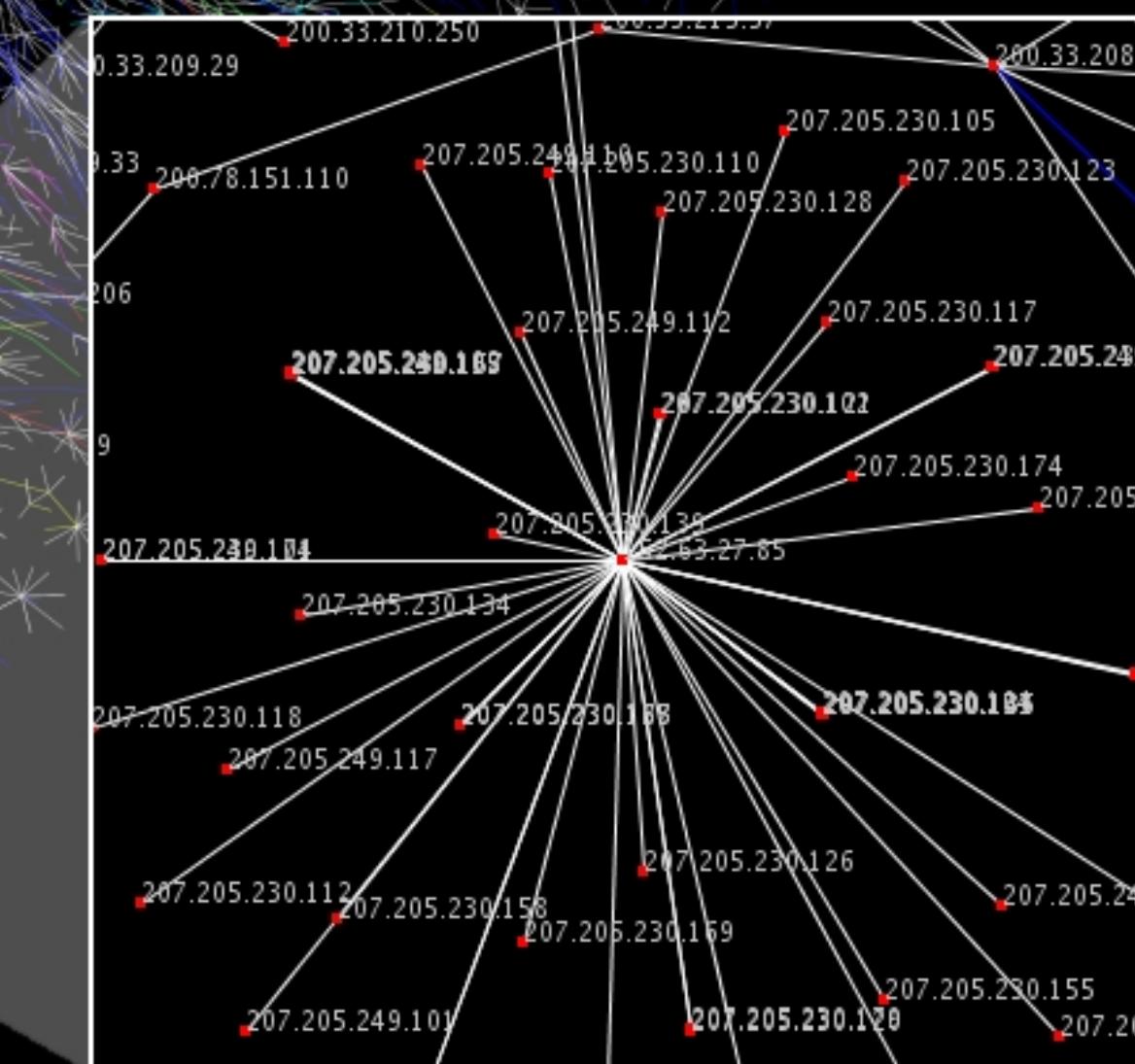
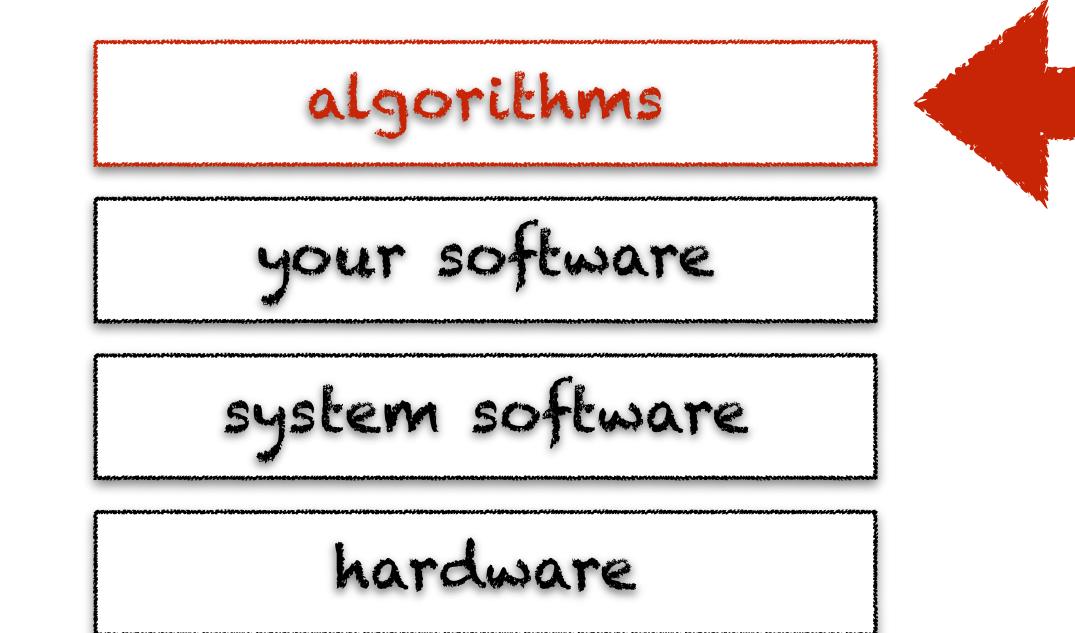


# graph algorithm



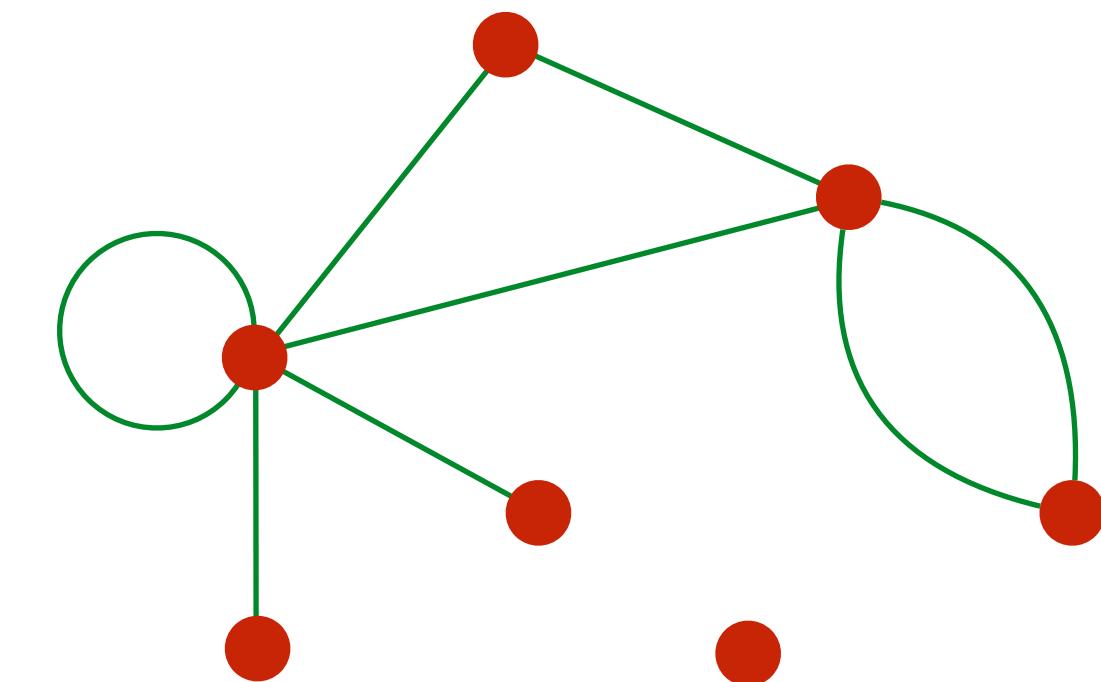
# learning objectives



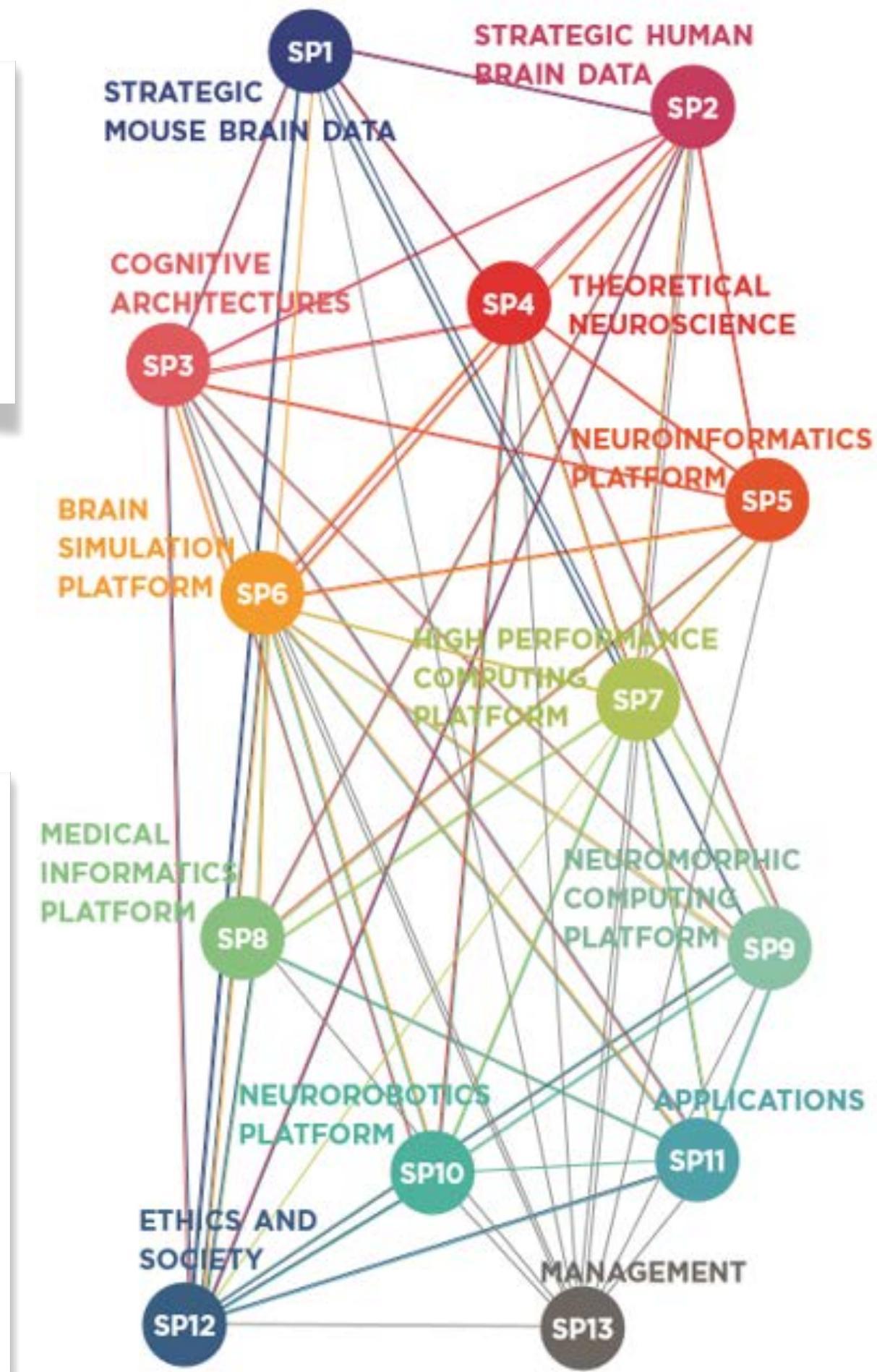
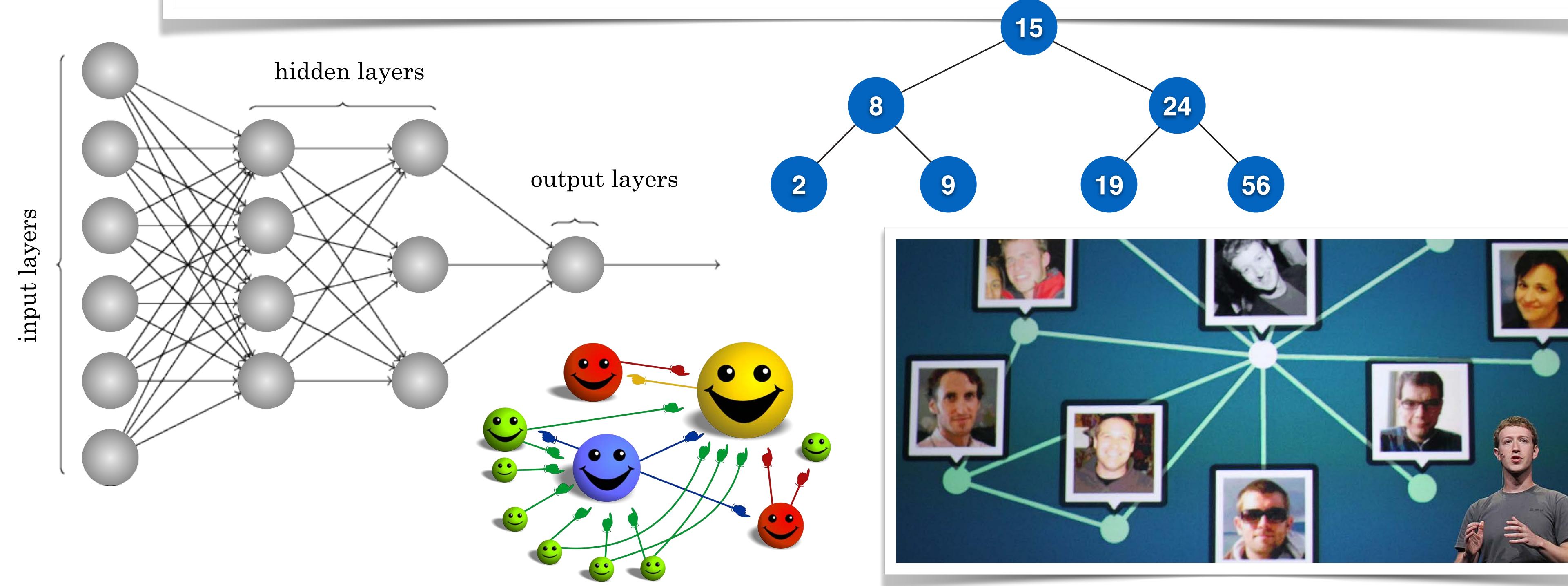
- learn what graphs are in mathematical terms
- learn how to represent graphs in computers
- learn about typical graph algorithms

# why graphs?

intuitively, a graph is formed by vertices and edges between vertices



graphs are used in numerous fields to model relationships (edges) between elements (vertices)



