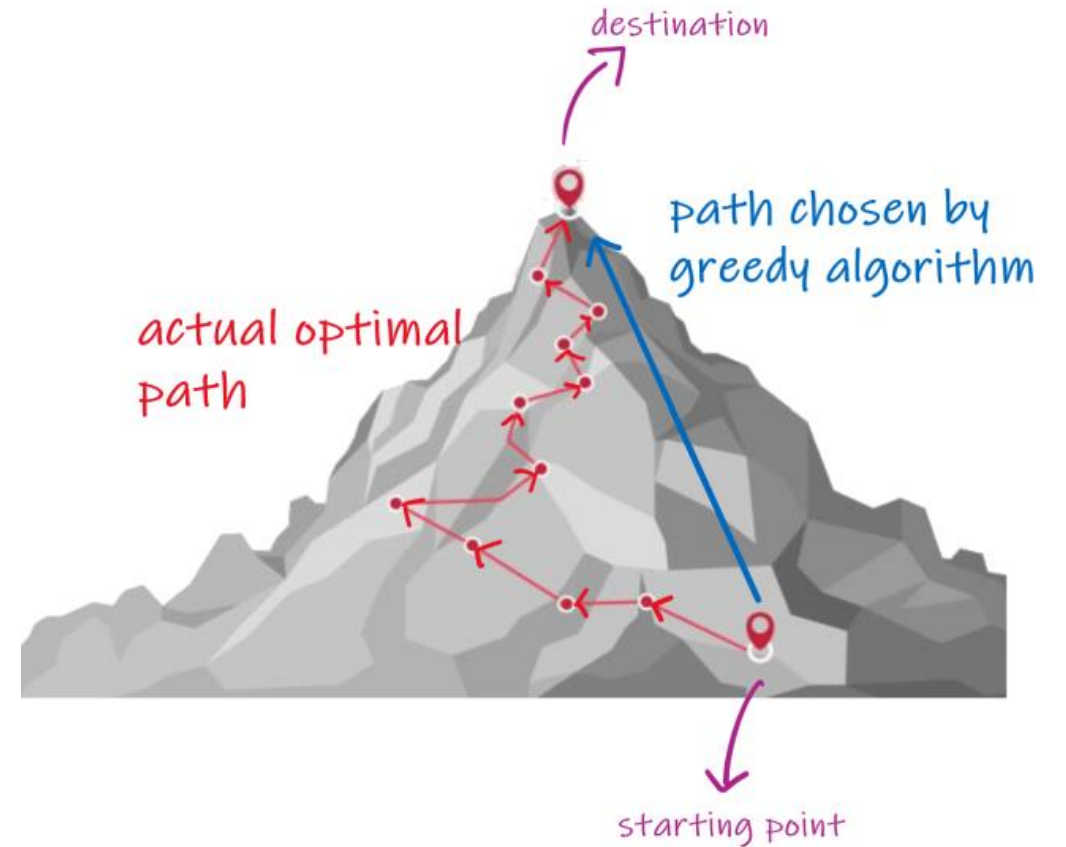


# DSE 2256 DESIGN & ANALYSIS OF ALGORITHMS

Lecture 36, 37, 38

## Greedy Technique

Prim's Algorithm  
Kruskal's Algorithm  
Dijkstra's Algorithm  
Huffman Coding



# Greedy Technique

- Constructs a solution to an optimization problem piece by piece through a sequence of choices that are:
  - **Feasible**, i.e. satisfying the problem constraints
  - **Locally optimal** (It has to be the best local choice among all choices in that step)
  - **Greedy** (in terms of some measure), **and irrevocable** (cannot be changed in subsequent steps)

For [some problems](#), it yields a globally optimal solution for every instance. For most, does not give globally optimal solution, but can be useful for fast approximations.

# Applications of the Greedy Strategy

- Optimal solutions:
  - Change making for “normal” coin denominations
  - Minimum spanning tree (MST)
  - Single-source shortest paths
  - Huffman codes
- Approximations/heuristics:
  - Traveling salesman problem (TSP)
  - Knapsack problem
  - Other combinatorial optimization problems

# Minimum Spanning Tree (MST)

- Spanning tree of a connected graph  $G = (V, E)$  is a **connected acyclic subgraph of  $G$**  that includes all of  $G$ 's vertices with the minimum possible no. of edges.

