 in $R^{(k)}$ if and only if r_{ik} and r_{kj} in $R^{(k-1)}$ are both 1.



$R^{(0)} =$

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	0	1	0

$R^{(1)} =$

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	**1**	1	0

$R^{(2)} =$

	a	b	c	d
a	0	1	0	**1**
b	0	0	0	1
c	0	0	0	0
d	1	1	1	**1**

Work out $R^{(3)}$ and $R^{(4)}$

Warshall's Algorithm

ALGORITHM *Warshall*$(A[1..n, 1..n])$

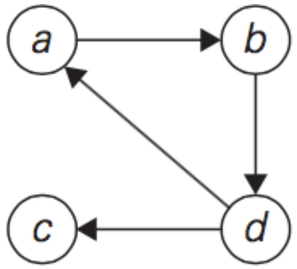
//Implements Warshall's algorithm for computing the transitive closure
//Input: The adjacency matrix A of a digraph with n vertices
//Output: The transitive closure of the digraph
$R^{(0)} \leftarrow A$
for $k \leftarrow 1$ **to** n **do**
    for $i \leftarrow 1$ **to** n **do**
        for $j \leftarrow 1$ **to** n **do**
            $R^{(k)}[i, j] \leftarrow R^{(k-1)}[i, j]$ **or** $(R^{(k-1)}[i, k]$ **and** $R^{(k-1)}[k, j])$
return $R^{(n)}$

Warshall's Algorithm

1. Apply Warshall's algorithm to find the transitive closure of the digraph defined by the following adjacency matrix:

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Warshall's Algorithm



$R^{(0)} =$

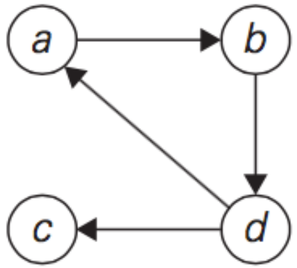
	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	0	1	0

1's reflect the existence of paths with no intermediate vertices ($R^{(0)}$ is just the adjacency matrix); boxed row and column are used for getting $R^{(1)}$.

$R^{(1)} =$

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	**1**	1	0

1's reflect the existence of paths with intermediate vertices numbered not higher than 1, i.e., just vertex *a* (note a new path from *d* to *b*); boxed row and column are used for getting $R^{(2)}$.



$R^{(0)} =$

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	0	1	0

1's reflect the existence of paths with no intermediate vertices ($R^{(0)}$ is just the adjacency matrix); boxed row and column are used for getting $R^{(1)}$.

$R^{(1)} =$

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	**1**	1	0

1's reflect the existence of paths with intermediate vertices numbered not higher than 1, i.e., just vertex *a* (note a new path from *d* to *b*); boxed row and column are used for getting $R^{(2)}$.

$R^{(2)} =$

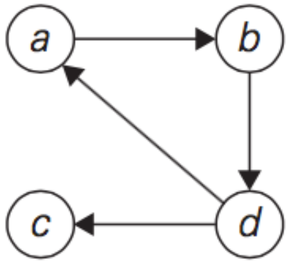
	a	b	c	d
a	0	1	0	**1**
b	0	0	0	1
c	0	0	0	0
d	1	1	1	**1**

1's reflect the existence of paths with intermediate vertices numbered not higher than 2, i.e., *a* and *b* (note two new paths); boxed row and column are used for getting $R^{(3)}$.

$R^{(3)} =$

	a	b	c	d
a	0	1	0	1
b	0	0	0	1
c	0	0	0	0
d	1	1	1	1

1's reflect the existence of paths with intermediate vertices numbered not higher than 3, i.e., *a, b,* and *c* (no new paths); boxed row and column are used for getting $R^{(4)}$.



$R^{(0)} =$

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	0	1	0

1's reflect the existence of paths with no intermediate vertices ($R^{(0)}$ is just the adjacency matrix); boxed row and column are used for getting $R^{(1)}$.

$R^{(1)} =$

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	**1**	1	0

1's reflect the existence of paths with intermediate vertices numbered not higher than 1, i.e., just vertex *a* (note a new path from *d* to *b*); boxed row and column are used for getting $R^{(2)}$.

$R^{(2)} =$

	a	b	c	d
a	0	1	0	**1**
b	0	0	0	1
c	0	0	0	0
d	1	1	1	**1**

1's reflect the existence of paths with intermediate vertices numbered not higher than 2, i.e., *a* and *b* (note two new paths); boxed row and column are used for getting $R^{(3)}$.

$R^{(3)} =$

	a	b	c	d
a	0	1	0	1
b	0	0	0	1
c	0	0	0	0
d	1	1	1	1

1's reflect the existence of paths with intermediate vertices numbered not higher than 3, i.e., *a, b,* and *c* (no new paths); boxed row and column are used for getting $R^{(4)}$.

$R^{(4)} =$

	a	b	c	d
a	**1**	1	**1**	1
b	**1**	**1**	**1**	1
c	0	0	0	0
d	1	1	1	1

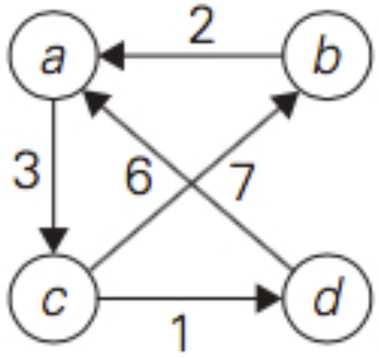
1's reflect the existence of paths with intermediate vertices numbered not higher than 4, i.e., *a, b, c,* and *d* (note five new paths).

Shortest Path Algorithm

- Minimum Spanning Tree
  - Prim's Algorithm
  - Kruskal's Algorithm
- Single Source Shortest Path Algorithm
  - Dijkstra's Algorithm
- All Pairs Shortest Path Algorithm
  - Floyd's Algorithm

All-Pairs Shortest-Paths Problem

- Consider a positive weighted connected graph W.
- Find the shortest paths from each vertex to all the others.
- Distance matrix D contains the shortest path length from i^{th} vertex to j^{th} vertex.



$$W = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

$$D = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ 7 & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{bmatrix} \end{array}$$

Floyd's Algorithm

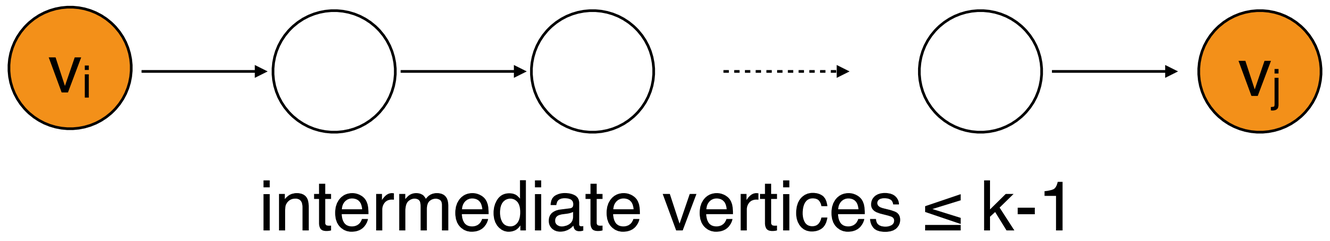
- Compute the distance matrix of a weighted graph with n vertices through a series of n x n matrices:

$$D^{(0)}, D^{(1)}, \ldots, D^{(k-1)}, D^{(k)}, \ldots, D^{(n)}$$

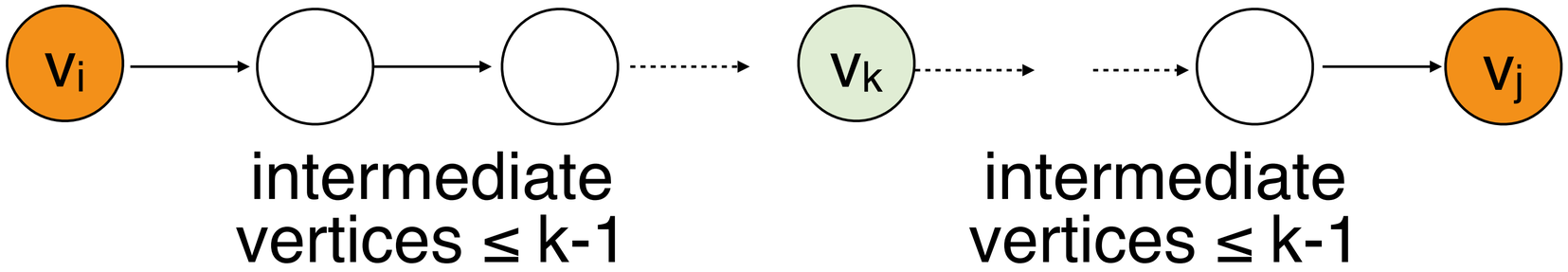
- d_{ij} in $D^{(k)}$ is the length of shortest path from i to j with intermediate vertices (if any) not higher than k.
  - $D^{(0)}$ is simply the weight matrix.
  - $D^{(k)}$: Length of shortest paths with only first k vertices as intermediates

Floyd's Algorithm

- Two situations regarding the path for $d_{ij}^{(k)}$ are possible.
  - I: List of intermediate vertices does not have k^{th} vertex.



  - II: List of intermediate vertices have k^{th} vertex.