# what's a graph?

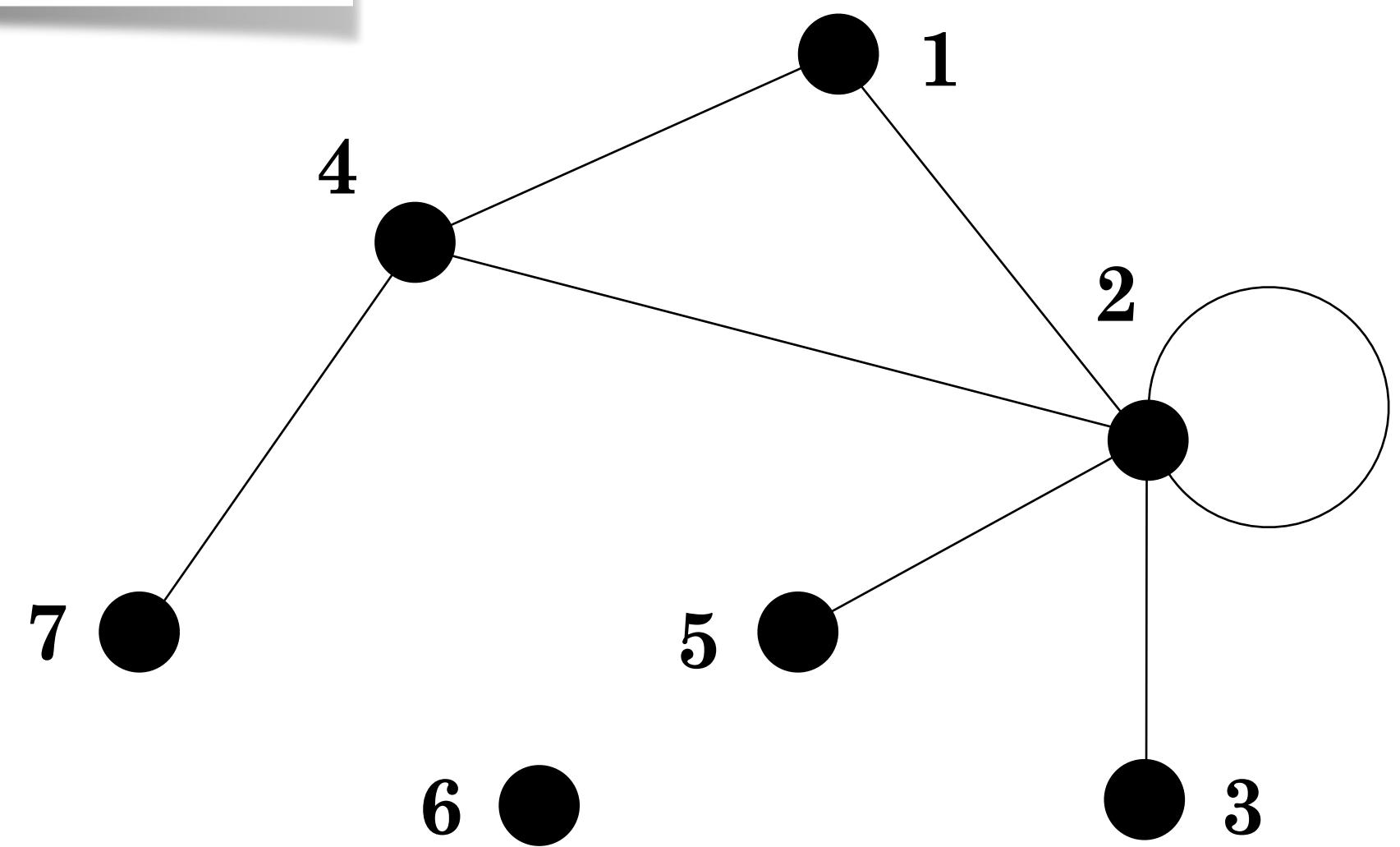
formally, a graph is a tuple  $G = (V, E)$  of sets,  
where  $V$  is a set of vertices (or nodes or points)  
and  $E$  is a set of edges such that:

$$E \subseteq V \times V$$

example:

$$V = \{1, 2, 3, 4, 5, 6, 7\}$$

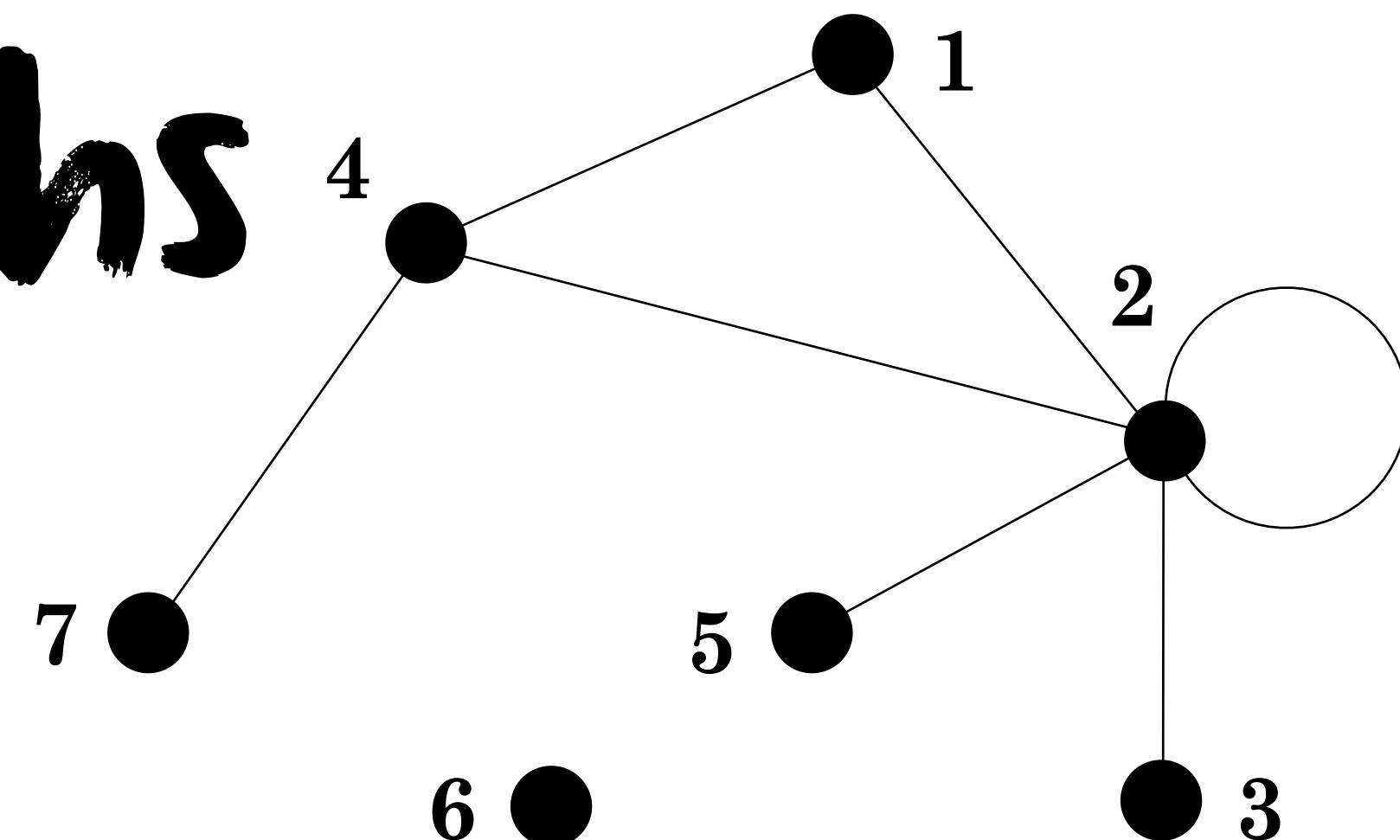
$$E = \{\{1, 2\}, \{1, 4\}, \{2\}, \{2, 3\}, \{2, 4\}, \{2, 5\}, \{4, 7\}\}$$



**undirected:**

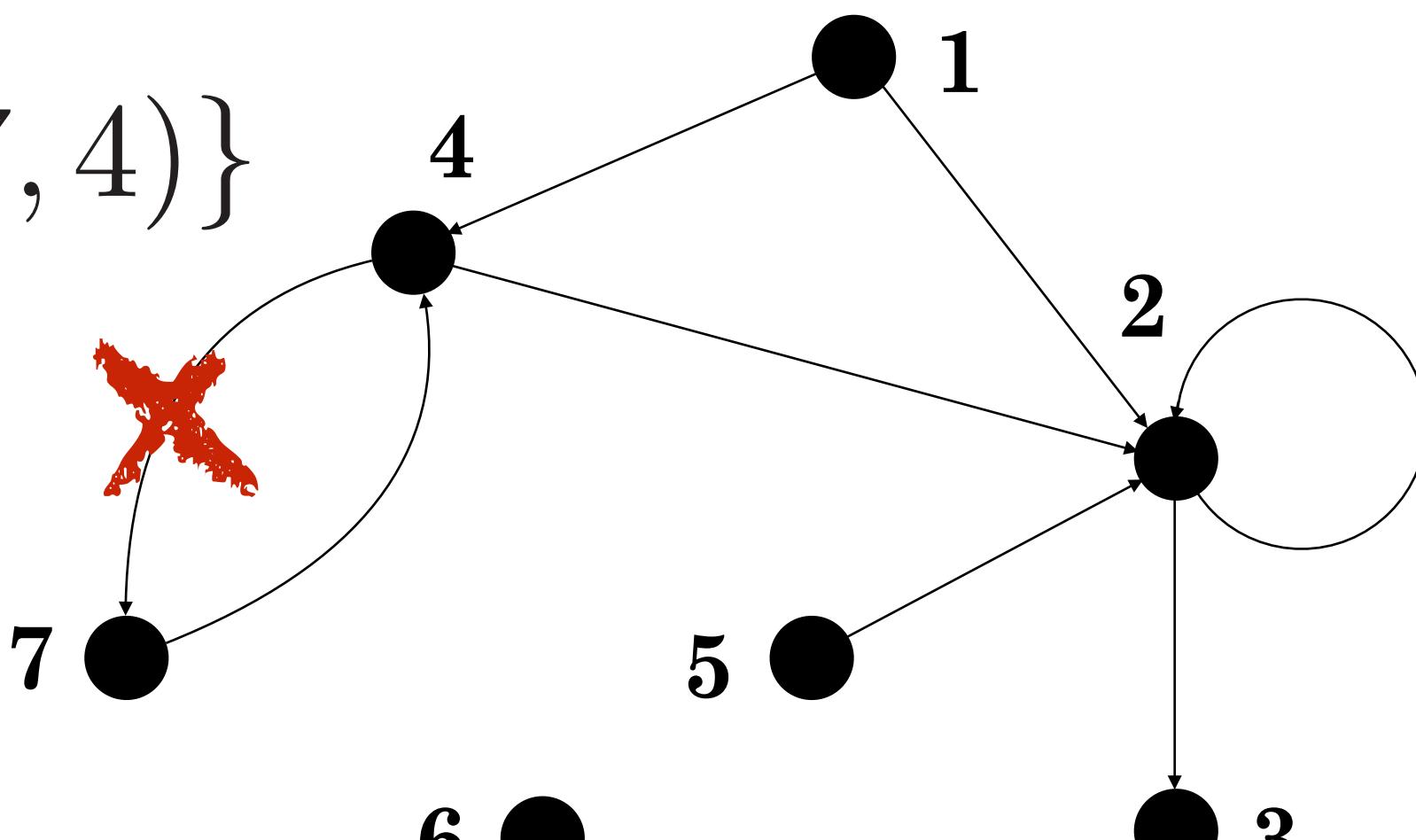
$$V = \{1, 2, 3, 4, 5, 6, 7\}$$

$$E = \{\{1, 2\}, \{1, 4\}, \{2\}, \{2, 3\}, \{2, 4\}, \{2, 5\}, \{4, 7\}\}$$



**directed:**

$$E = \{(1, 2), (1, 4), (2, 2), (2, 3), (4, 2), (4, 7), (5, 2), (7, 4)\}$$



**oriented:**

$$E = \{(1, 2), (1, 4), (2, 2), (2, 3), (4, 2), (4, 7), (5, 2), (7, 4)\}$$

# notations & metrics

let  $G$  be graph,  $G.V$  denotes its set of vertices and  $G.E$  its set of edges

the edge between vertices  $x$  and  $y$  is noted  $\{x,y\}$ ,  $(x,y)$  or simply  $xy$

the order of  $G$ , written  $|G|$ , is the number of its vertices, whereas  $\|G\|$  denotes its number of edges

graph  $G$  is sparse if  $\|G\| \ll |G|^2$  and it is dense if  $\|G\| \approx |G|^2$

two vertices  $x$  and  $y$  are adjacent or neighbors if  $xy \in G$

if all the vertices of  $G$  are pairwise adjacent, then  $G$  is complete

# notations & metrics

a path from vertex  $x$  to vertex  $y$  is a sequence  $\langle v_0, v_1, \dots, v_k \rangle$  of vertices  $v_i \in V$  where  $x = v_0$  and  $y = v_k$ , such that  $\forall i \in \{1, \dots, k\} : (v_{i-1}, v_i) \in E$

a graph is connected if every pair of vertices is connected via a path

a path  $\langle v_0, v_1, \dots, v_k \rangle$  is a cycle if vertices  $v_0 = v_k$

we can store attributes in vertices and edges using the dotted notation,  
e.g.,  $v.\text{color}$  stores a color attribute in vertex  $v$ , while  $e.\text{weight}$  and  
 $(x,y).\text{weight}$  store a weight attribute in edge  $e$  and edge  $(x,y)$  respectively

# notations & metrics

let  $G = (V, E)$  and  $G' = (V', E')$  be two graphs, if  $V' \subseteq V$  and  $E' \subseteq E$ ,  
then  $G'$  is a **subgraph** of  $G$ , which we write  $G' \subseteq G$

let  $G = (V, E)$  and  $G' = (V', E')$  be two graphs and  $G' \subseteq G$ ,  
**if  $V' = V$ ,  $G'$  is a **spanning subgraph** of  $G$**

the **degree** (or **valency**) of a vertex  $v$  is the  
**number of neighbors** of  $v$  and is noted  $d(v)$

