Edsger Wybe Dijkstra
(1930 – 2002)

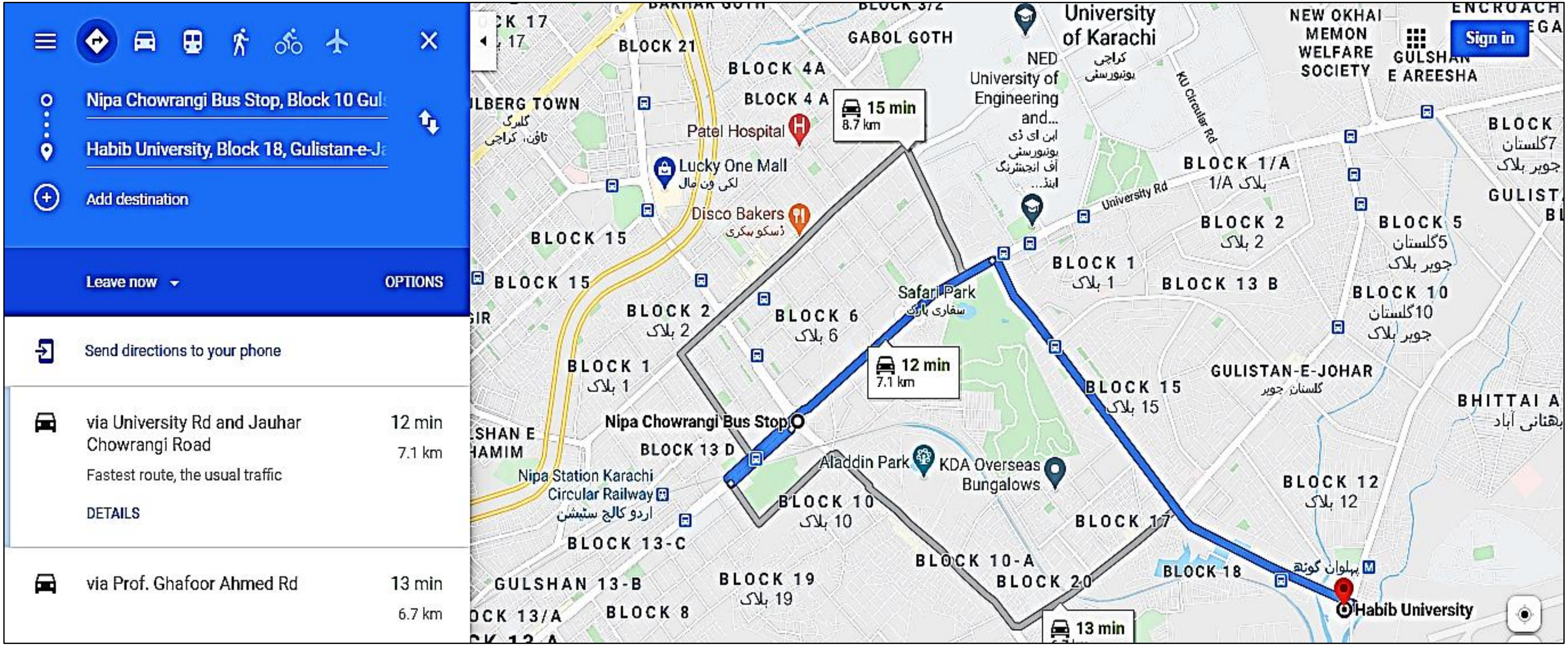
Greedy Algorithms:

A Recap of Dijkstra’s Algorithm and MST

Shah Jamal Alam

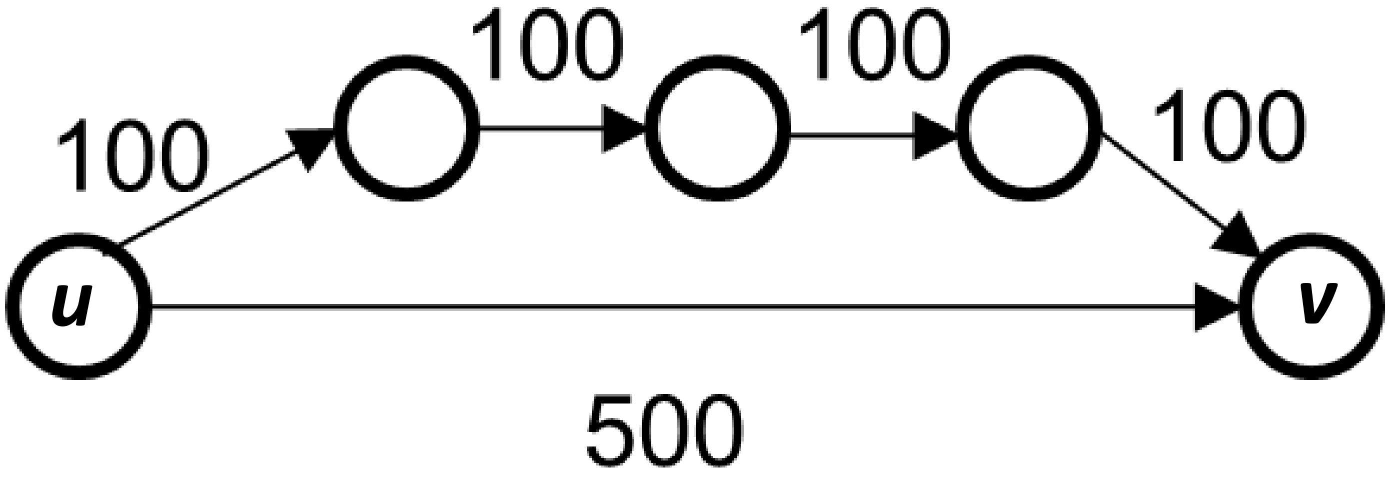
Single-source shortest path in a graph with non-negative weights

A single-pair shortest path problem on a weighted directed graph

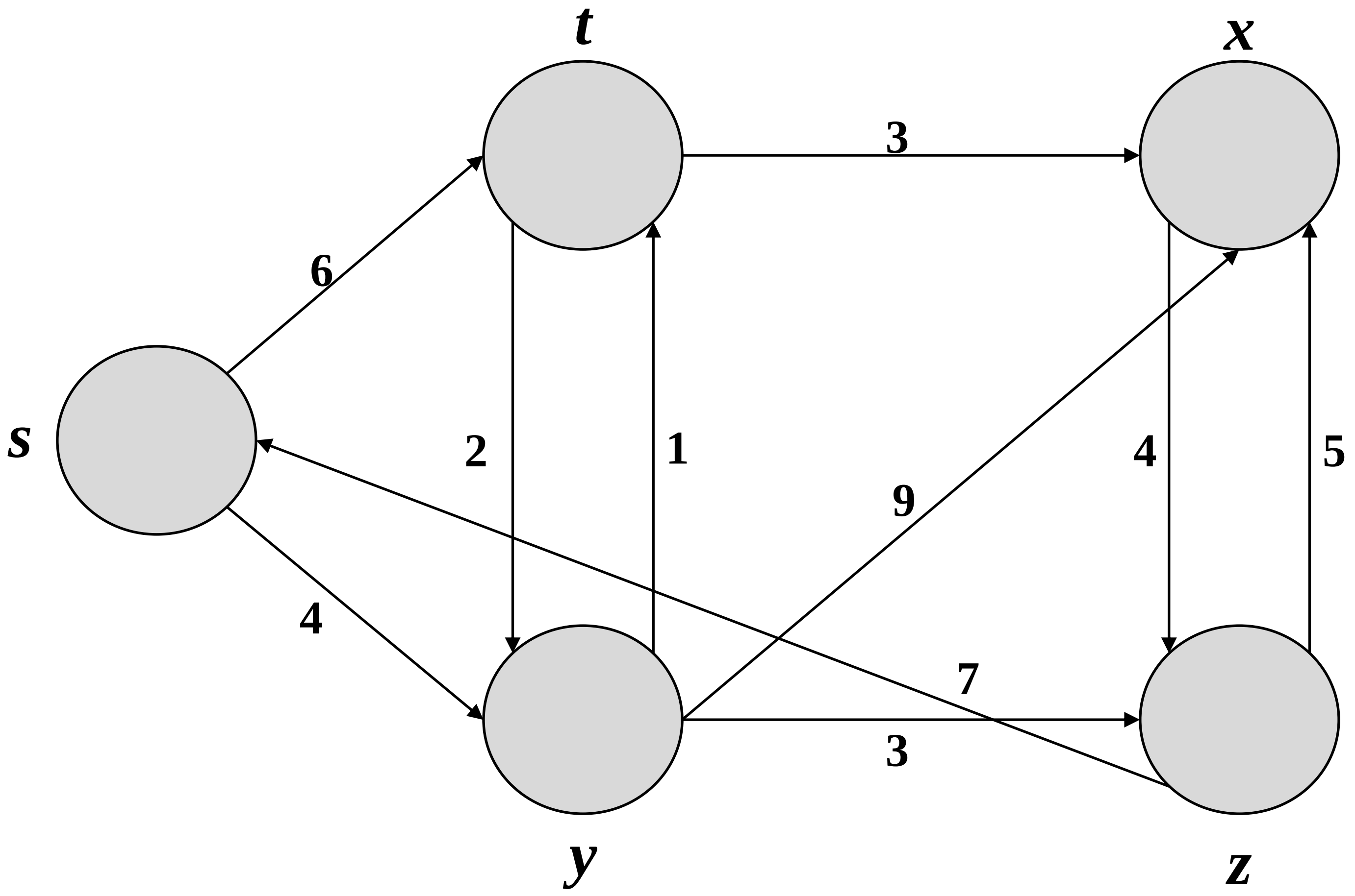


Dijkstra's algorithm: the premise

- Named after **Edsger Wybe Dijkstra** (1930 – 2002)
- Used for finding **shortest path** from a single-source vertex to all other vertices in a graph with weighted edges.
- All edge weights **must be nonnegative**.
- The underlying graph may contain cycles.