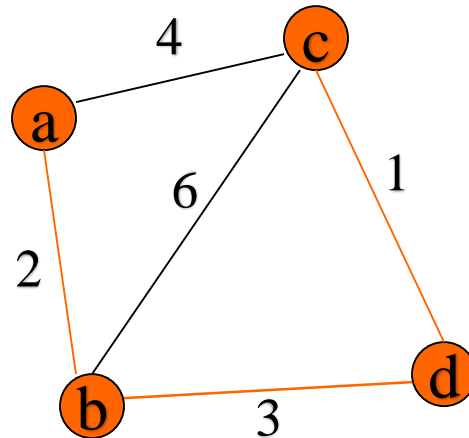
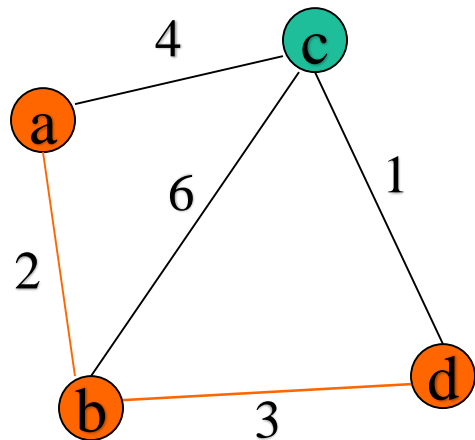
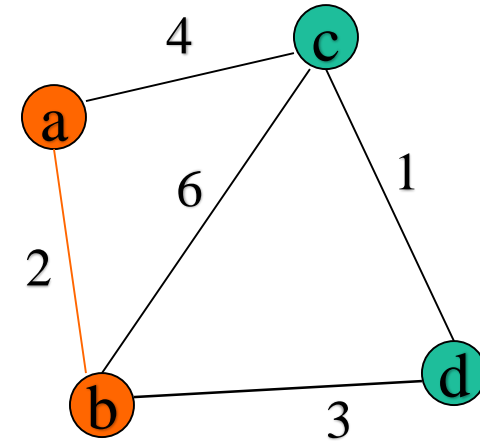
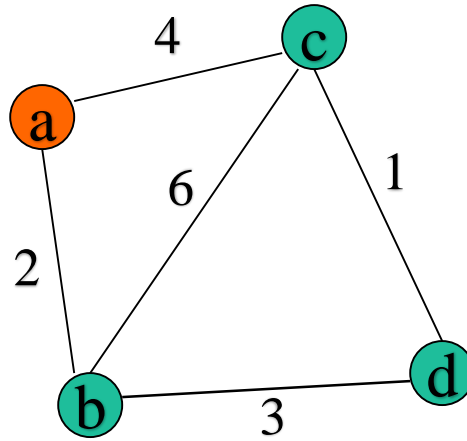
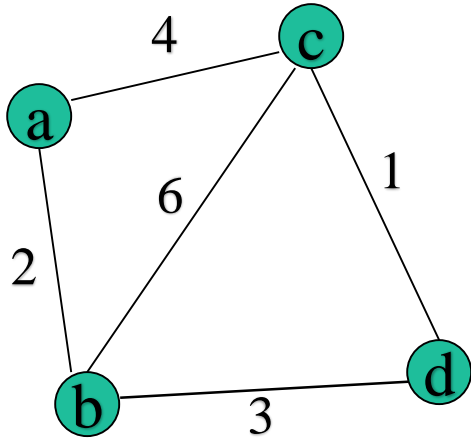
## Some useful properties of spanning trees:

- The spanning tree of a graph  $G$  has  $|V| - 1$  edges ( where,  $|V|$  =the total no. of vertices in  $G$ ).
- For any given graph, multiple spanning trees are possible.
- For a cycle graph (a graph with a single cycle) with  $n$  vertices, a total of  $n$  spanning trees are possible.
- For a complete graph ( $K_n$ ) with  $n$  vertices a total of  $n^{(n-2)}$  spanning trees are possible.
- Minimum spanning tree of a weighted, connected graph  $G$ : is a spanning tree of  $G$  of the **minimum total weight**.

