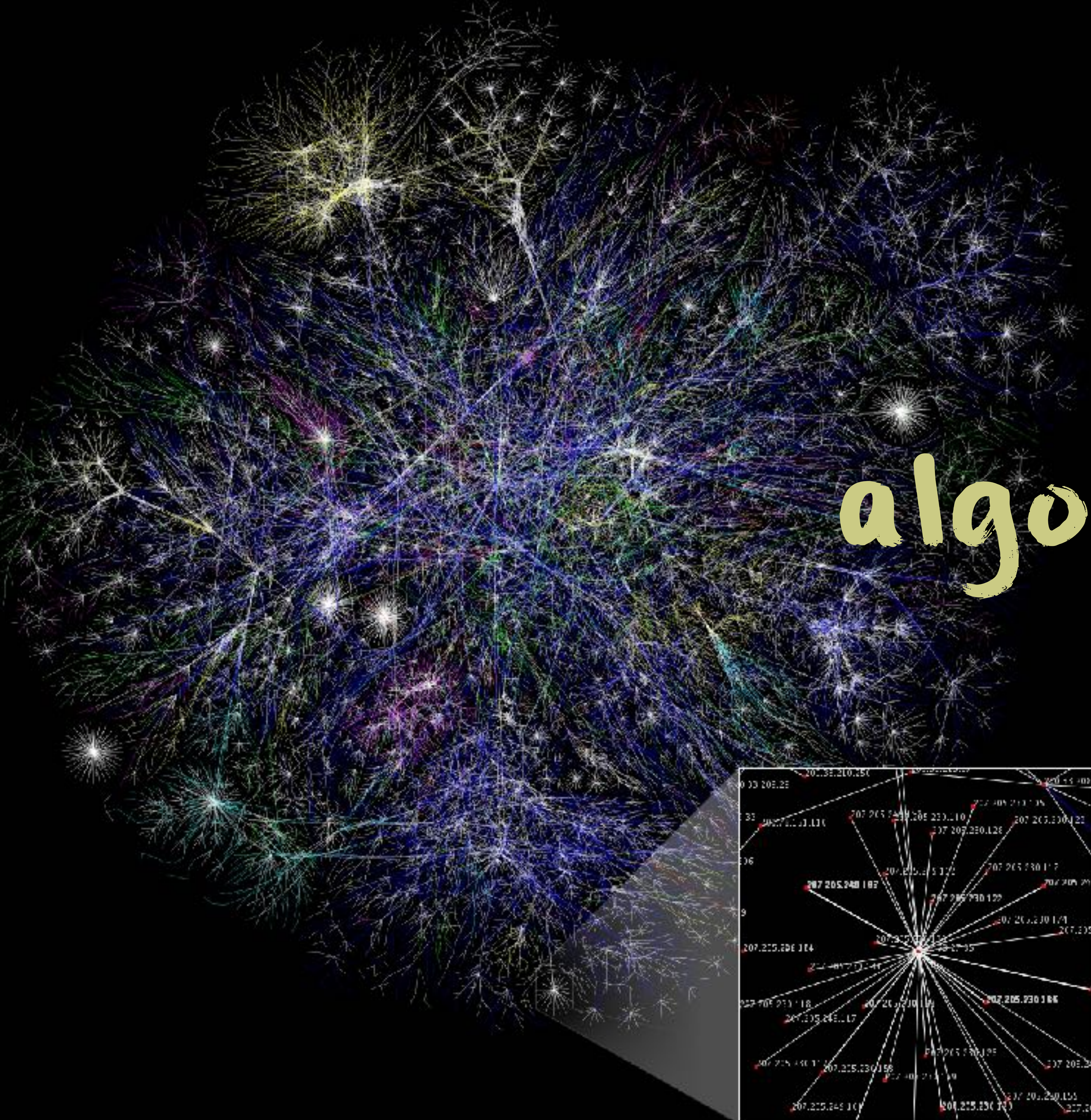


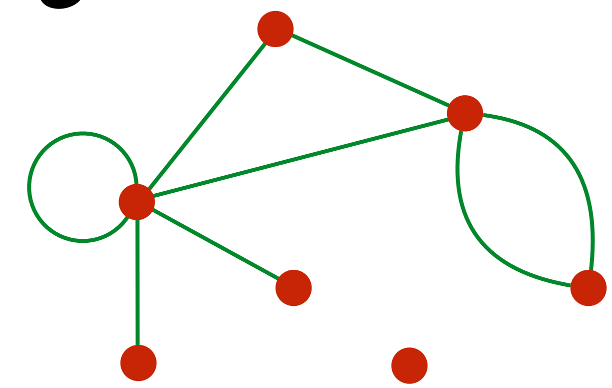
graph algorithms

learning objectives

algorithms ←
your software
system software
hardware

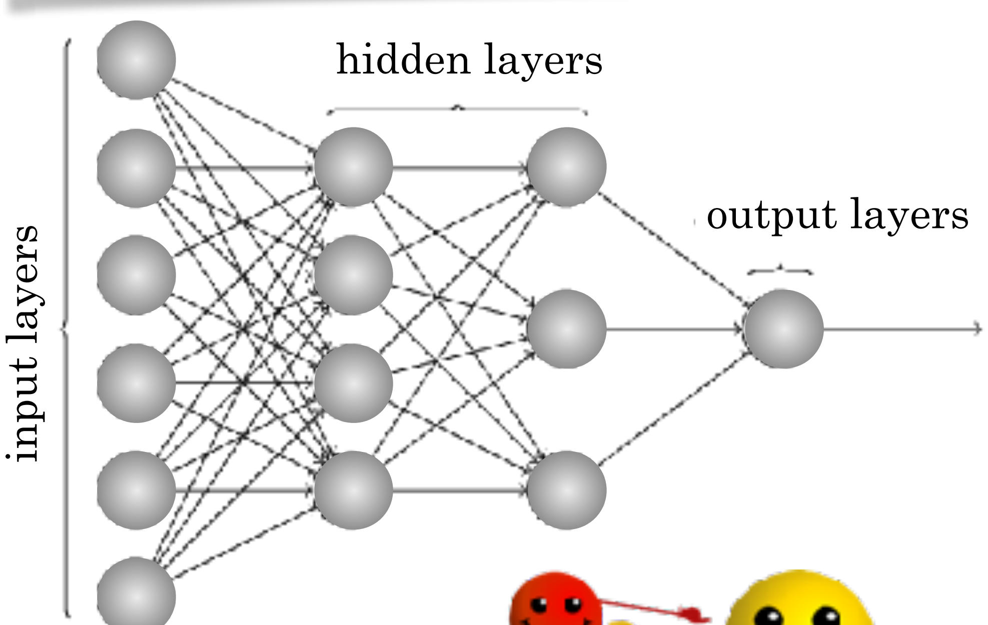
- learn what graphs are in mathematical terms
- learn how to represent graphs in computers
- learn about typical graph algorithms