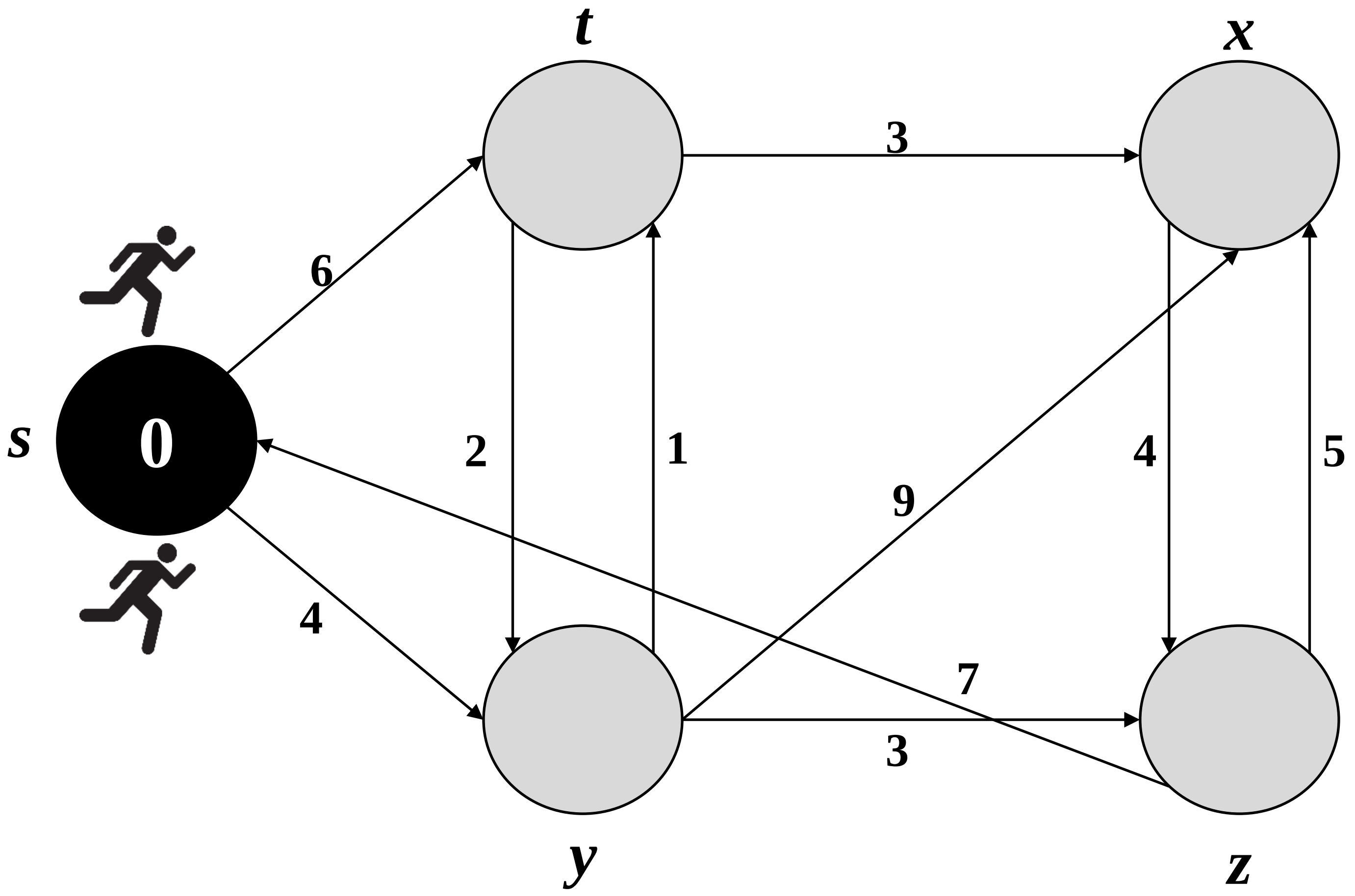


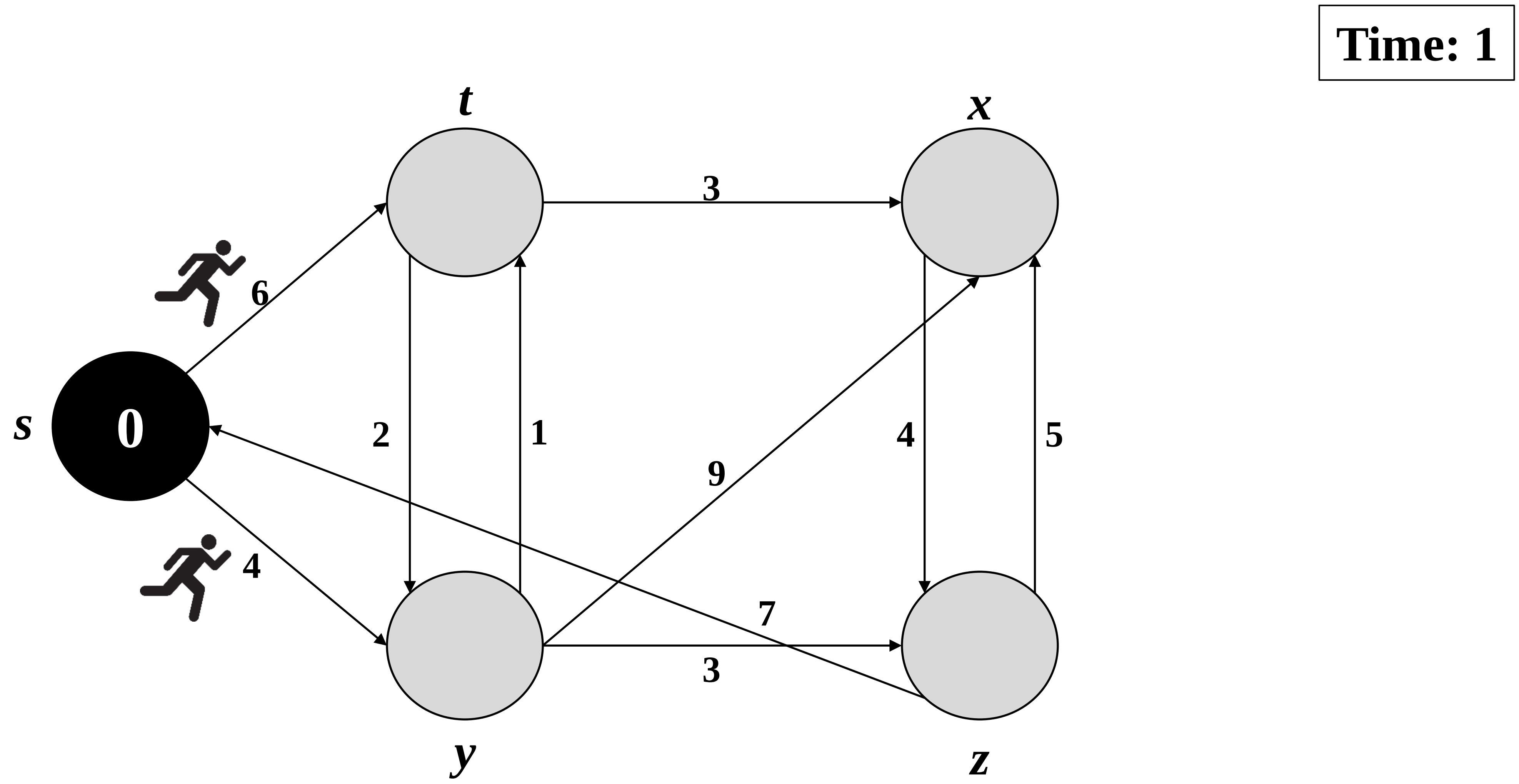
Finding Single-source Shortest Path:
A discrete event simulation