# Prim's MST algorithm

- Start with tree  $T_1$  consisting of one (any) vertex and “grow” tree one vertex at a time to produce MST through a series of expanding subtrees  $T_1, T_2, \dots, T_n$
- On each iteration, construct  $T_{i+1}$  from  $T_i$  by adding vertex not in  $T_i$  that is closest to those already in  $T_i$  (this is a “greedy” step!)
- Stop when all vertices are included

# Prim's MST algorithm: Example



# Prim's MST algorithm

## **ALGORITHM** *Prim*( $G$ )

//Prim'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  $G$

$V_T \leftarrow \{v_0\}$  //the set of tree vertices can be initialized with any vertex

$E_T \leftarrow \emptyset$

**for**  $i \leftarrow 1$  **to**  $|V| - 1$  **do**

    find a minimum-weight edge  $e^* = (v^*, u^*)$  among all the edges  $(v, u)$

    such that  $v$  is in  $V_T$  and  $u$  is in  $V - V_T$

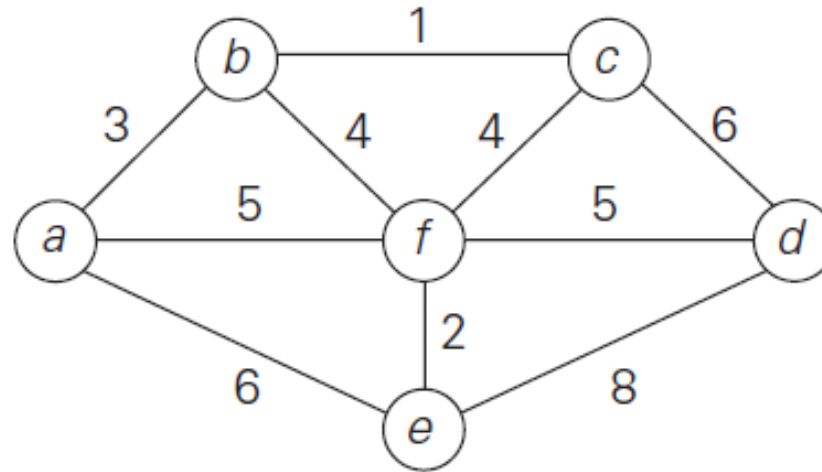
$V_T \leftarrow V_T \cup \{u^*\}$

$E_T \leftarrow E_T \cup \{e^*\}$

**return**  $E_T$

# Prim's MST algorithm: Example

**Apply Prim's algorithm to the following graph to find the minimum spanning tree**





# Prim's MST algorithm: Example

Start with any arbitrary vertex.  
Here, let us start with **a**



- At any step check whether there is a connection between all the intermediate tree vertices and the remaining vertices of the input graph.
- If so, choose the connection with the minimum cost.
- Repeat until there are no remaining graph vertices to be considered.
- The resultant tree obtained will be a spanning tree with the minimum total weight (MST).