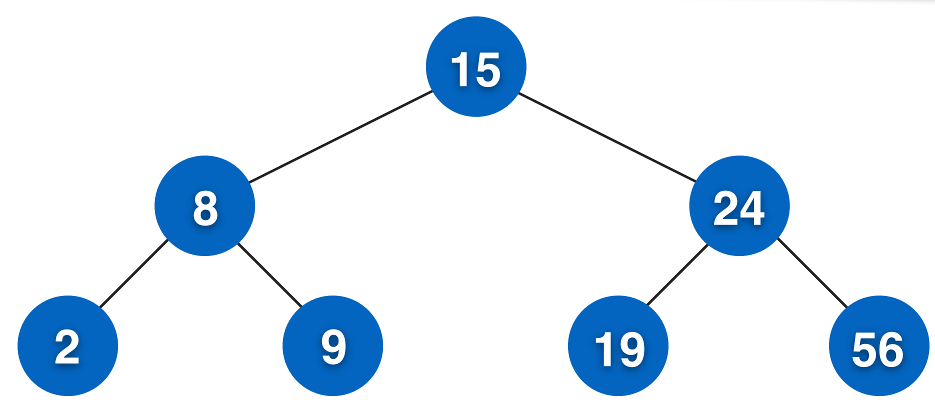
why graphs?

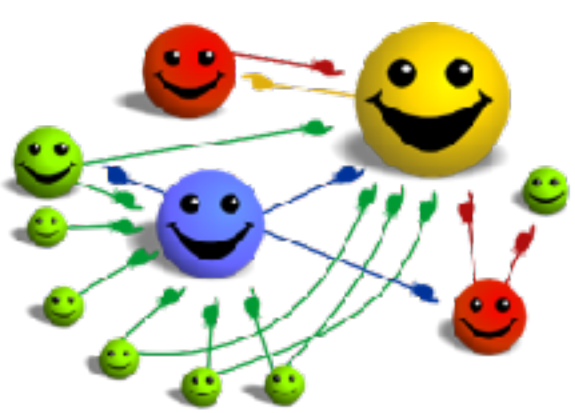


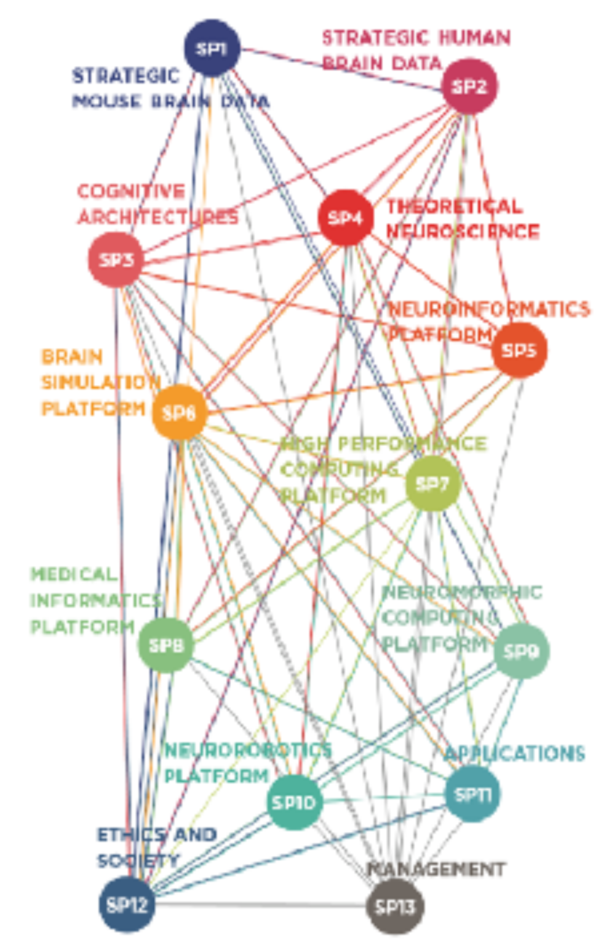
intuitively, a graph is formed by vertices and edges between vertices

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





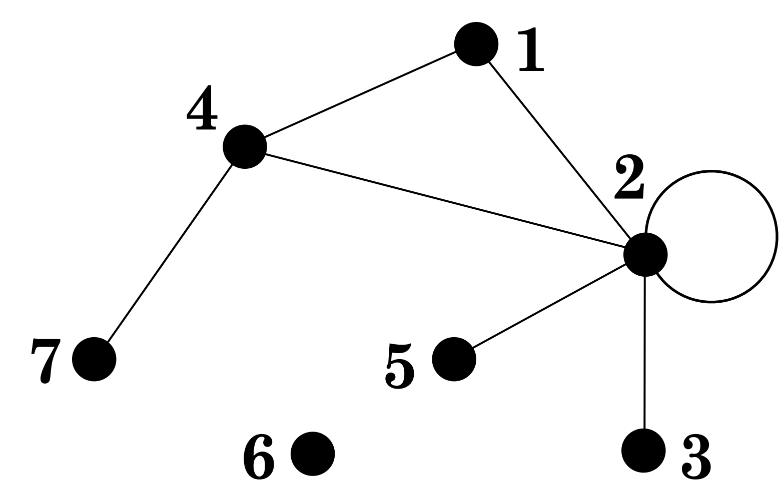




what's a graph?