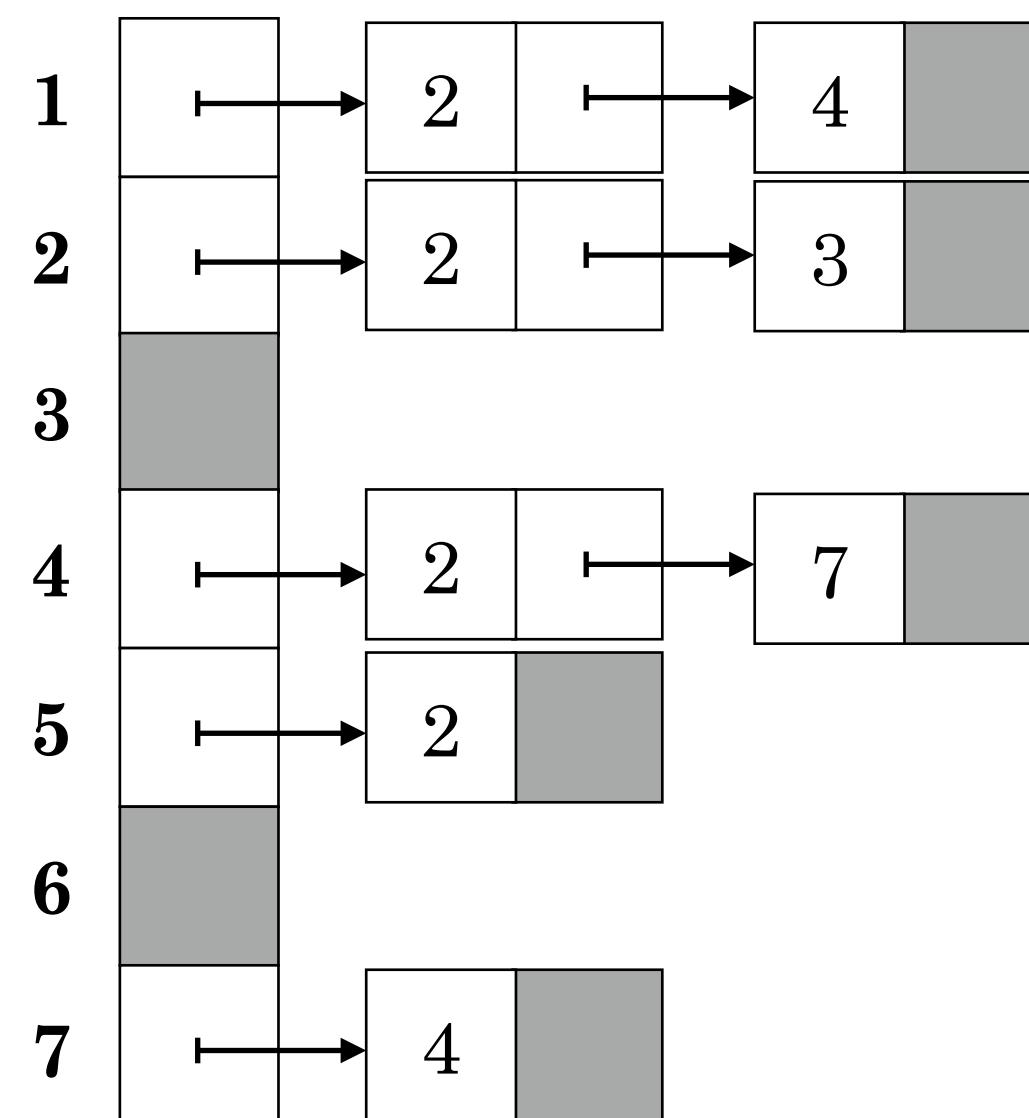
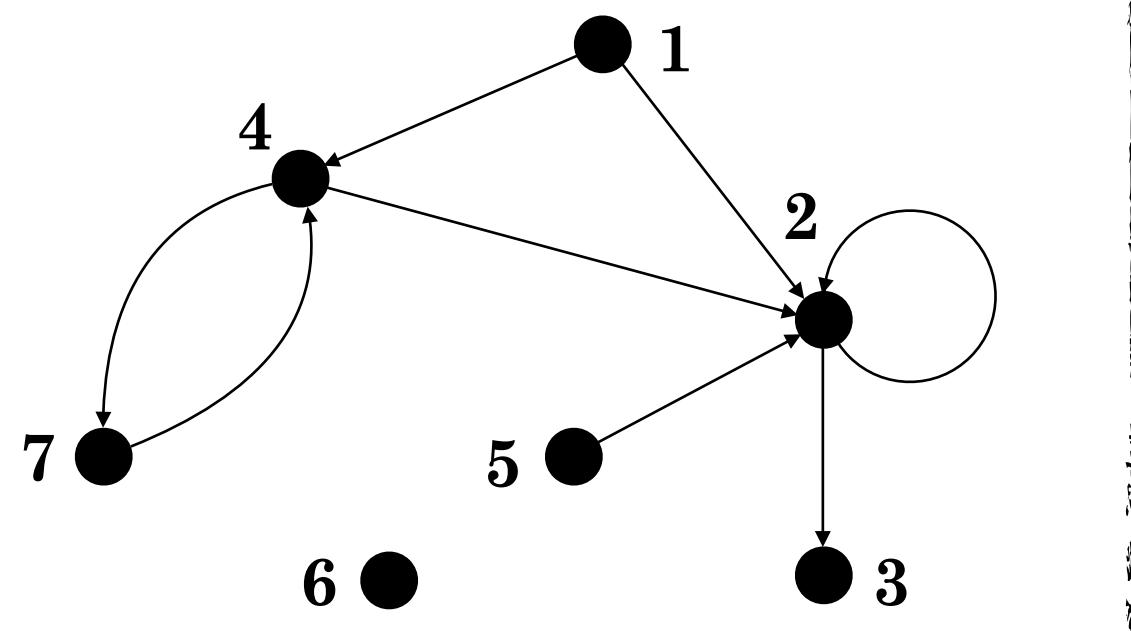
we defined  $\delta(G) = \min \{ d(v) \mid v \in V \}$   
as the **minimum degree** of  $G$

we defined  $\Delta(G) = \max \{ d(v) \mid v \in V \}$   
as the **maximum degree** of  $G$

we defined  $d(G) = \frac{1}{|V|} \sum_{v \in V} d(v)$  as the **average degree** of  $G$

# representing graphs

directed



the adjacency list consist of

- ◆ an array  $Adj$  of  $|V|$  lists, one for each vertex in  $V$
- ◆ for each  $u \in V$ , the adjacency list  $Adj[u]$  consists of all the vertices adjacent to  $u$  in  $G$
- ◆ that is, the adjacency list  $Adj[u]$  contains all the vertices  $v$  such that there is an edge  $(u, v) \in E$

the adjacency matrix consist of

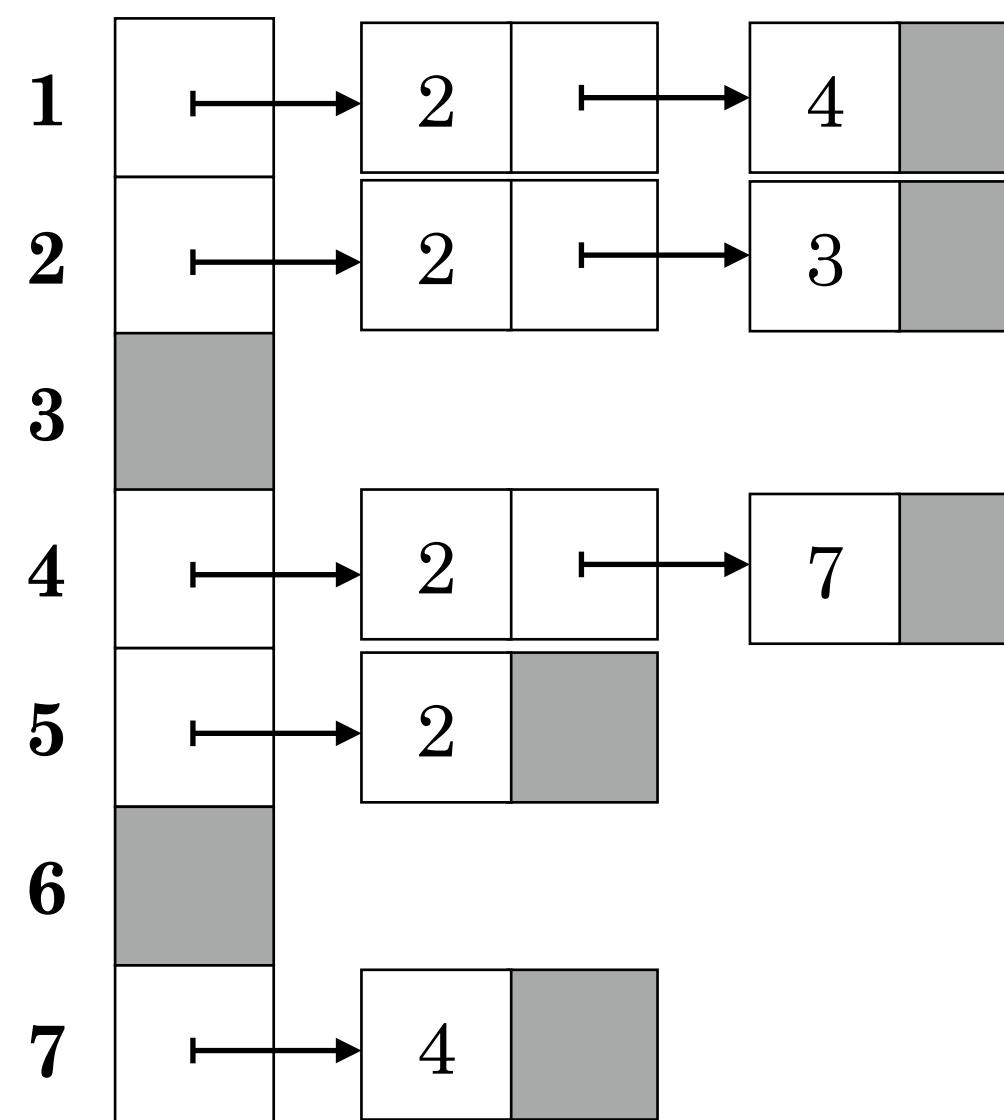
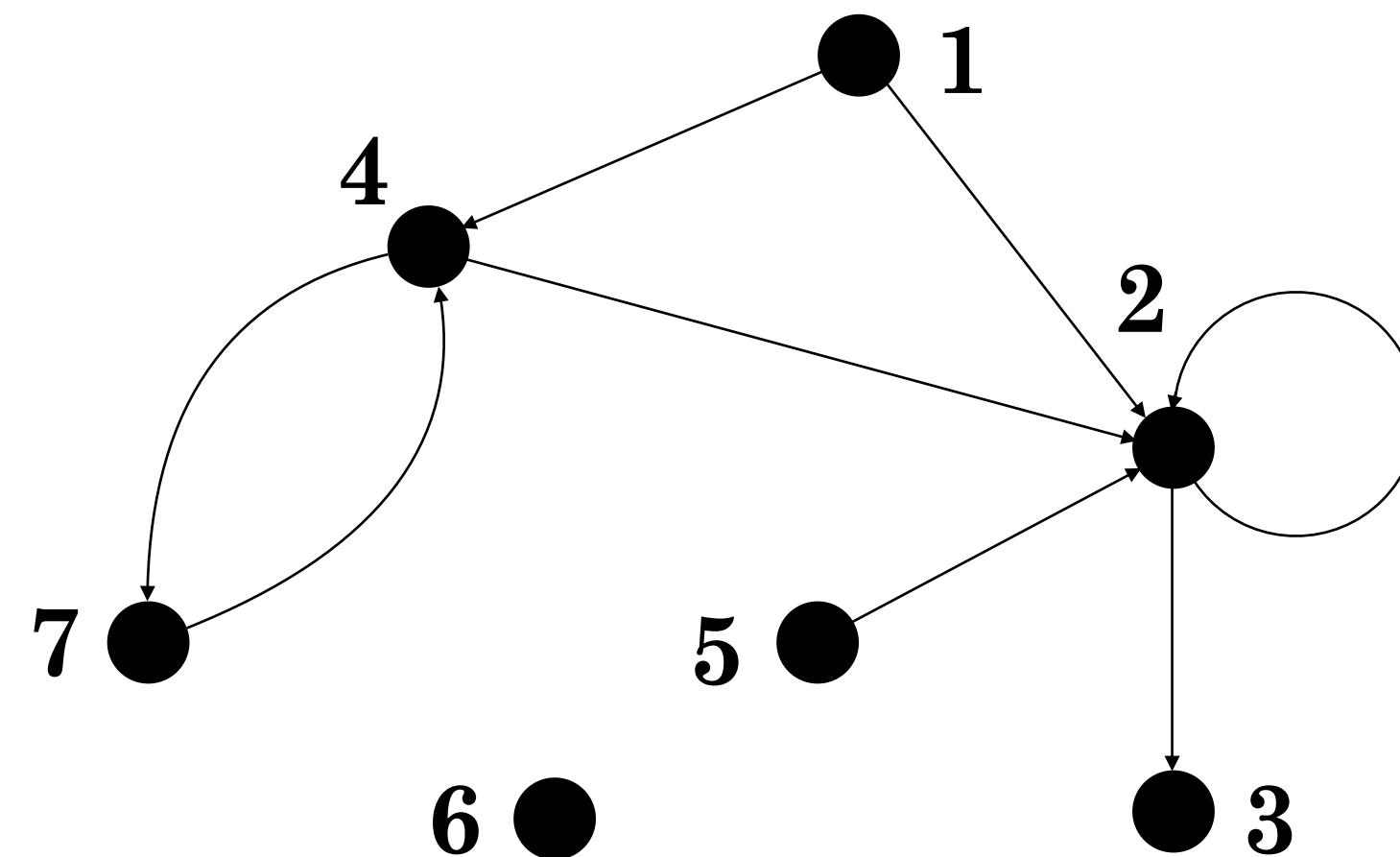
- ◆ a  $|V| \times |V|$  matrix  $A = (a_{ij})$  such that

$$a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

	1	2	3	4	5	6	7
1	0	1	0	1	0	0	0
2	0	1	1	0	0	0	0
3	0	0	0	0	0	0	0
4	0	1	0	0	0	0	1
5	0	1	0	0	0	0	0
6	0	0	0	0	0	0	0
7	0	0	0	1	0	0	0

# representing graphs

directed



adjacency list

	1	2	3	4	5	6	7
1	0	1	0	1	0	0	0
2	0	1	1	0	0	0	0
3	0	0	0	0	0	0	0
4	0	1	0	0	0	0	1
5	0	1	0	0	0	0	0
6	0	0	0	0	0	0	0
7	0	0	0	1	0	0	0