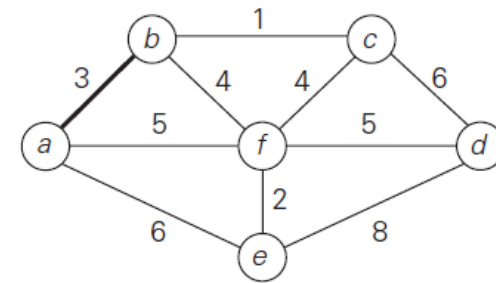
**Tree vertices**

**Remaining vertices**

**Illustration**

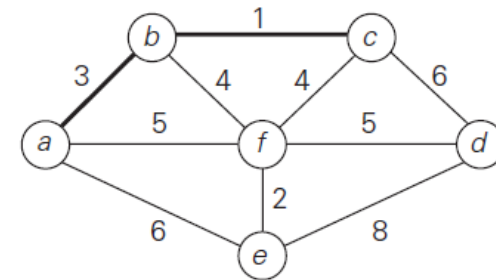
$a(-, -)$

**b(a, 3)**  $c(-, \infty)$   $d(-, \infty)$   
 $e(a, 6)$   $f(a, 5)$



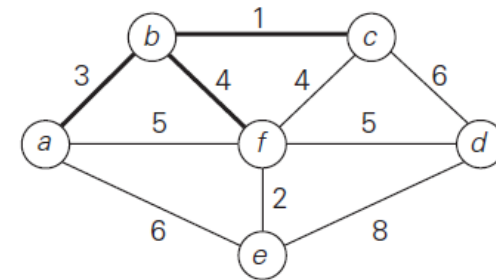
**b(a, 3)**

**c(b, 1)**  $d(-, \infty)$   $e(a, 6)$   
 $f(b, 4)$



**c(b, 1)**

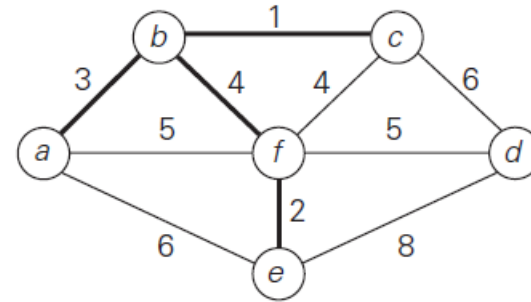
$d(c, 6)$   $e(a, 6)$  **f(b, 4)**



# Prim's MST algorithm: Example

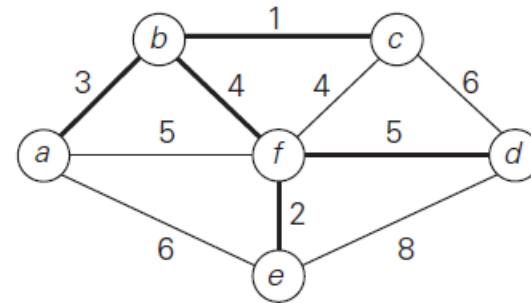
$f(b, 4)$

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



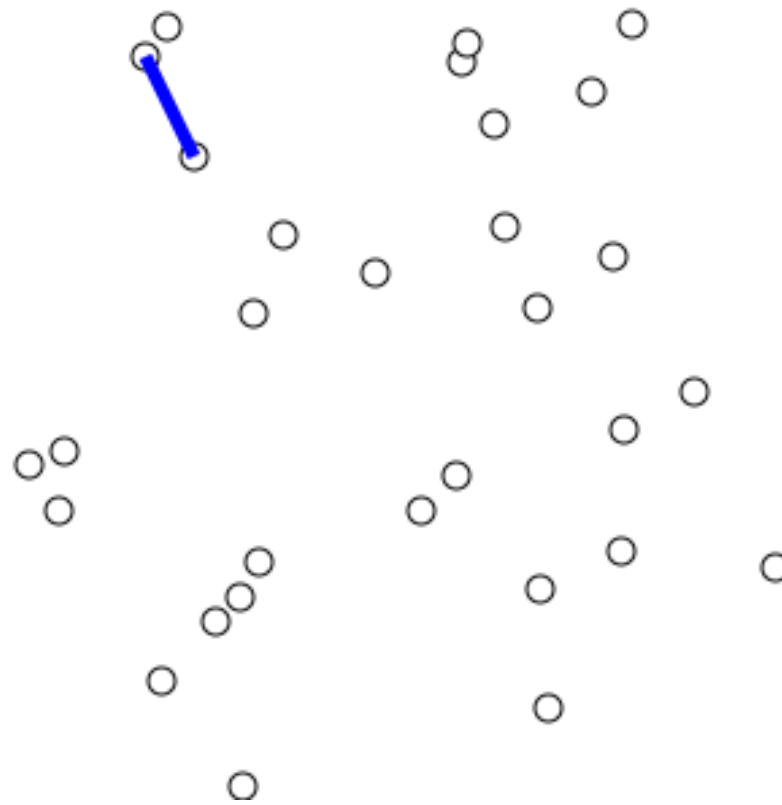
$e(f, 2)$

**$d(f, 5)$**



$d(f, 5)$

# Prim's MST algorithm: Example



Courtesy - Shiyu Ji (Wikipedia)

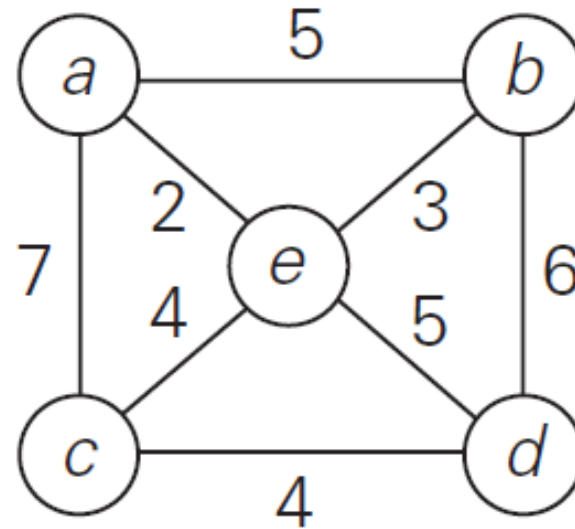
# Prim's MST algorithm : Efficiency

- Depends on data structures chosen for graph.
  - Generally **implemented using priority queue** (which contains the priorities of the remaining graph vertices based on their distances to the nearest tree vertices).
- I. If the **graph is implemented using weighted adjacency matrix** and **the priority queue is implemented using arrays**, then Prim's algorithm's running time will be:  **$O(|V|^2)$**
  - II. If the graph is implemented using adjacency lists and the priority queue is implemented as a min-heap, then the algorithm performs  **$|V|-1$  deletions of the smallest element and makes  $|E|$  verifications**.  
In this case, the running time of the Prim's algorithm is in :