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:

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

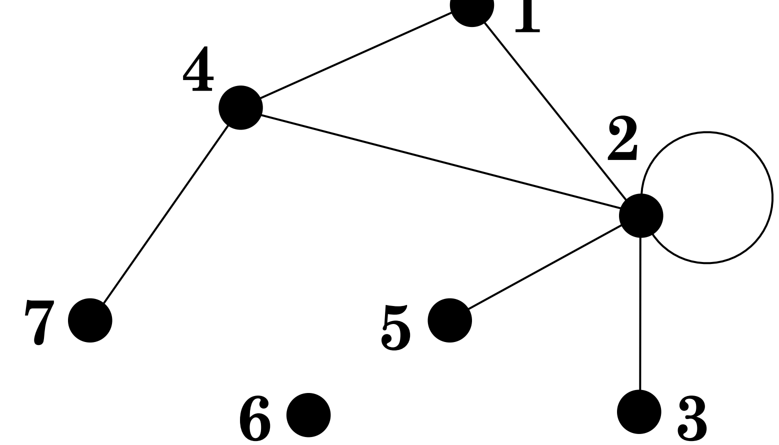
$$E = \{\{1, 2\}, \{1, 4\}, \{2\}, \{2, 3\}, \{2, 4\}, \{2, 5\}, \{4, 7\}\}$$

types of graphs

undirected:

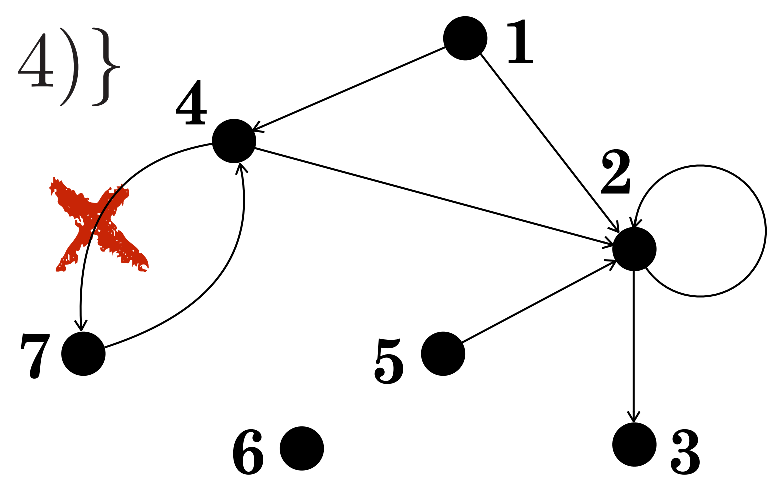
$V = \{1, 2, 3, 4, 5, 6, 7\}$

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



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** $\langle v_0, v_1, \ldots, v_k \rangle$ **of vertices** $v_i \in V$ where $x = v_0$ and $y = v_k$, such that $\forall i \in \{1, \ldots, k\} : (v_{i-1}, v_i) \in E$

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

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

we can store **attributes** in vertices and edges using the **dotted notation**, e.g., $v.color$ stores a $color$ attribute in vertex v, while $e.weight$ and $(x,y).weight$ 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 $d(v)$

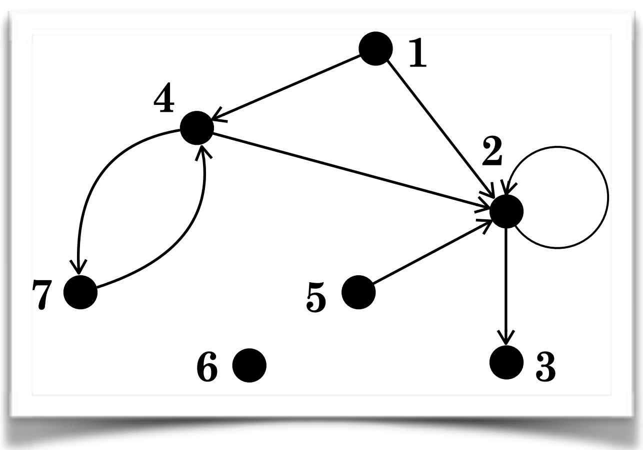
we defined $\delta(G) = \min \{ d(v) \mid v \in V \}$ as the minimum degree of G

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

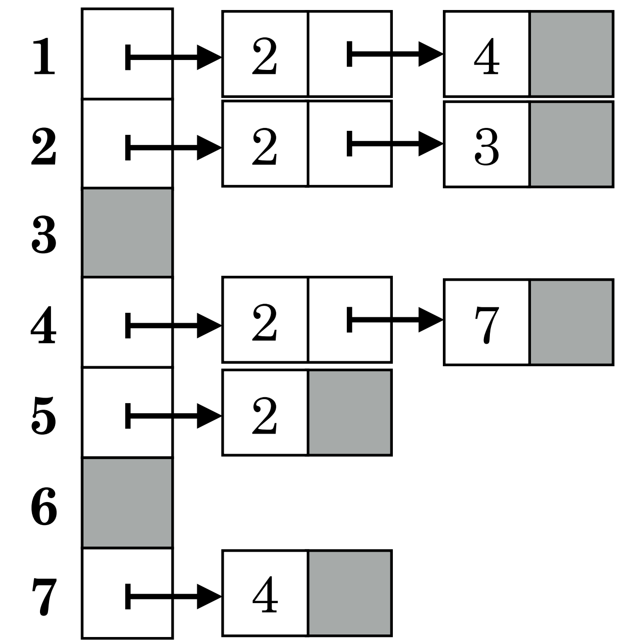
representing graphs

directed



	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 list

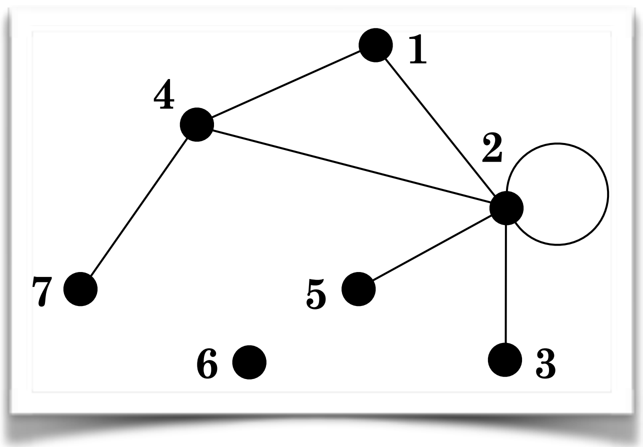
an adjacency list is best suited for representing a sparse graph