From *CLRS*

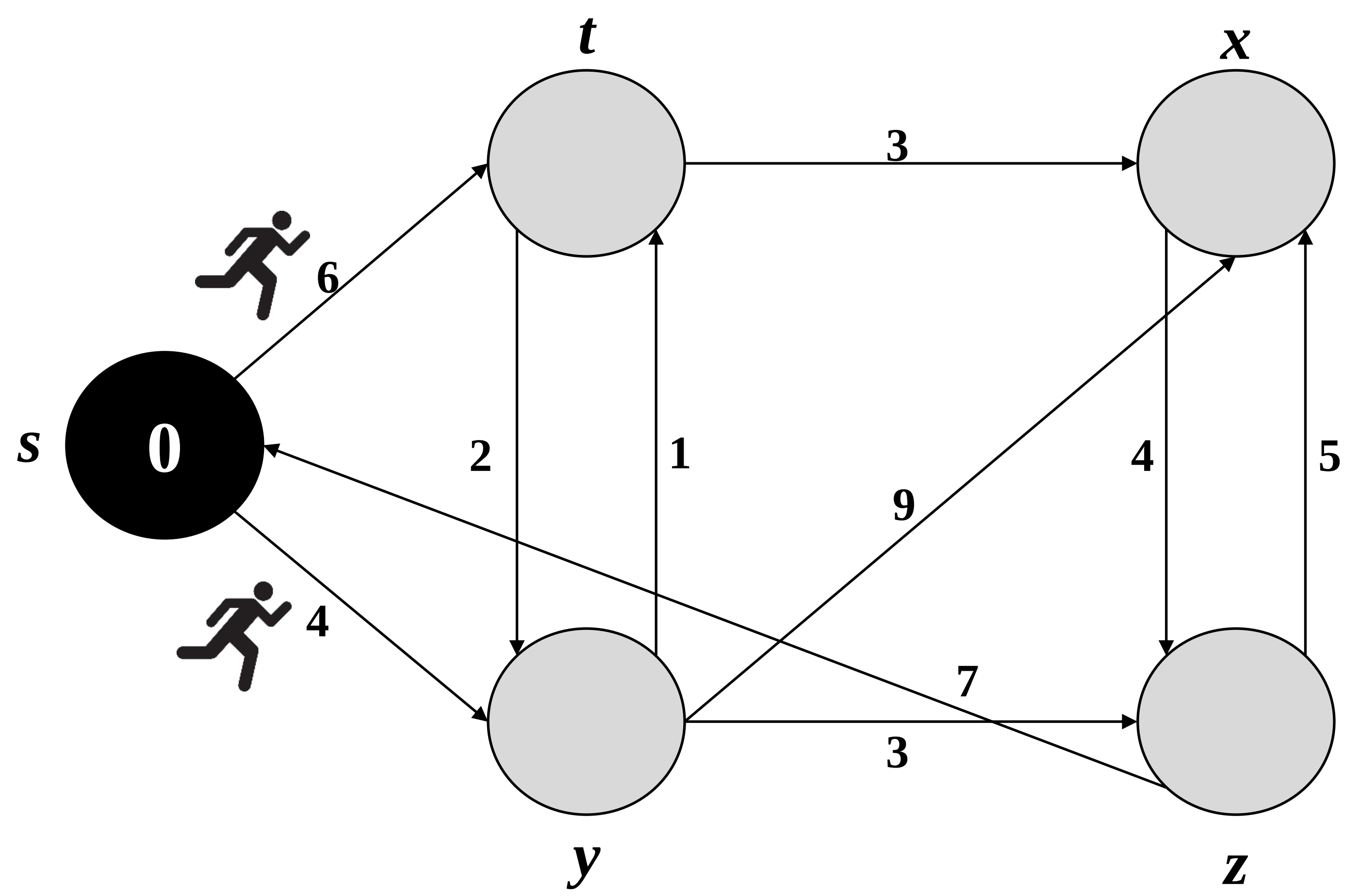
Time: 0

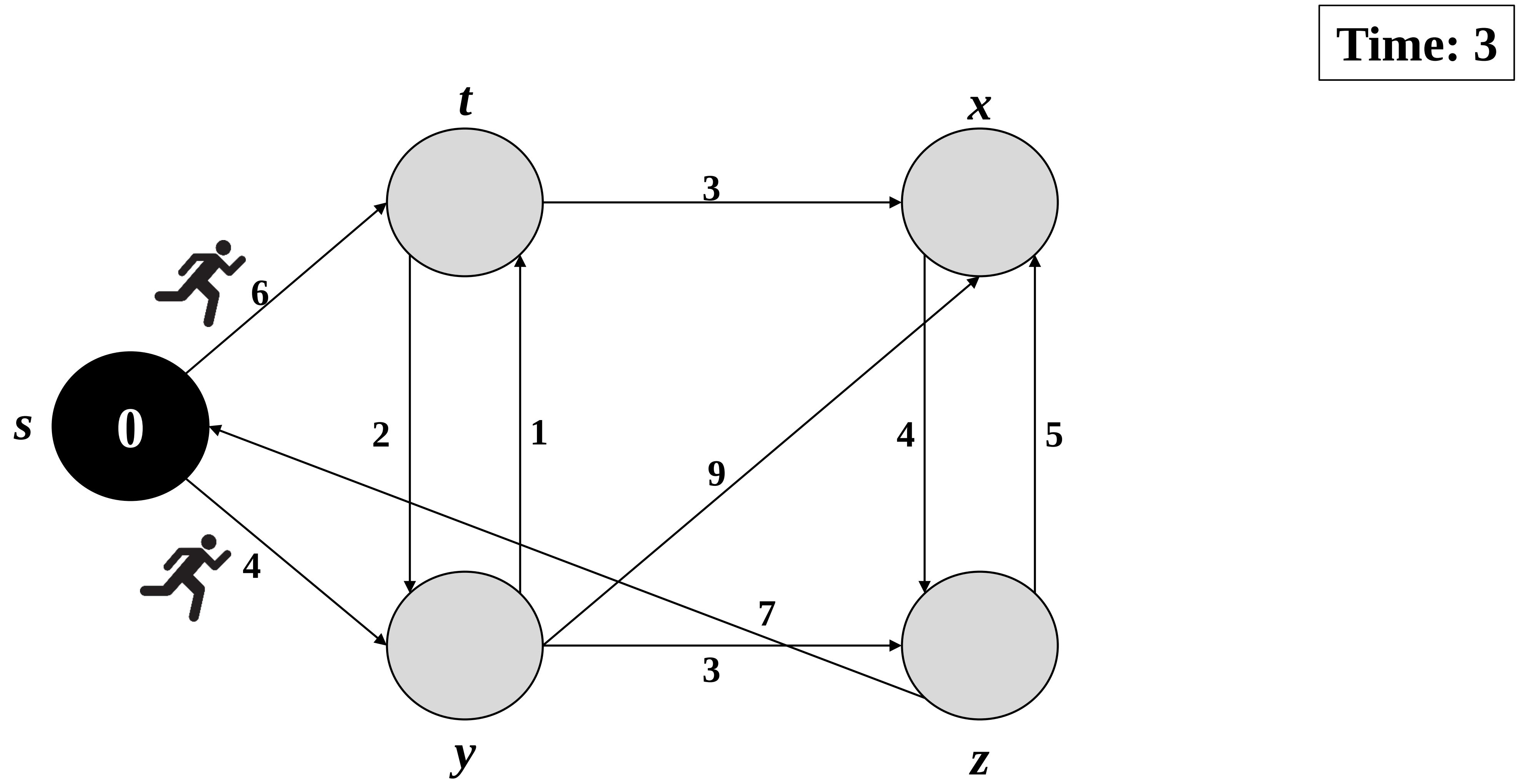


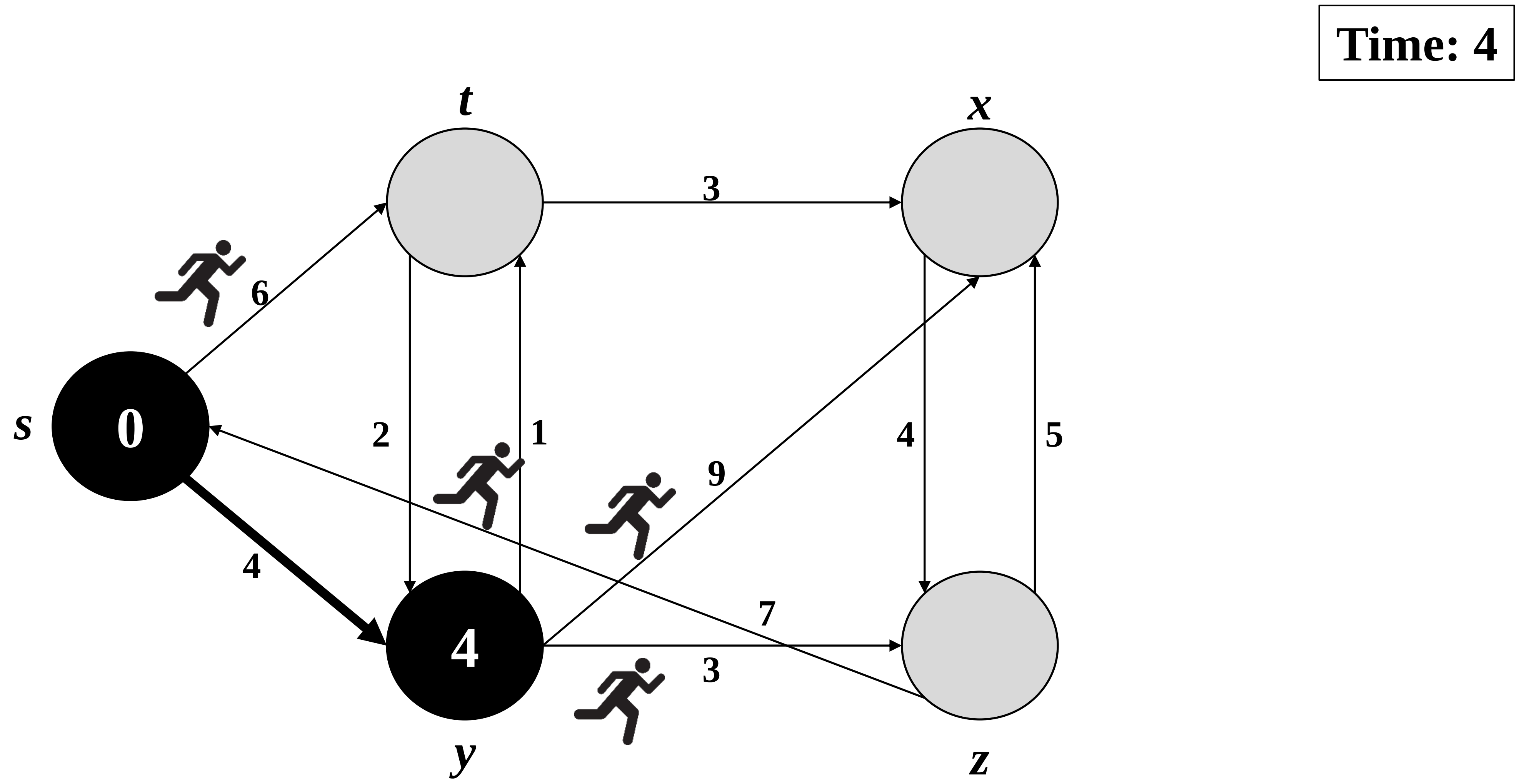


Time: 1
t
x
s
0
y
z
6
4
3
2
1
9
4
5
7
3

Time: 2

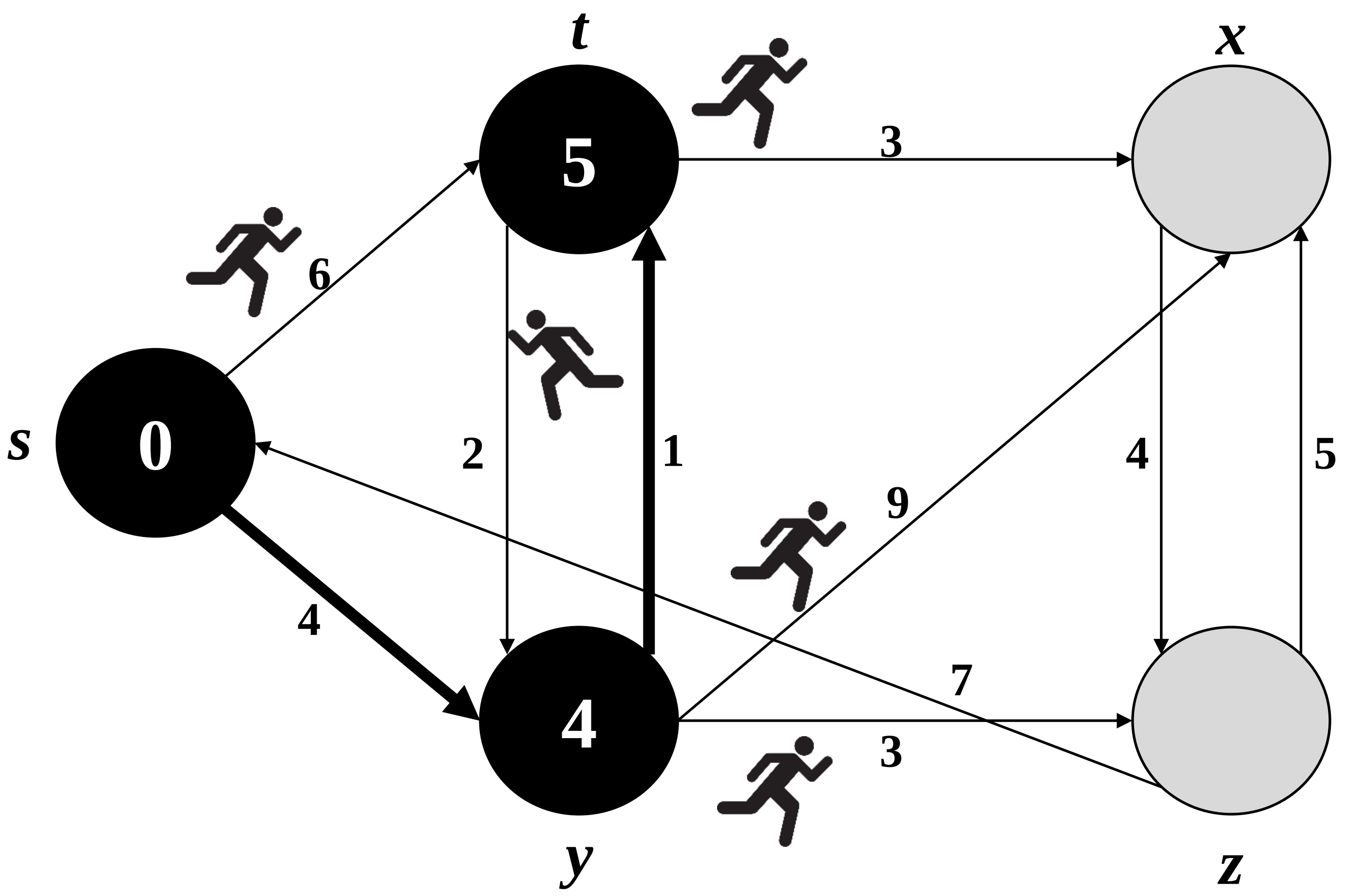


