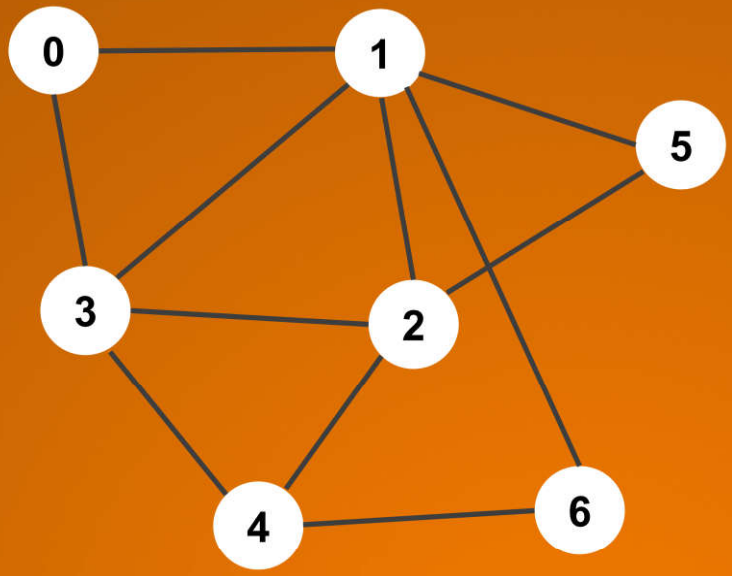
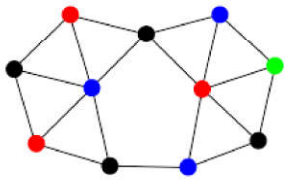
Graph



ITD62-124 Data Structure

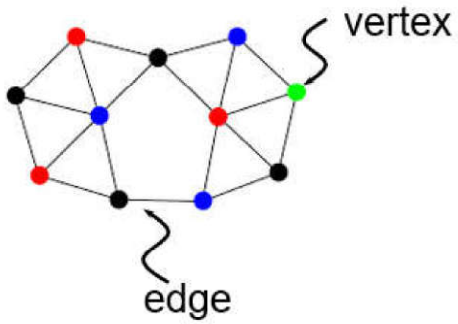
Outline:

- Elementary graph algorithms
- Minimum spanning trees
- Single-source shortest paths



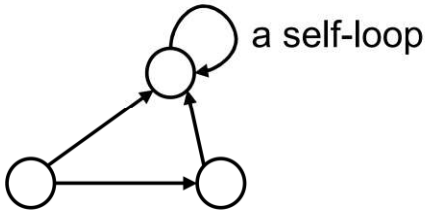
Elementary graph algorithms:

- A *graph* is a bunch of vertices connected by edges
- A *graph* is an ordered pair (V, E)
  - ✓V is the set of vertices
  - ✓E is the set of edges. An edge is a pair of vertices: (u, v)
- There is at most one edge between two vertices
- Example:

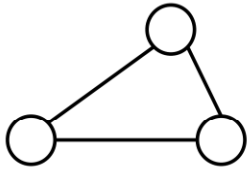


Elementary graph algorithms:

- *directed graph*, the edges are “one-way”

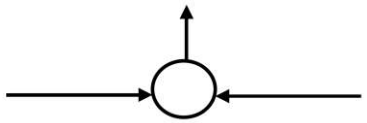


- *undirected graph*, there is no direction on the edges



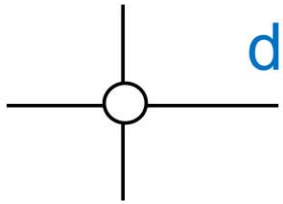
Elementary graph algorithms:

- Directed graph: the *in-degree* is the number of edges entering the vertex, and the *out-degree* is the number leaving it. The *degree* is the *in-degree* + the *out-degree*.



in-degree 2, out-degree 1

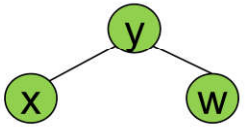
- Undirected graph: The *degree* of a vertex is the number of edges touching it



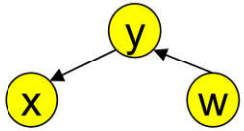
degree 4

Elementary graph algorithms:

- Two vertices are *adjacent* if there is an edge between them
- For a directed graph, *u* is adjacent to *v* iff there is an edge (*v*, *u*)
- Example:



x is adjacent to *y*
y is adjacent to *x* and *w*
w is adjacent to *y*



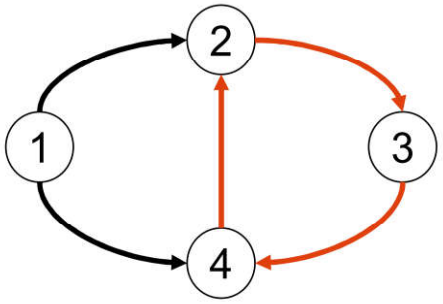
x is adjacent to *y*
y is adjacent to *w*

Elementary graph algorithms:

- *Weight* is a cost parameter associated with each edge
- A *path* is a sequence of vertices $w_1, w_2, \ldots, w_n$ where there is an edge for each pair of consecutive vertices
- The *length* of a path is the number of edges it contains (one less than the number of vertices)
- *Cost of a path* is sum of the weights of the edges along the path
- The graph is a *complete graph* if there is an edge between every pair of vertices

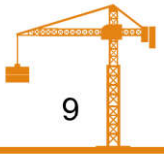
Elementary graph algorithms:

- A *cycle* is a path of length at least 1 from a vertex to itself
  - ✓Example: *cycle*



- A graph with no cycles is *acyclic*
- A path with no cycles is a *simple* path

Elementary graph algorithms:



- Types of graphs:
  - ✓Undirected: edge $(u, v) = (v, u)$; for all v, $(v, v) \notin E$ (No self loops)
  - ✓Directed: (u, v) is edge from u to v, denoted as $u \rightarrow v$. Self loops are allowed
  - ✓Weighted: each edge has an associated weight, given by a weight function $w : E \rightarrow R$ (R = set of all possible real numbers)

Elementary graph algorithms:

- Graphs can be represented by their adjacency matrix or adjacency list
- Adjacency matrices:

✓have a value $a_{i,j} = 1$ if nodes i and j share an edge; 0 otherwise

✓in case of a weighted graph, $a_{i,j} = w_{i,j}$, the weight of the edge

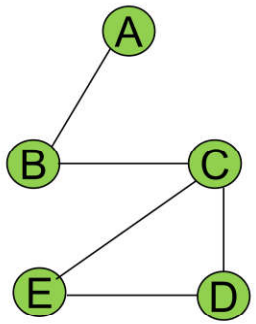
- Adjacency list of a graph $G = (V,E)$

✓consists of an array Adj of $|V|$ lists, one for each vertex in V

✓For each $u \in V$, the adjacency list $Adj[u]$ contains all vertices v such that there is an edge $(u,v) \in E$

Elementary graph algorithms:

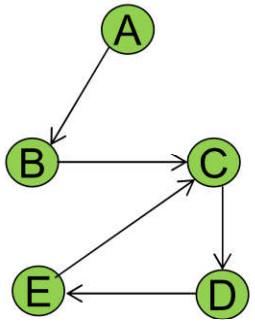
- Example: undirected graph



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

Elementary graph algorithms:

- Example: directed graph



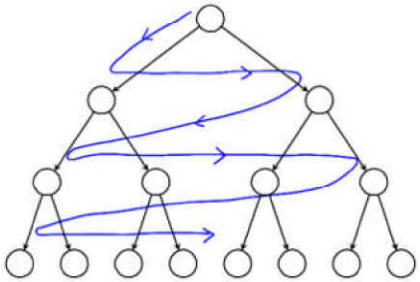
$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Elementary graph algorithms:

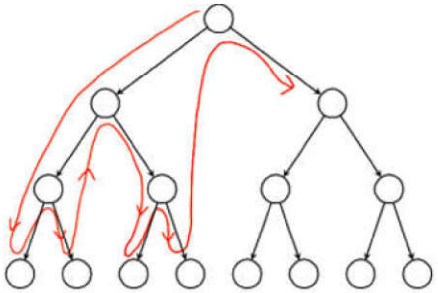
- Graph traversal:
  - ✓Searching a graph: systematically follow the edges of a graph to visit the vertices of the graph
  - ✓Used to discover the structure of a graph
  - ✓Standard graph-searching algorithms:
    - ➢Breadth-first Search (BFS)
    - ➢Depth-first Search (DFS)

Elementary graph algorithms:

- Graph traversal: Breadth-first Search (BFS)



- Graph traversal: Depth-first Search (DFS)



Elementary graph algorithms:

❑Graph traversal: Breadth-first Search (BFS)

- Input: Graph $G = (V, E)$, either directed or undirected, and source vertex $s \in V$
- Output:
  - ✓$d[v]$ = distance (smallest # of edges, or shortest path) from s to v, for all $v \in V$
  - ✓$d[v] = \infty$ if v is not reachable from s
  - ✓$\pi[v] = u$ such that (u, v) is last edge on shortest path $s \rightsquigarrow v$
    - ✓u is v's predecessor
  - ✓Builds breadth-first tree with root s that contains all reachable vertices

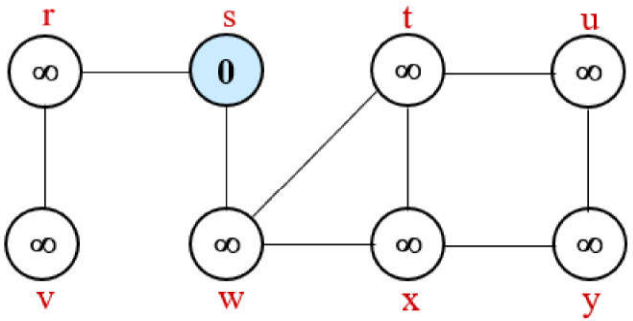
Elementary graph algorithms:

- Graph traversal: Breadth-first Search (BFS)

```
BFS(G,s)
1. for each vertex u in V[G] – {s}
2            u.color = white
3            u.d = ∞
4             u.π = nil
5   s.color = gray
6   s.d = 0
7   s.π = nil
8  Q = ∅
9  enqueue(Q, s)
10 while Q ≠ ∅
11           u = dequeue(Q)
12           for each v in G.Adj[u]
13                if v.color == white
14                    v.color = gray
15                    v.d = u.d + 1
16                    v.π = u
17                    enqueue(Q, v)
18         u.color = black
```

Elementary graph algorithms:

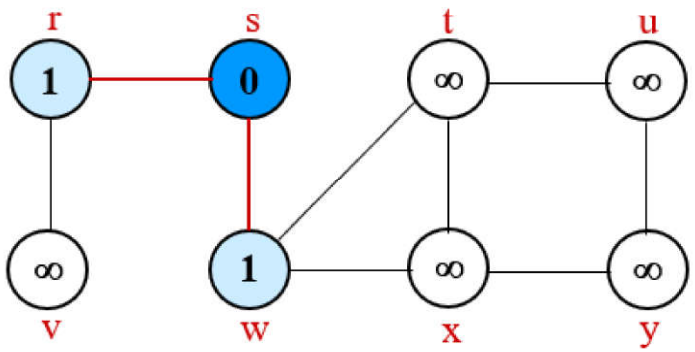
- Graph traversal: Breadth-first Search (BFS)
  - ✓Example:



Q:	s
	0

Elementary graph algorithms:

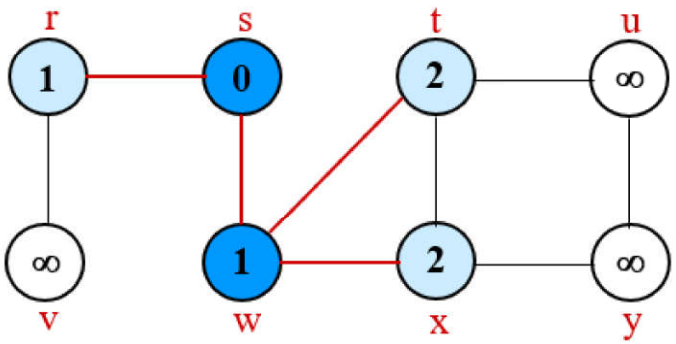
- Graph traversal: Breadth-first Search (BFS)
  - ✓Example:



Q:	w	r
	1	1

Elementary graph algorithms:

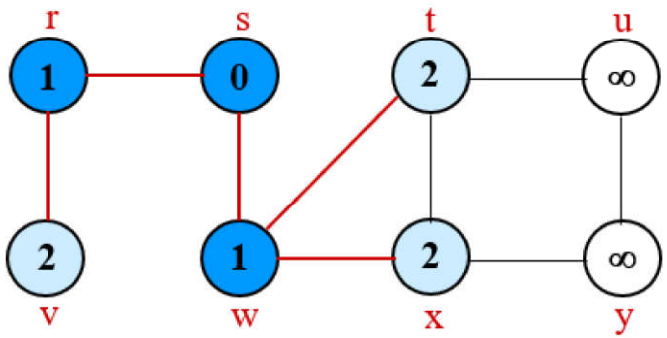
- Graph traversal: Breadth-first Search (BFS)
  - ✓Example:



Q:	r	t	x
	1	2	2

Elementary graph algorithms:

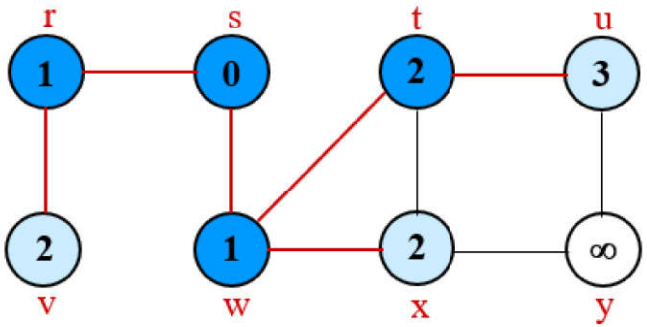
- Graph traversal: Breadth-first Search (BFS)
  - ✓Example:



Q:	t	x	v
	2	2	2

Elementary graph algorithms:

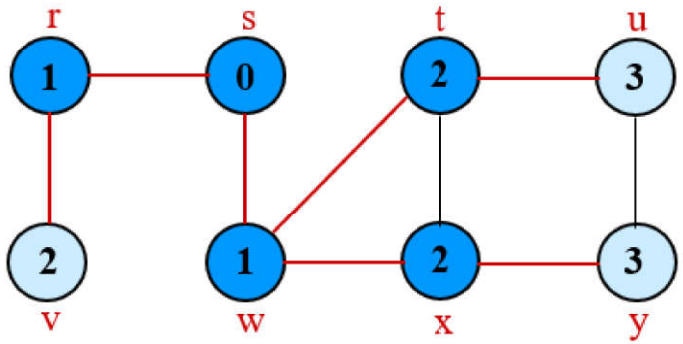
- Graph traversal: Breadth-first Search (BFS)
  - ✓Example:



Q:	x	v	u
	2	2	3

Elementary graph algorithms:

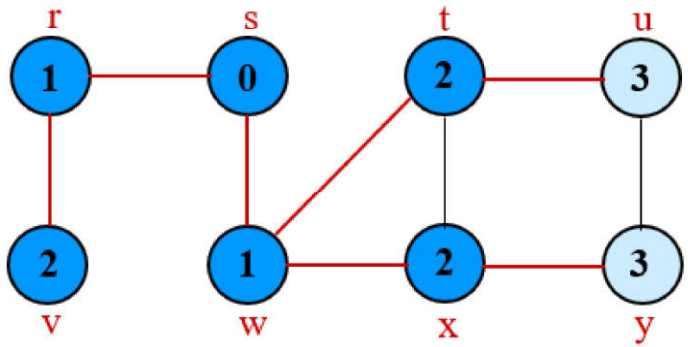
- Graph traversal: Breadth-first Search (BFS)
  - ✓Example:



Q:	v	u	y
	2	3	3

Elementary graph algorithms:

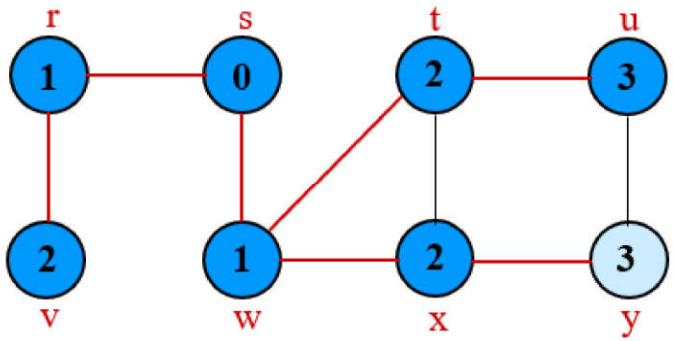
- Graph traversal: Breadth-first Search (BFS)
  - ✓Example:



Q:	u	y
	3	3

Elementary graph algorithms:

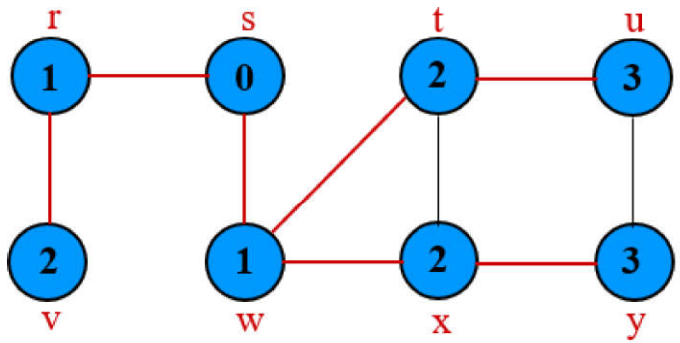
- Graph traversal: Breadth-first Search (BFS)
  - ✓Example:



Q:	y
	3

Elementary graph algorithms:

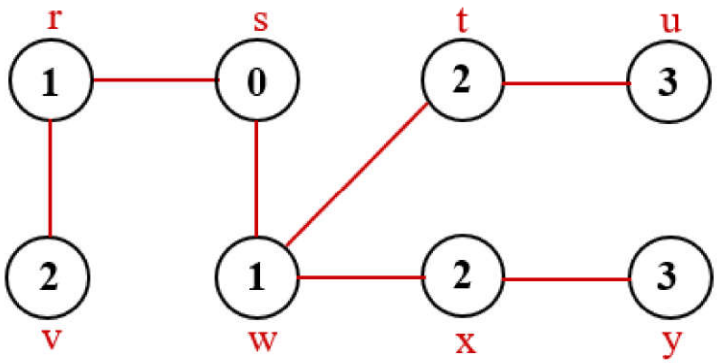
- Graph traversal: Breadth-first Search (BFS)
  - ✓Example:



Q: ∅

Elementary graph algorithms:

- Graph traversal: Breadth-first Search (BFS)
  - ✓Example:



Elementary graph algorithms:

- Explore edges out of the most recently discovered vertex *v*
- When all edges of *v* have been explored, backtrack to explore other edges leaving the vertex from which *v* was discovered (its predecessor)
- Continue until all vertices reachable from the original source are discovered
- If any undiscovered vertices remain, then one of them is chosen as a new source and search is repeated from that source

Elementary graph algorithms:

- **Tree edge** are edges in the depth-first forest G_π. Edge (*u, v*) is a tree edge if *v* was first discovered by exploring edge (*u, v*)
- Back edges (B) are those edges (*u, v*) connecting a vertex *u* to an ancestor *v* in a depth-first tree
- Forward edges (F) are those nontree edges (*u, v*) connecting a vertex *u* to a descendant *v* in a depth-first tree
- Cross edges (C) are all other edges. They can go between vertices in the same depth-first tree, as long as one vertex is not an ancestor of the other, or they can go between vertices in different depth-first tree

Elementary graph algorithms:

- Graph traversal: Depth-first Search (DFS)
- Input: Graph $G = (V, E)$, directed or undirected
- Output:
  - ✓2 **timestamps** on each vertex. Integers between 1 and $2|V|$
    - $d[v]$ = ***discovery time*** (v turns from white to gray)
    - $f[v]$ = ***finishing time*** (v turns from gray to black)
  - ✓$\pi[v]$: predecessor of $v = u$, such that v was discovered during the scan of u's adjacency list

Elementary graph algorithms:

- Graph traversal: Depth-first Search (DFS)

DFS(*G*)

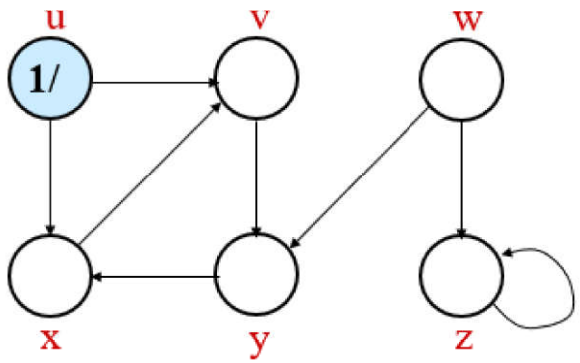
```
1 for each vertex u in V[G]
2     u.color = white
3     u.π = nil
4 time = 0
5 for each vertex u in V[G]
6     If u.color == white
7         DFS-VISIT(G, u)
```

DFS-VISIT(*G*, *u*)

```
1. time = time + 1
2. u.d = time
3. u.color = gray
4. for each vertex v in G.Adj[u]
5.     If v.color == white
6         v.π = u
7         DFS-VISIT(G, v)
8   u.color = black
9   time = time + 1
10 u.f = time
```

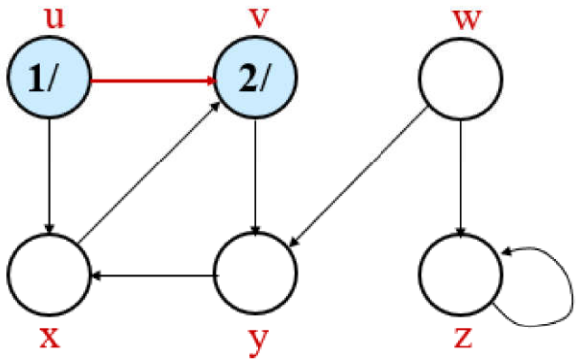
Elementary graph algorithms:

- Graph traversal: Depth-first Search (DFS)
  - ✓Example:



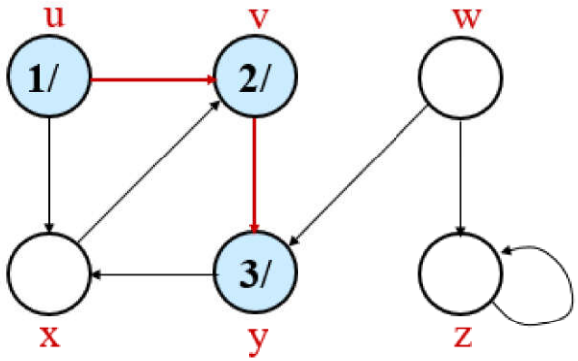
Elementary graph algorithms:

- Graph traversal: Depth-first Search (DFS)
  - ✓Example:



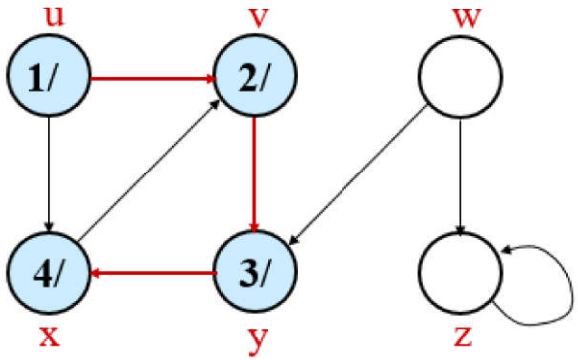
Elementary graph algorithms:

- Graph traversal: Depth-first Search (DFS)
  - ✓Example:



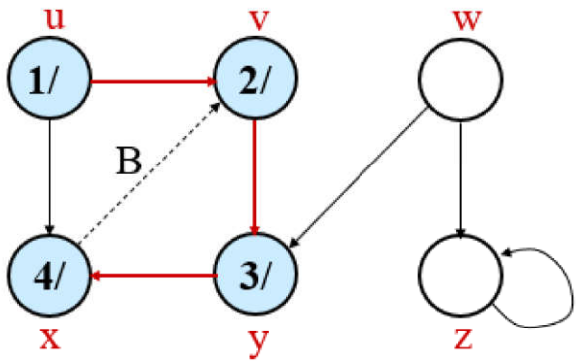
Elementary graph algorithms:

- Graph traversal: Depth-first Search (DFS)
  - ✓Example:



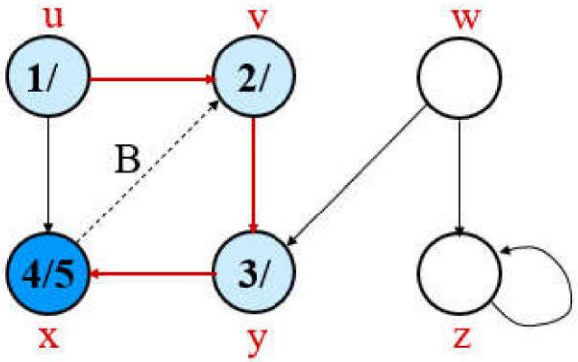
Elementary graph algorithms:

- Graph traversal: Depth-first Search (DFS)
  - ✓Example:



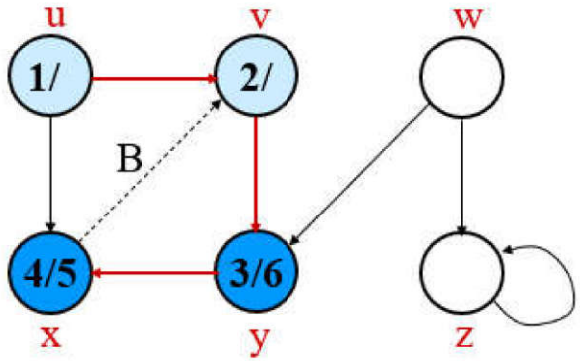
Elementary graph algorithms:

- Graph traversal: Depth-first Search (DFS)
  - ✓Example:



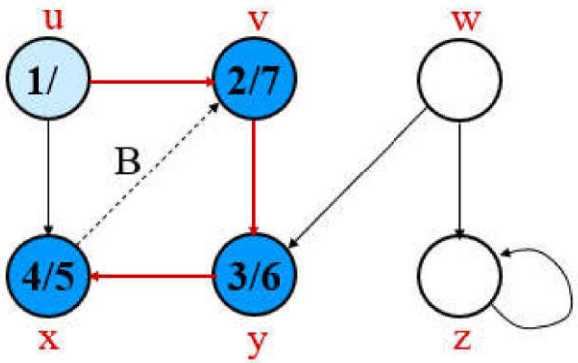
Elementary graph algorithms:

- Graph traversal: Depth-first Search (DFS)
  - ✓Example:



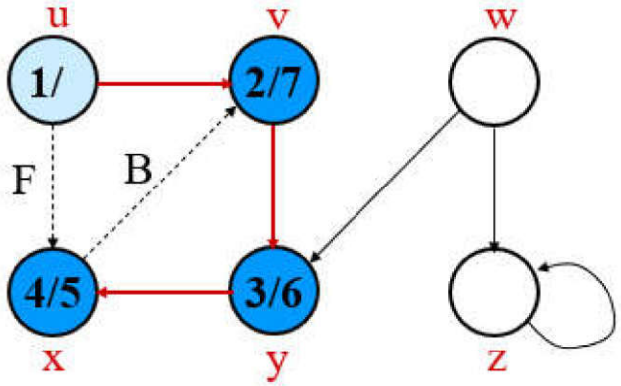
Elementary graph algorithms:

- Graph traversal: Depth-first Search (DFS)
  - ✓Example:



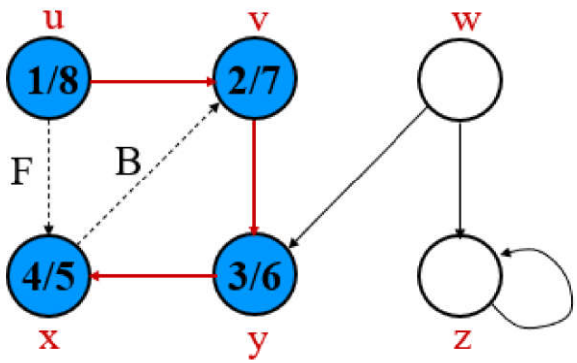
Elementary graph algorithms:

- Graph traversal: Depth-first Search (DFS)
  - ✓Example:



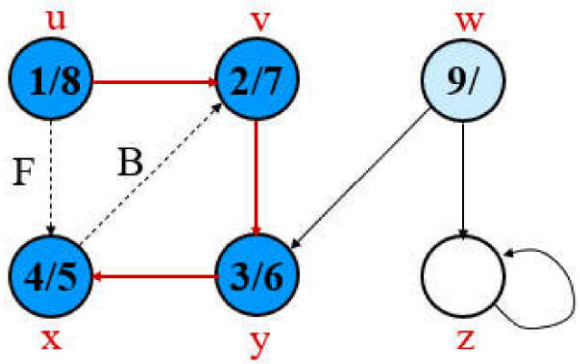
Elementary graph algorithms:

- Graph traversal: Depth-first Search (DFS)
  - ✓Example:



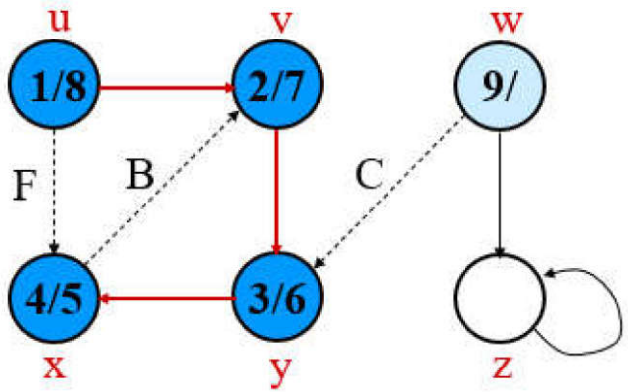
Elementary graph algorithms:

- Graph traversal: Depth-first Search (DFS)
  - ✓Example:



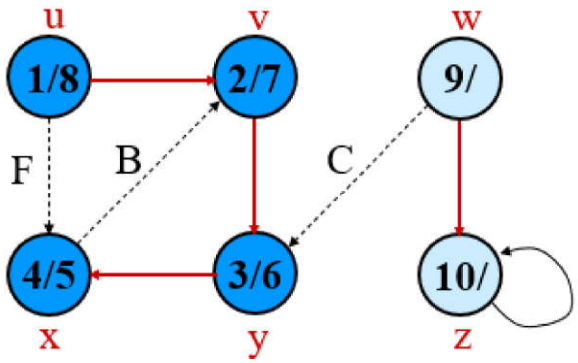
Elementary graph algorithms:

- Graph traversal: Depth-first Search (DFS)
  - ✓Example:



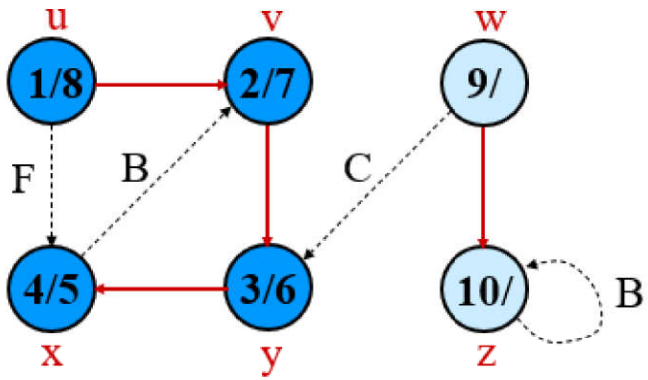
Elementary graph algorithms:

- Graph traversal: Depth-first Search (DFS)
  - ✓Example:



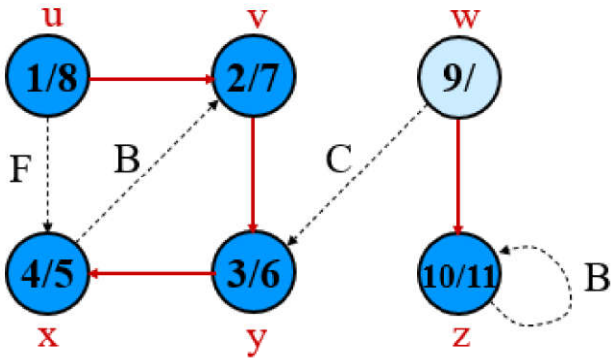
Elementary graph algorithms:

- Graph traversal: Depth-first Search (DFS)
  - ✓Example:



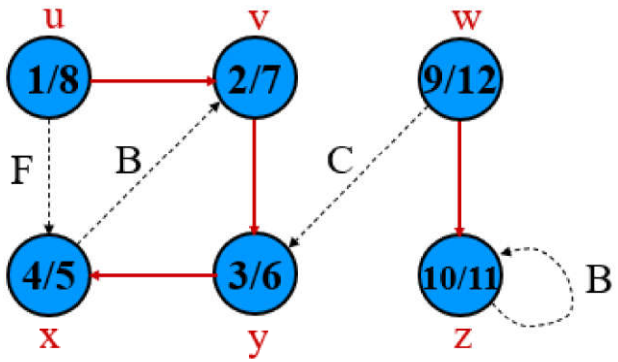
Elementary graph algorithms:

- Graph traversal: Depth-first Search (DFS)
  - ✓Example:



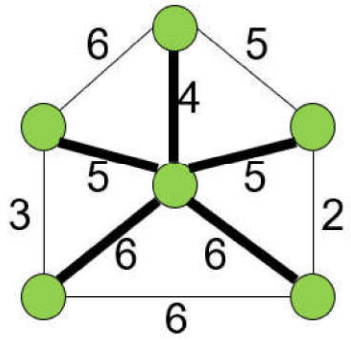
Elementary graph algorithms:

- Graph traversal: Depth-first Search (DFS)
  - ✓Example:

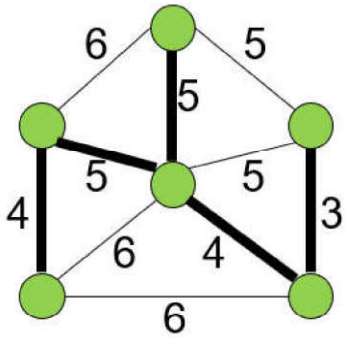


Minimum Spanning Tree :

- A *spanning tree* of an undirected graph *G* is a tree that contains every vertex of *G*
- The *weight* of a tree is the sum of its edges' weights
- Example:



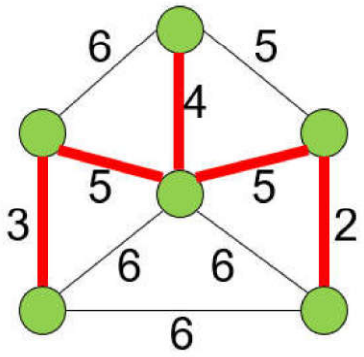
weight = 26



weight = 21

Minimum Spanning Tree :

- A *minimal spanning tree (mst)* is a spanning tree with lowest weight
- Example:



Minimum Spanning Tree :

- **Kruskal's algorithm**
  - ✓find a safe edge to add to the growing forest by finding an edge (u, v) of least weight

Minimum Spanning Tree :

- **Kruskal's algorithm**

```
MST-Kruskal (G, w)
 1  A = ∅
 2   for each vertex v ∈ G.V
 3      MAKE-SET (v)
 4   Sort the edges of G.E into nondecreasing order by weight w
 5   for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight w
 6         if FIND-SET (u) ≠ FIND-SET (v)
 7               A = A U {(u, v)}
 8               UNION {(u, v)}
 9  return A
```

Minimum Spanning Tree :

- **Kruskal's algorithm**

```
MAKE-SET (x)
 1  x.p = x                          // x.p = parent of x
 2  x.rank = 0
//  x.rank = the number of edges in the longest simple path between x and a descendant leaf
```

```
FIND-SET (x)
 1  if x ≠ x.p                       // x.p = parent of x
 2      x.p = FIND-SET (x.p)
 3  return x.p
```

Minimum Spanning Tree :

- **Kruskal's algorithm**

UNION (*x,y*)

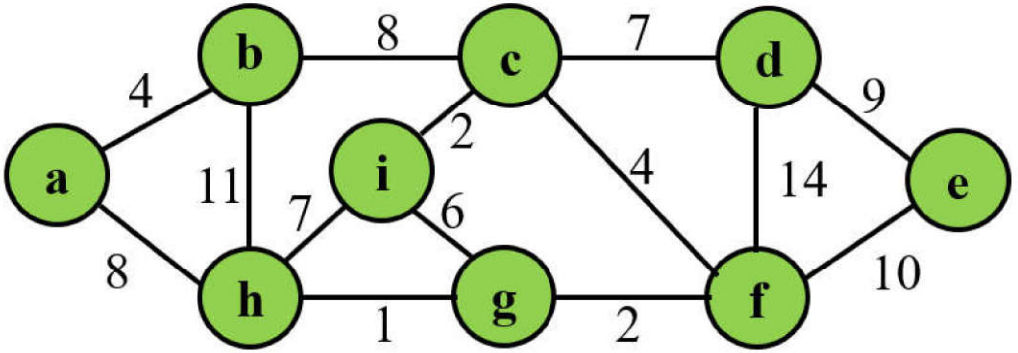
```
1  LINK(FIND-SET(x), FIND-SET(y))
```

LINK (*x,y*)

```
1  if x.rank > y.rank
2      y.p = x
3  else x.p = y
4    if x.rank == y.rank
5        y.rank = y.rank +1
```

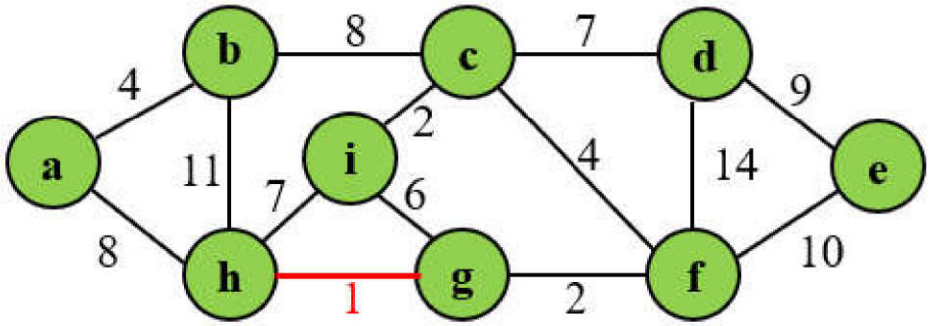
Minimum Spanning Tree :