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

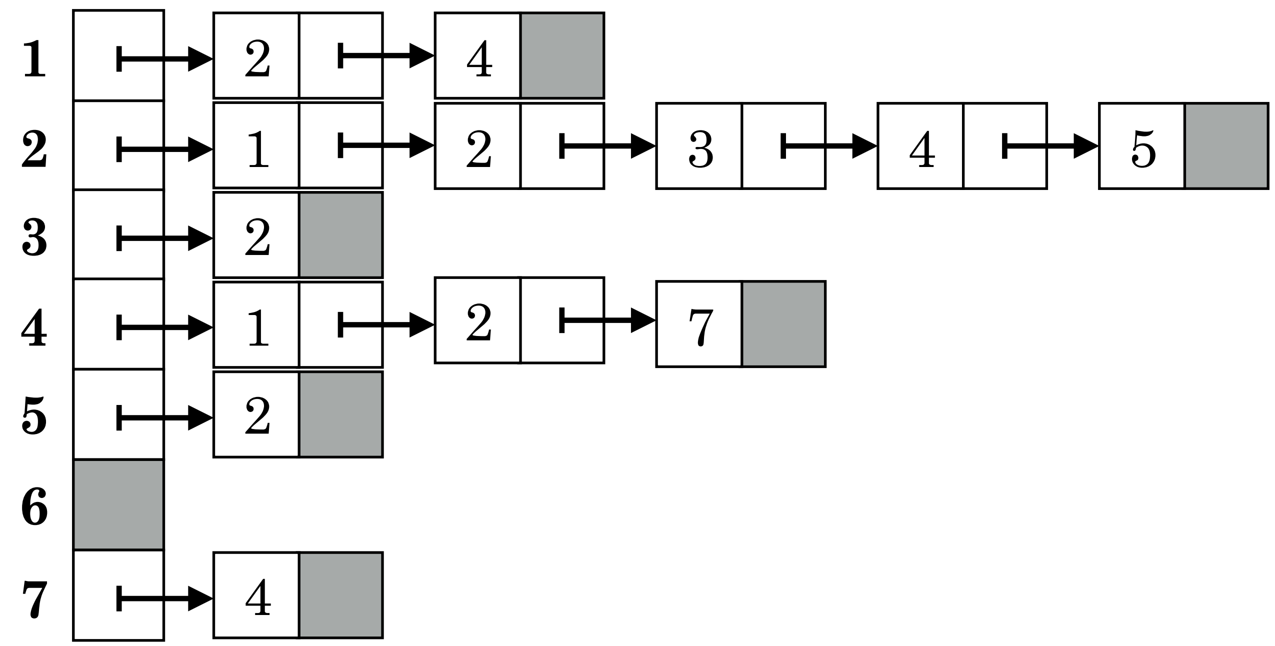
representing graphs

undirected



	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

adjacency matrix



adjacency list

typical problems

breadth-first search

minimum spanning tree

single-source shortest paths

breadth-first search

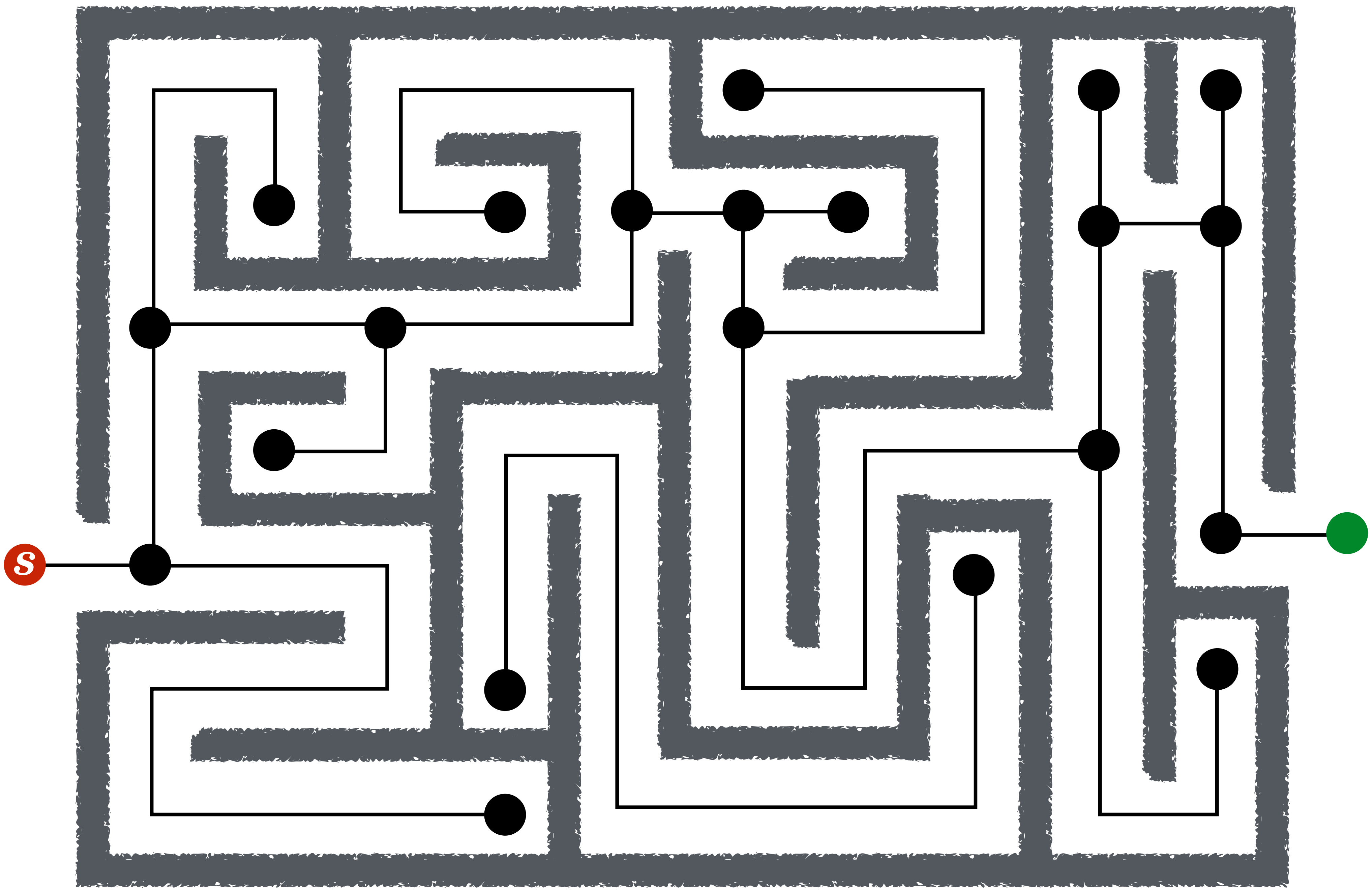
given graph G and a source vertex $s \in G$,
it discover every vertex reachable from s