adjacency matrix

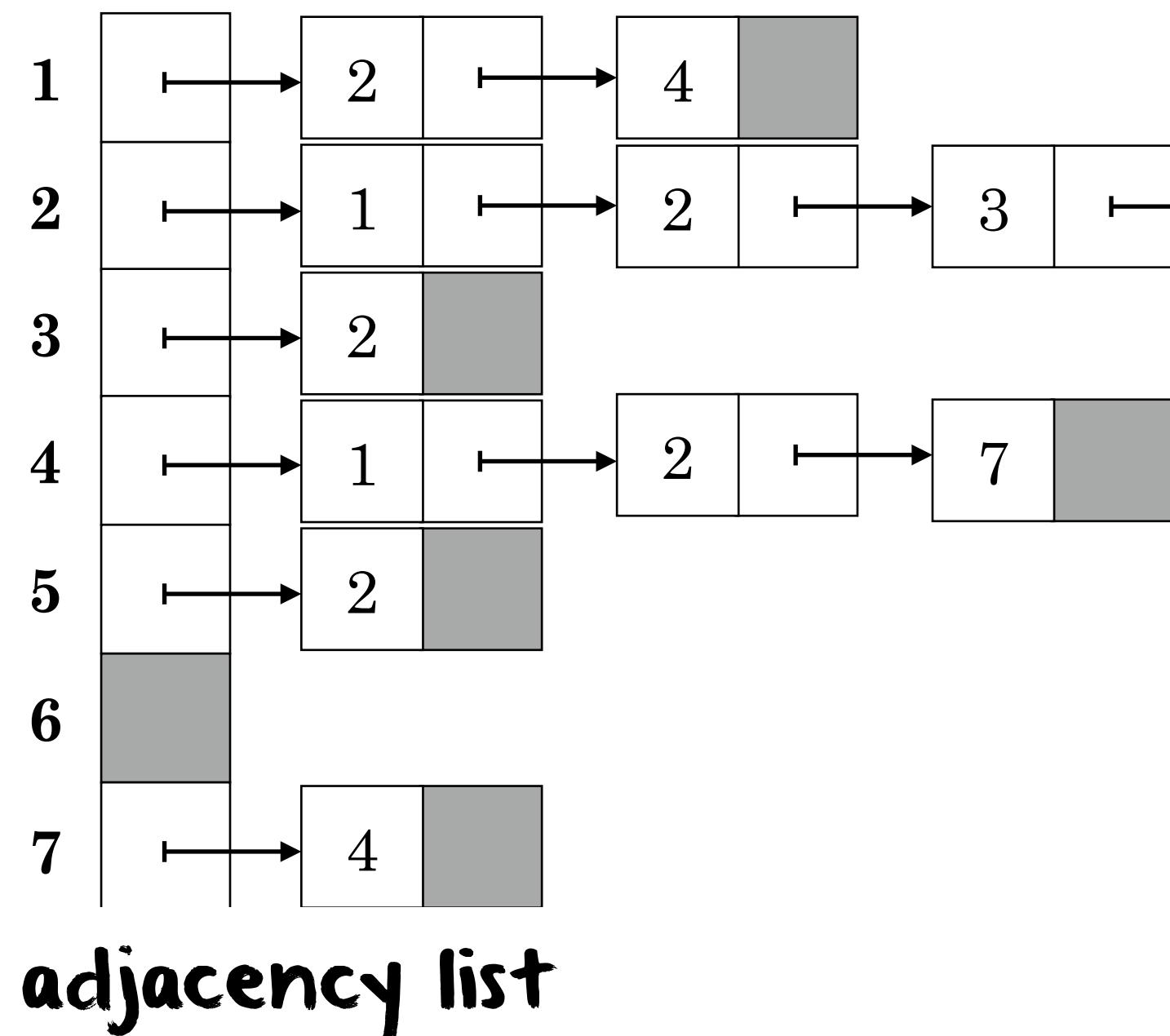
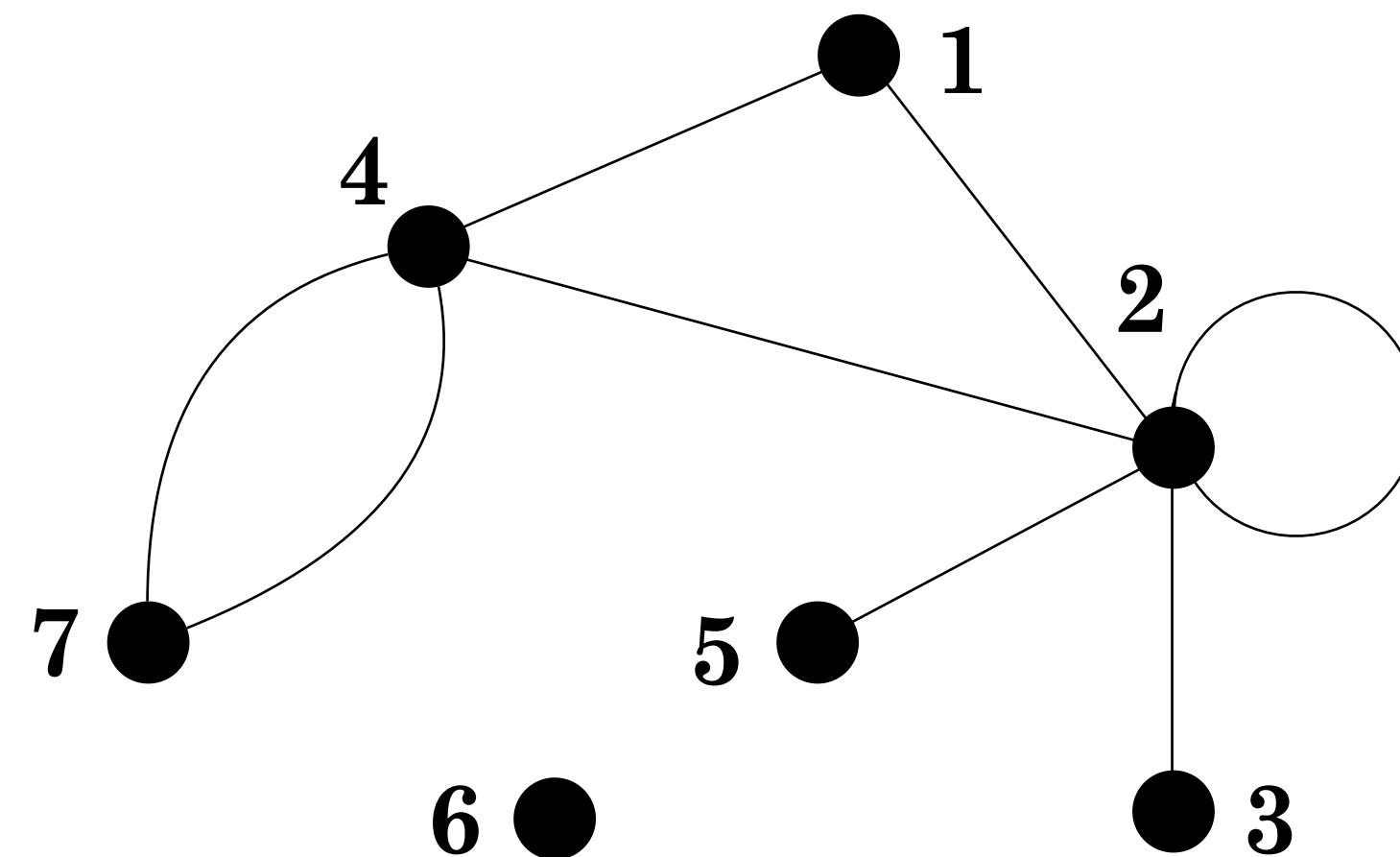
adjacency lists are best suited for representing a sparse graph

most graph algorithms rely on adjacency lists

an adjacency matrix is best suited for representing a dense graph or when the algorithm needs to know quickly if there exists an edge connecting two vertices

# representing graphs

undirected



adjacency matrix

	1	2	3	4	5	6	7
1	0	1	0	1	0	0	0
2	1	1	1	1	1	0	0
3	0	1	0	0	0	0	0
4	1	1	0	0	0	0	1
5	0	1	0	0	0	0	0
6	0	0	0	0	0	0	0
7	0	0	0	1	0	0	0

space complexity (memory footprint)

the amount of memory required by an adjacency list is  $\Theta(V + E)$

the amount of memory required by an adjacency matrix is  $\Theta(V^2)$

# typical problems

breadth-first search

minimum spanning tree

single-source shortest paths

# breadth-first search

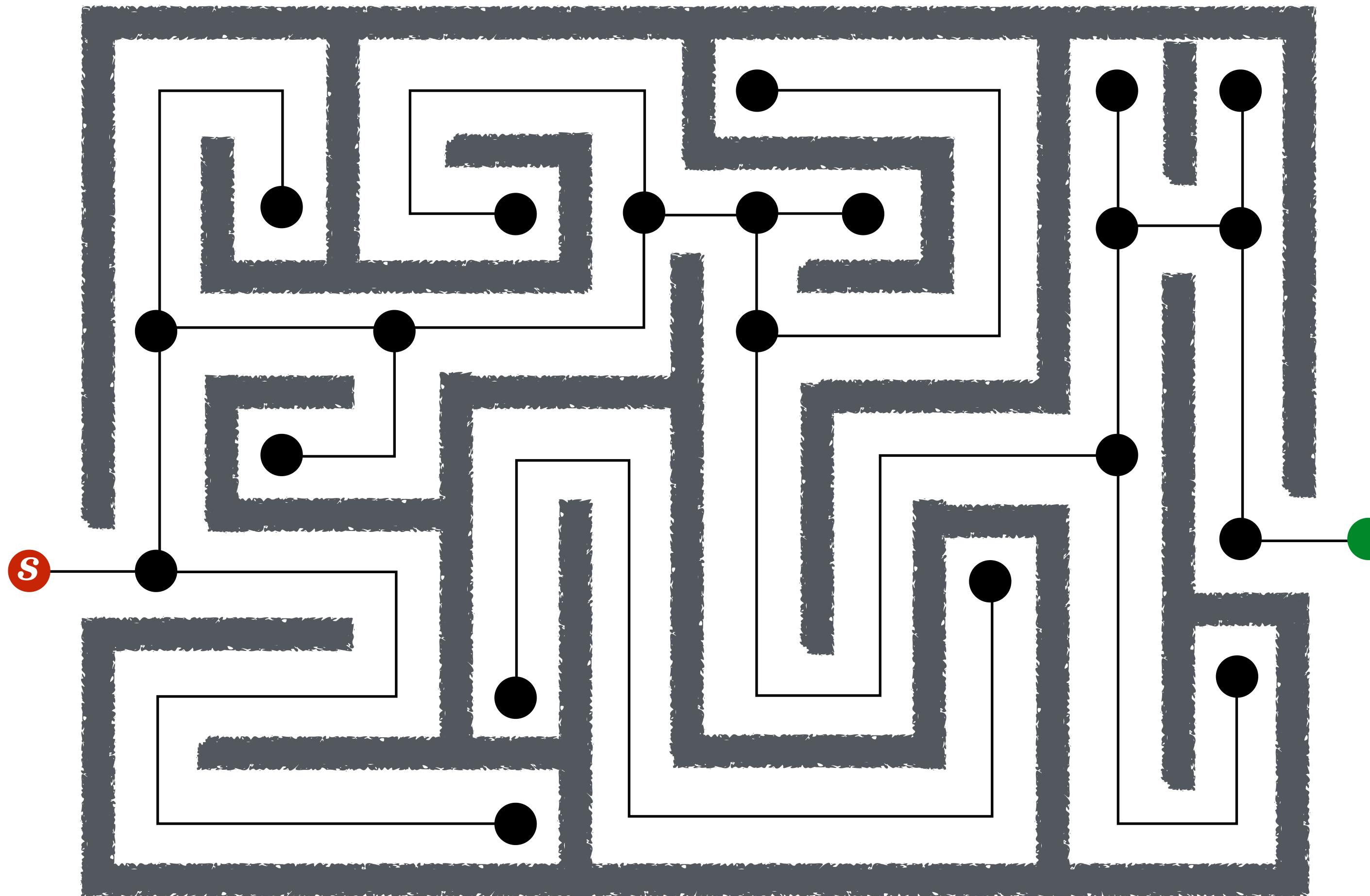
given graph  $G$  and a source vertex  $s \in G$ ,  
it discover every vertex reachable from  $s$

it computes the distance from  $s$  to every vertex  $v \in G$

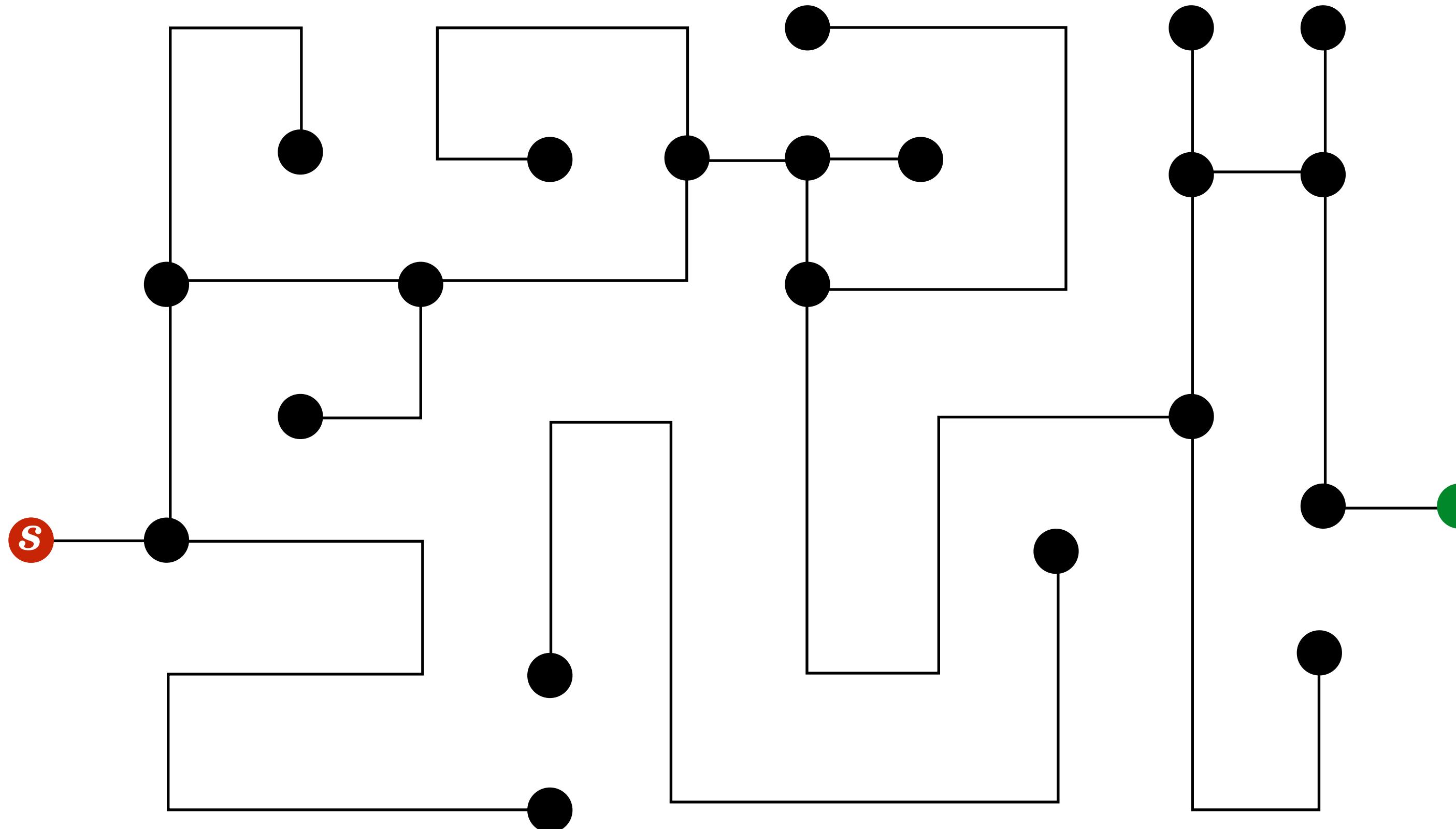
it produces a breadth-first tree rooted at  $s$   
that contains all reachable vertices from  $s$

the search is said to be breadth-first because it  
discovers all vertices at distance  $k$  from  $s$  before  
discovering any vertices at distance  $k + 1$