- **Kruskal's algorithm**
  - ✓Example:



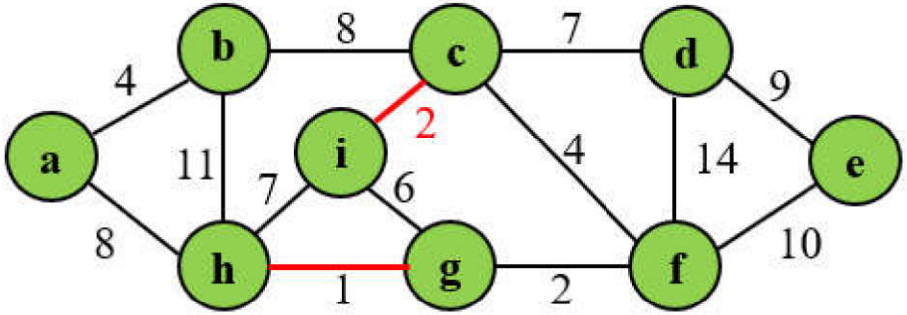
Minimum Spanning Tree :

- **Kruskal's algorithm**
  - ✓Example:



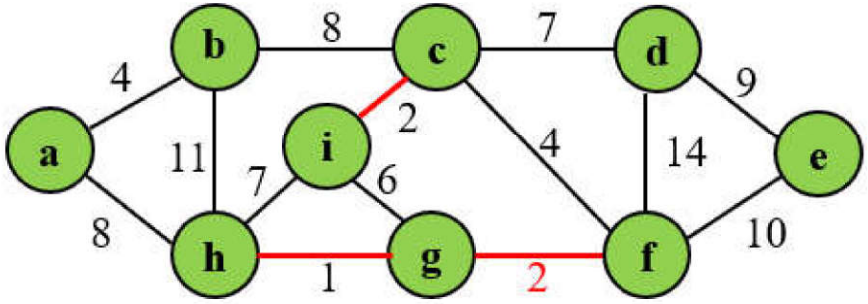
Minimum Spanning Tree :

- **Kruskal's algorithm**
  - ✓Example:



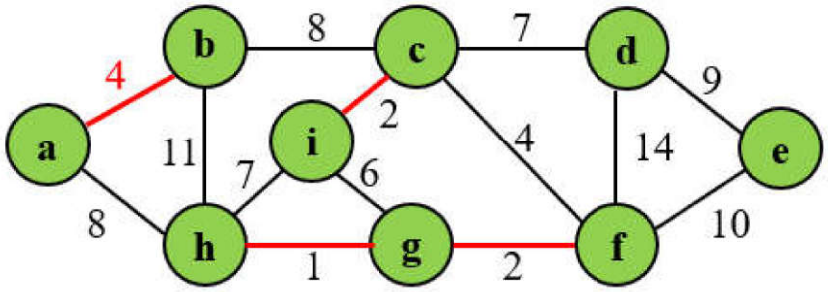
Minimum Spanning Tree :

- **Kruskal's algorithm**
  - ✓Example:



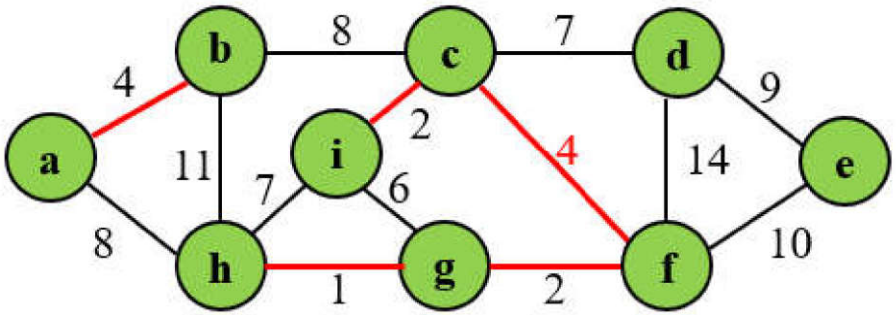
Minimum Spanning Tree :

- **Kruskal's algorithm**
  - ✓Example:



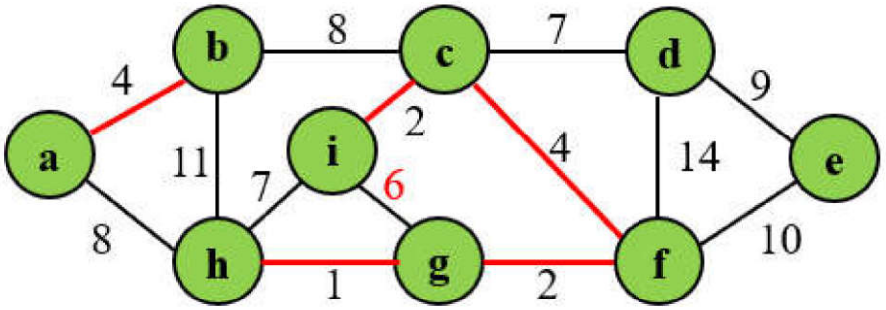
Minimum Spanning Tree :

- **Kruskal's algorithm**
  - ✓Example:



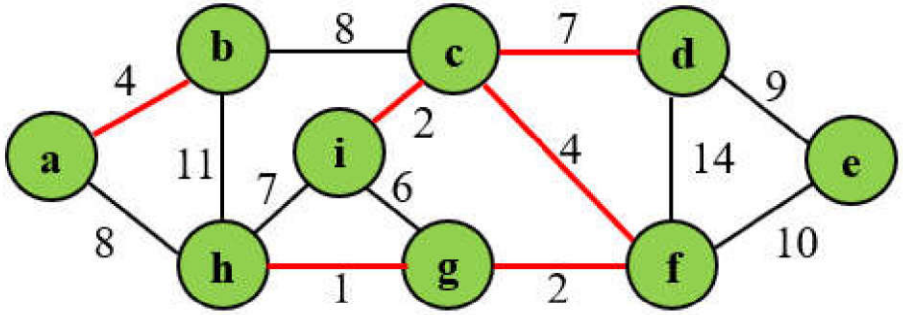
Minimum Spanning Tree :

- **Kruskal's algorithm**
  - ✓Example:



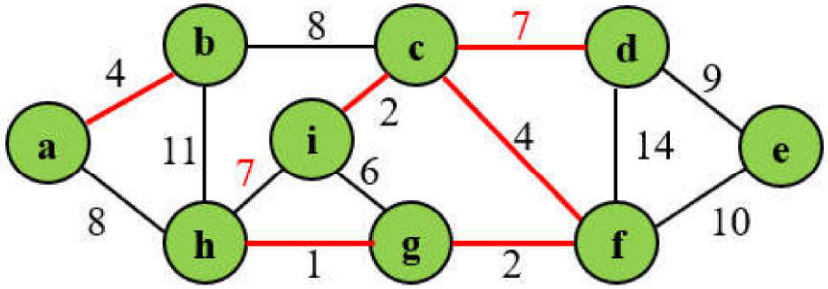
Minimum Spanning Tree :

- **Kruskal's algorithm**
  - ✓Example:



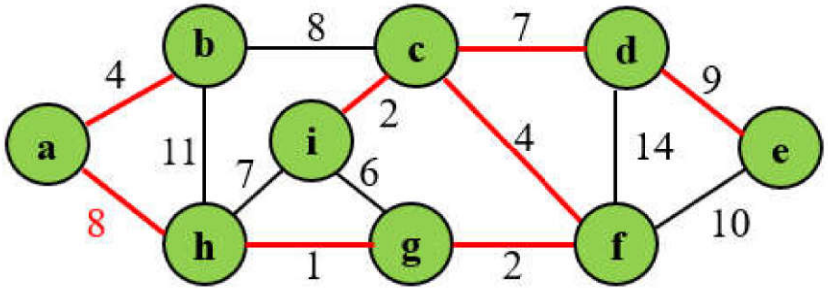
Minimum Spanning Tree :

- **Kruskal's algorithm**
  - ✓Example:



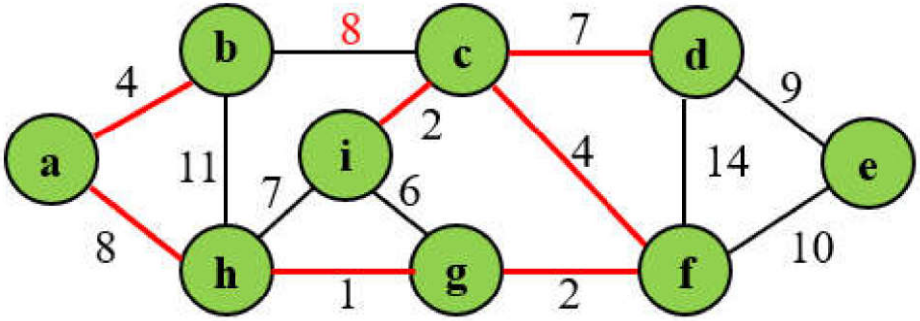
Minimum Spanning Tree :

- **Kruskal's algorithm**
  - ✓Example:



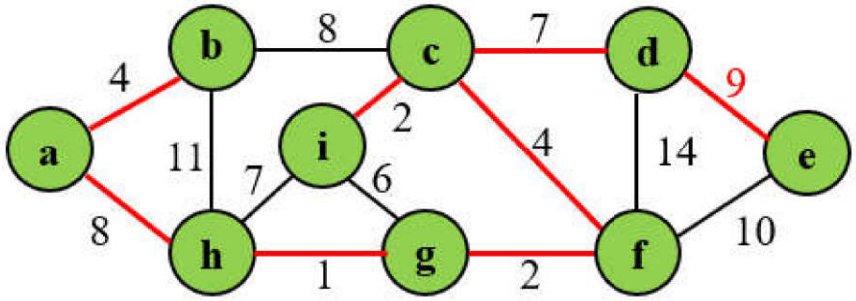
Minimum Spanning Tree :