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

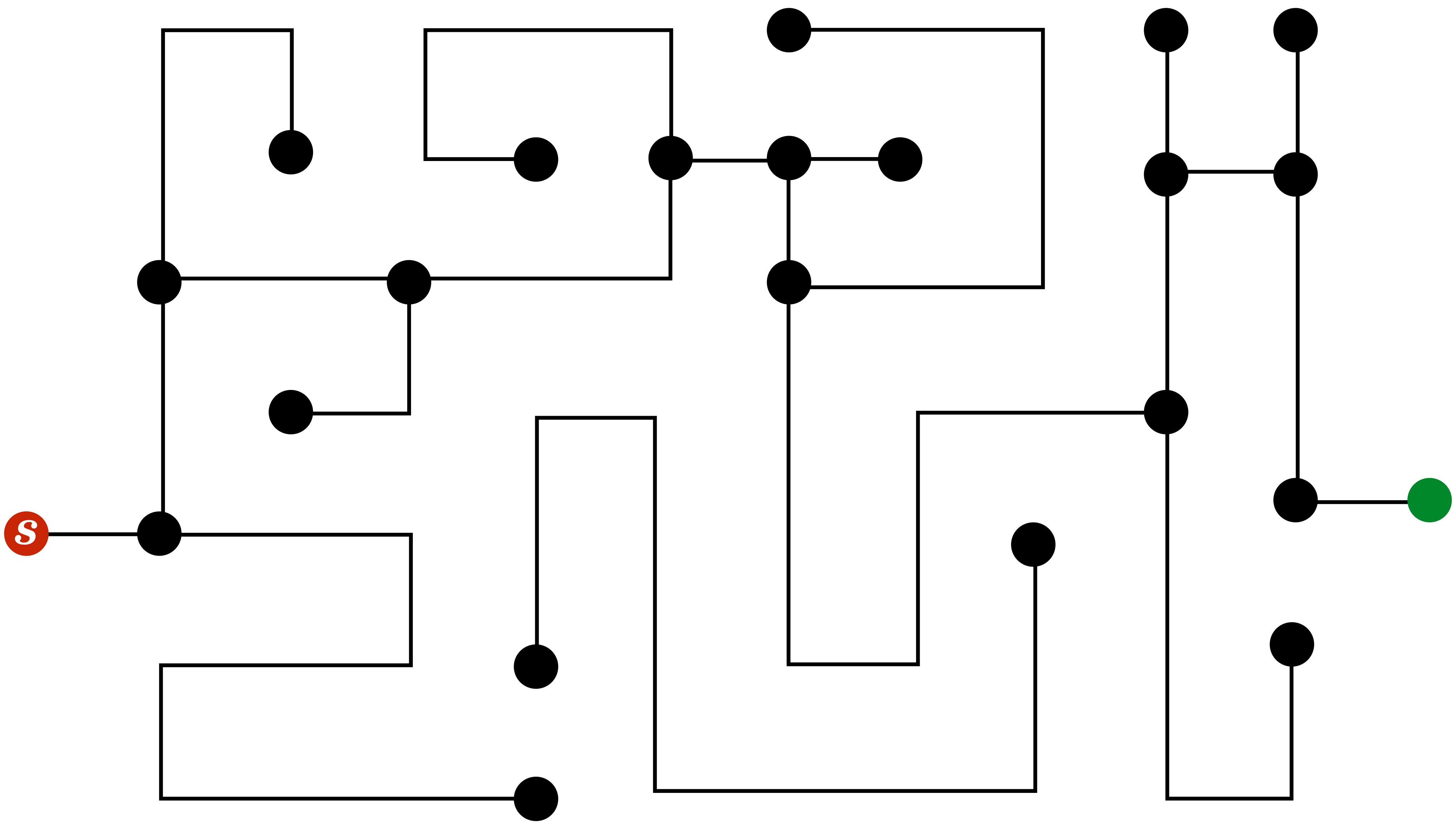
it produces a breadth-first tree rooted at s
that contains all reachable vertices from s