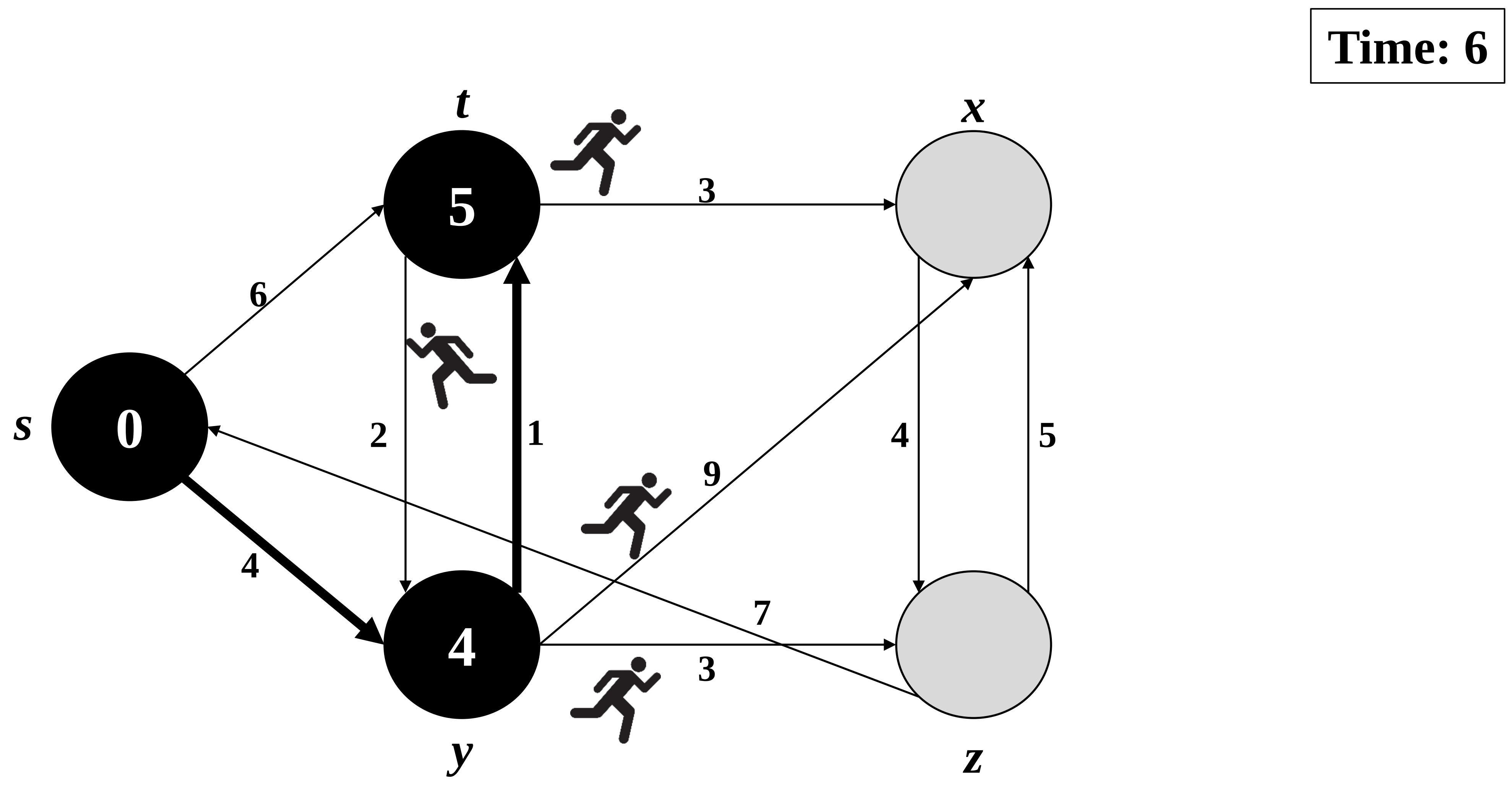

Time: 3
t
x
s
0
y
z
6
4
3
2
1
9
4
5
7
3



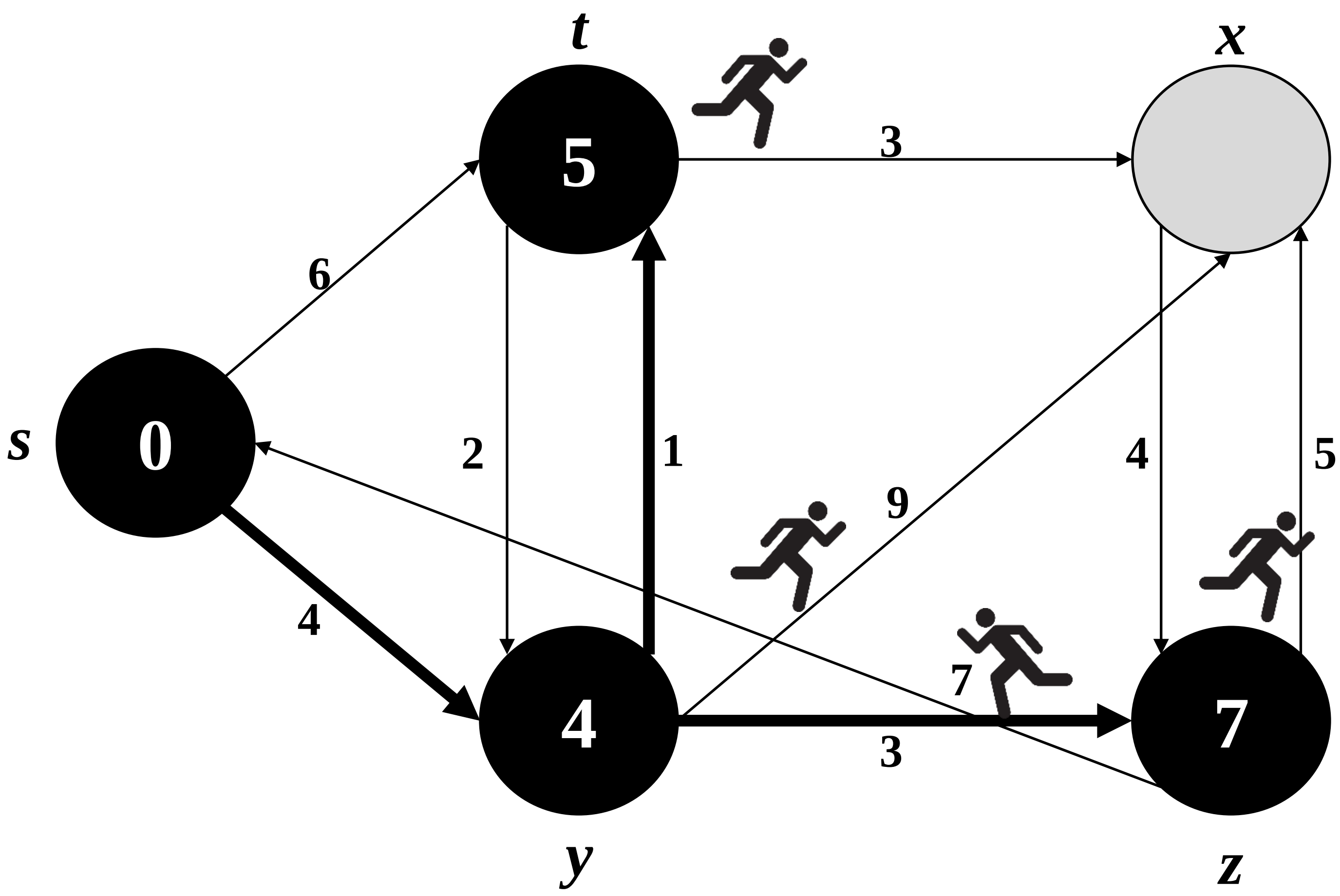
Time: 4
t
x
3
6
s
0
2
1
4
5
9
4
7
4
3
y
z

Time: 5

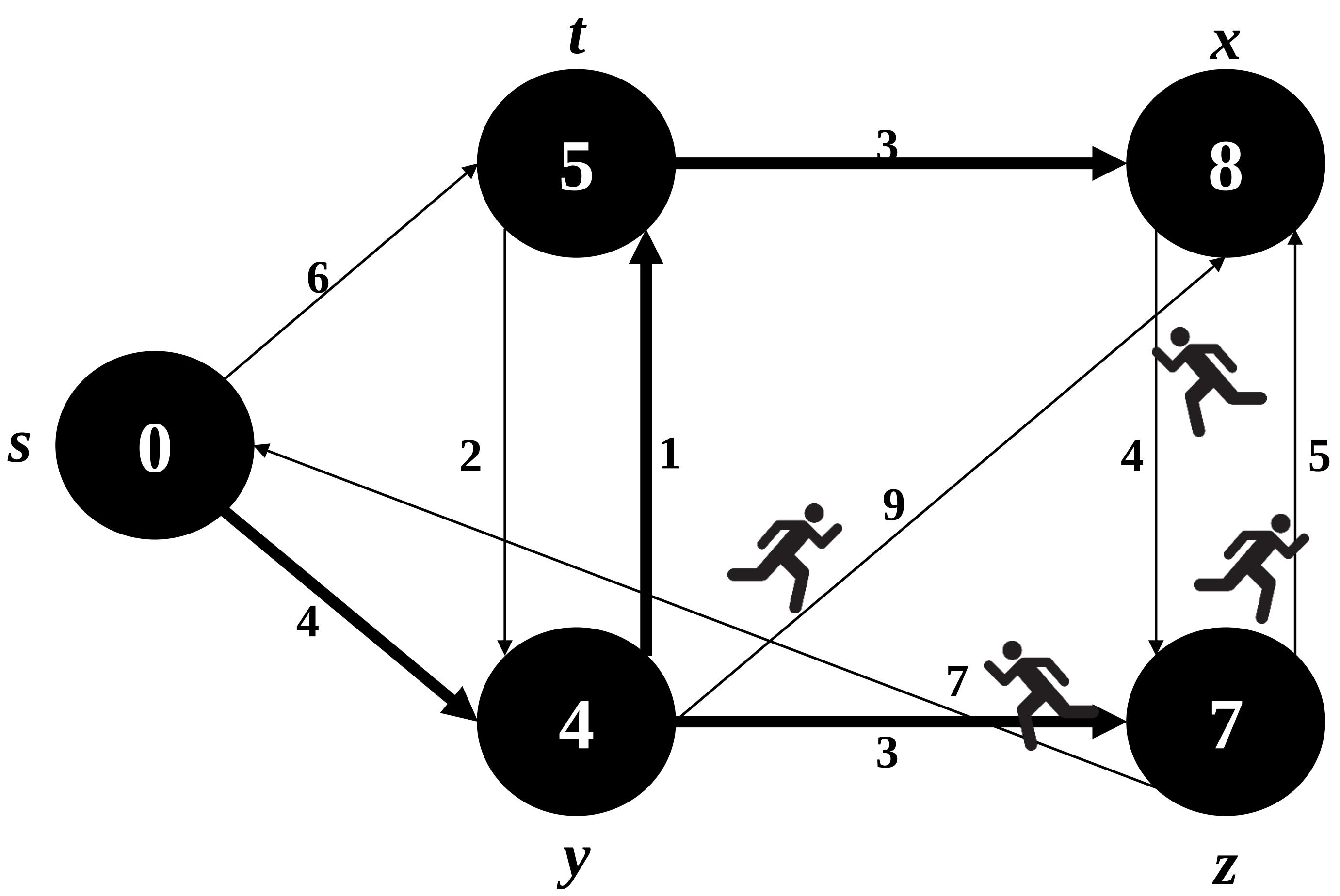



Time: 6
t
x
s
y
z
5
0