- **Kruskal's algorithm**
  - ✓Example:



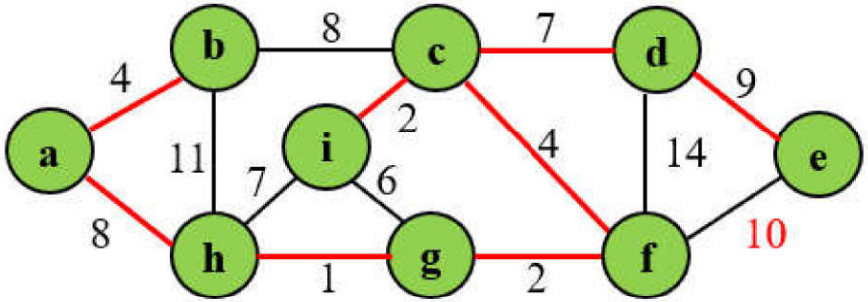
Minimum Spanning Tree :

- **Kruskal's algorithm**
  - ✓Example:



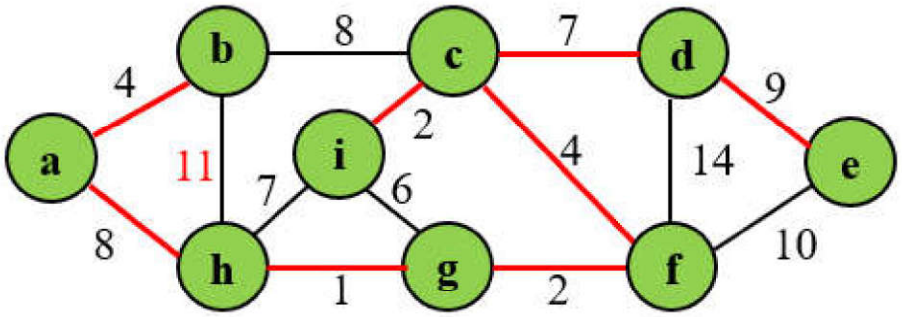
Minimum Spanning Tree :

- **Kruskal's algorithm**
  - ✓Example:



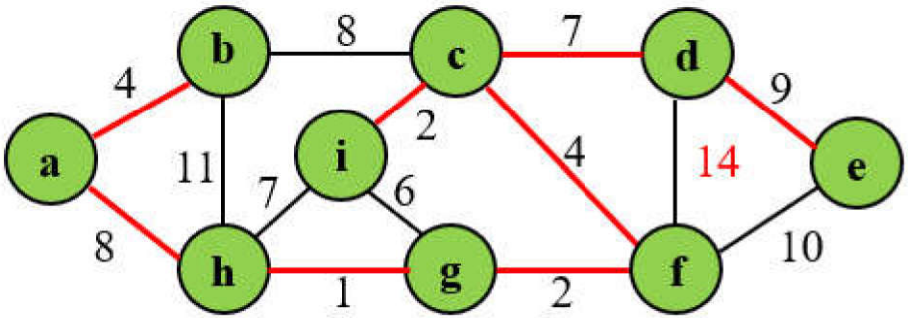
Minimum Spanning Tree :

- **Kruskal's algorithm**
  - ✓Example:



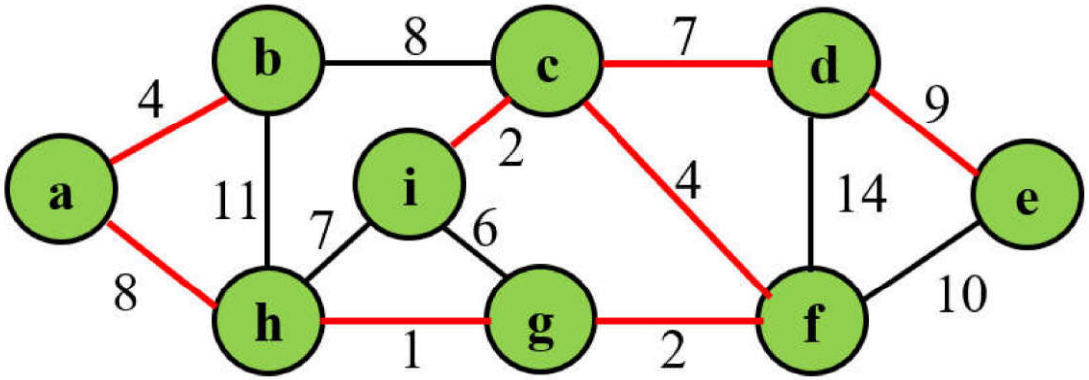
Minimum Spanning Tree :

- **Kruskal's algorithm**
  - ✓Example:



Minimum Spanning Tree :

- **Kruskal's algorithm**
  - ✓Example:



Minimum Spanning Tree :

- **Prim's algorithm**
  - ✓Operate like Dijkstra's algorithm for finding shortest paths in a graph
  - ✓The edges in the set *A* form a single tree

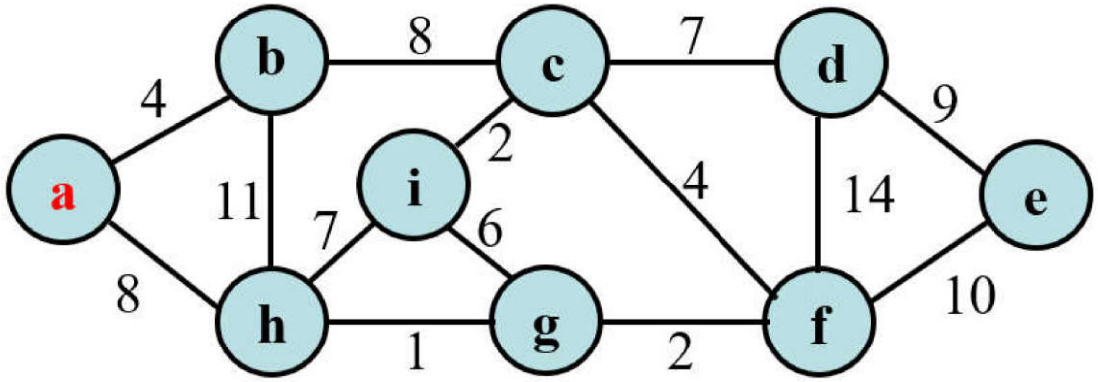
Minimum Spanning Tree :

- **Prim's algorithm**

```
MST-Prim (G, w, r)
 1  for each u ∈ G.V
 2       u.key = ∞
 3       u.π = nil
 4   r.key = 0
 5  Q = G.V
 6  while Q ≠ 0
 7      u = EXTRACT-MIN (Q)
 8      for each v ∈ G.Adj[u]
 9           if  v ∈ Q and w (u, v) < v.key
10                v.π = u
11                v.key = w (u, v)
```

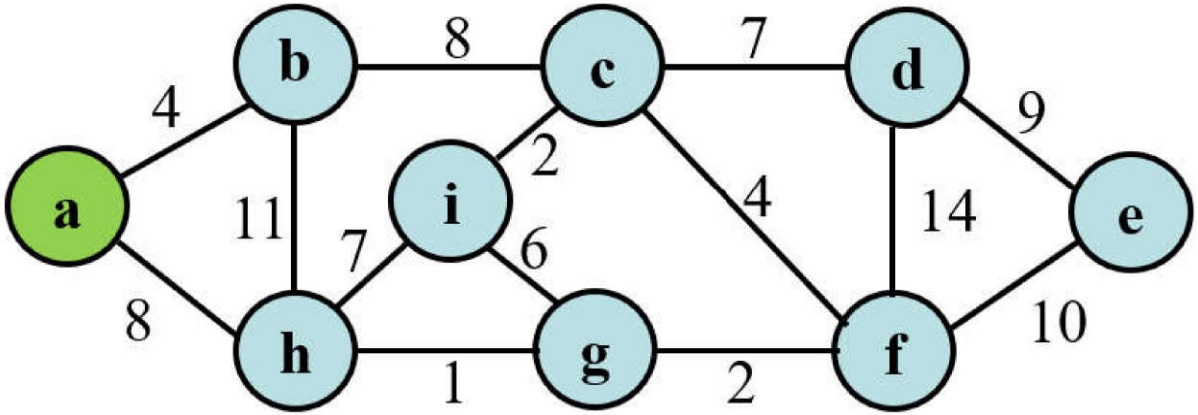
Minimum Spanning Tree :

- **Prim's algorithm**
  - ✓Example:



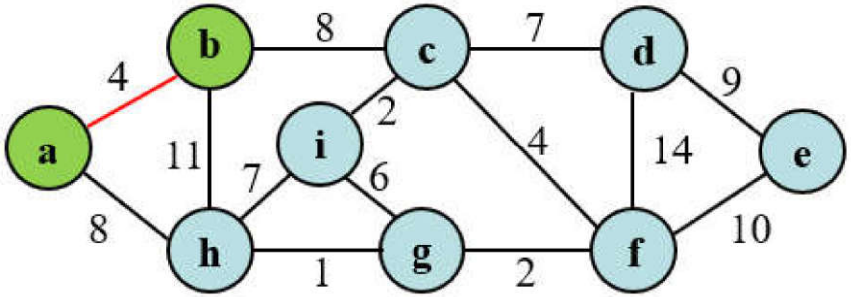
Minimum Spanning Tree :

- **Prim's algorithm**
  - ✓Example:



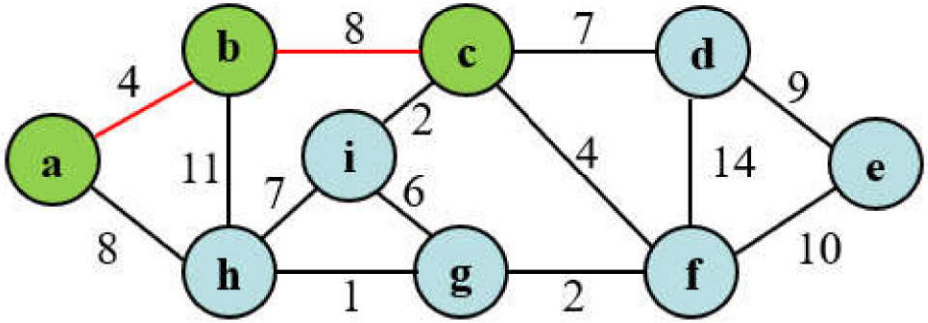
Minimum Spanning Tree :

- **Prim's algorithm**
  - ✓Example:



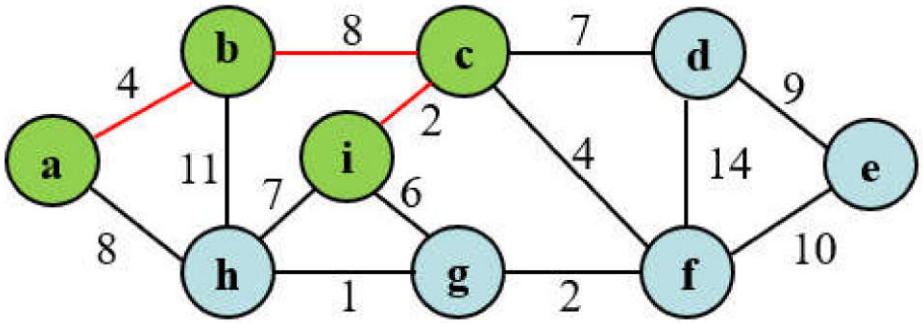
Minimum Spanning Tree :

- **Prim's algorithm**
  - ✓Example:



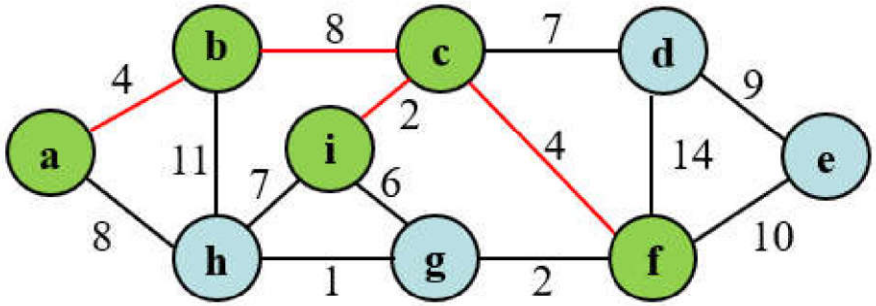
Minimum Spanning Tree :

- **Prim's algorithm**
  - ✓Example:



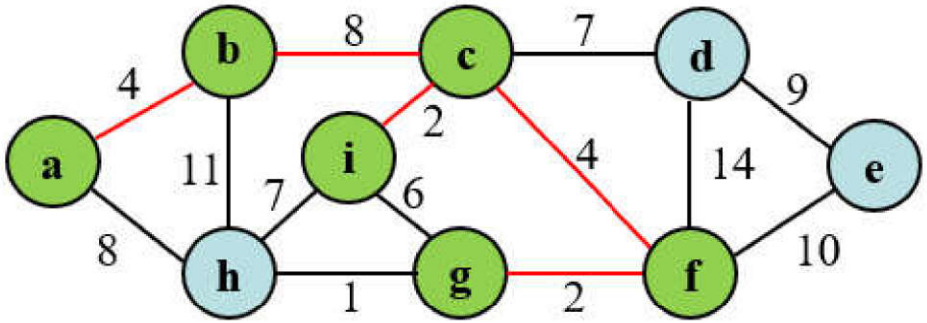
Minimum Spanning Tree :

- **Prim's algorithm**
  - ✓Example:



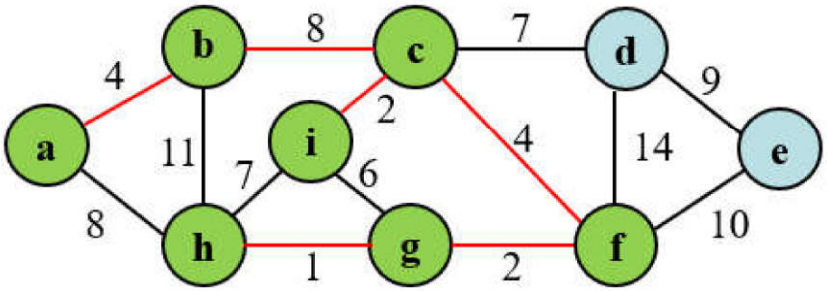
Minimum Spanning Tree :

- **Prim's algorithm**
  - ✓Example:



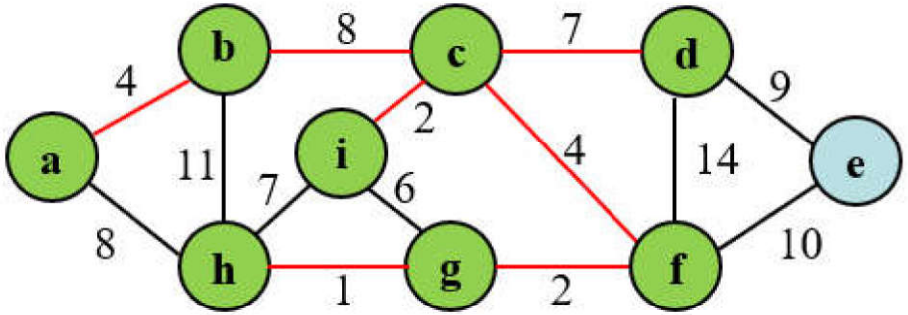
Minimum Spanning Tree :

- **Prim's algorithm**
  - ✓Example:



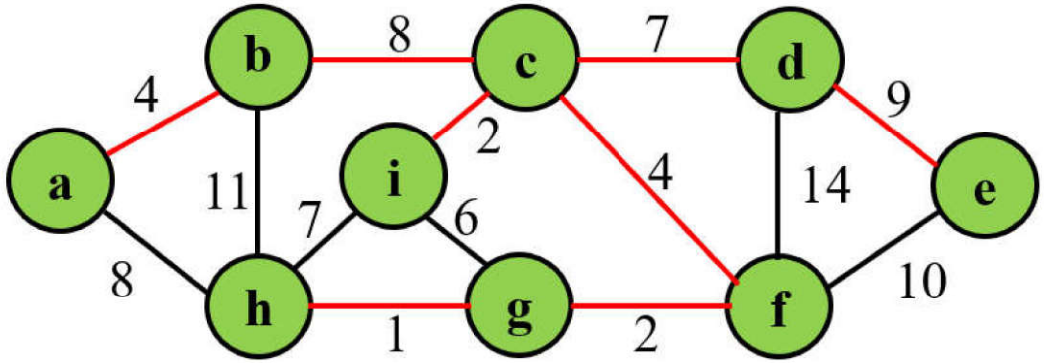
Minimum Spanning Tree :

- **Prim's algorithm**
  - ✓Example:



Minimum Spanning Tree :

- **Prim's algorithm**
  - ✓Example:



Single-source shortest paths :

- Given a weighted, directed graph $G = (V, E)$
- weight $w(p)$ of path $p = \langle v_0, v_1, \ldots, v_k \rangle$ is the sum of the weights of its constituent edges:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

- *Shortest-path weight* $\delta(u, v)$ from u to v by

$$\delta(u, v) = \begin{cases} \min\{w(p) : u \overset{p}{\rightsquigarrow} v\} & \text{if there is a path from } u \text{ to } v, \\ \infty & \text{otherwise} \end{cases}$$

- *Shortest path* from vertex u to vertex v is defined as any path p with weight $w(p) = \delta(u, v)$

Single-source shortest paths :

- **Dijkstra's algorithm**

Dijkstra (*G*, *w*, *s*)

1. Initialize-Single-Source (*G*, *s*)
2. S = ∅
3. $Q = G.V$
4. while $Q \neq \emptyset$
5. u = EXTRACT-MIN (Q)
6. $S = S \cup \{u\}$
7. for each vertex $v \in G.\text{Adj}[u]$
8. Relax (u, v, w)

Single-source shortest paths :

- **Dijkstra's algorithm**

Initialize-Single-Source (*G*, *s*)

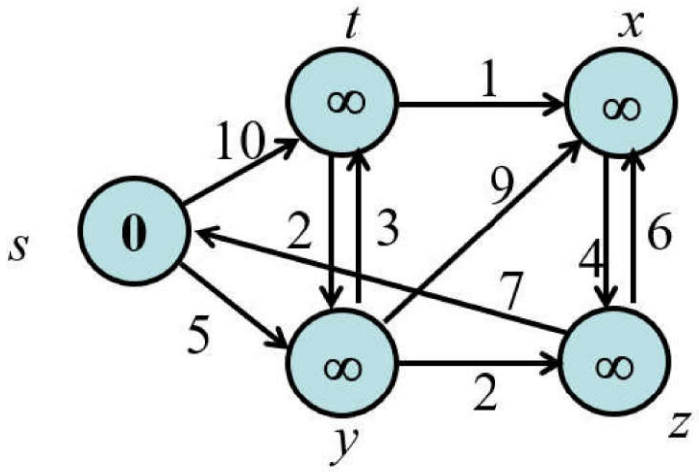
```
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = nil
4  s.d = 0
```

Relax (*u*, *v*, *w*)

```
1  if v.d > u.d + w (u, v)
2      v.d = u.d + w (u, v)
3      v.π = u
```