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

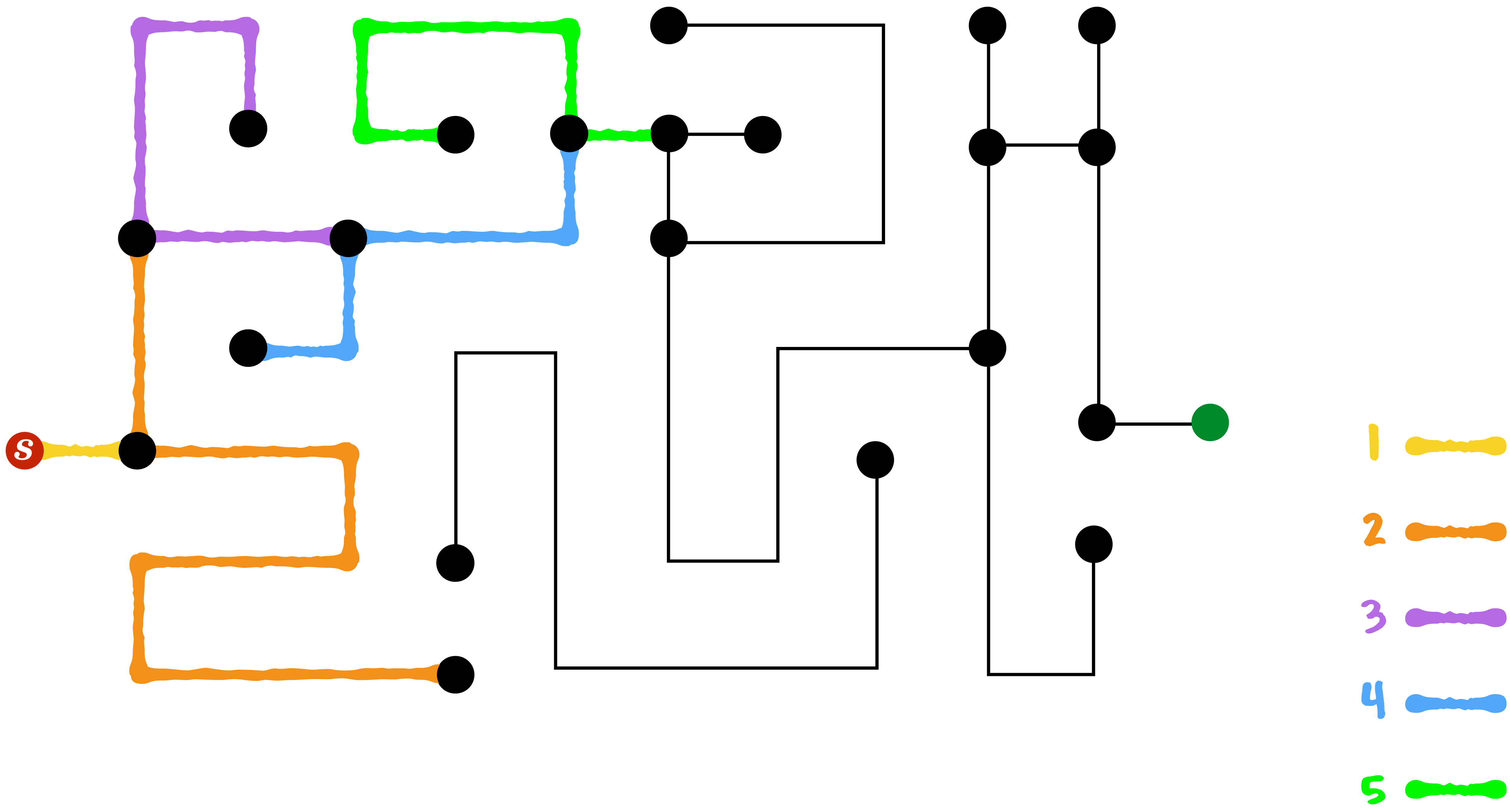
breadth-first search



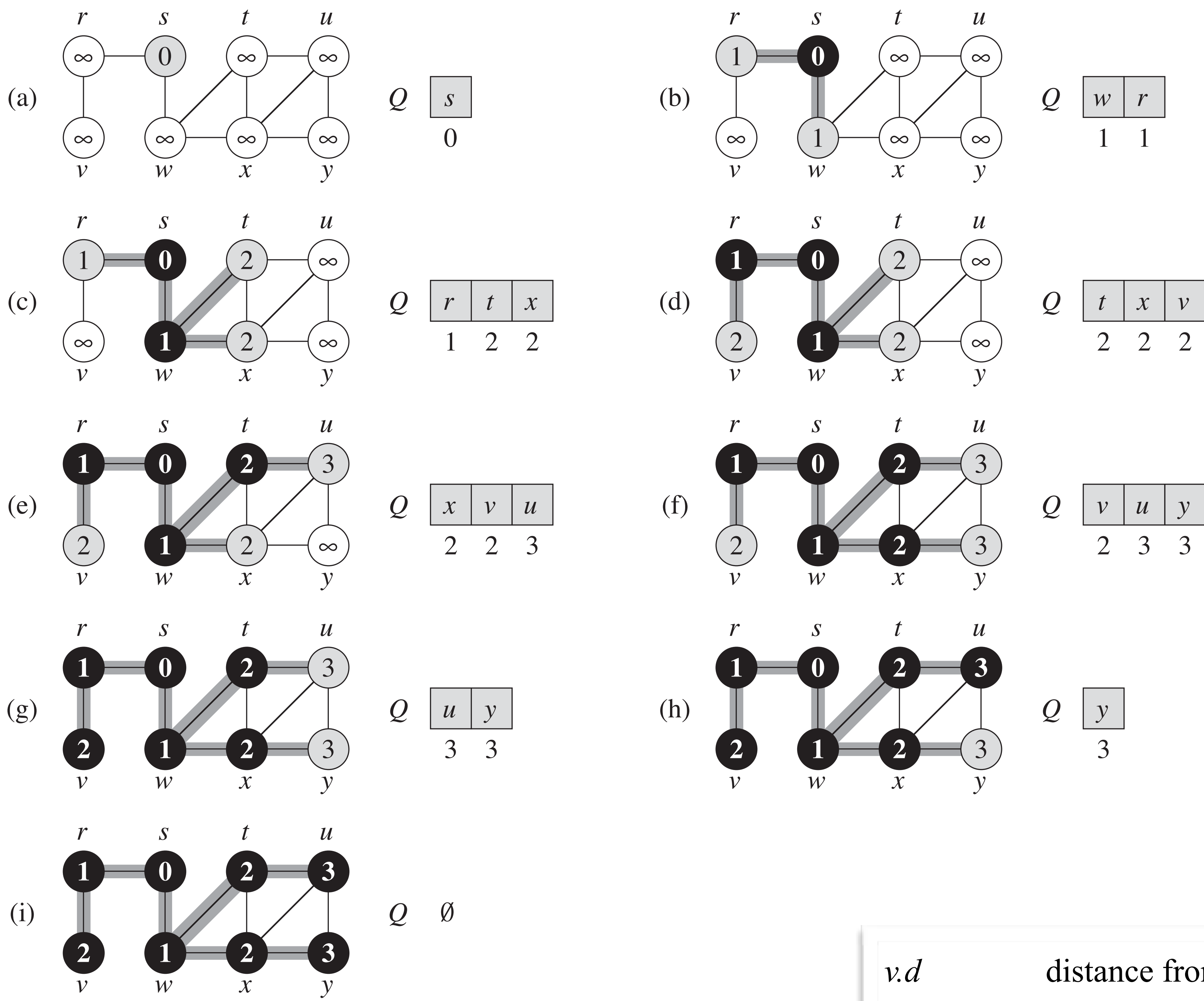
breadth-first search



breadth-first search



breadth-first search

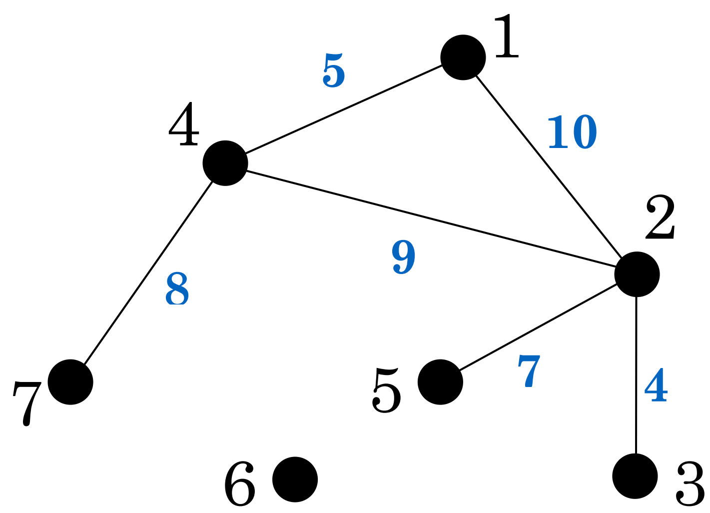



```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = Ø
ENQUEUE(Q, s)
while Q ≠ Ø
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

v.d	distance from source *s*	
v.color	white :	undiscovered
	grey :	discovered with some neighbors discovered
	black :	discovered with all neighbors discovered
v.π	predecessor in bread-first three	

from: ***Introduction to Algorithms***
by T. H. Cormen et al.
3rd Edition