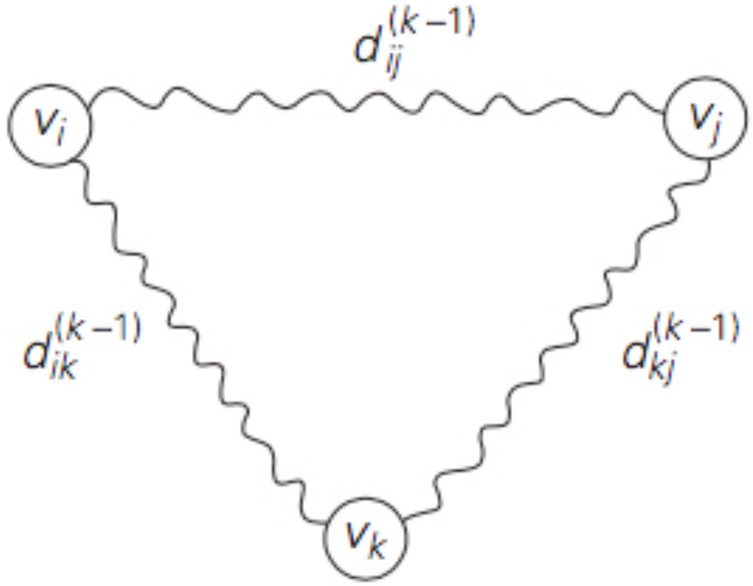


Floyd's Algorithm

ALGORITHM *Floyd*$(W[1..n, 1..n])$

//Implements Floyd's algorithm for the all-pairs shortest-paths problem
//Input: The weight matrix W of a graph with no negative-length cycle
//Output: The distance matrix of the shortest paths' lengths
$D \leftarrow W$ //is not necessary if W can be overwritten
for $k \leftarrow 1$ **to** n **do**
    for $i \leftarrow 1$ **to** n **do**
        for $j \leftarrow 1$ **to** n **do**
            $D[i, j] \leftarrow \min\{D[i, j], D[i, k] + D[k, j]\}$
return D

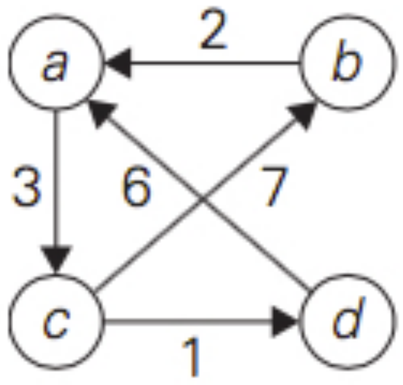
- For subset II, $d_{ij}^{(k)}$ will be different from $d_{ij}^{(k-1)}$



- Both subsets lead to the following recurrence:

$$d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}, \quad k \geq 1$$

$$d_{ij}^{(0)} = w_{ij}$$



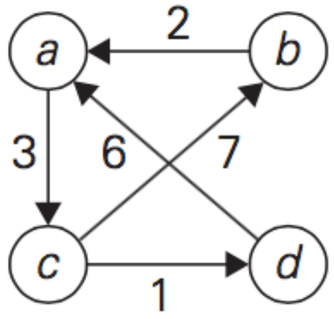
$$D^{(0)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

$$D^{(1)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \mathbf{5} & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \mathbf{9} & 0 \end{bmatrix} \end{array}$$

$$D^{(2)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ \mathbf{9} & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix} \end{array}$$

Work out $D^{(3)}$ and $D^{(4)}$

Floyd's Algorithm

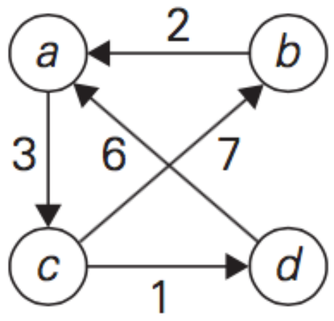


$D^{(0)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	∞	∞
c	∞	7	0	1
d	6	∞	∞	0

Lengths of the shortest paths with no intermediate vertices ($D^{(0)}$ is simply the weight matrix).

Floyd's Algorithm



$D^{(0)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	∞	∞
c	∞	7	0	1
d	6	∞	∞	0

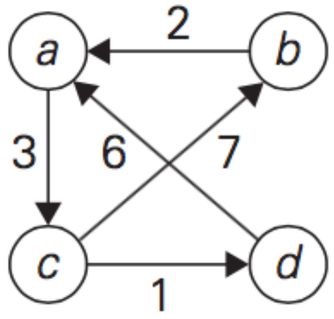
Lengths of the shortest paths with no intermediate vertices ($D^{(0)}$ is simply the weight matrix).

$D^{(1)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	**5**	∞
c	∞	7	0	1
d	6	∞	**9**	0

Lengths of the shortest paths with intermediate vertices numbered not higher than 1, i.e., just *a* (note two new shortest paths from *b* to *c* and from *d* to *c*).

Floyd's Algorithm



$D^{(0)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	∞	∞
c	∞	7	0	1
d	6	∞	∞	0

Lengths of the shortest paths with no intermediate vertices ($D^{(0)}$ is simply the weight matrix).

$D^{(1)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	**5**	∞
c	∞	7	0	1
d	6	∞	**9**	0

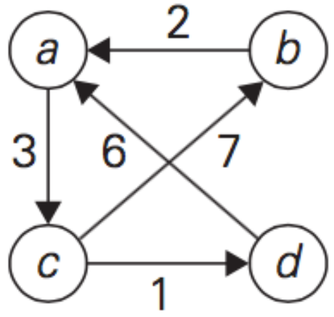
Lengths of the shortest paths with intermediate vertices numbered not higher than 1, i.e., just *a* (note two new shortest paths from *b* to *c* and from *d* to *c*).

$D^{(2)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	5	∞
c	**9**	7	0	1
d	6	∞	9	0

Lengths of the shortest paths with intermediate vertices numbered not higher than 2, i.e., a and *b* (note a new shortest path from *c* to *a*).

Floyd's Algorithm

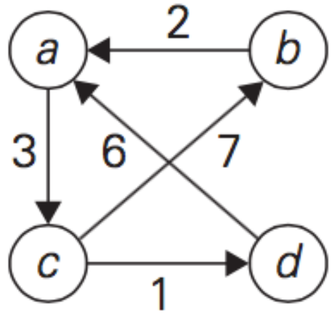


$D^{(2)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	5	∞
c	**9**	7	0	1
d	6	∞	9	0

Lengths of the shortest paths with intermediate vertices numbered not higher than 2, i.e., a and *b* (note a new shortest path from *c* to *a*).

Floyd's Algorithm



$D^{(2)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	5	∞
c	**9**	7	0	1
d	6	∞	9	0

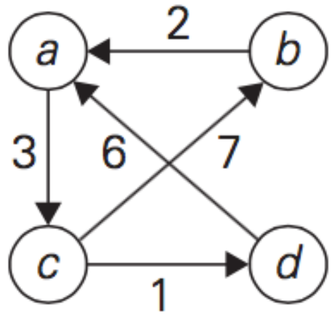
Lengths of the shortest paths with intermediate vertices numbered not higher than 2, i.e., a and *b* (note a new shortest path from *c* to *a*).

$D^{(3)} =$

	a	b	c	d
a	0	**10**	3	**4**
b	2	0	5	**6**
c	9	7	0	1
d	6	**16**	9	0

Lengths of the shortest paths with intermediate vertices numbered not higher than 3, i.e., a, *b*, and *c* (note four new shortest paths from *a* to *b*, from a to *d*, from *b* to *d*, and from *d* to *b*).

Floyd's Algorithm



$D^{(2)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	5	∞
c	**9**	7	0	1
d	6	∞	9	0

Lengths of the shortest paths with intermediate vertices numbered not higher than 2, i.e., a and *b* (note a new shortest path from *c* to *a*).

$D^{(3)} =$

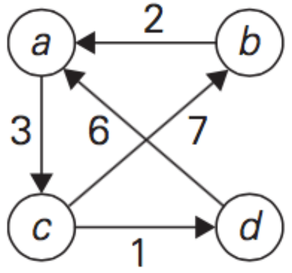
	a	b	c	d
a	0	**10**	3	**4**
b	2	0	5	**6**
c	9	7	0	1
d	6	**16**	9	0

Lengths of the shortest paths with intermediate vertices numbered not higher than 3, i.e., a, *b*, and *c* (note four new shortest paths from *a* to *b*, from a to *d*, from *b* to *d*, and from *d* to *b*).

$D^{(4)} =$

	a	b	c	d
a	0	10	3	4
b	2	0	5	6
c	**7**	7	0	1
d	6	16	9	0

Lengths of the shortest paths with intermediate vertices numbered not higher than 4, i.e., a, *b*, *c*, and *d* (note a new shortest path from *c* to *a*).



$D^{(0)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	∞	∞
c	∞	7	0	1
d	6	∞	∞	0

Lengths of the shortest paths with no intermediate vertices ($D^{(0)}$ is simply the weight matrix).

$D^{(1)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	**5**	∞
c	∞	7	0	1
d	6	∞	**9**	0

Lengths of the shortest paths with intermediate vertices numbered not higher than 1, i.e., just *a* (note two new shortest paths from *b* to *c* and from *d* to *c*).

$D^{(2)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	5	∞
c	**9**	7	0	1
d	6	∞	9	0

Lengths of the shortest paths with intermediate vertices numbered not higher than 2, i.e., a and *b* (note a new shortest path from *c* to *a*).

$D^{(3)} =$

	a	b	c	d
a	0	**10**	3	**4**
b	2	0	5	**6**
c	9	7	0	1
d	6	**16**	9	0

Lengths of the shortest paths with intermediate vertices numbered not higher than 3, i.e., a, *b,* and *c* (note four new shortest paths from *a* to *b,* from a to *d,* from *b* to *d,* and from *d* to *b*).