$$(|V|-1+|E|) O(\log|V|) = \mathbf{O(|E| \log |V|)} \quad (\text{because in a connected graph, } |V|-1 \leq |E|)$$

# Exercise I

**Apply Prim's algorithm to the following graph to find the minimum spanning tree.**



# Kruskal's Algorithm

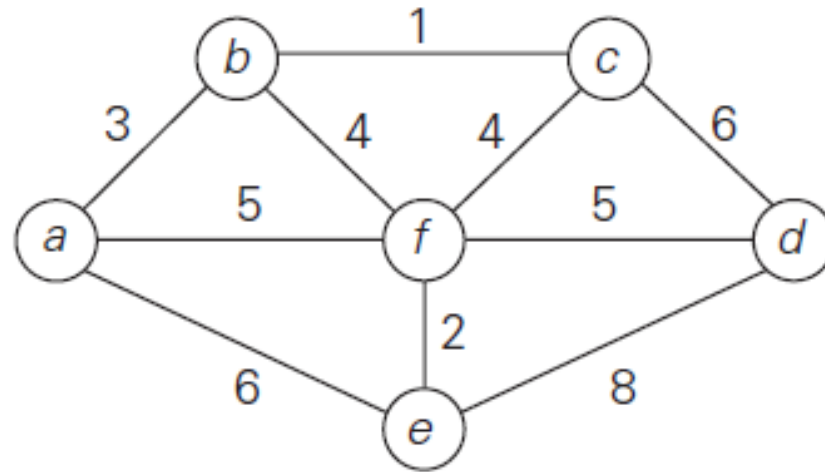
- Another greedy algorithm for computing the Minimum Spanning Tree.
- Developed by Joseph Kruskal in 1956.

## Kruskal's Algorithm (Working):

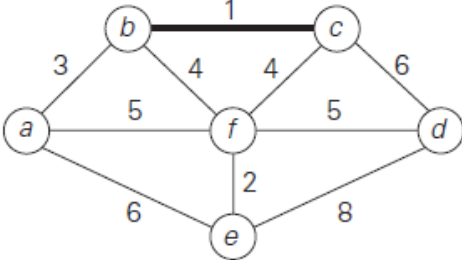
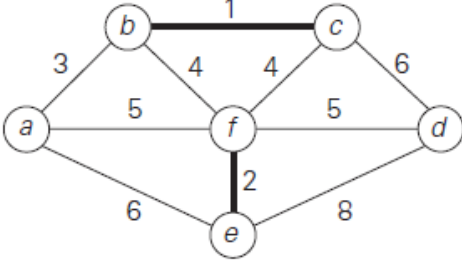
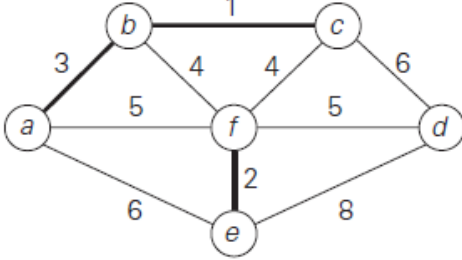
- Sort the edges in nondecreasing order of lengths
- "Grow" tree one edge at a time to produce MST through a series of expanding forests\*  $F_1, F_2, \dots, F_{n-1}$
- On each iteration, add the next edge on the sorted list unless this would create a cycle. (If it would, skip the edge.)