MIT Press, 2009

minimum spanning tree



a **weighted graph** $G_w = (G, w)$ is a tuple composed of a graph $G = (V, E)$ and of a **function** $w : E \to \mathbb{R}$ 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 $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ 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:

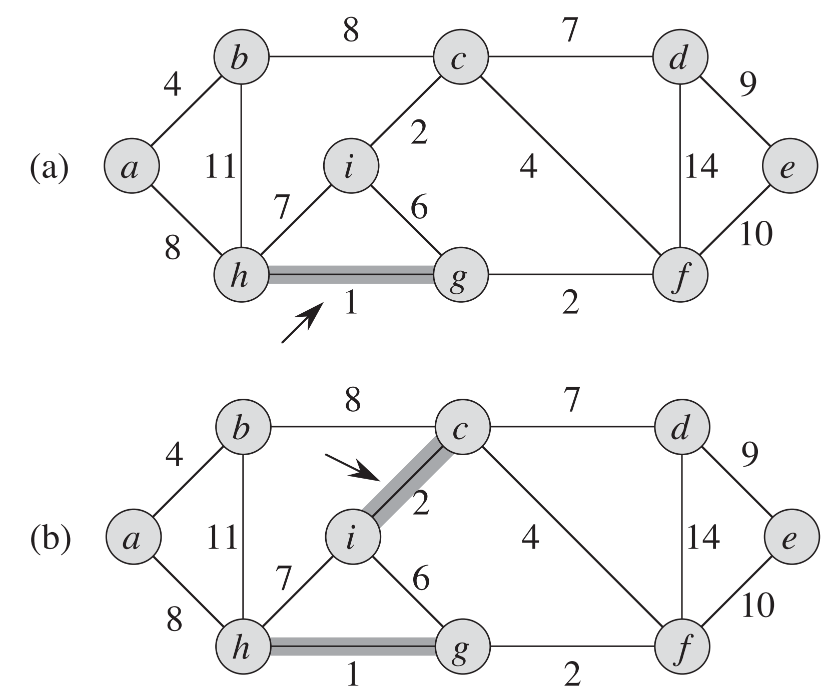
MAKE-SET(x) creates a new set whose only member and its representative is x

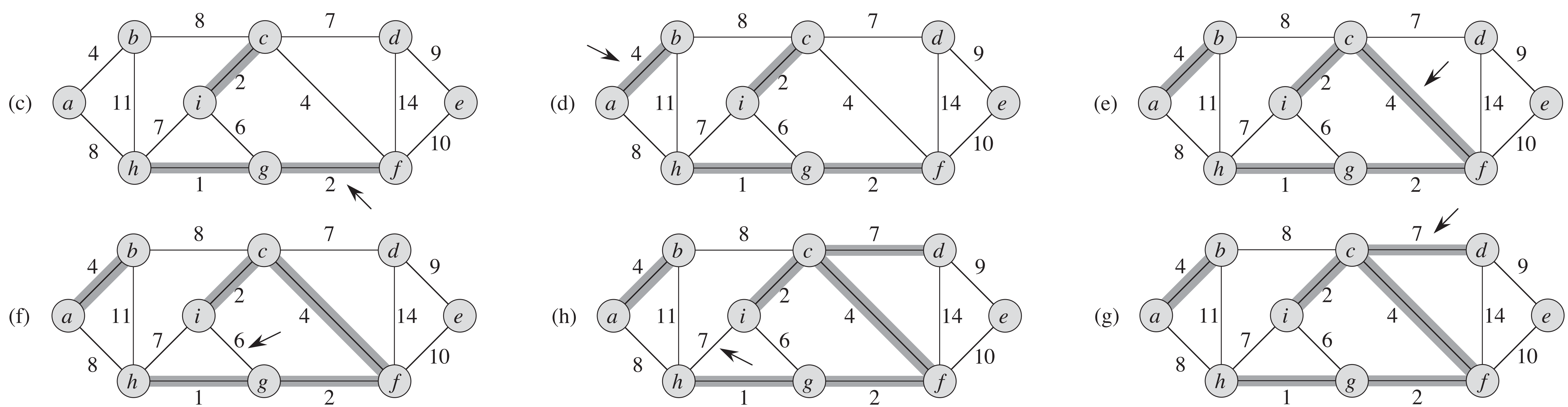
UNION(x,y) merges the dynamic sets that contain x and y, say S_x and S_y, into a new set that is the union of these two sets

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

minimum spanning tree