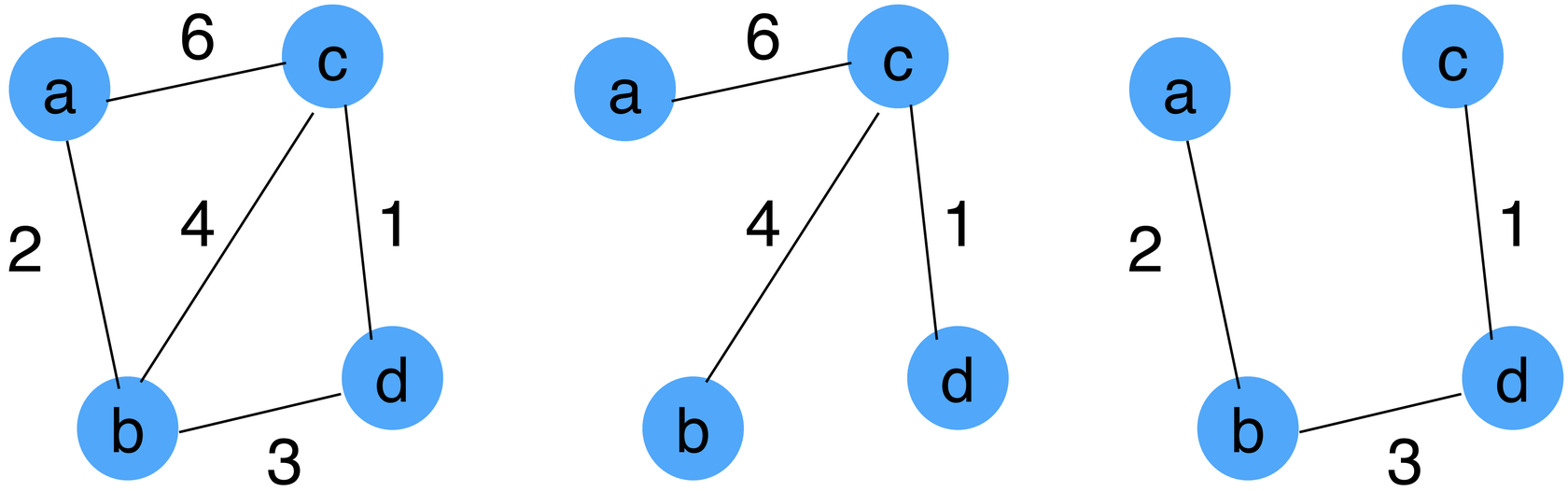
$D^{(4)} =$

	a	b	c	d
a	0	10	3	4
b	2	0	5	6
c	**7**	7	0	1
d	6	16	9	0

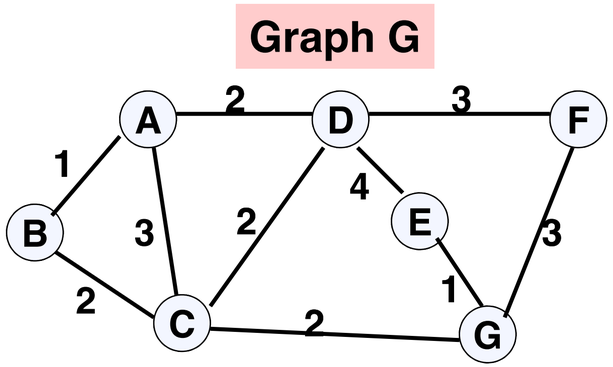
Lengths of the shortest paths with intermediate vertices numbered not higher than 4, i.e., a, *b, c,* and *d* (note a new shortest path from *c* to *a*).

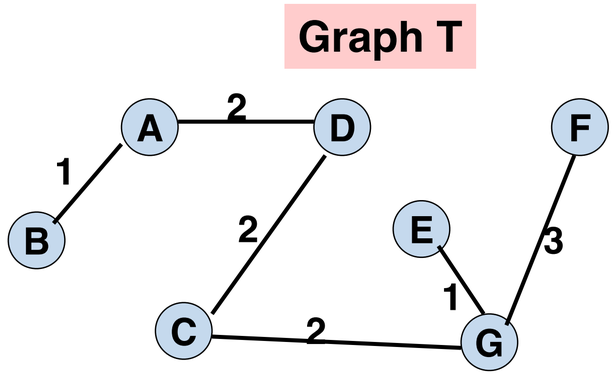
Minimum Spanning Tree (MST)

- Spanning tree of a connected graph G: a connected acyclic subgraph of G that includes all of G's vertices
- MST of a weighted, connected graph G: a spanning tree of G of the minimum total weight



Prim's MST Algorithm

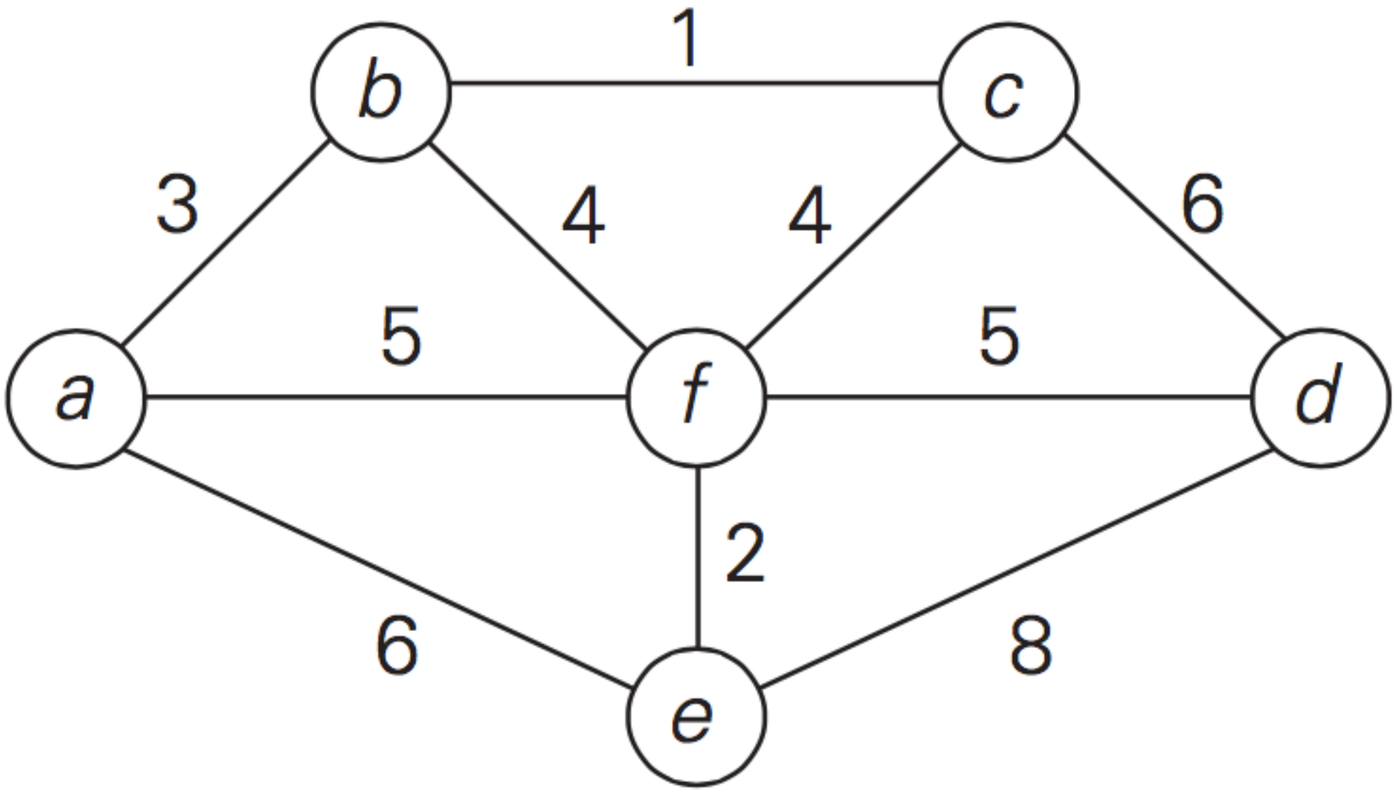




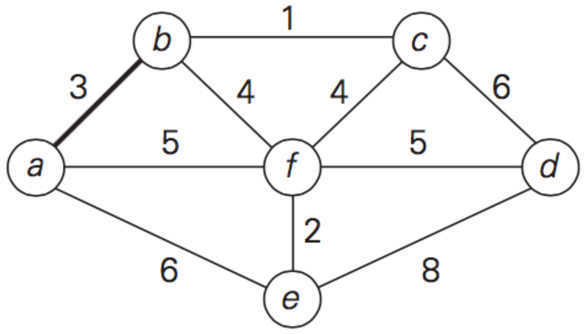
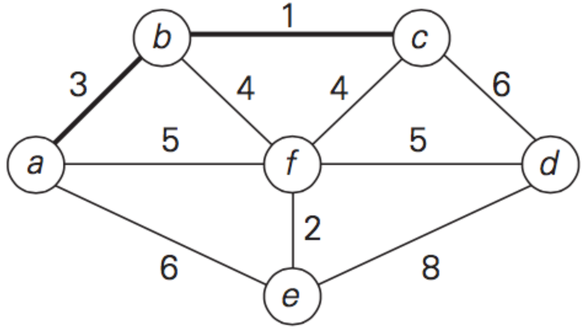
1. Add arbitrary vertex (node) from G in T.
2. Grow T by adding a vertex from G closest to those already in T with the corresponding edge. *(Greedy step)*
3. Repeat 2) until all the vertices in G are added in T.

The algorithm does not necessarily give a unique MST.