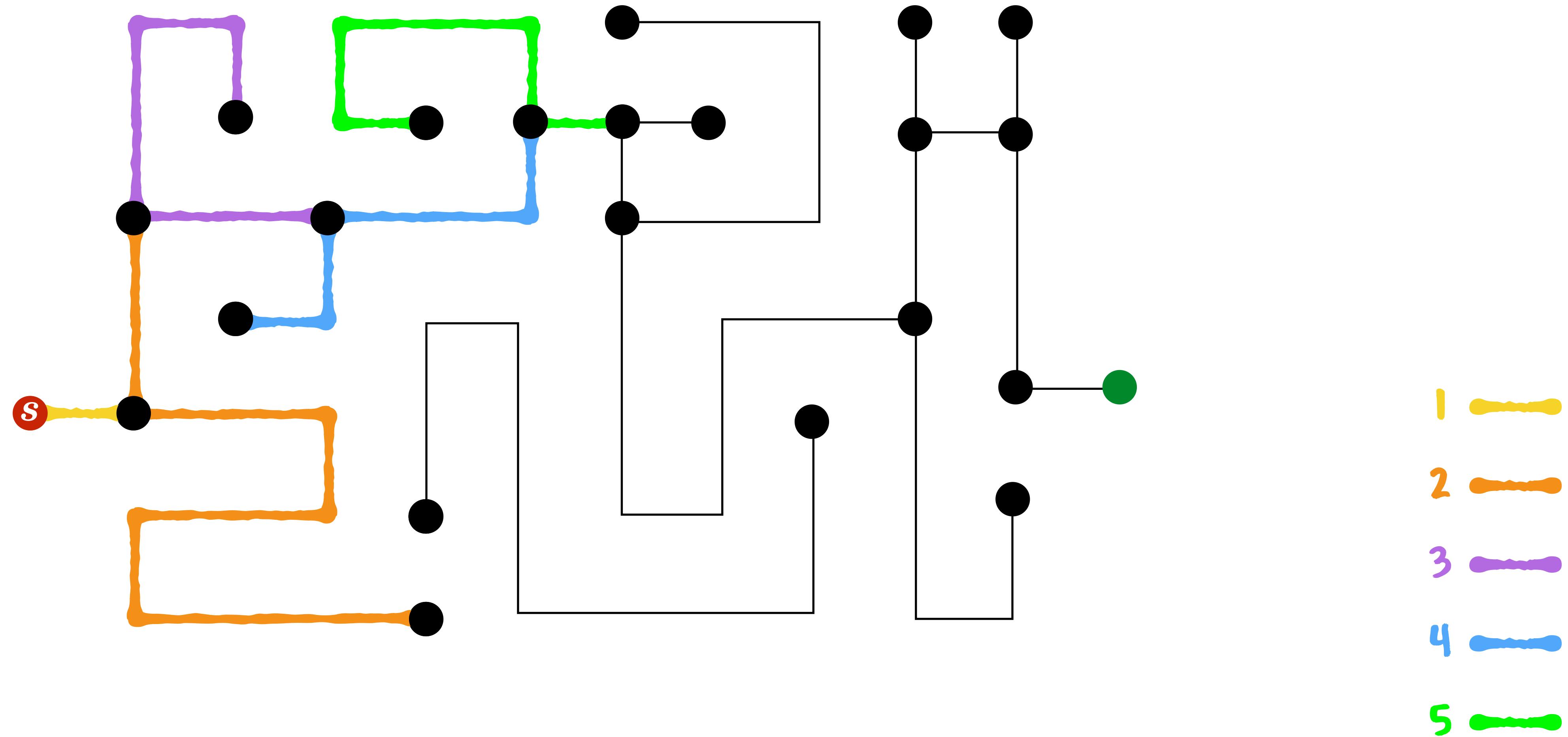
# breadth-first search



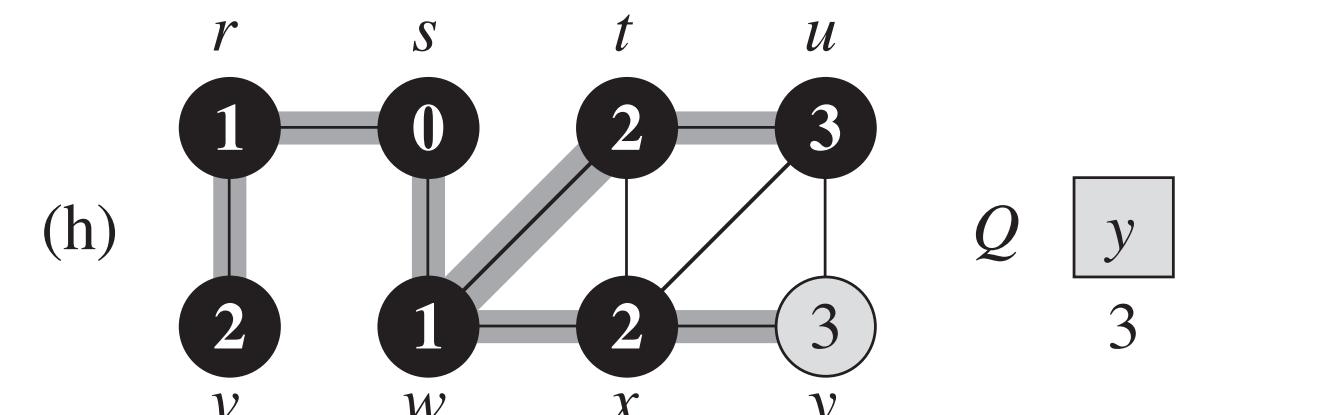
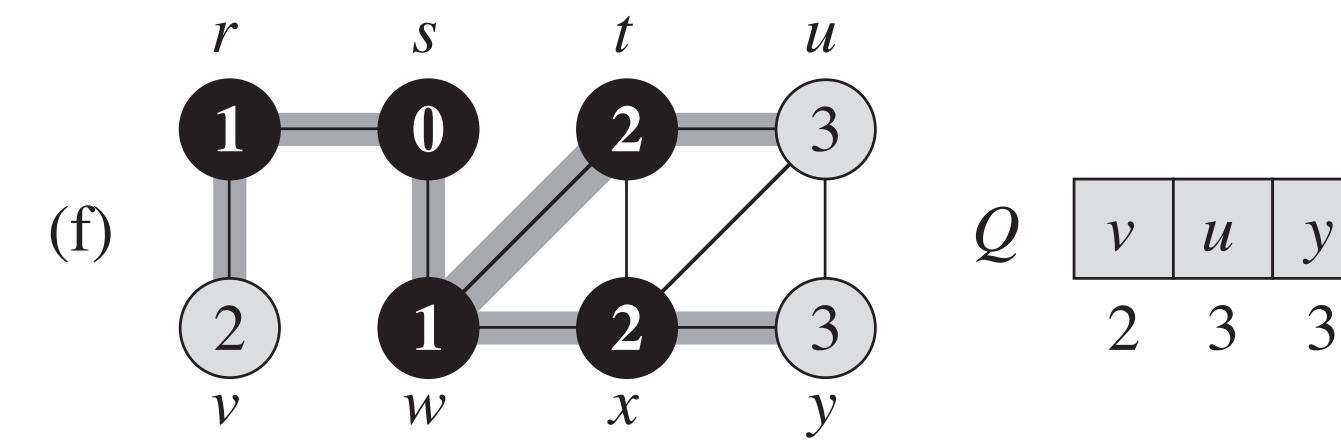
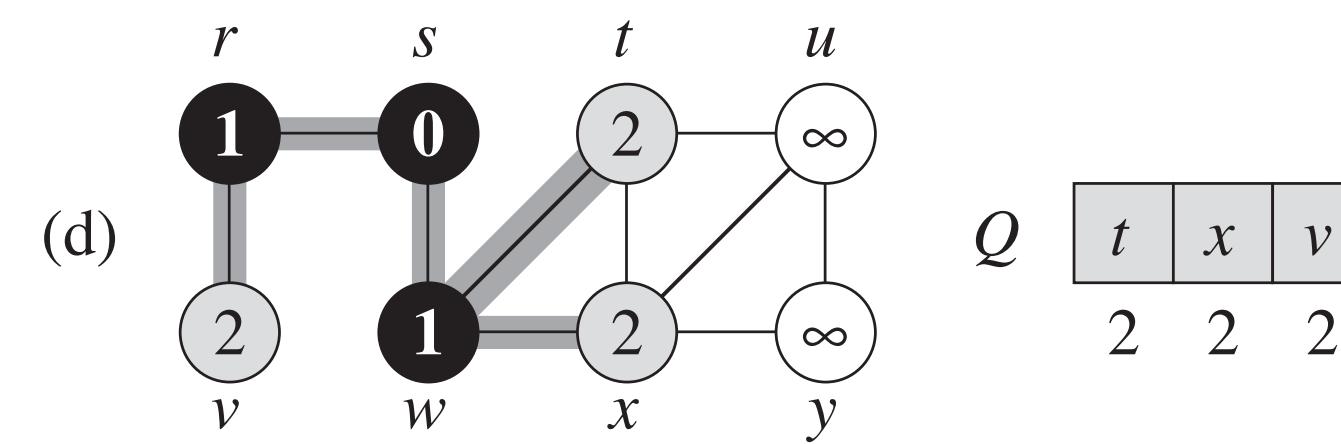
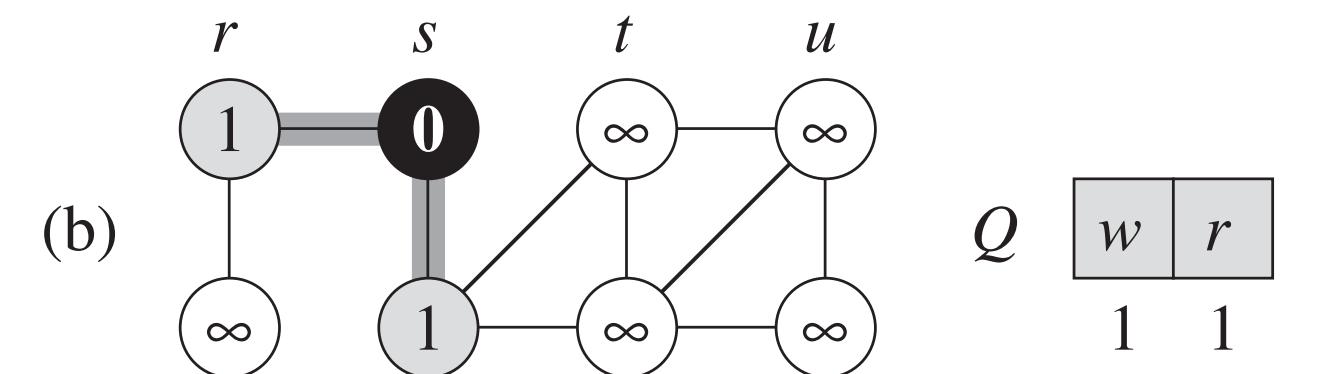
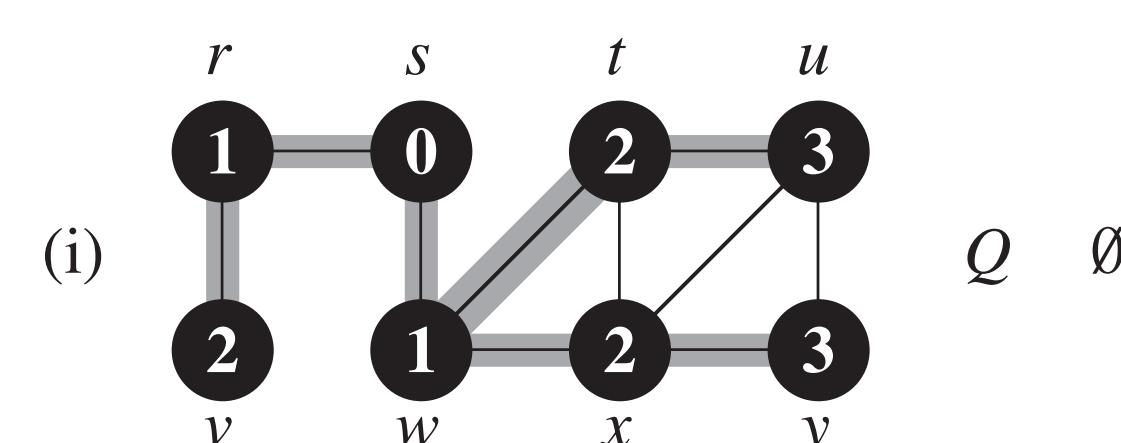
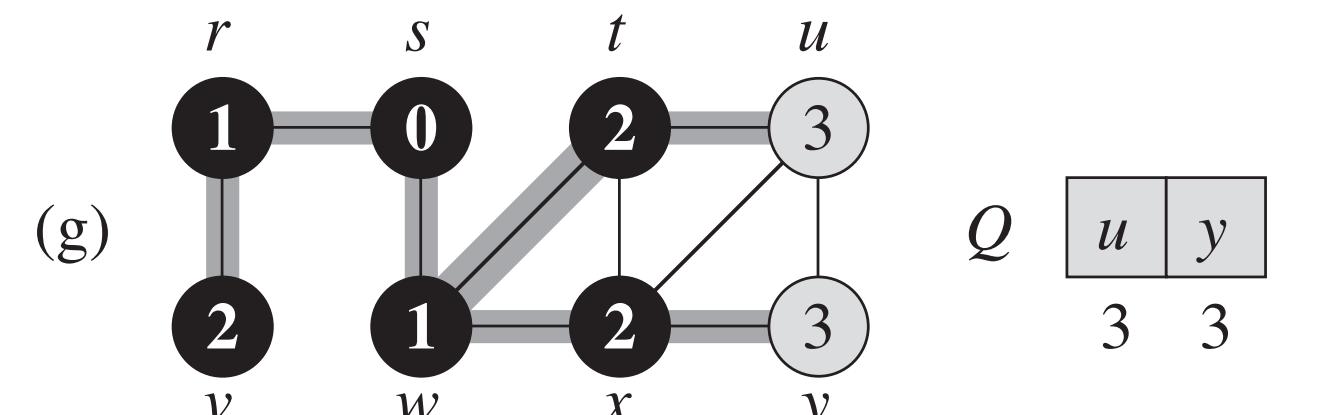
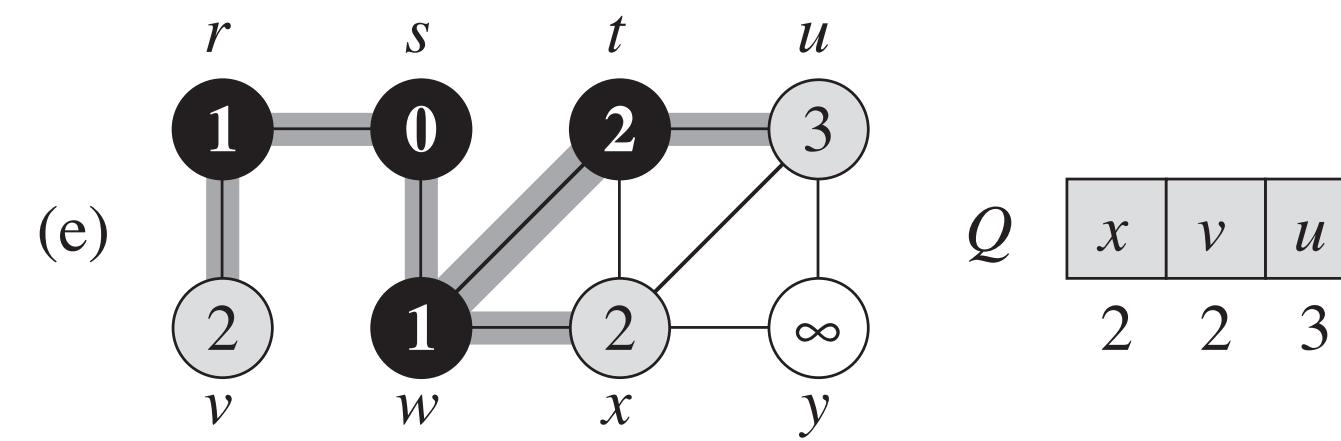
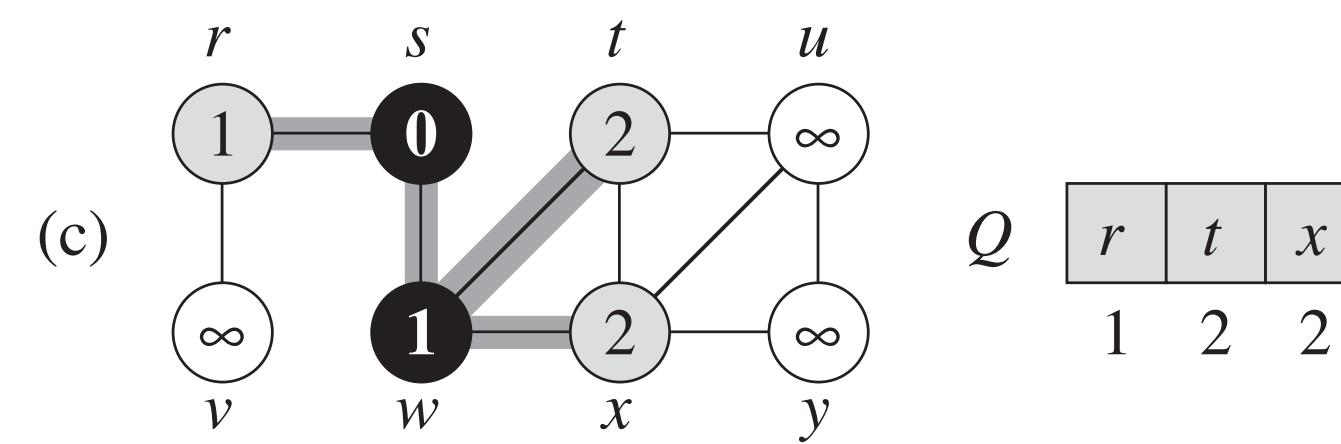
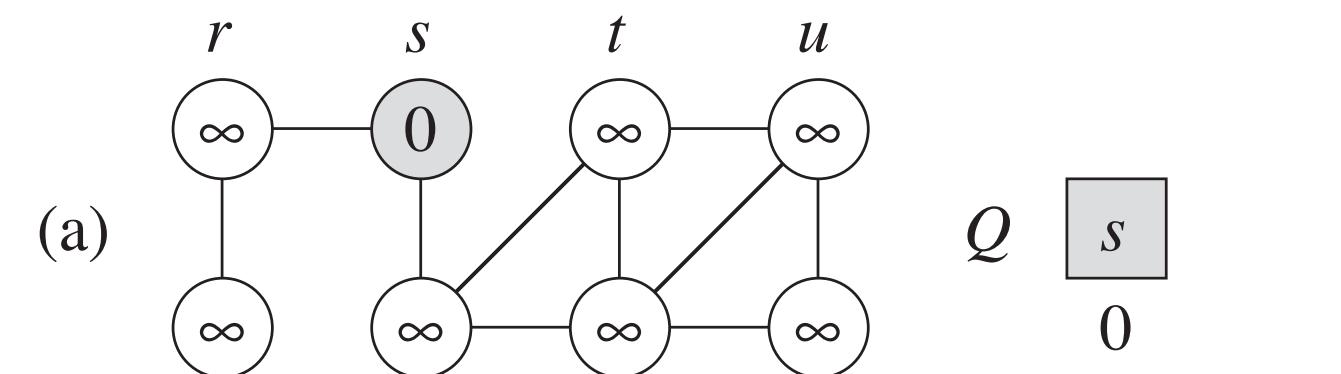
# breadth-first search