Kruskal's algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

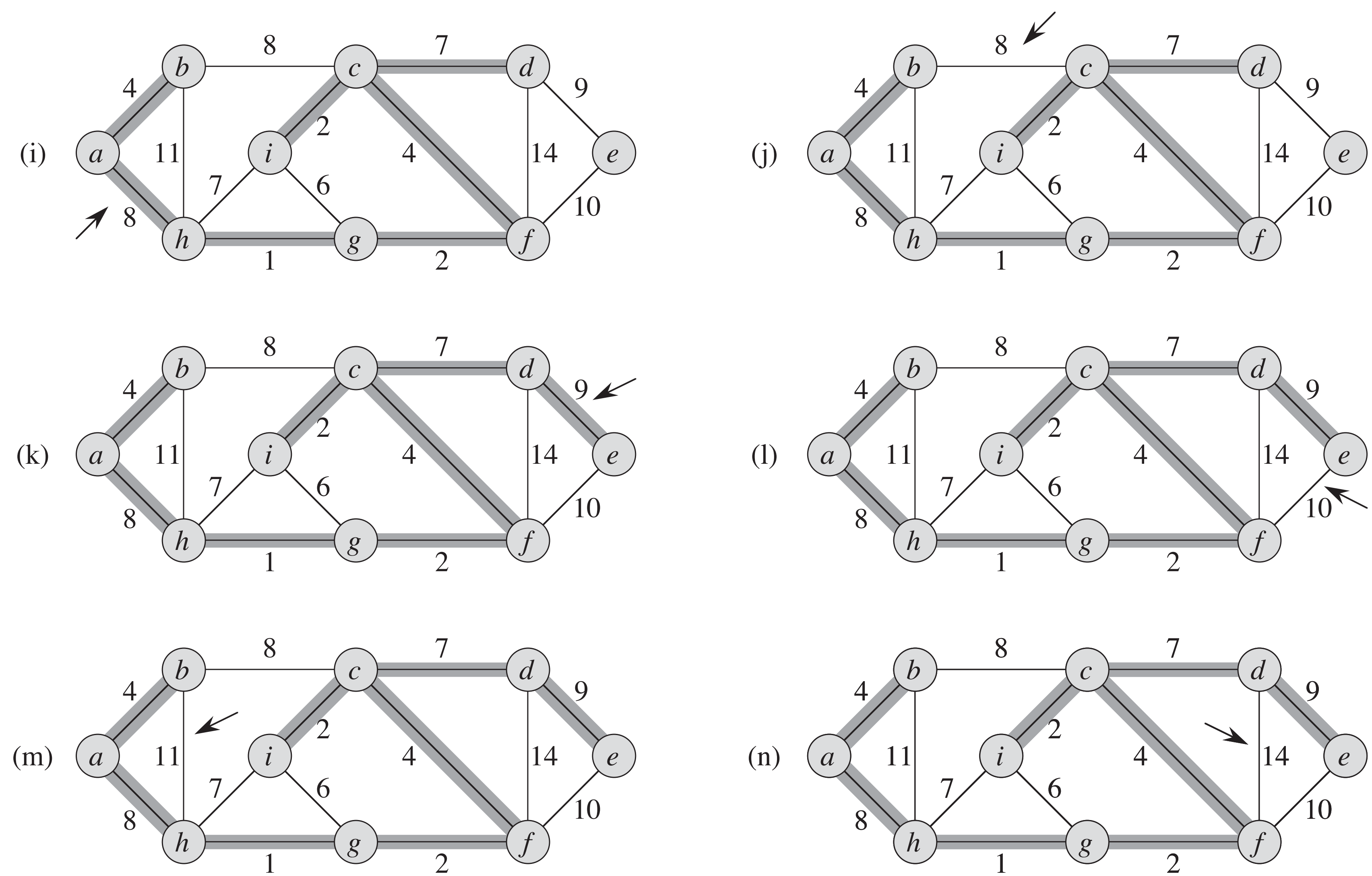




from: ***Introduction to Algorithms***
by T. H. Cormen et al.
3rd Edition
MIT Press, 2009

minimum spanning tree
Kruskal's algorithm

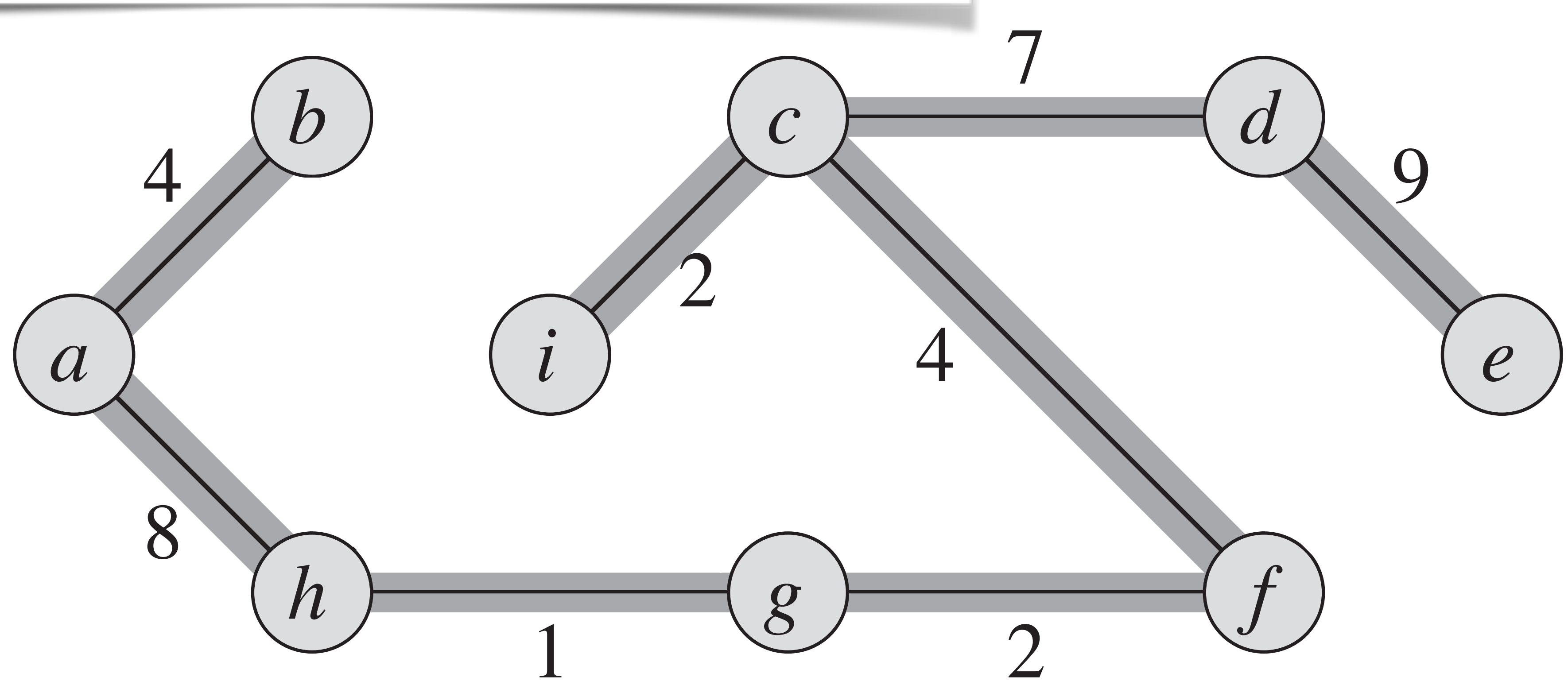
```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```



from: ***Introduction to Algorithms***
by T. H. Cormen et al.
3rd Edition
MIT Press, 2009

minimum spanning tree
Kruskal's algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```



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

from: ***Introduction to Algorithms***
by T. H. Cormen et al.
3rd Edition
MIT Press, 2009