4
6
3
2
1
9
4
5
4
7
3

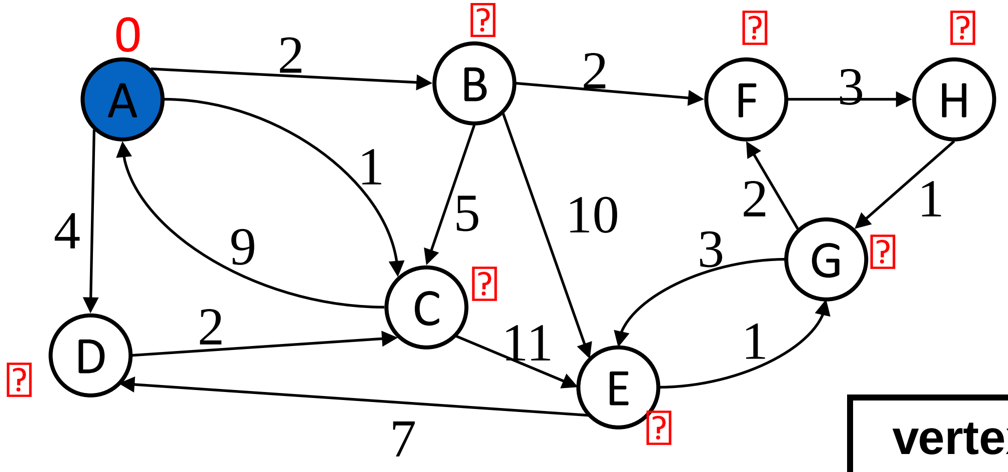
Time: 7



Time: 8



Dijkstra's algorithm: an example

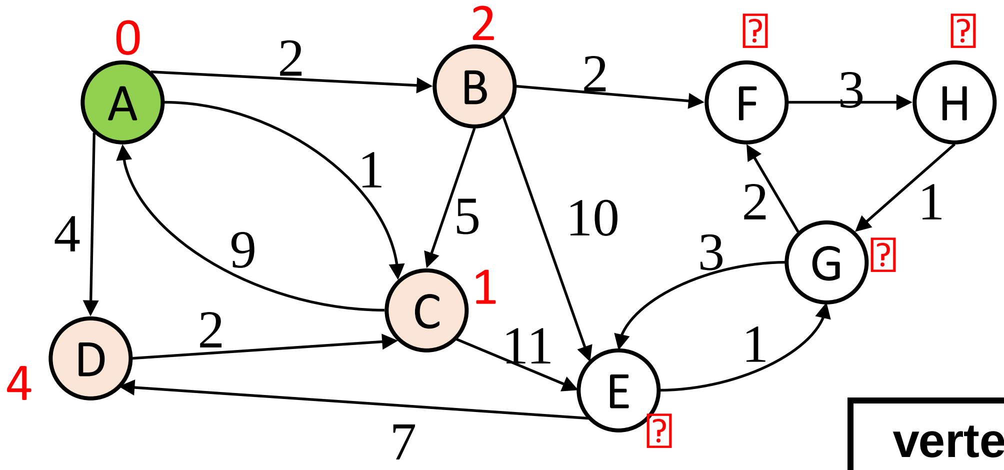


vertex	known?	cost	path
A		0	
B		??	
C		??	
D		??	
E		??	
F		??	
G		??	
H		??	