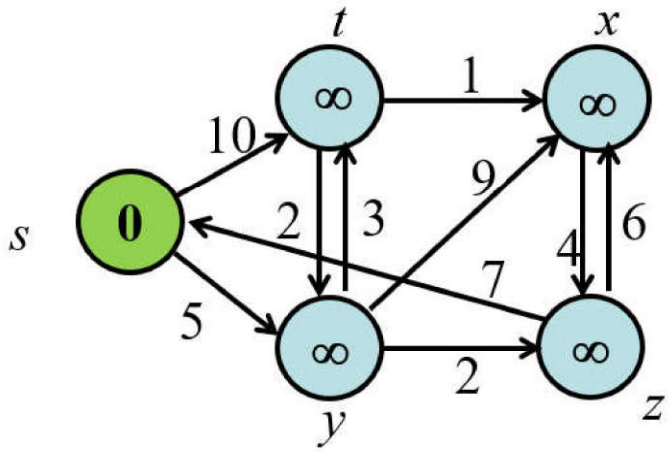
Single-source shortest paths :

- **Example:**



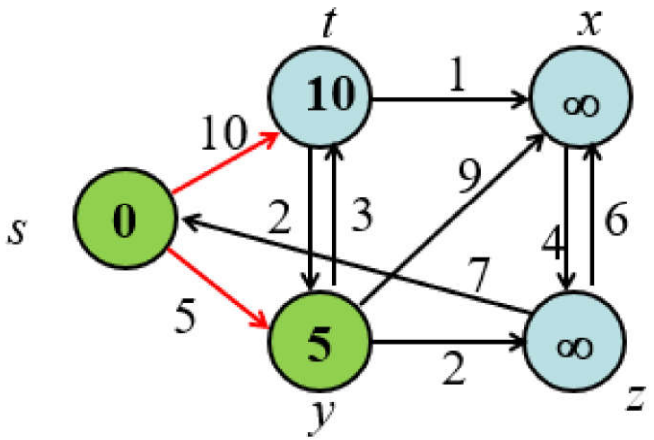
Single-source shortest paths :

- **Example:**



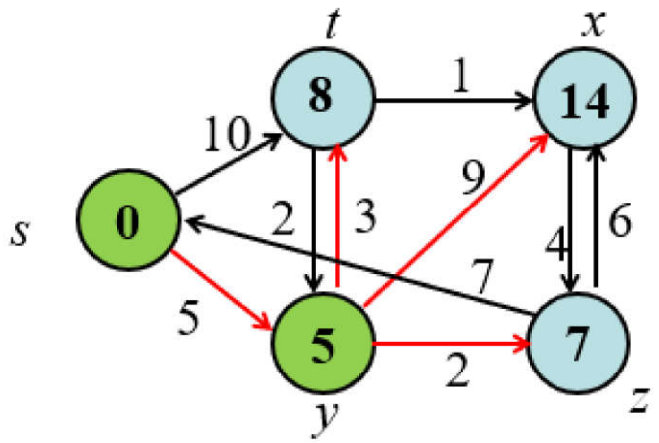
Single-source shortest paths :

- **Example:**



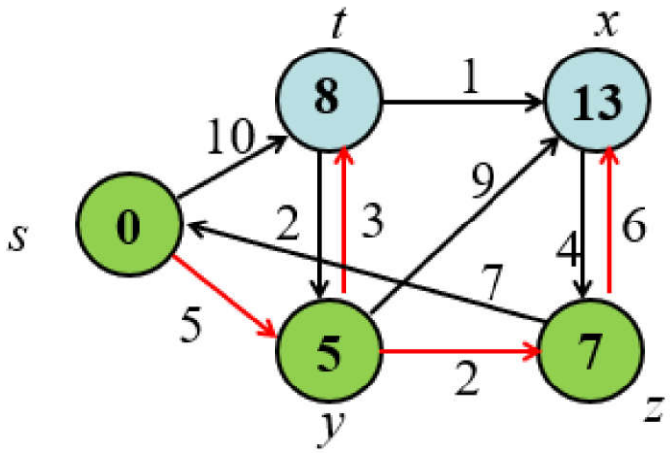
Single-source shortest paths :

- **Example:**



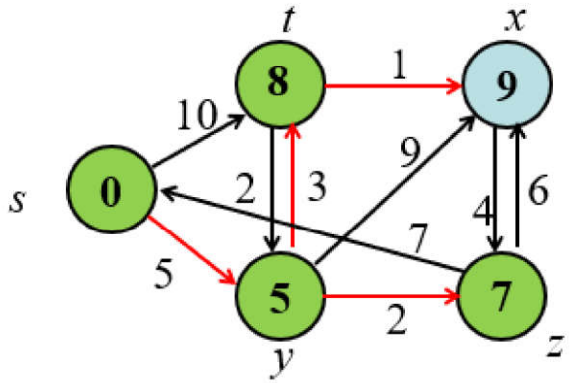
Single-source shortest paths :

- **Example:**



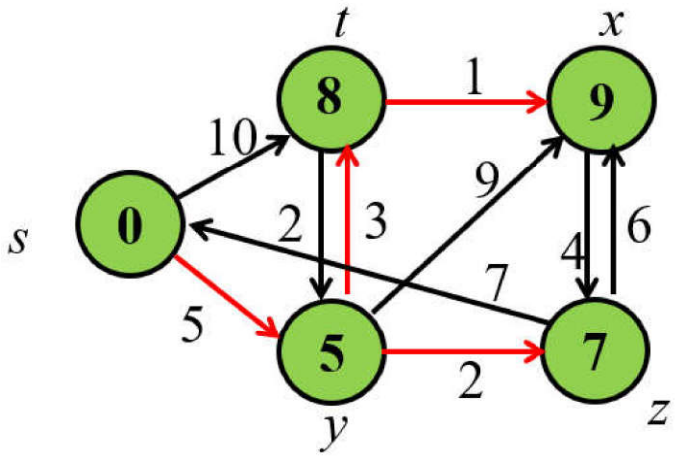
Single-source shortest paths :

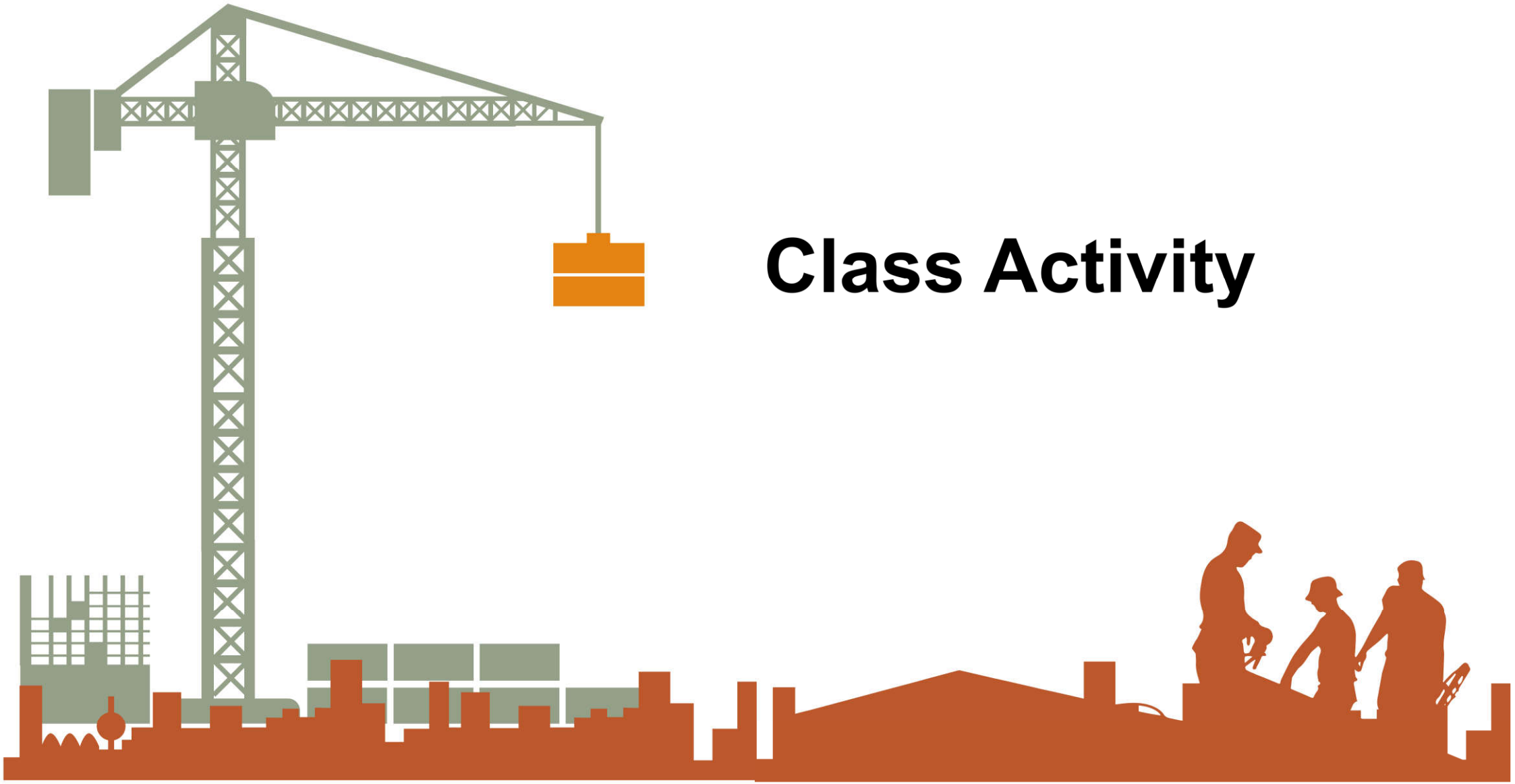
- **Example:**



Single-source shortest paths :

- **Example:**





Class Activity

Question & Answer

Formative Assessment