Prim's MST Algorithm

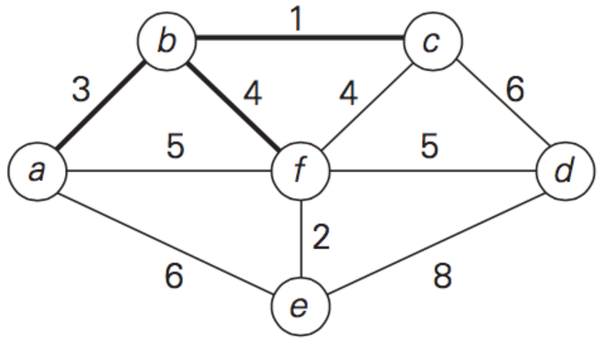


Prim's MST Algorithm

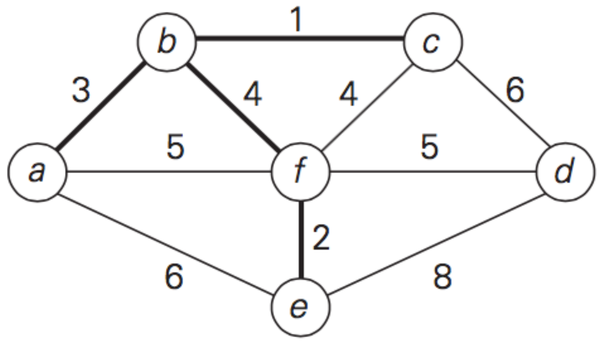
Tree vertices	Remaining vertices	Illustration
a(−, −)	**b(a, 3)** c(−, ∞) d(−, ∞) e(a, 6) f(a, 5)	
b(a, 3)	**c(b, 1)** d(−, ∞) e(a, 6) f(b, 4)	

Prim's MST Algorithm

c(b, 1) d(c, 6) e(a, 6) **f(b, 4)**

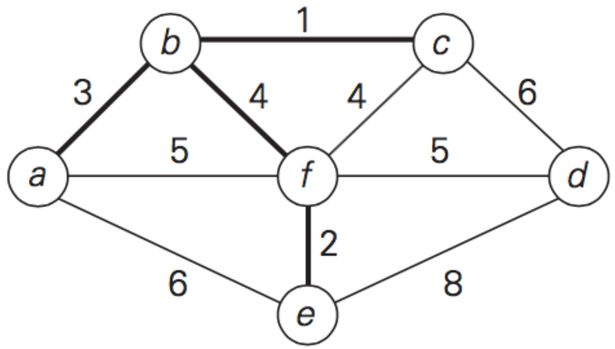


f(b, 4) d(f, 5) **e(f, 2)**

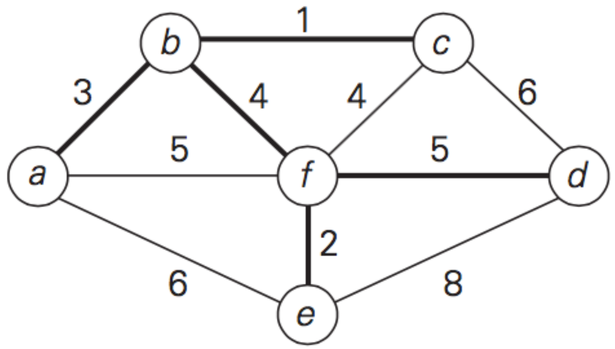


Prim's MST Algorithm

f(b, 4) d(f, 5) **e(f, 2)**



e(f, 2) **d(f, 5)**



d(f, 5)

Prim Algorithm

- If a graph is represented by its adjacency lists and the priority queue is implemented as a min-heap, the running time of the algorithm is in $O(|E| \log |V|)$.
- This is because the algorithm performs $|V| - 1$ deletions of the smallest element and makes $|E|$ verifications and, possibly, changes of an element's priority in a min-heap of size not exceeding $|V|$.
- Each of these operations, as noted earlier, is a $O(\log |V|)$ operation.

Prim Algorithm

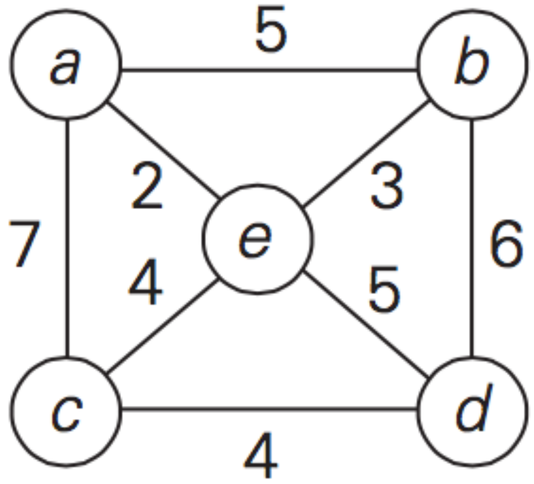
- Each of these operations, as noted earlier, is a O(log |V|) operation. Hence, the running time of this implementation of Prim's algorithm is in

$$(|V| - 1 + |E|)O(\log |V|) = O(|E| \log |V|)$$

- because, in a connected graph, |V| − 1 ≤ |E|.

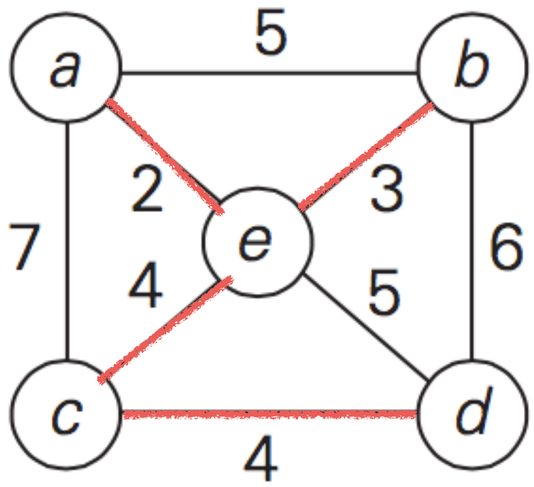
In-Class Exercise

- Apply Prim's algorithm to find a minimum spanning tree of the following graph:



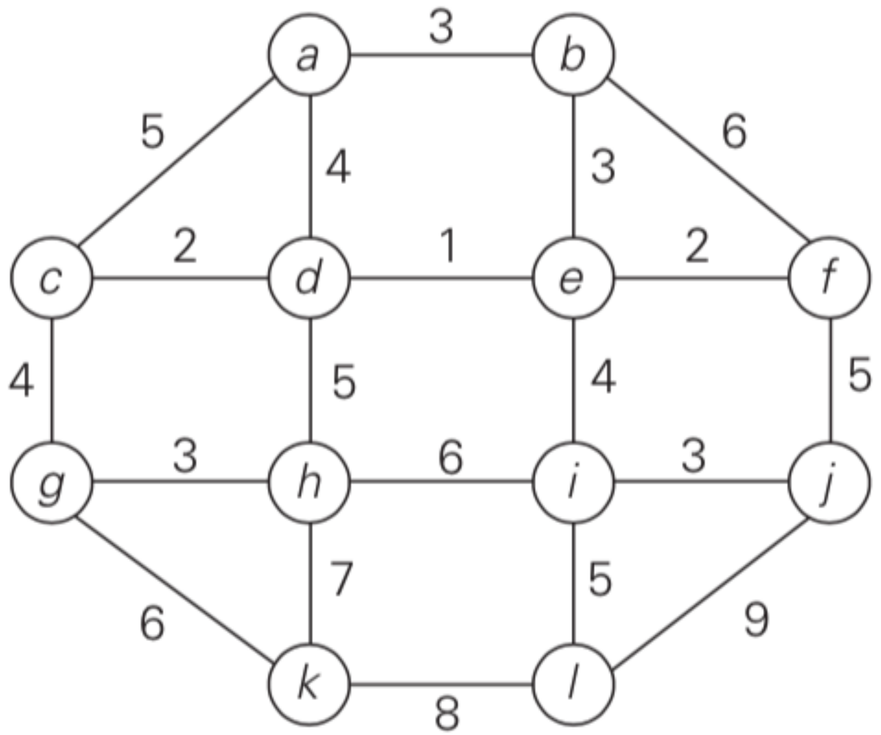
In-Class Exercise

- Apply Prim's algorithm to find a minimum spanning tree of the following graph:



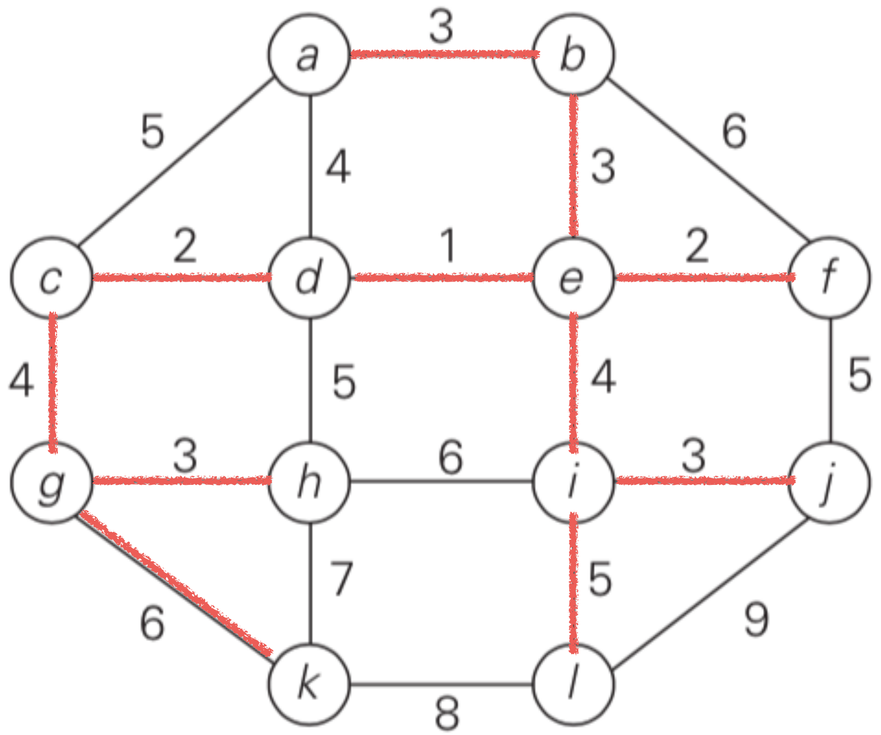
In-Class Exercise

- Apply Prim's algorithm to find a minimum spanning tree of the following graph:

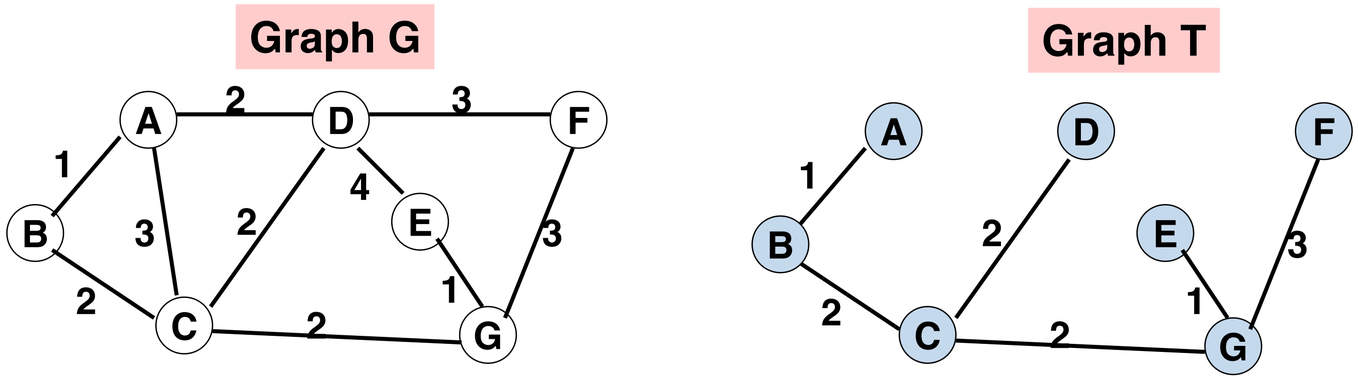


In-Class Exercise

- Apply Prim's algorithm to find a minimum spanning tree of the following graph:



Kruskal's MST Algorithm



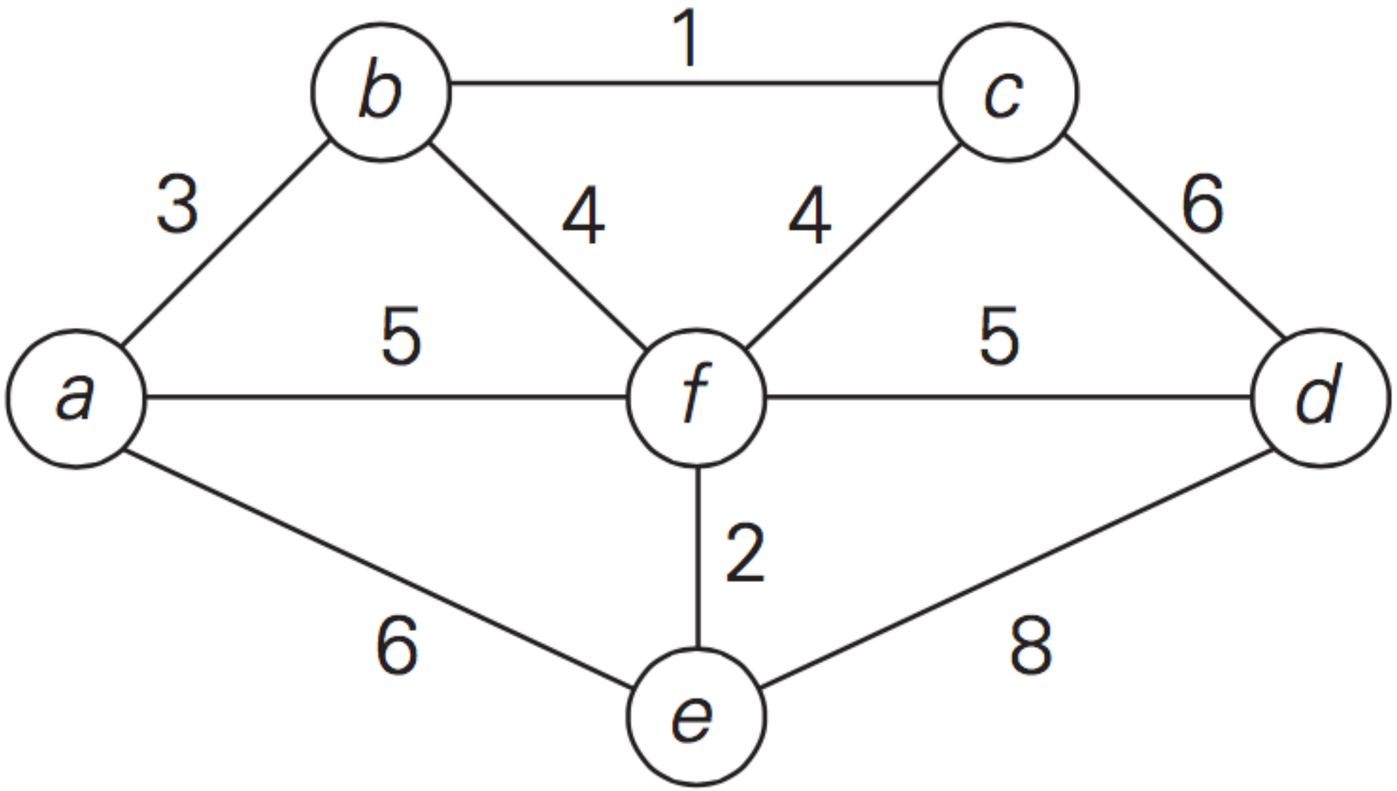
1. Sort the edges in the ascending order. T = ϕ
2. Select an edge with smallest weight and add to T.
3. Select the edge (u, v) with smallest weight such that T is acyclic if added.
4. Add the edge in Step 3 to T. *(Greedy step)*
5. Repeat step 3 until all the nodes are included in T.

Kruskal's MST Algorithm

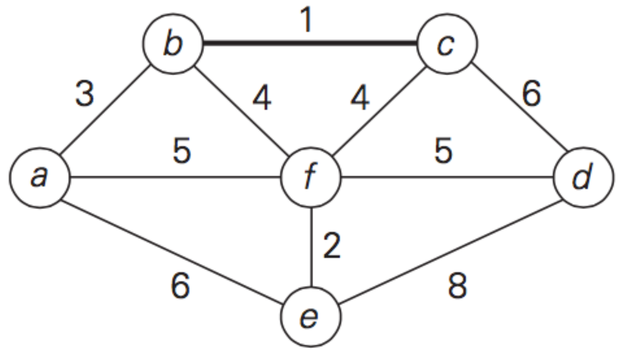
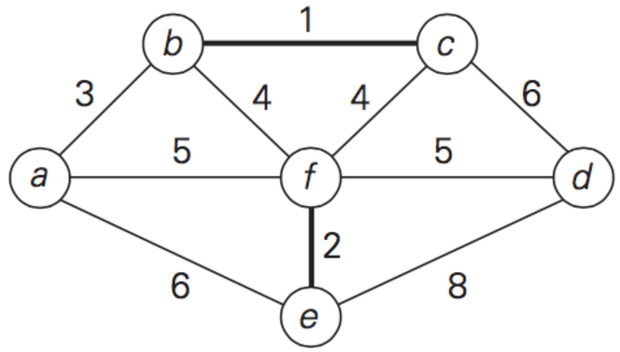
ALGORITHM *Kruskal*(G)

//Kruskal's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph $G = \langle V, E \rangle$
//Output: E_T, the set of edges composing a minimum spanning tree of G
sort E in nondecreasing order of the edge weights $w(e_{i_1}) \leq \cdots \leq w(e_{i_{|E|}})$
$E_T \leftarrow \varnothing$; $ecounter \leftarrow 0$ //initialize the set of tree edges and its size
$k \leftarrow 0$ //initialize the number of processed edges
while $ecounter < |V| - 1$ **do**
    $k \leftarrow k + 1$
    if $E_T \cup \{e_{i_k}\}$ is acyclic
        $E_T \leftarrow E_T \cup \{e_{i_k}\}$; $ecounter \leftarrow ecounter + 1$
return E_T

Kruskal's MST Algorithm

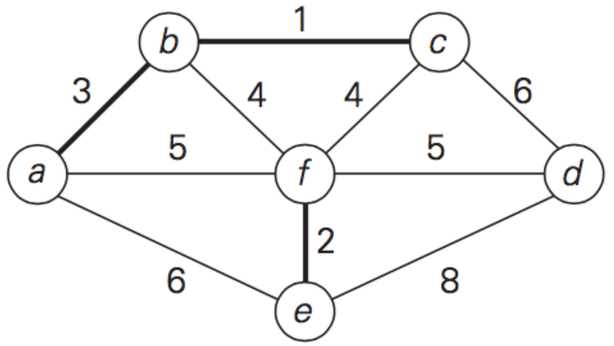


Kruskal's MST Algorithm

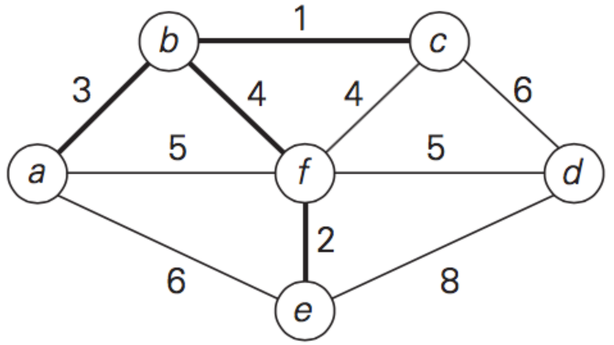
Tree edges	Sorted list of edges	Illustration
	bc ef ab bf cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	
bc 1	bc **ef** ab bf cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	