Order Added to Known Set:

Source: TBA

Dijkstra's algorithm: an example



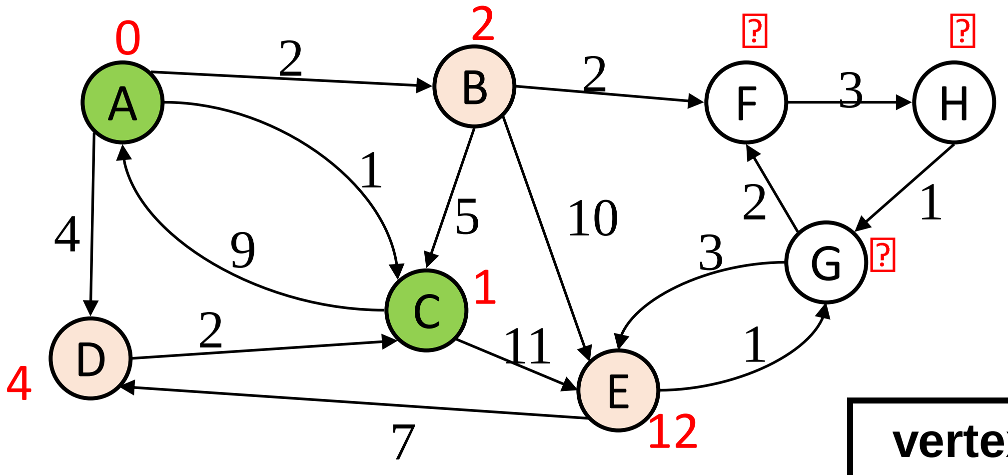
vertex	known?	cost	path
A	Yes	0	
B		≤ 2	A
C		≤ 1	A
D		≤ 4	A
E		??	
F		??	
G		??	
H		??	

Order Added to Known Set:

A

Source: TBA

Dijkstra's algorithm: an example



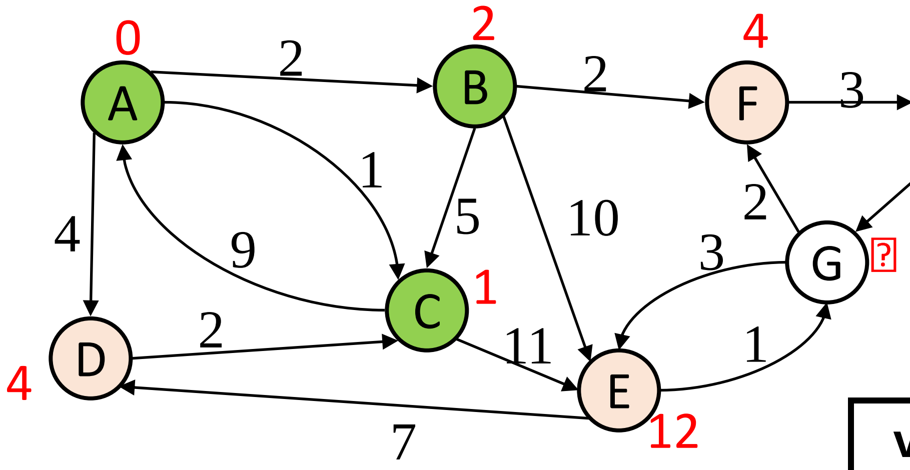
vertex	known?	cost	path
A	Yes	0	
B		≤ 2	A
C	Yes	1	A
D		≤ 4	A
E		≤ 12	C
F		??	
G		??	
H		??	

<u>Order Added to Known Set:</u>

A, C

Source: TBA

Dijkstra's algorithm: an example



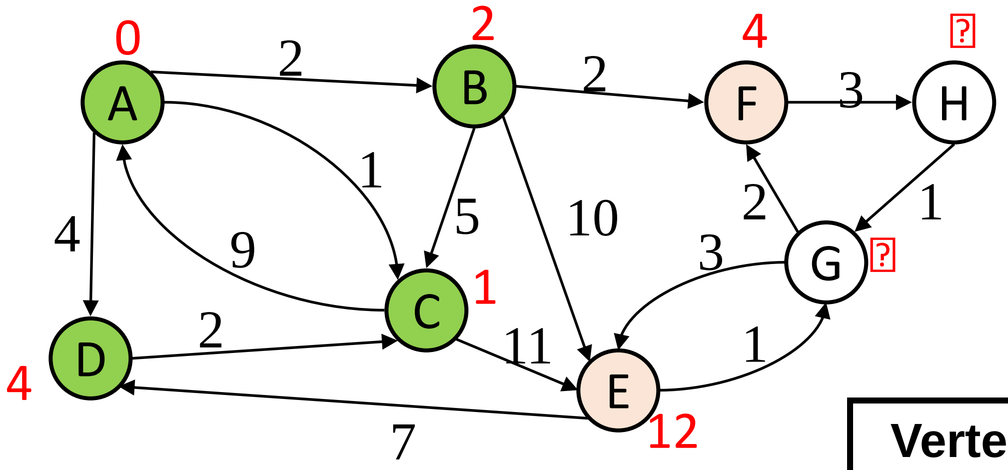
vertex	known?	cost	path
A	Yes	0	
B	Yes	2	A
C	Yes	1	A
D		≤ 4	A
E		≤ 12	C
F		≤ 4	B
G		??	
H		??	

<u>Order Added to Known Set:</u>

A, C, B

Source: TBA

Dijkstra's algorithm: an example



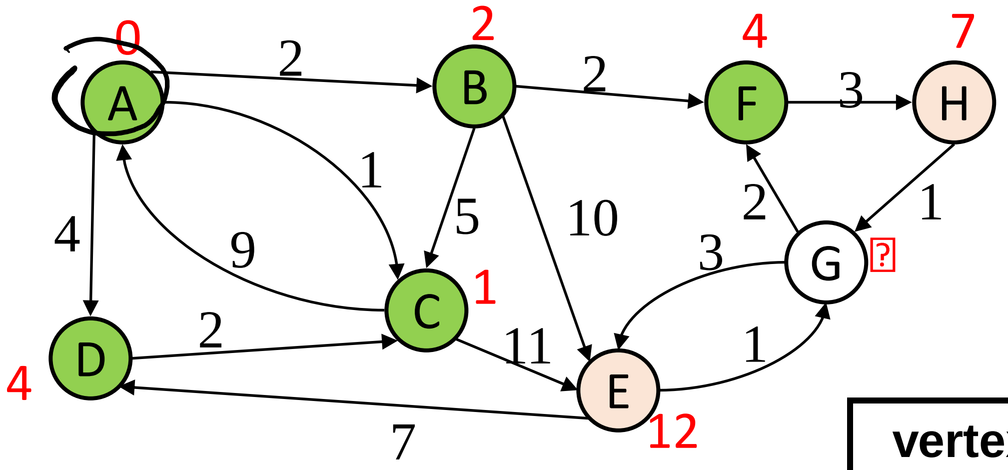
Vertex	known?	cost	path
A	Yes	0	
B	Yes	2	A
C	Yes	1	A
D	Yes	4	A
E		≤ 12	C
F		≤ 4	B
G		??	
H		??	