# breadth-first search



# breadth-first search



$\text{BFS}(G, s)$

```

1 for each vertex  $u \in G.V - \{s\}$ 
2    $u.\text{color} = \text{WHITE}$ 
3    $u.d = \infty$ 
4    $u.\pi = \text{NIL}$ 
5    $s.\text{color} = \text{GRAY}$ 
6    $s.d = 0$ 
7    $s.\pi = \text{NIL}$ 
8    $Q = \emptyset$ 
9   ENQUEUE( $Q, s$ )
10  while  $Q \neq \emptyset$ 
11     $u = \text{DEQUEUE}(Q)$ 
12    for each  $v \in G.\text{Adj}[u]$ 
13      if  $v.\text{color} == \text{WHITE}$ 
14         $v.\text{color} = \text{GRAY}$ 
15         $v.d = u.d + 1$ 
16         $v.\pi = u$ 
17        ENQUEUE( $Q, v$ )
18     $u.\text{color} = \text{BLACK}$ 

```

$v.d$

distance from source  $s$

$v.\text{color}$

white : undiscovered

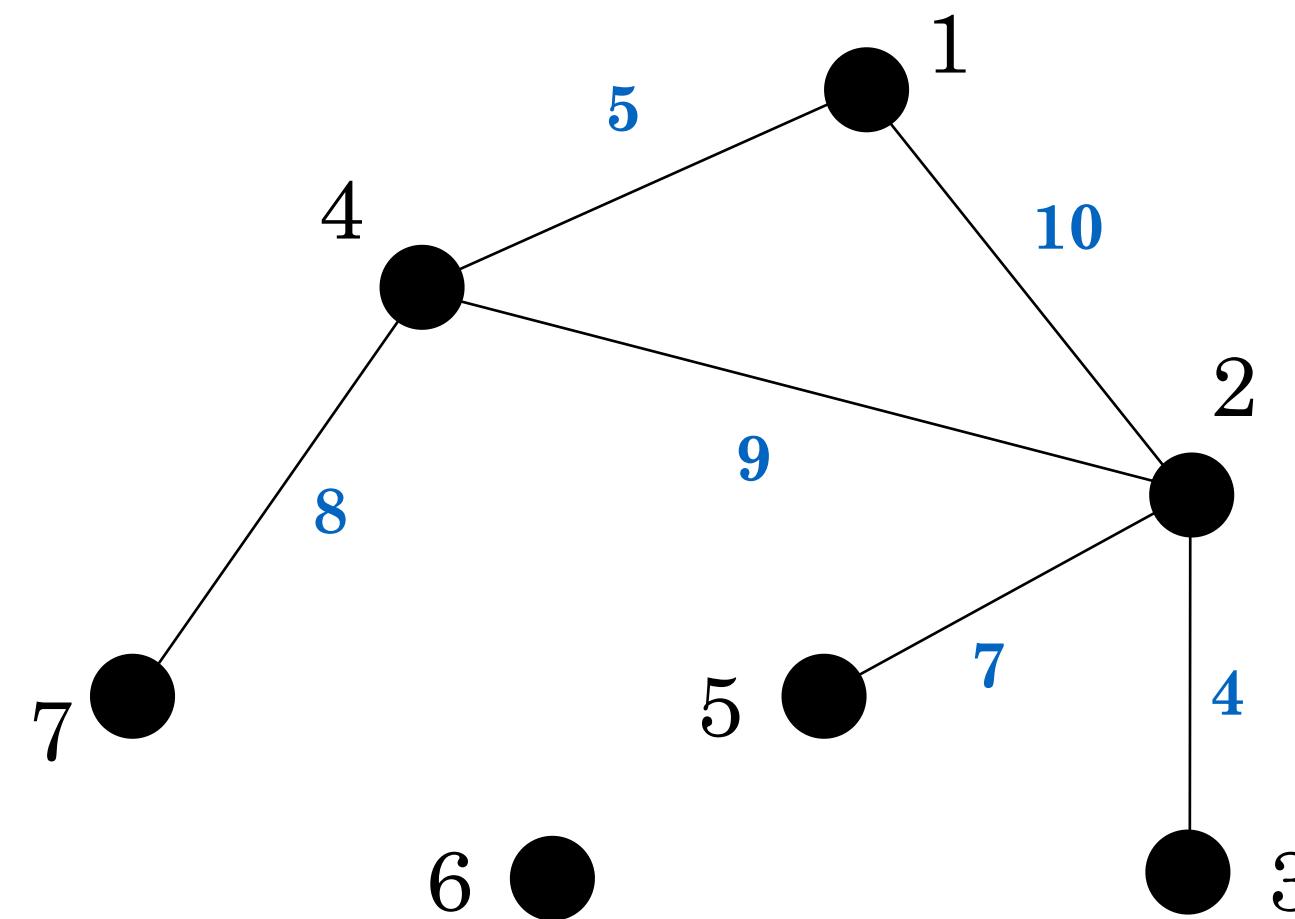
grey : discovered with some neighbors discovered

black : discovered with all neighbors discovered

$v.\pi$

predecessor in breadth-first tree

# minimum spanning tree



a **weighted graph**  $G_w = (G, w)$  is a tuple composed of a graph  $G = (V, E)$  and of a **function**  $w : E \rightarrow \mathbb{R}$  associating a **weight**  $w_e$  to each edge  $e \in E$

a **minimum (weight) spanning tree** of graph  $G_w = (G, w)$  is a connected subgraph  $(V', E')$  such that:

1

$$V' = V$$

2

$(V', E')$  does not contain any cycles

3

$\sum_{e \in E'} w_e$  is minimal across all subgraphs fulfilling 1 and 2

# minimum spanning tree

a disjoint-set data structure maintains a collection  $S = \{S_1, S_2, \dots, S_k\}$  of disjoint dynamic sets where each set is identified by a member of the set known as its representative

a disjoint-set data structure supports the following operations:

MAKE-SET( $x$ ) creates a new set whose only member and representative is  $x$

UNION( $x, y$ ) merges the dynamic sets that contain  $x$  and  $y$ , say  $S_x$  and  $S_y$ , into a new set that is the union of these two sets