Kruskal's MST Algorithm

ef		bc	ef	**ab**	bf	cf	af	df	ae	cd	de
2		1	2	3	4	4	5	5	6	6	8

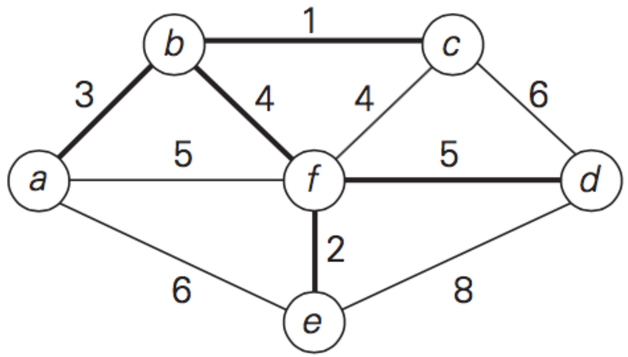


ab		bc	ef	ab	**bf**	cf	af	df	ae	cd	de
3		1	2	3	4	4	5	5	6	6	8



Kruskal's MST Algorithm

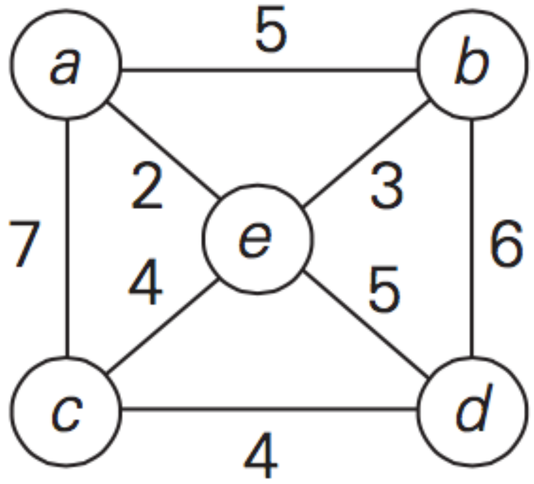
bf		bc	ef	ab	bf	cf	af	**df**	ae	cd	de
4		1	2	3	4	4	5	5	6	6	8



df
5

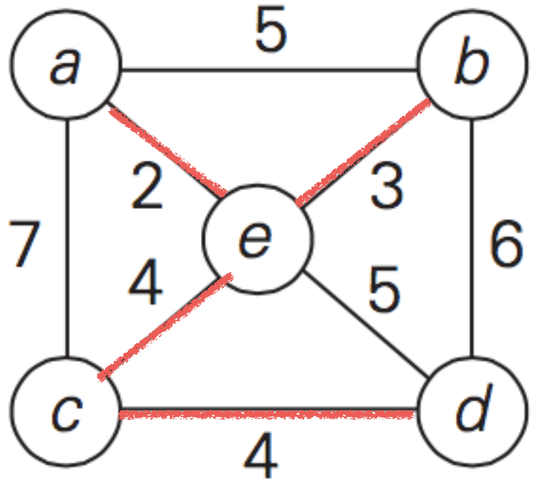
In-Class Exercise

- Apply Kruskal's algorithm to find a minimum spanning tree of the following graph:



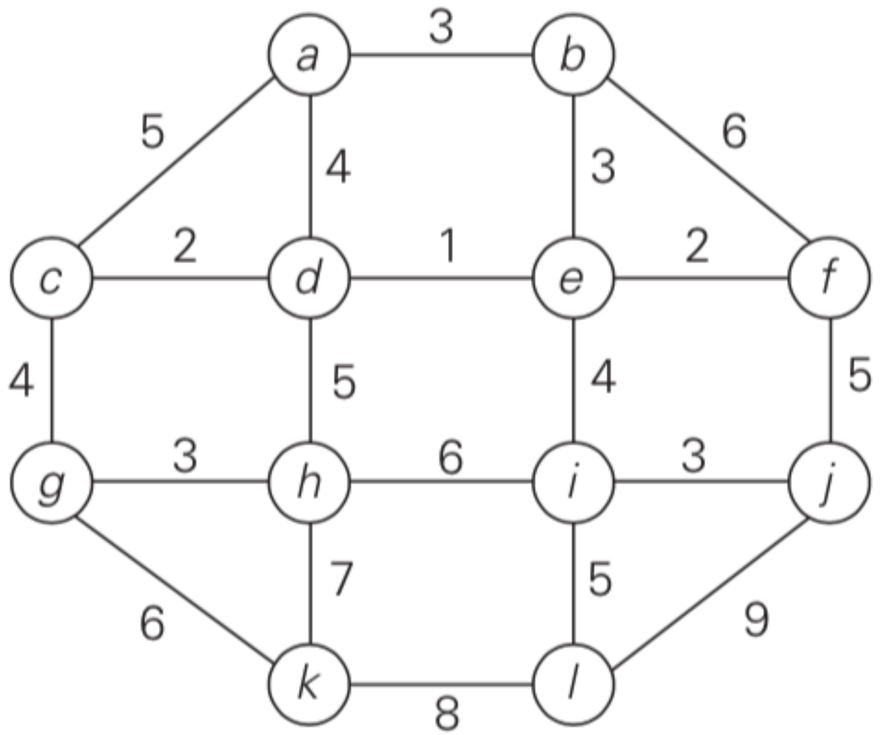
In-Class Exercise

- Apply Kruskal's algorithm to find a minimum spanning tree of the following graph:



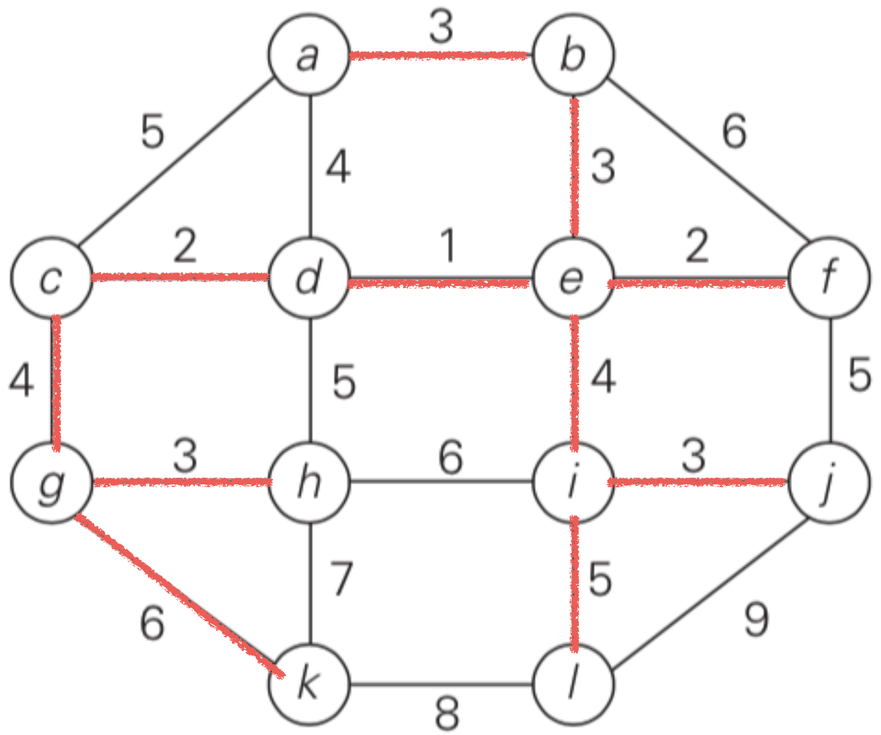
In-Class Exercise

- Apply Kruskal's algorithm to find a minimum spanning tree of the following graph:



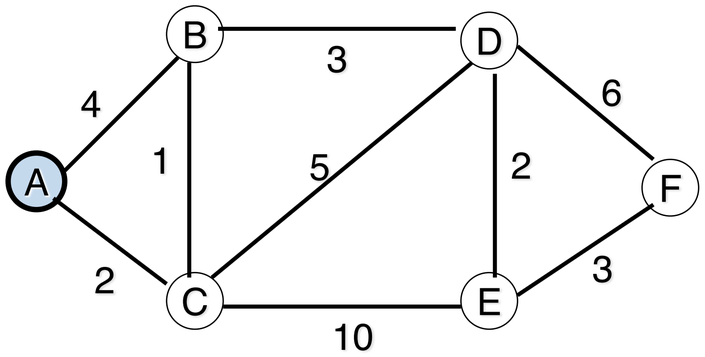
In-Class Exercise

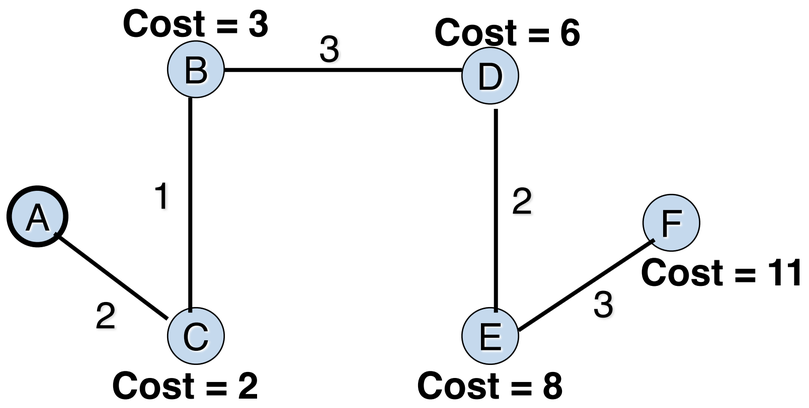
- Apply Kruskal's algorithm to find a minimum spanning tree of the following graph:



Single-Source Shortest Paths Problem

- Given a weighted connected (directed) graph G, find shortest paths from source vertex ***s*** to each of the other vertices.