# Kruskal's Algorithm: Example

**Apply Kruskal's algorithm to the following graph to find the minimum spanning tree**



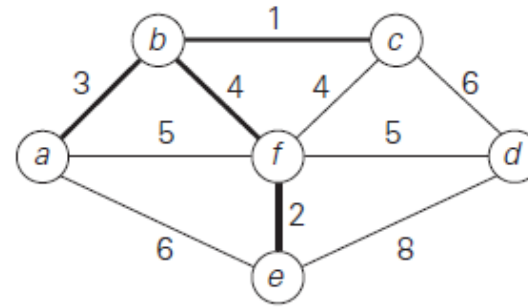
# Kruskal's Algorithm: Example

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

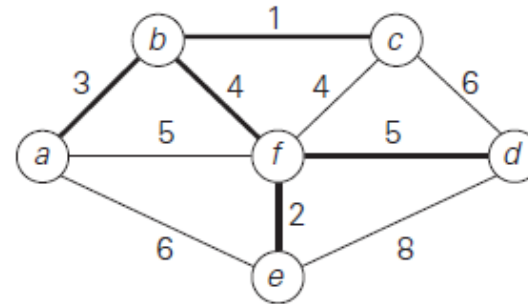


# Kruskal's Algorithm: Example

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

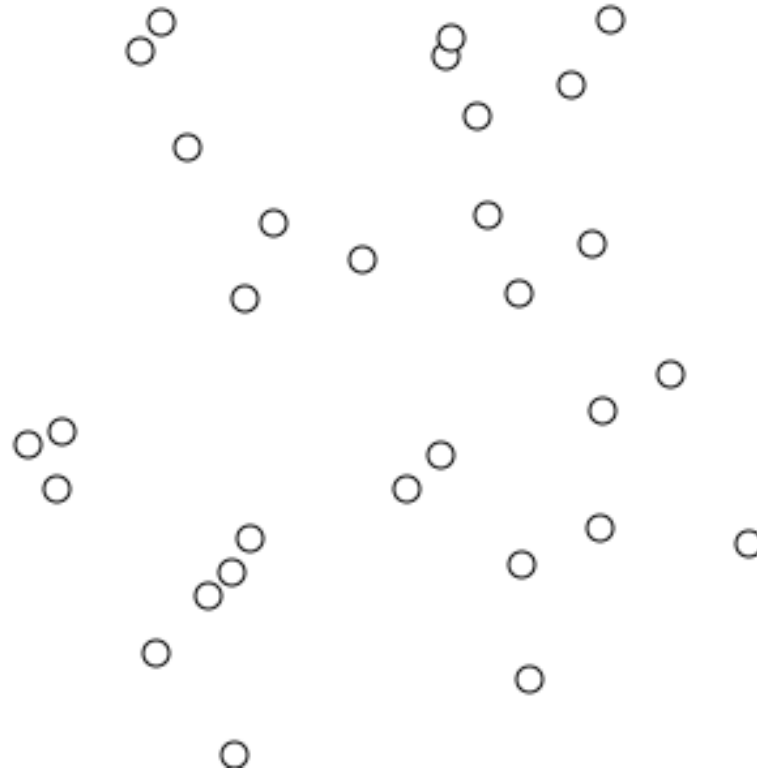


bf 4      bc 1   ef 2   ab 3   bf 4   cf 4   af 5   **df** 5   ae 6   cd 6   de 8



df 5

# Kruskal's Algorithm: Example



Courtesy - Shiyu Ji (Wikipedia)

# Kruskal's Algorithm and its efficiency

## 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  $w(e_{i_1}) \leq \dots \leq w(e_{i_{|E|}})$   
 $E_T \leftarrow \emptyset$ ;  $ecounter \leftarrow 0$      //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**  $E_T$

### Time Complexity:

$$T_{\text{kruskal (worst case)}} = T_{\text{sort\_edges}} + T_{\text{check\_acyclic}}$$

$$\text{Where, } T_{\text{check\_acyclic}} = T_{\text{create\_subsets}} + T_{\text{union\_find}}$$

$$T_{\text{check\_acyclic}} = O(|V|) + O(|E| \log |E|) = O(|E| \log |E|)$$

and

$$T_{\text{sort\_edges}} = O(|E| \log |E|)$$

$$\text{Therefore, } T_{\text{kruskal (worst case)}} = O(|E| \log |E|)$$

# Dijkstra's Algorithm

- It is applied to **Single source shortest problem**: for a given vertex called the source in weighted connected graph, find shortest paths to its other vertices.
- Introduced by Edsger W. Dijkstra in 1956.
- Applicable to both directed and undirected graphs with non-negative weights.

## Applications:

- Transportation planning.
- Packet routing in communication networks.
- Shortest paths social networks, robotics, speech recognition, document formatting, robotics, compilers, airline crew scheduling.
- Path finding in video games, puzzles etc.