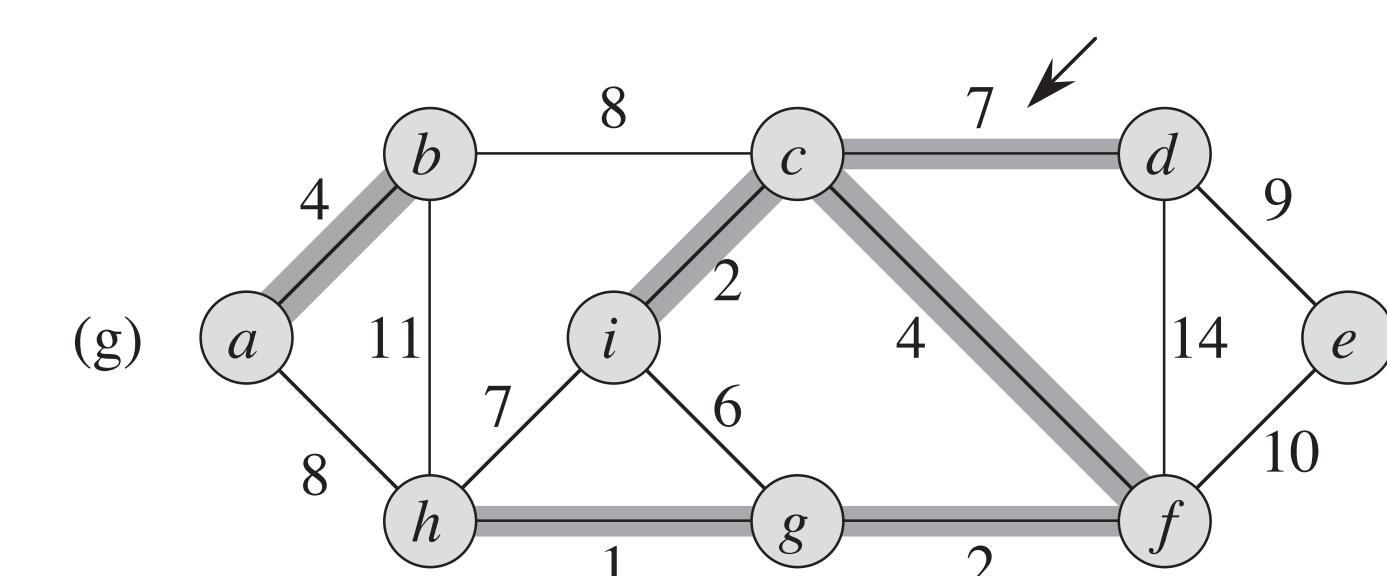
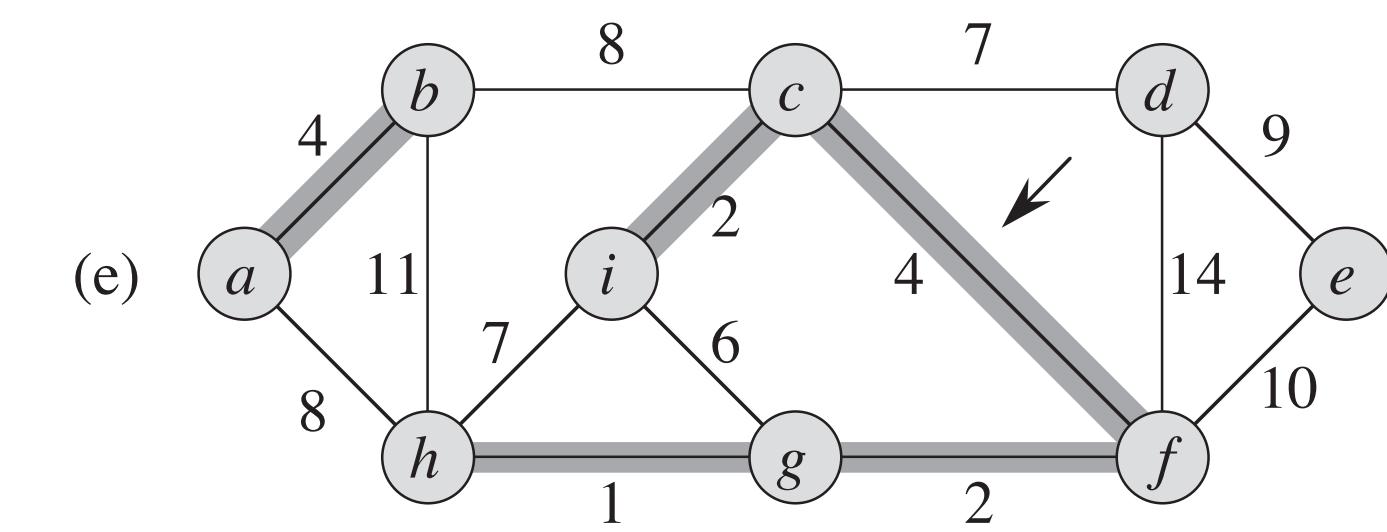
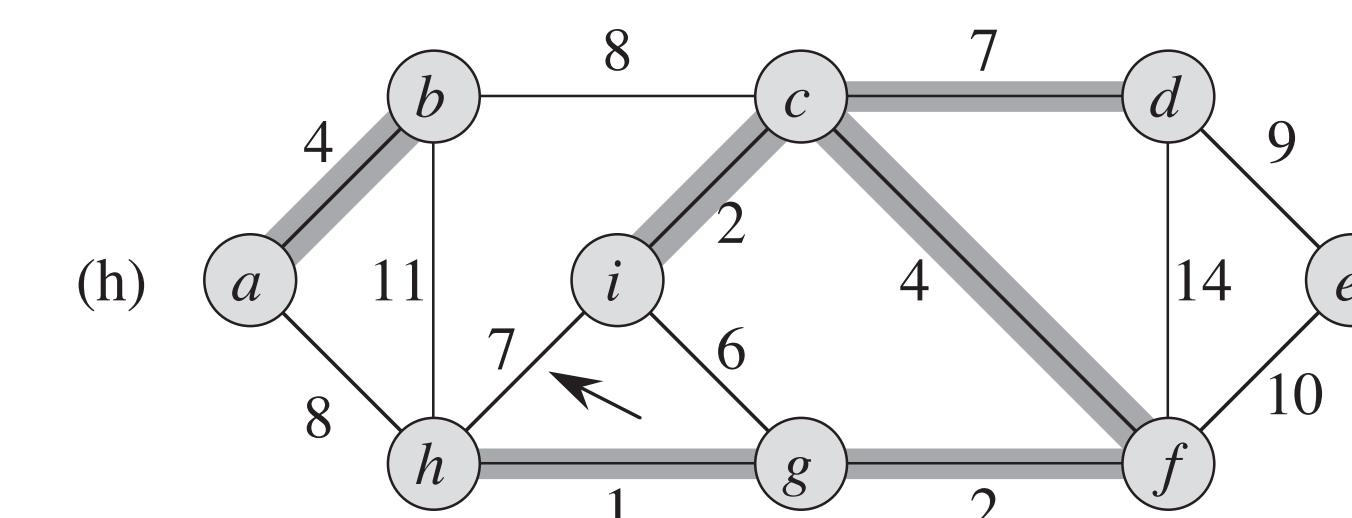
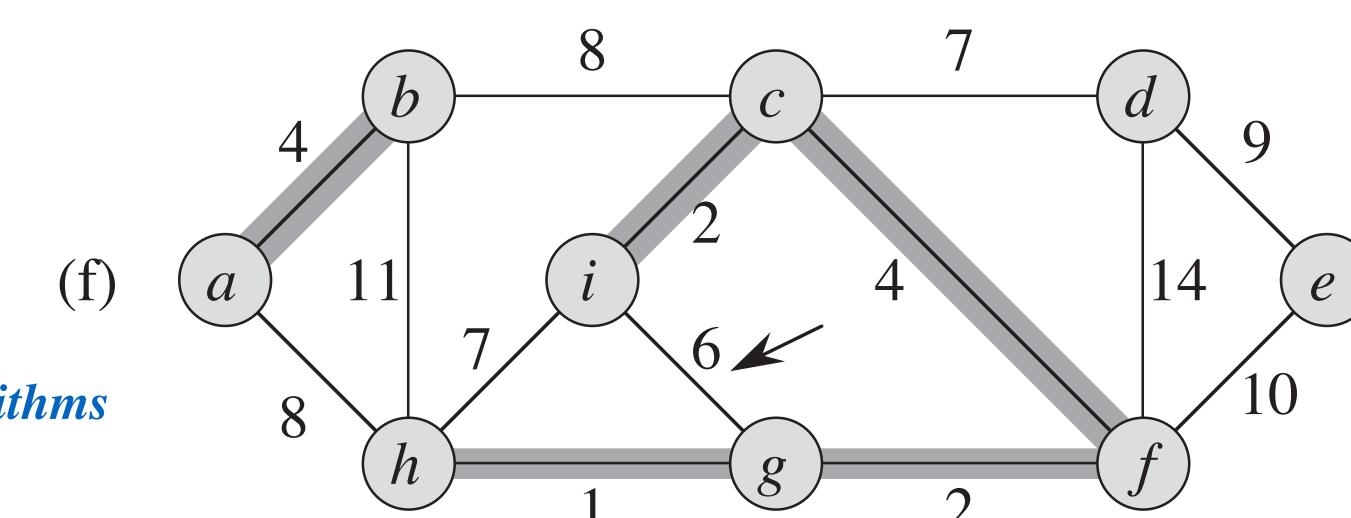
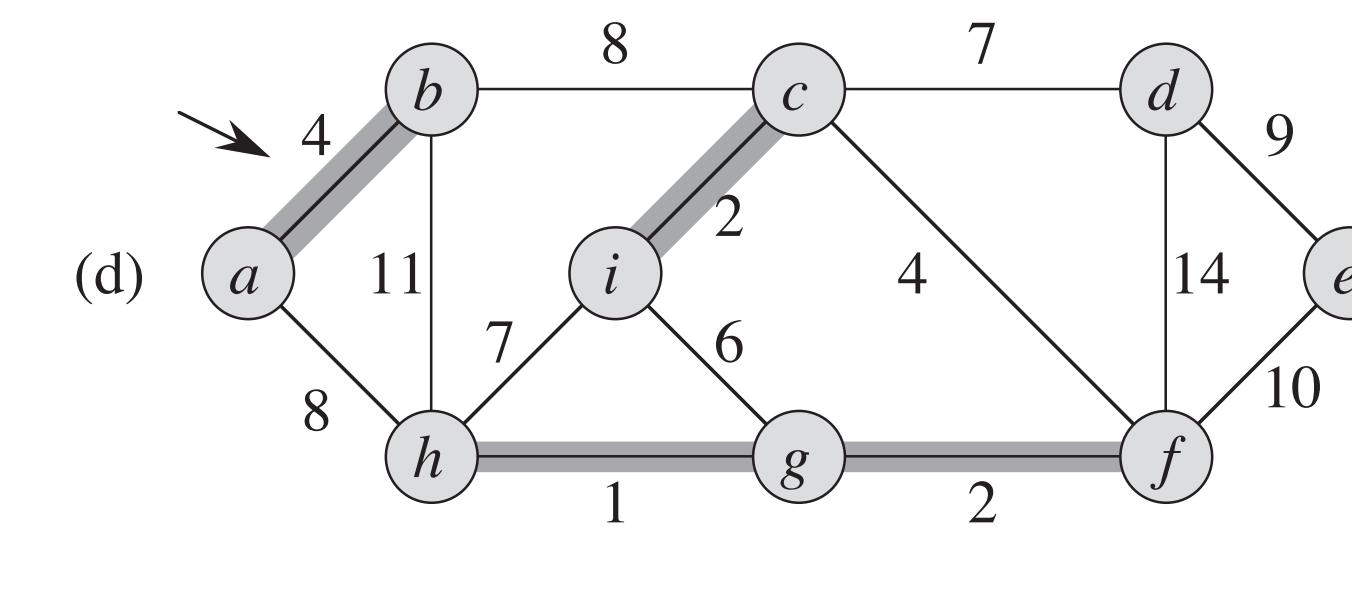
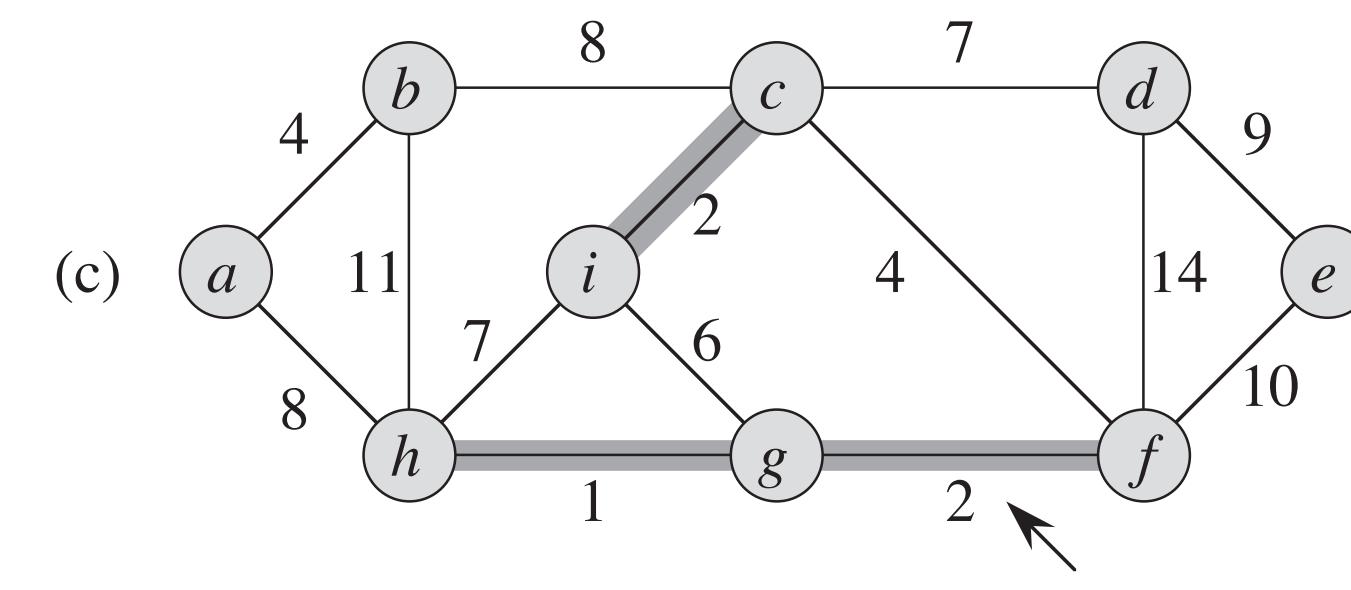
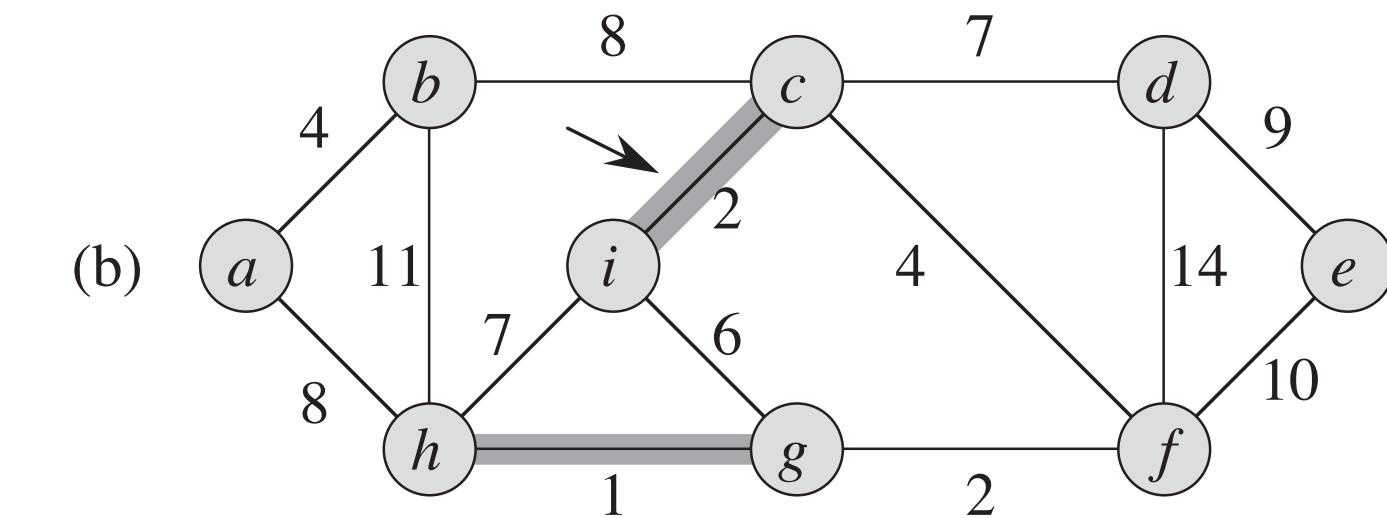
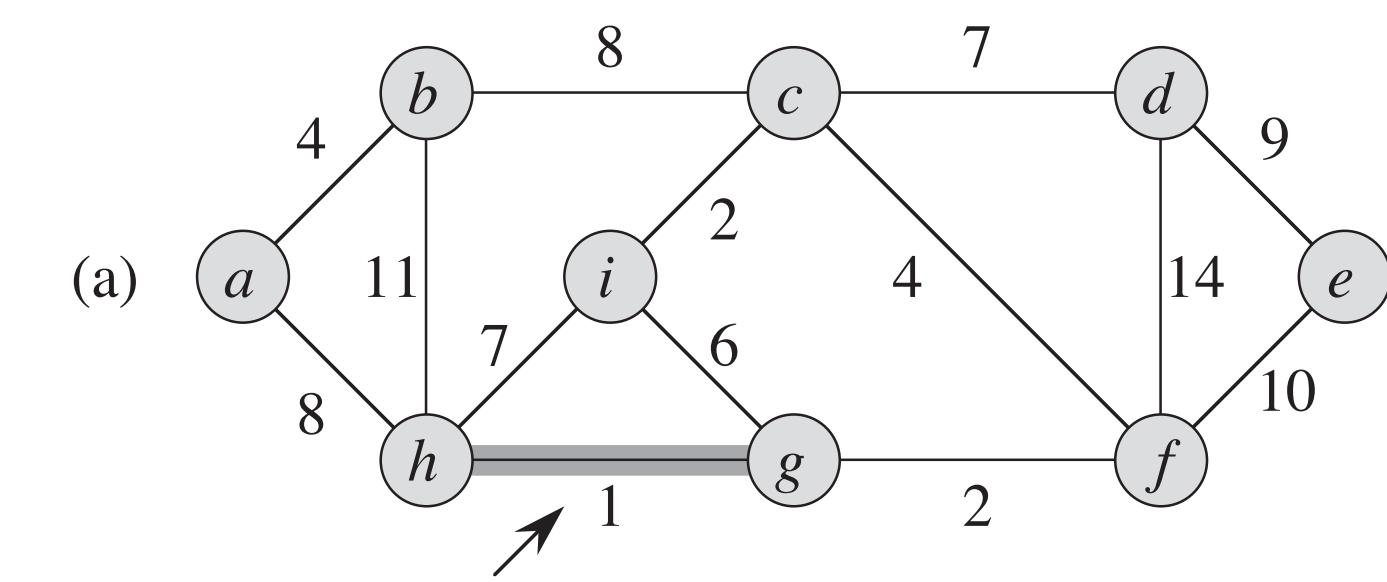
FIND-SET( $x$ ) returns the representative of the set containing  $x$

# minimum spanning tree - Kruskal's algorithm

MST-KRUSKAL( $G, w$ )

```

1   $A = \emptyset$ 
2  for each vertex  $v \in G.V$ 
3      MAKE-SET( $v$ )
4  sort the edges of  $G.E$  into nondecreasing order by weight  $w$ 
5  for each edge  $(u, v) \in G.E$ , taken in nondecreasing order by weight
6      if FIND-SET( $u$ )  $\neq$  FIND-SET( $v$ )
7           $A = A \cup \{(u, v)\}$ 
8          UNION( $u, v$ )
9  return  $A$ 
```