Dijkstra's Algorithm

- Similar to Prim's MST algorithm, with a different way of computing numerical labels: Among vertices not already in the tree, it finds vertex v with the smallest sum

$$d_v + w(v, u)$$

  - u: Vertex for which shortest path has been already found on preceding iterations (such vertices form a tree rooted at s)
  - d_v : Length of the shortest path from source s to v
  - w(v,u): Length (weight) of edge (v, u)

Dijkstra's Algorithm

ALGORITHM *Dijkstra*(G, s)

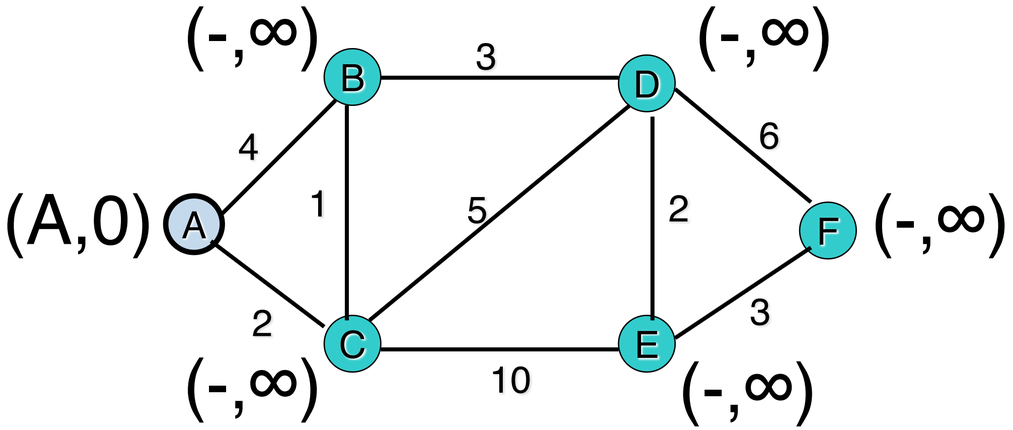
```
//Dijkstra's algorithm for single-source shortest paths
//Input: A weighted connected graph G = ⟨V, E⟩ with nonnegative weights
//        and its vertex s
//Output: The length d_v of a shortest path from s to v
//         and its penultimate vertex p_v for every vertex v in V
Initialize(Q)  //initialize priority queue to empty
for every vertex v in V
    d_v ← ∞;   p_v ← null
    Insert(Q, v, d_v)  //initialize vertex priority in the priority queue
d_s ← 0;  Decrease(Q, s, d_s)  //update priority of s with d_s
V_T ← ∅
for i ← 0 to |V| − 1 do
    u* ← DeleteMin(Q)  //delete the minimum priority element
    V_T ← V_T ∪ {u*}
    for every vertex u in V − V_T that is adjacent to u* do
        if d_u* + w(u*, u) < d_u
            d_u ← d_u* + w(u*, u);   p_u ← u*
            Decrease(Q, u, d_u)
```

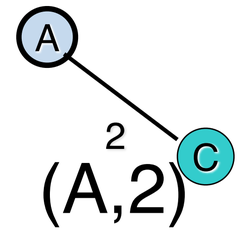
Notation

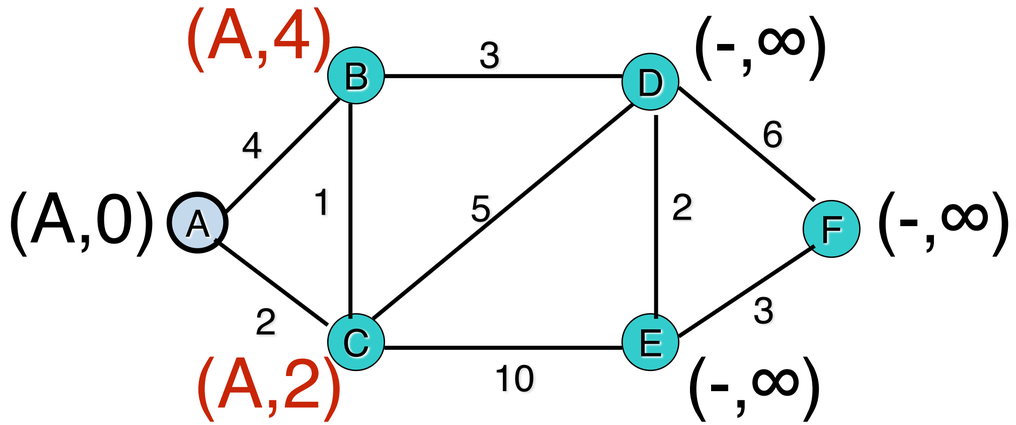
- Let s be the source node (root)
- Let d_u be the current minimum distance from s to u (Node u's label).
- Let p_u be the previous node before u on the least cost path from s to u.
- Let c(u, v) is a cost (weight) of link (u, v).

1. Initialize labels $d_u = \infty, \forall u \in V, u \neq s, \ d_s = 0, p_s = s$
2. From a set of nodes not in the tree, find node **u** whose d_u is smallest and include it to the tree.
3. For all nodes v not in the tree and connected to **u**, update their labels with
   - $d_v = \min [d_u + c(u, v), \ d_v]$ *(Greedy step)*

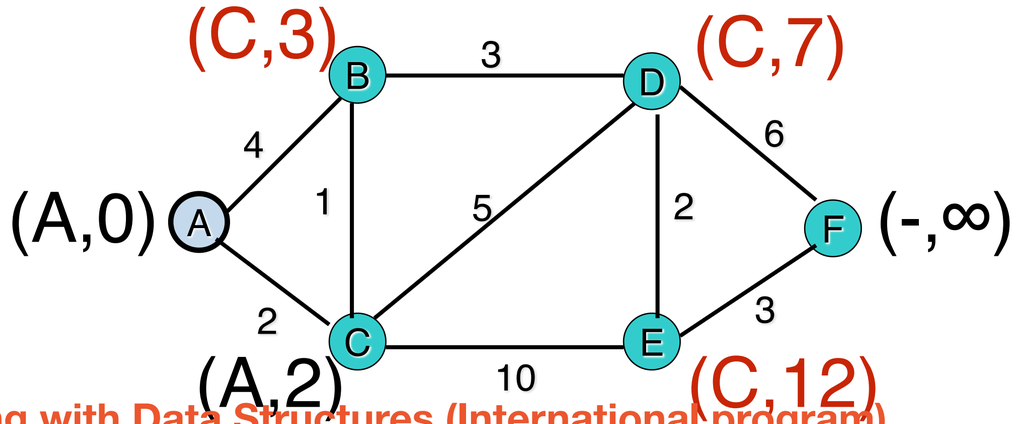
     Distance from root to u | Distance from u to v | Current distance from root to v.
   - Update p_v to u if necessary.
4. Repeat step 2 until all nodes are included in the tree.

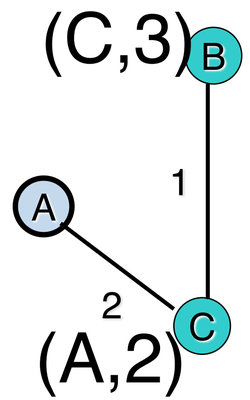

(-,∞)
(-,∞)
B
3
D
4
6
(A,0)
A
1
5
2
F
(-,∞)
2
3
C
E
(-,∞)
10
(-,∞)

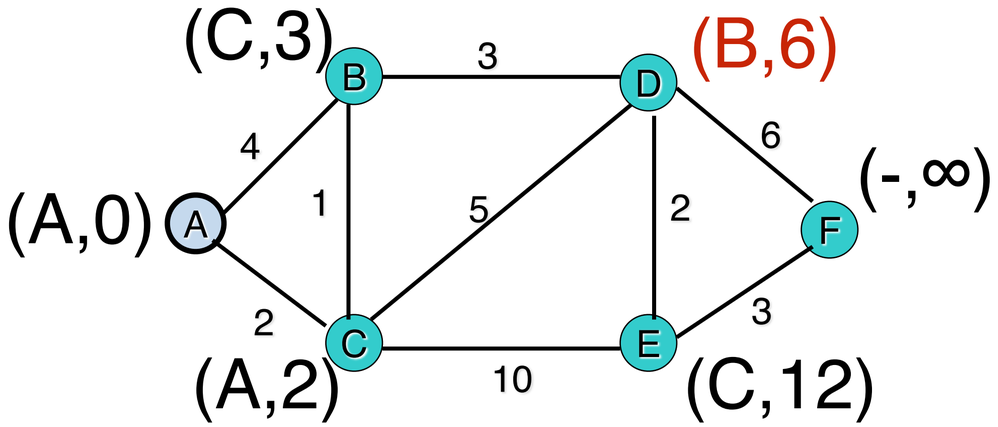

(A,0)
A

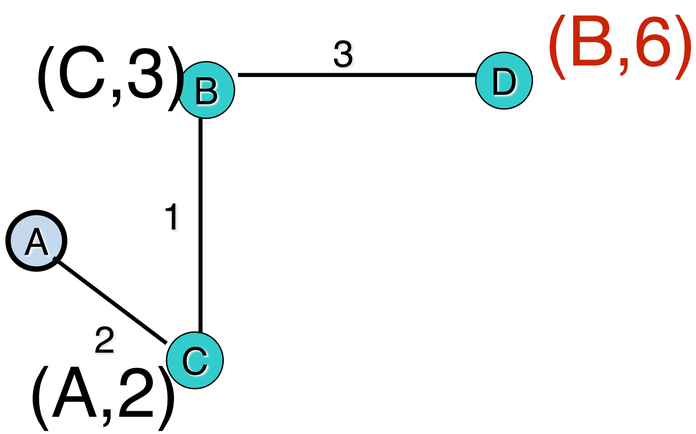

(A,4)
(-,∞)
B
3
D
4
6
(A,0)
A
1
5
2
F
(-,∞)
2
3
C
E
(A,2)
10
(-,∞)