# Dijkstra's Algorithm

It is similar to Prim's MST algorithm, with a different way of computing numerical labels:

Among vertices not already in the tree, it finds vertex  $u$  with the smallest sum

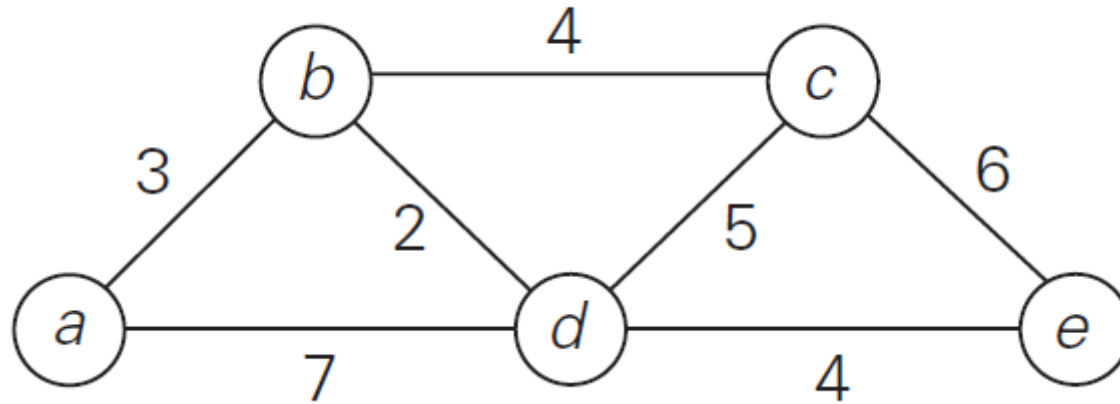
$$d_v + w(v,u)$$

Where,

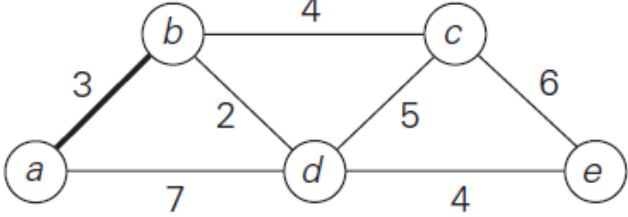
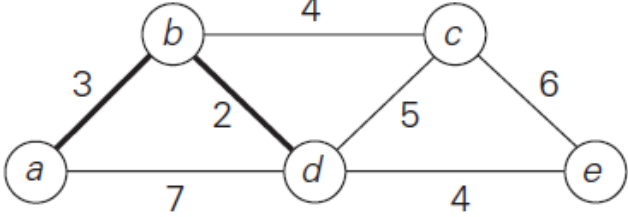
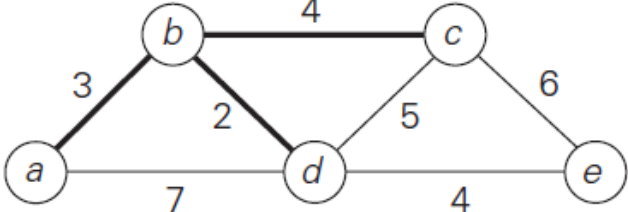
- $v$  is a vertex for which shortest path has been already found on preceding iterations (such vertices form a tree rooted at  $s$ )
- $d_v$  is the length of the shortest path from source  $s$  to  $v$
- $w(v,u)$  is the length (weight) of edge from  $v$  to  $u$

# Dijkstra's Algorithm : Example

**Apply Dijkstra's algorithm to the following graph to find the shortest paths from source vertex "a" to all other vertices.**



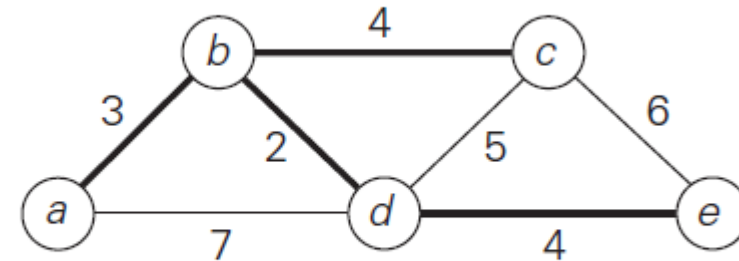
# Dijkstra's Algorithm : Example

Tree vertices	Remaining vertices	Illustration
$a(-, 0)$	$\mathbf{b(a, 3)}$ $c(-, \infty)$ $d(a, 7)$ $e(-, \infty)$	
$\mathbf{b(a, 3)}$	$c(\mathbf{b}, 3 + 4)$ $\mathbf{d(b, 3 + 2)}$ $e(-, \infty)$	
$\mathbf{d(b, 5)}$	$\mathbf{c(b, 7)}$ $e(\mathbf{d}, 5 + 4)$	