<u>Order Added to Known Set:</u>

A, C, B, D

Source: TBA

Dijkstra's algorithm: an example



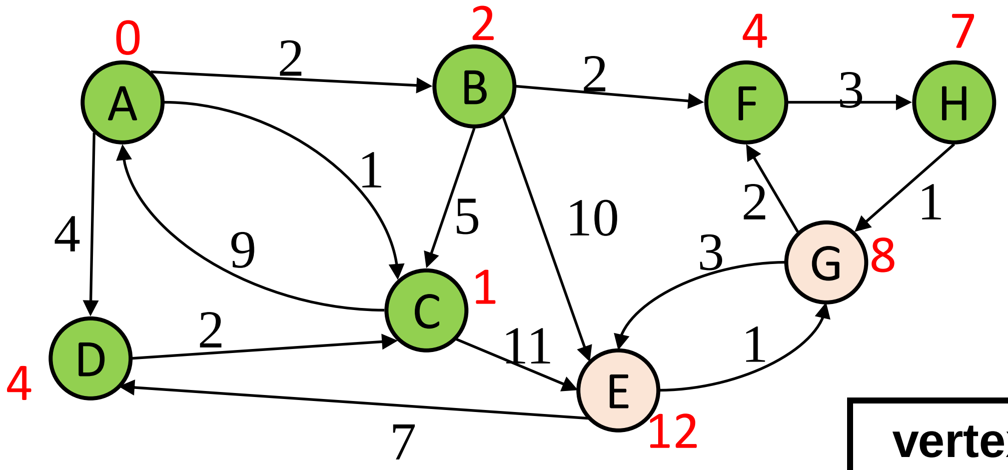
vertex	known?	cost	path
A	Yes	0	
B	Yes	2	A
C	Yes	1	A
D	Yes	4	A
E		≤ 12	C
F	Yes	4	B
G		??	
H		≤ 7	F

Order Added to Known Set:

A, C, B, D, F

Source: TBA

Dijkstra’s algorithm: an example



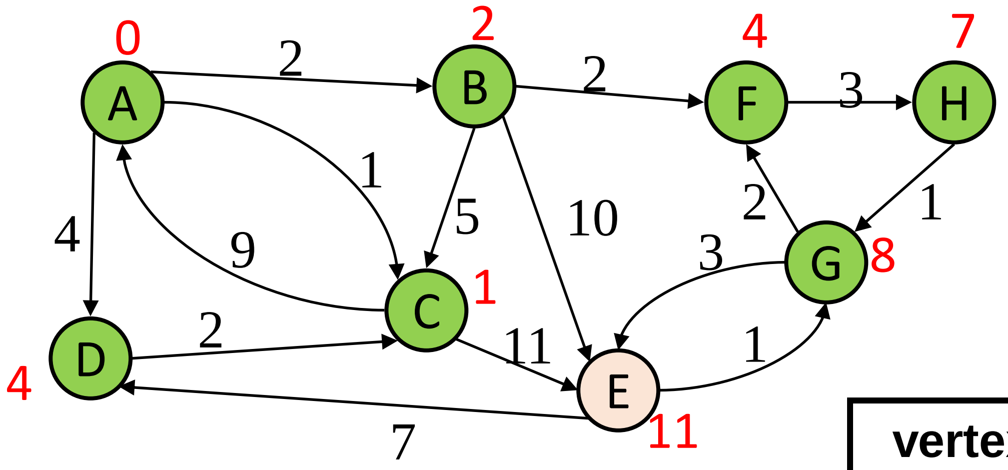
vertex	known?	cost	path
A	Yes	0	
B	Yes	2	A
C	Yes	1	A
D	Yes	4	A
E		$\leq$ 12	C
F	Yes	4	B
G		$\leq$ 8	H
H	Yes	7	F

<u>Order Added to Known Set:</u>

A, C, B, D, F, H

Source: TBA

Dijkstra's algorithm: an example



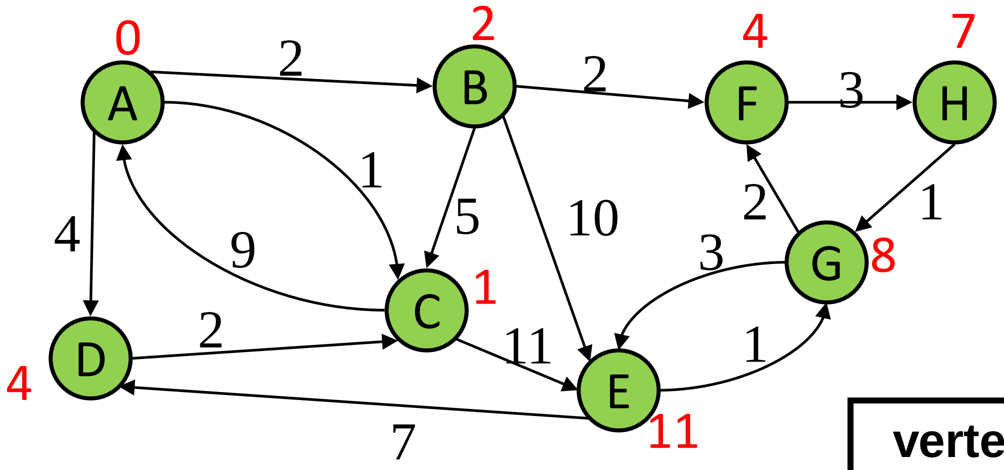
vertex	known?	cost	path
A	Yes	0	
B	Yes	2	A
C	Yes	1	A
D	Yes	4	A
E		**≤ 11**	**G**
F	Yes	4	B
G	Yes	8	H
H	Yes	7	F

Order Added to Known Set:

A, C, B, D, F, H, G

Source: TBA

Dijkstra's algorithm: an example



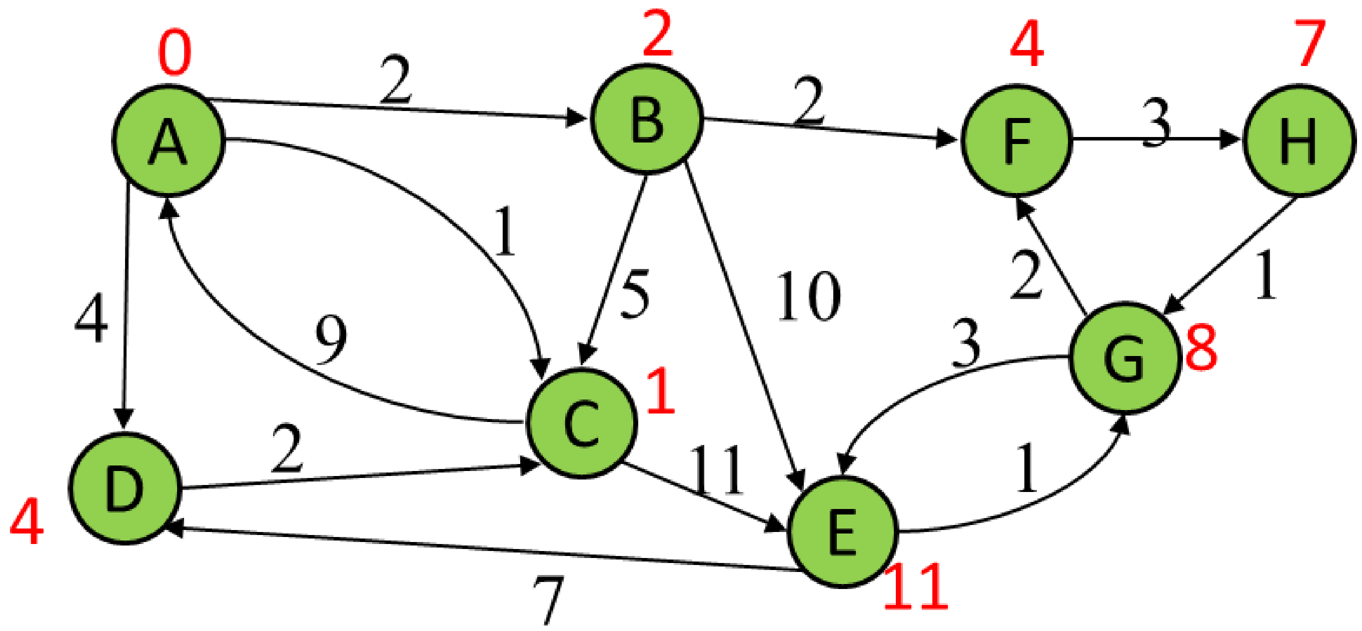
vertex	known?	cost	path
A	Yes	0	
B	Yes	2	A
C	Yes	1	A
D	Yes	4	A
E	Yes	11	G
F	Yes	4	B
G	Yes	8	H
H	Yes	7	F

Order Added to Known Set:

A, C, B, D, F, H, G, E

Source: TBA

Dijkstra’s algorithm: the skeleton



Initially, source node has cost **0** and all other nodes have cost ∞.

At each step:

- Pick **closest** unknown vertex ***v***.
- Add it to the ‘set’ of known vertices.
- Update distances for nodes with edges from ***v***.

Source: TBA

Dijkstra's algorithm: key features

- Reminiscent of BFS, but adapted to handle weights.
- Grow the set of nodes whose shortest distance has been computed.
- Nodes not in the set will have a **"best distance so far"**.
- When a vertex is marked **known**, the cost of the shortest path to that node is known [the path is known as well].
- While a vertex is still **not known**, another shorter path to it might still be found.
- To keep track of which vertex should be added next in the **known set**, we use a **priority queue**. Each of the |V| vertices will need to be added into the priority queue.

Dijkstra's algorithm: the pseudocode

```
1. For each node v, set  v.cost = ∞ and v.known = false.

2. Set source.cost = 0.

3. While there are unknown nodes in the graph
   a) Select the unknown node v with the lowest cost.
   b) Mark v as known.
   c) For each edge (v, u) with weight w,
           c1 ← v.cost + w     # cost of best path through v to u.
             c2 ← u.cost       # cost of best path to u previously known.
             if c1 < c2        # if the path through v is lesser cost.
              u.cost ← c1
                 u.path ← v # for computing actual paths.
```

Source: TBA

Dijkstra's algorithm: the pseudocode

Dijkstra$(\boldsymbol{G}, \boldsymbol{w}, \boldsymbol{s})$**:**

InitializeSingleSource(G, s)

$S = \emptyset$ *#A set of vertices whose final shortest path from* $\boldsymbol{s}$ *is determined.*

$Q = G.V$ *#Inserting all vertices.*

while $Q \neq \emptyset$

$\quad u \leftarrow ExtractMin(Q)$ *#Remove from the priority queue.*

$\quad S = S \cup \{u\}$

$\quad$ **for** each vertex $v \in G.Adj[u]$ *#for each edge* $O(E)$

$\quad\quad Relax(u, v, w)$ *#Decrease-key operation.*

Source: *CLRS.*

The Priority Queue ADT

The key operations of the priority queue ADT are:

- **Insert** (Q, v): insert an element *v* (vertex) into the set *Q*.
- **Extract_Min** (Q): remove the element (vertex) in *Q* with the minimum (shortest) value and return this element (vertex) to the caller.
- **Decrease_Key** (Q, v): record that the ‘shortest’ value associated with an element (vertex) *v* is decreased (updated).