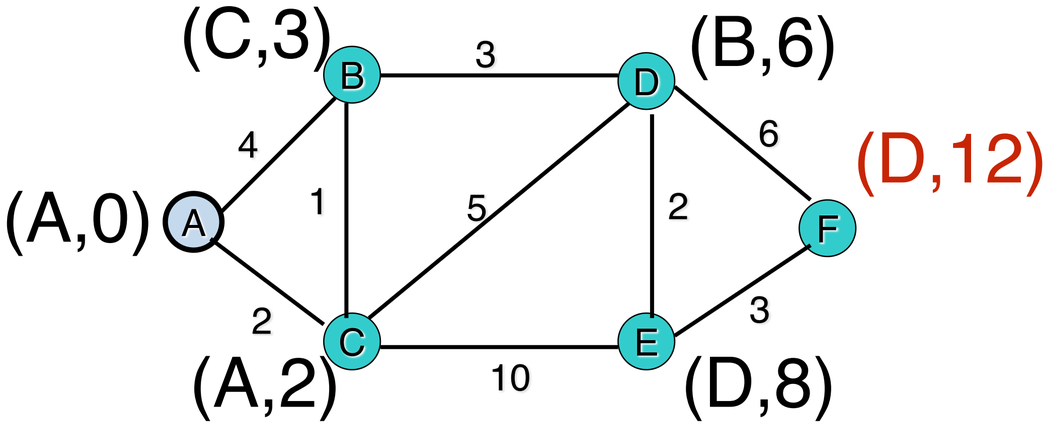
A
2
C
(A,2)

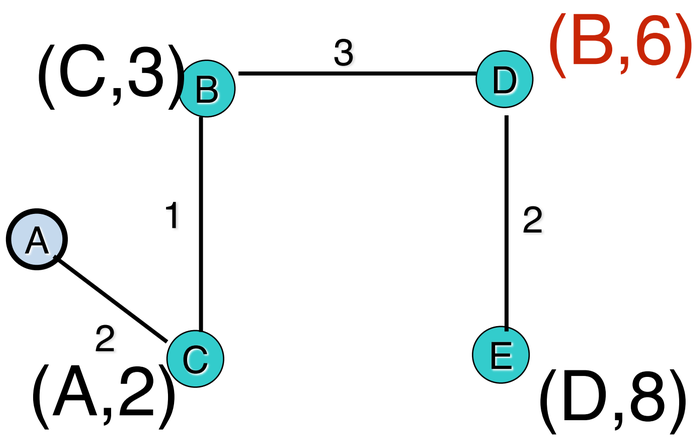

(C,3)
(C,7)
B
3
D
4
6
(A,0)
A
1
5
2
F
(-,∞)
2
3
C
E
(A,2)
10
(C,12)

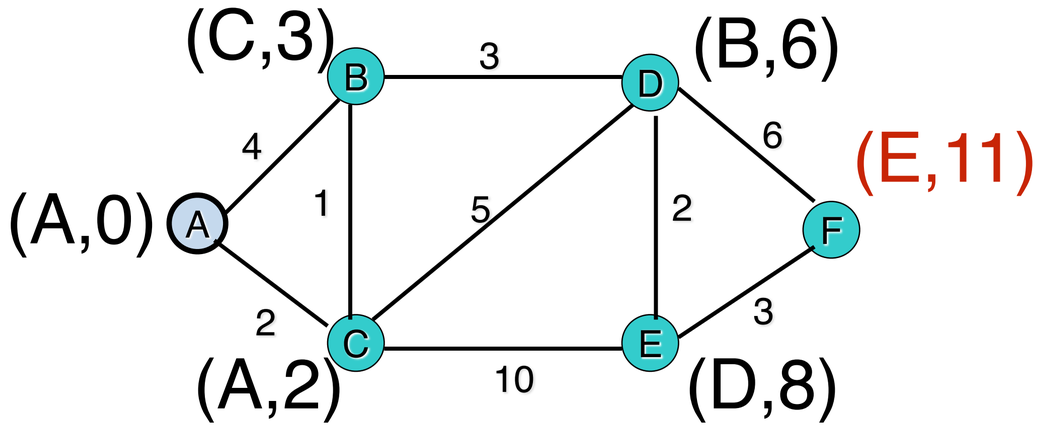

(C,3)
B
1
A
2
C
(A,2)

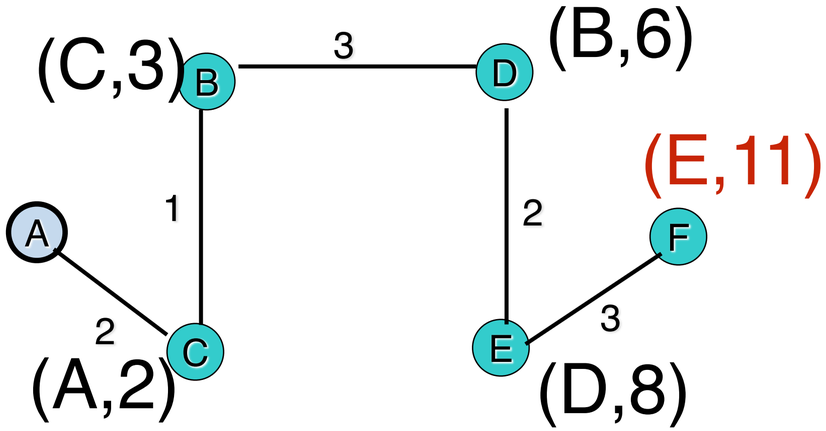

(C,3)
B
3
D
(B,6)
4
6
(-,∞)
(A,0)
A
1
5
2
F
2
3
C
E
(A,2)
10
(C,12)


(C,3)
B
3
D
(B,6)
A
1
2
C
(A,2)

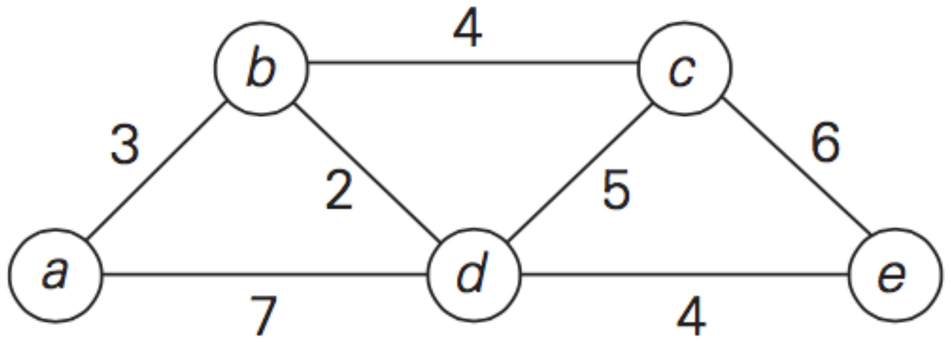

(C,3)
B
3
D
(B,6)
4
6
(D,12)
(A,0)
A
1
5
2
F
2
3
C
E
(A,2)
10
(D,8)


(C,3)
B
3
D
(B,6)
A
1
2
2
C
E
(A,2)
(D,8)

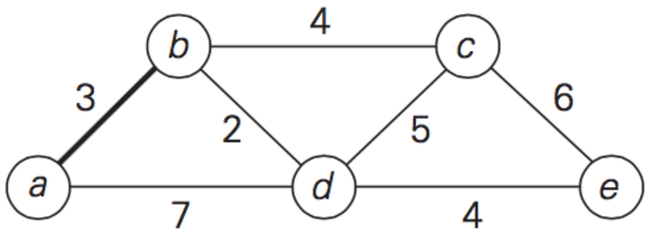
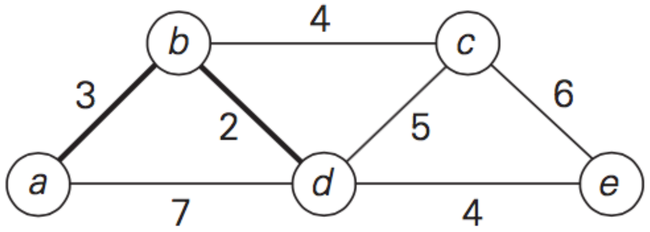

(C,3)
B
3
D
(B,6)
4
6
(E,11)
(A,0)
A
1
5
2
F
2
3
C
E
(A,2)
10
(D,8)


(C,3)
B
3
D
(B,6)
(E,11)
A
1
2
F
2
3
C
E
(A,2)
(D,8)

Dijkstra's Algorithm



Dijkstra's Algorithm

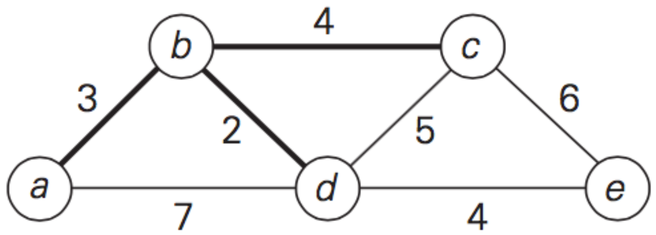
Tree vertices	Remaining vertices	Illustration
a(−, 0)	**b(a, 3)** c(−, ∞) d(a, 7) e(−, ∞)	
b(a, 3)	c(b, 3 + 4) **d(b, 3 + 2)** e(−, ∞)	

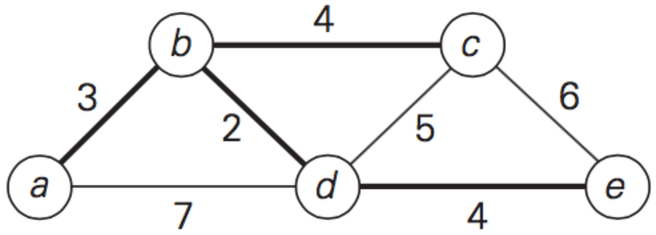
Dijkstra's Algorithm

d(b, 5) **c(b, 7)** e(d, 5 + 4)

c(b, 7) **e(d, 9)**

e(d, 9)





Dijkstra's Algorithm

The shortest paths (identified by following nonnumeric labels backward from a destination vertex in the left column to the source) and their lengths (given by numeric labels of the tree vertices) are as follows:

from a to b :	$a - b$	of length 3
from a to d :	$a - b - d$	of length 5
from a to c :	$a - b - c$	of length 7
from a to e :	$a - b - d - e$	of length 9

Question and Answer