# Dijkstra's Algorithm : Example

$c(b, 7)$

$e(d, 9)$



$e(d, 9)$

---

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



# Dijkstra's Algorithm

## ALGORITHM *Dijkstra*( $G, s$ )

//Dijkstra's algorithm for single-source shortest paths

//Input: A weighted connected graph  $G = \langle V, E \rangle$  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  $V$

$d_v \leftarrow \infty$ ;  $p_v \leftarrow \text{null}$

*Insert*( $Q, v, d_v$ ) //initialize vertex priority in the priority queue

$d_s \leftarrow 0$ ; *Decrease*( $Q, s, d_s$ ) //update priority of  $s$  with  $d_s$

$V_T \leftarrow \emptyset$

**for**  $i \leftarrow 0$  **to**  $|V| - 1$  **do**

$u^* \leftarrow \text{DeleteMin}(Q)$  //delete the minimum priority element

$V_T \leftarrow V_T \cup \{u^*\}$

**for** every vertex  $u$  in  $V - V_T$  that is adjacent to  $u^*$  **do**

**if**  $d_{u^*} + w(u^*, u) < d_u$

$d_u \leftarrow d_{u^*} + w(u^*, u)$ ;  $p_u \leftarrow u^*$

*Decrease*( $Q, u, d_u$ )

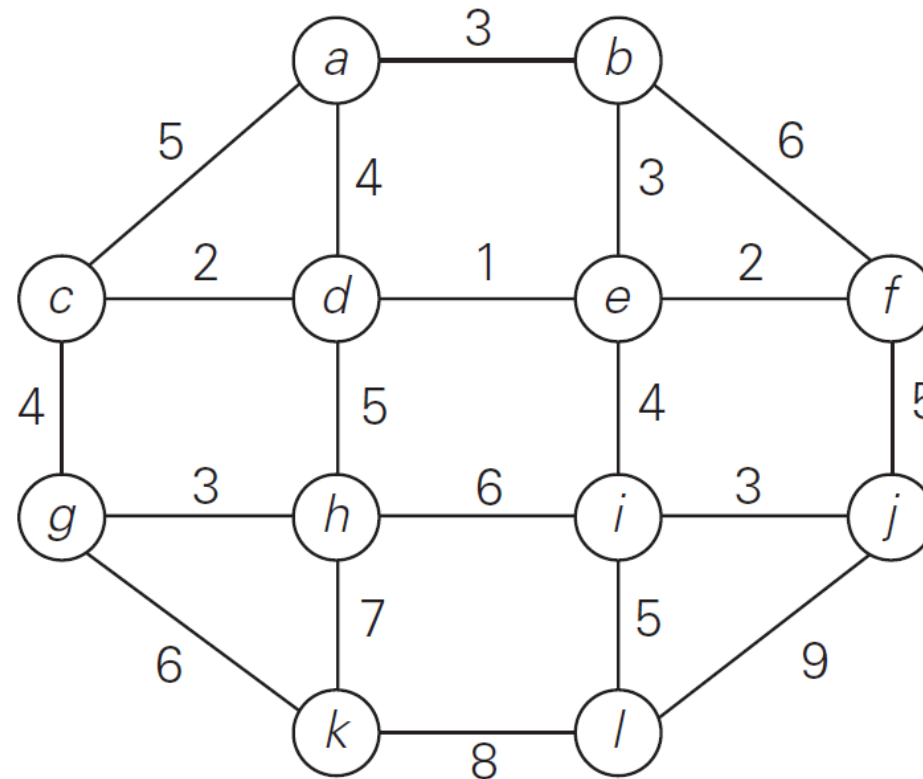
Time Complexity is similar to Prim's Algorithm

If implemented using :

- Weighted adjacency matrix + array-based priority queue, the complexity is  $O(|V|^2)$
- Adjacency lists + Min-heap based priority queue, the complexity is  $O(|E| \log |E|)$

# Exercise II

**Apply Prim's, Kruskal's and Dijkstra's algorithms to the following graph (assume starting/source vertex as "a" for Prim's and Dijkstra's).**



# Thank you!

## Any queries?