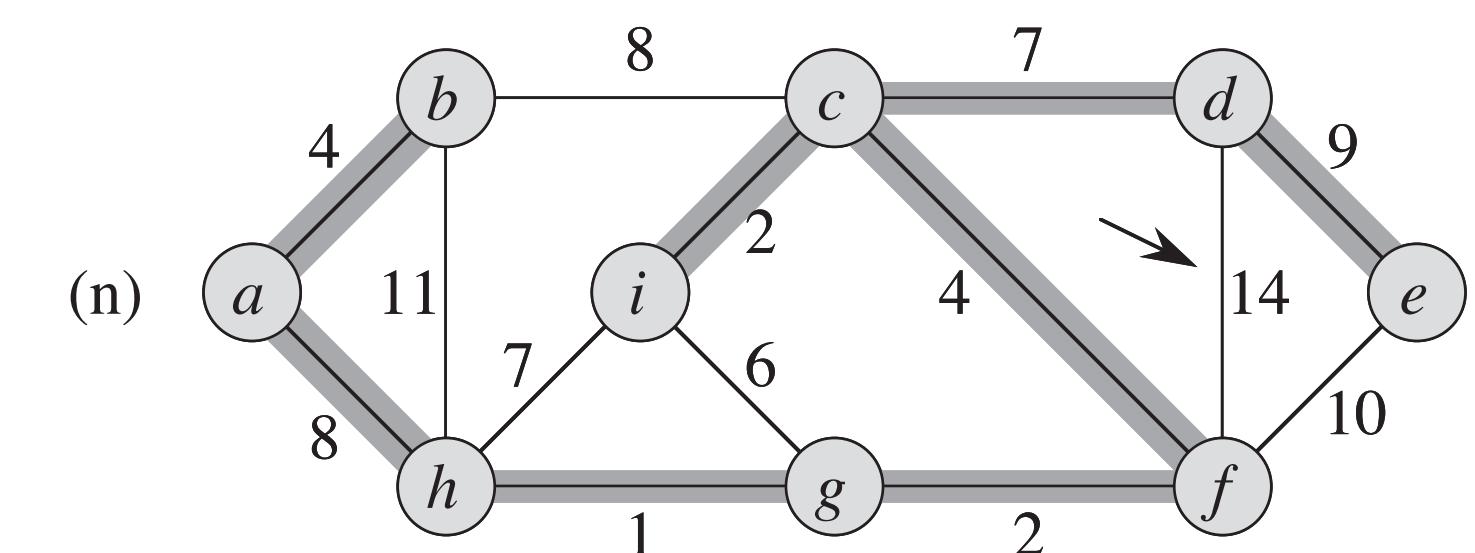
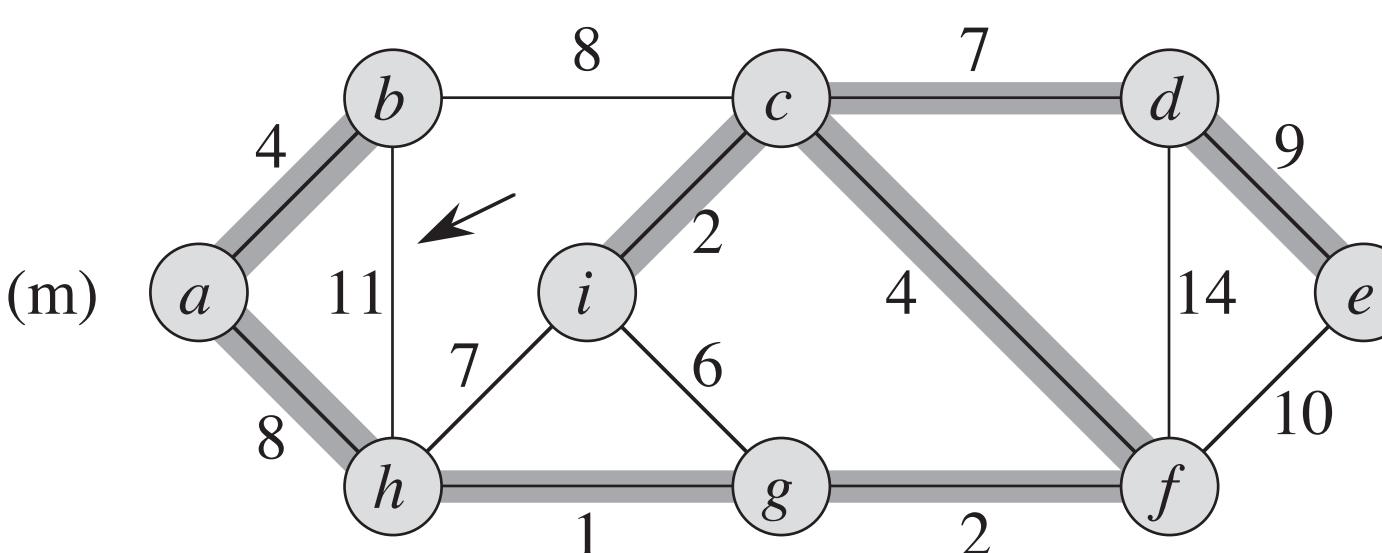
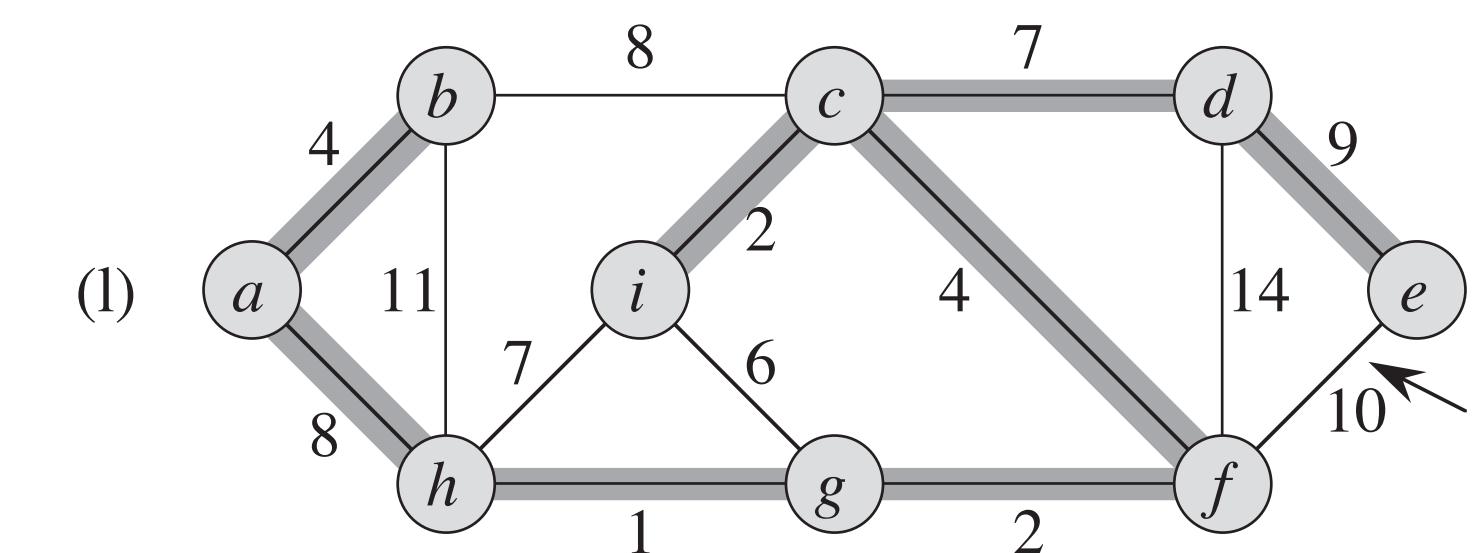
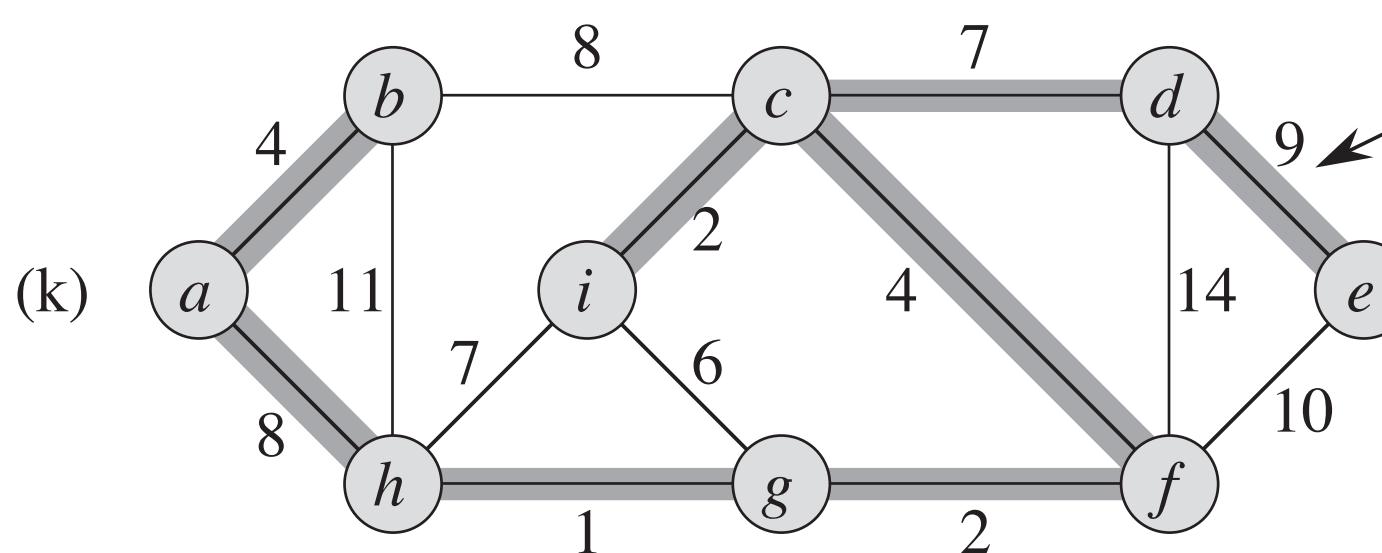
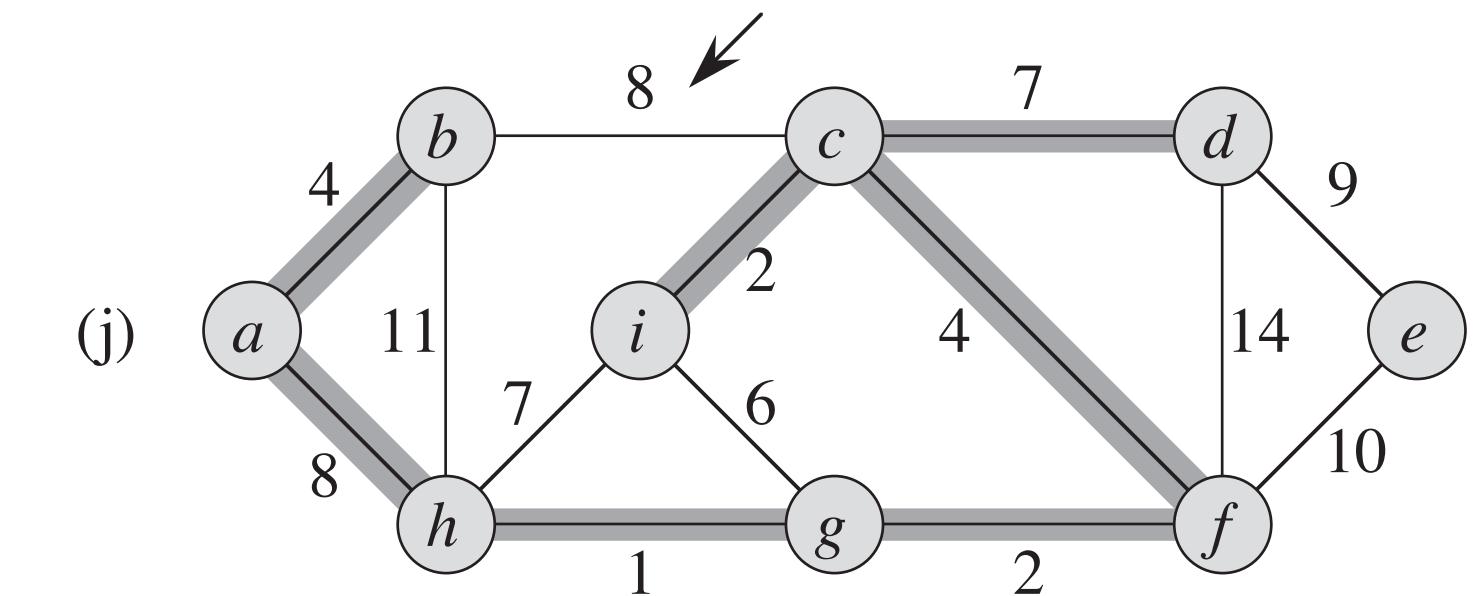
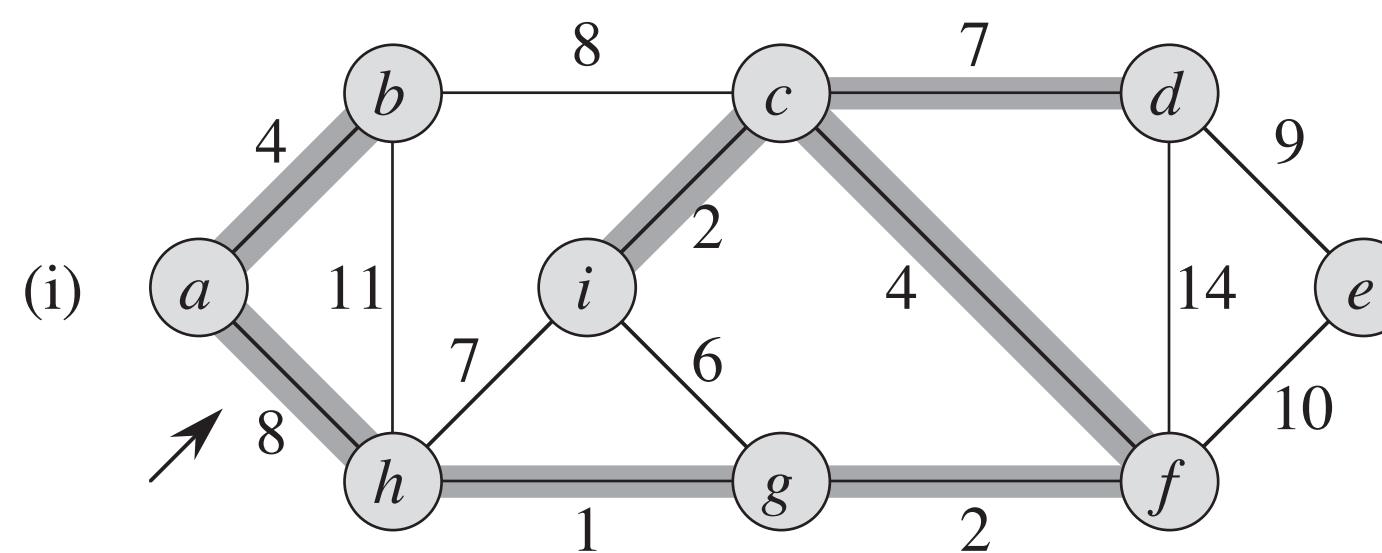


# minimum spanning tree - Kruskal's algorithm

MST-KRUSKAL( $G, w$ )

```

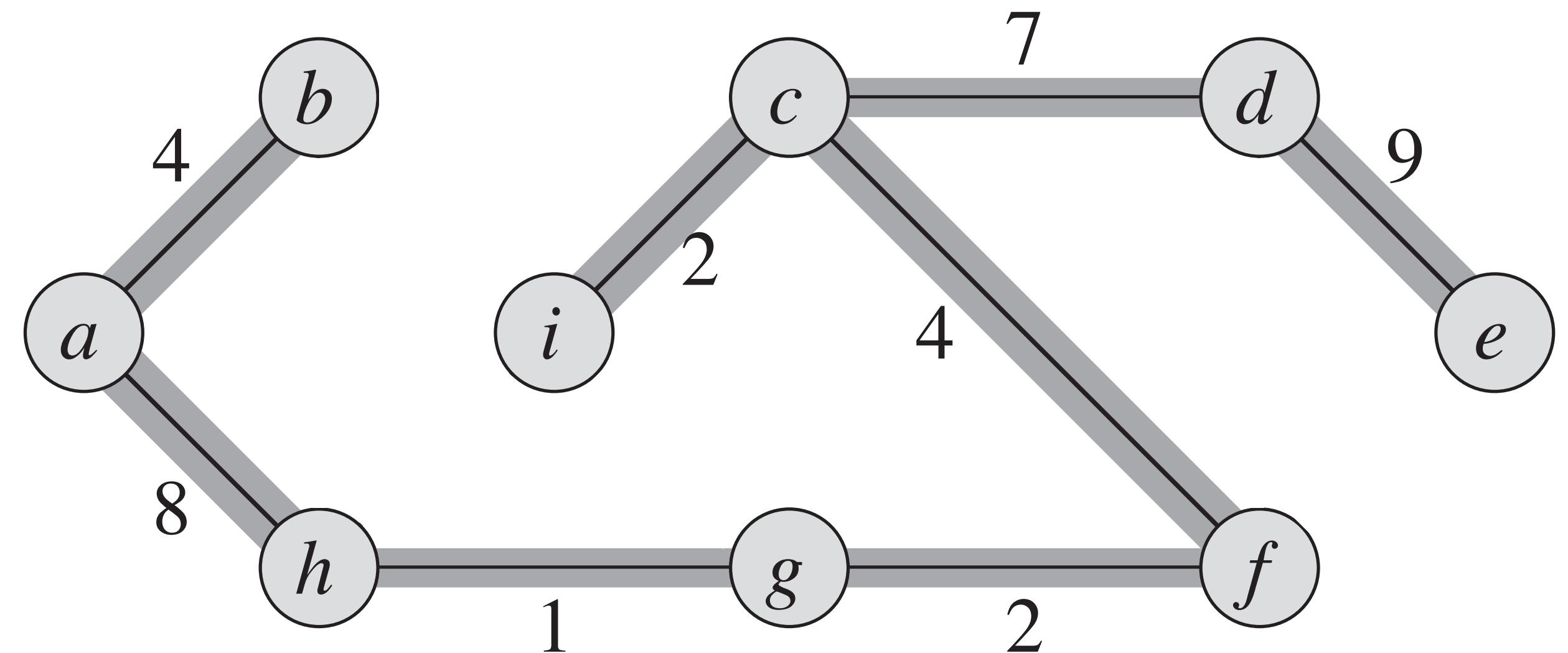
1    $A = \emptyset$ 
2   for each vertex  $v \in G.V$ 
3       MAKE-SET( $v$ )
4   sort the edges of  $G.E$  into nondecreasing order by weight  $w$ 
5   for each edge  $(u, v) \in G.E$ , taken in nondecreasing order by weight
6       if FIND-SET( $u$ )  $\neq$  FIND-SET( $v$ )
7            $A = A \cup \{(u, v)\}$ 
8       UNION( $u, v$ )
9   return  $A$ 
```



# minimum spanning tree - Kruskal's algorithm

MST-KRUSKAL( $G, w$ )

```
1  $A = \emptyset$ 
2 for each vertex  $v \in G.V$ 
3   MAKE-SET( $v$ )
4 sort the edges of  $G.E$  into nondecreasing order by weight  $w$ 
5 for each edge  $(u, v) \in G.E$ , taken in nondecreasing order by weight
6   if FIND-SET( $u$ )  $\neq$  FIND-SET( $v$ )
7      $A = A \cup \{(u, v)\}$ 
8     UNION( $u, v$ )
9 return  $A$ 
```



the Kruskal's algorithm is **greedy**, i.e., it makes  
locally optimal choice at each step