The Priority Queue ADT can be implemented using, e.g., a **simple array**, a **binary heap**, or a **Fibonacci heap** data structures.

Binary Heap (a recap?)

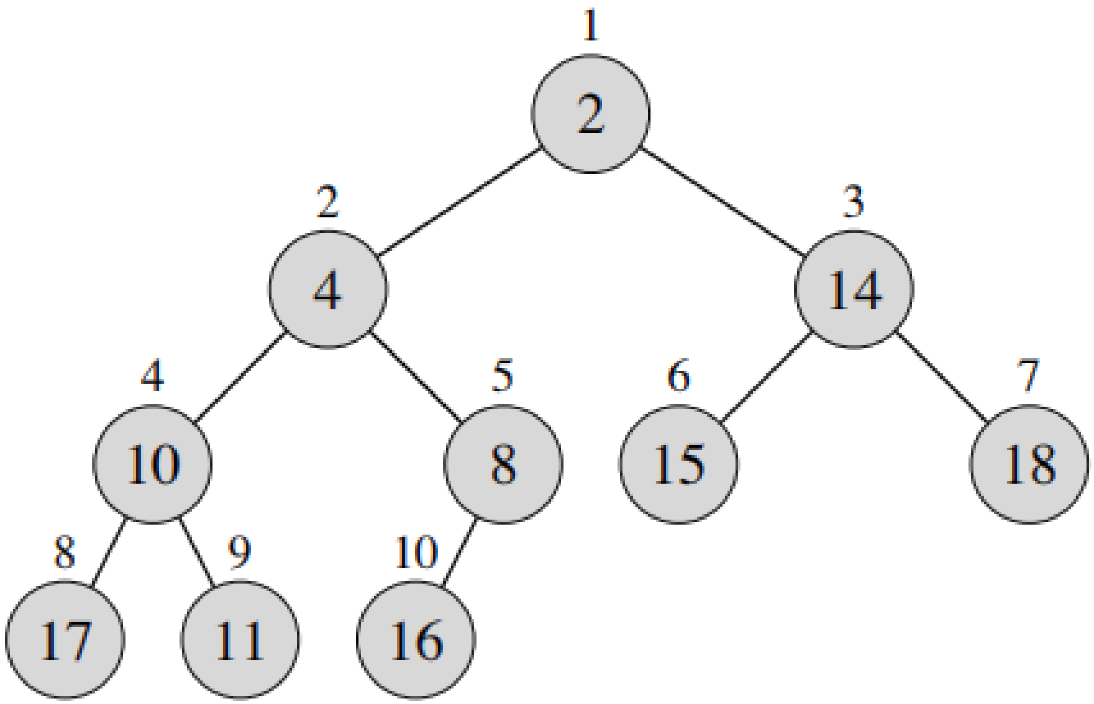
- A full binary tree (i.e., completely filled except possibly the last level) where a new node is added from left to first available slot.
- The **heap property** must be satisfied.
- The **min-heap** (we can define max-heap) property is given as follows:

For every node i except the $root$, $parent(i)._{value} \leq i._{value}$

i.e., the value of every node must not be greater than its children.

➔ The smallest element in min-heap is at the root.

Binary Heap (a recap?)



1	2	3	4	5	6	7	8	9	10
2	4	14	10	8	15	18	17	11	16

All three Priority Queue operations can be performed in $O(\lg n)$ time using binary heaps.

Dijkstra's algorithm: Analysis

Dijkstra's algorithm relies on a **priority queue**, the implementation of the data structure influences the overall performance of the algorithm.

So, let's analyze the Binary Heap ADT operations [see the handwritten notes].

Spanning Trees

Given an **undirected** graph, a ***spanning tree*** T is a subgraph of G, where T:

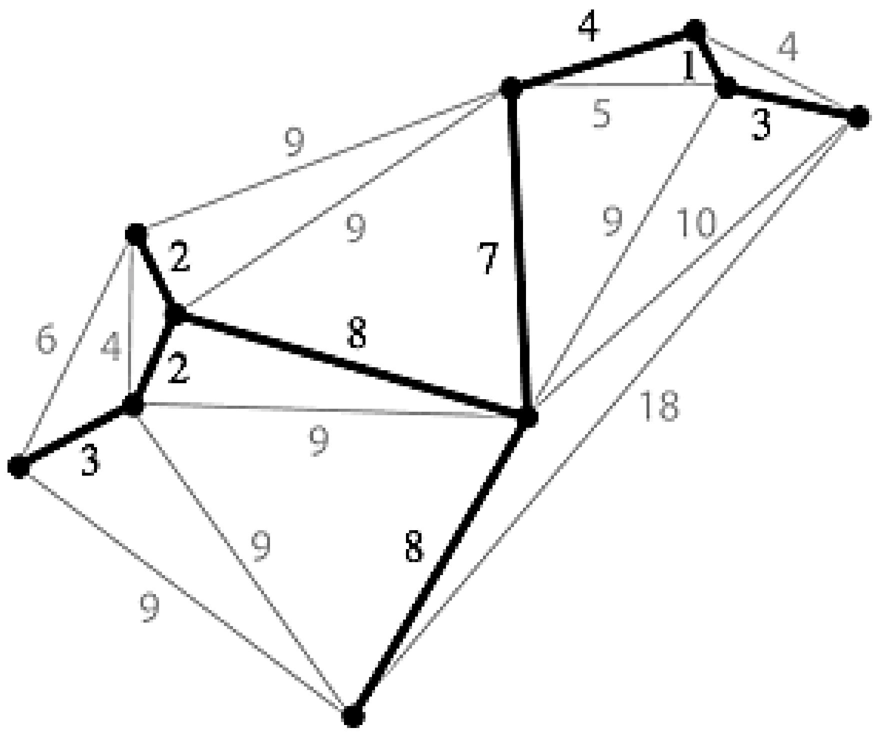
- Is connected.
- Is acyclic.
- Includes all of the vertices.

These two properties make it a tree.

This makes it spanning.

Example:

Spanning tree is the black edges and vertices.



A ***minimum spanning tree*** is a spanning tree of minimum total weight.

- Example: Directly connecting buildings by power lines.

Prim’s & Dijkstra’s

Prim’s and Dijkstra’s algorithms are same, except Dijkstra’s considers “distance from the source”, and Prim’s considers “distance from the tree.”

Visit order:

- Dijkstra’s algorithm visits vertices in order of distance from the source.
- Prim’s algorithm visits vertices in order of distance from the MST under construction.

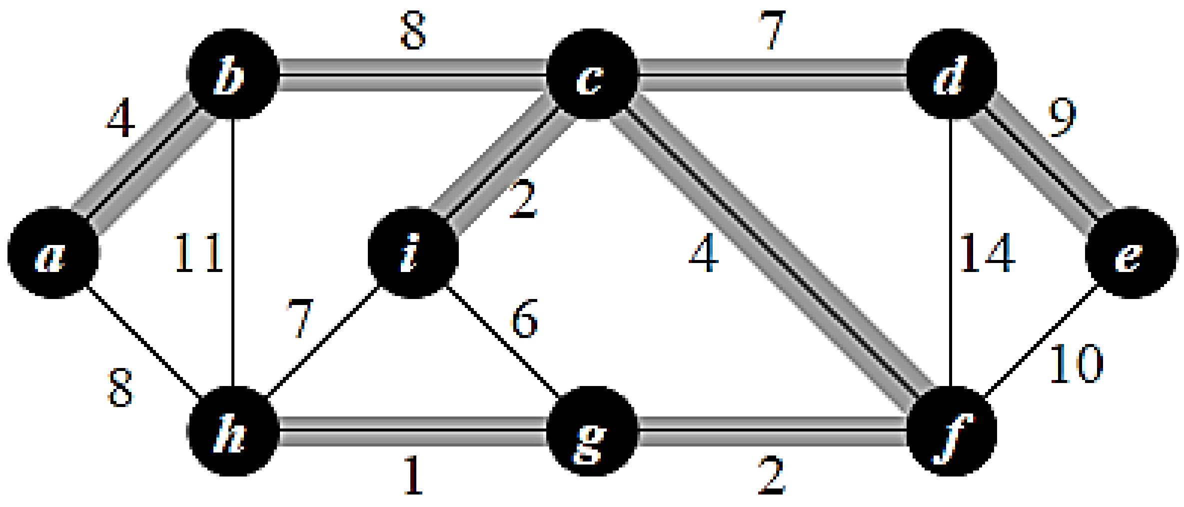
Relaxation:

- Relaxation in Dijkstra’s considers an edge better based on distance to source.
- Relaxation in Prim’s considers an edge better based on distance to tree.

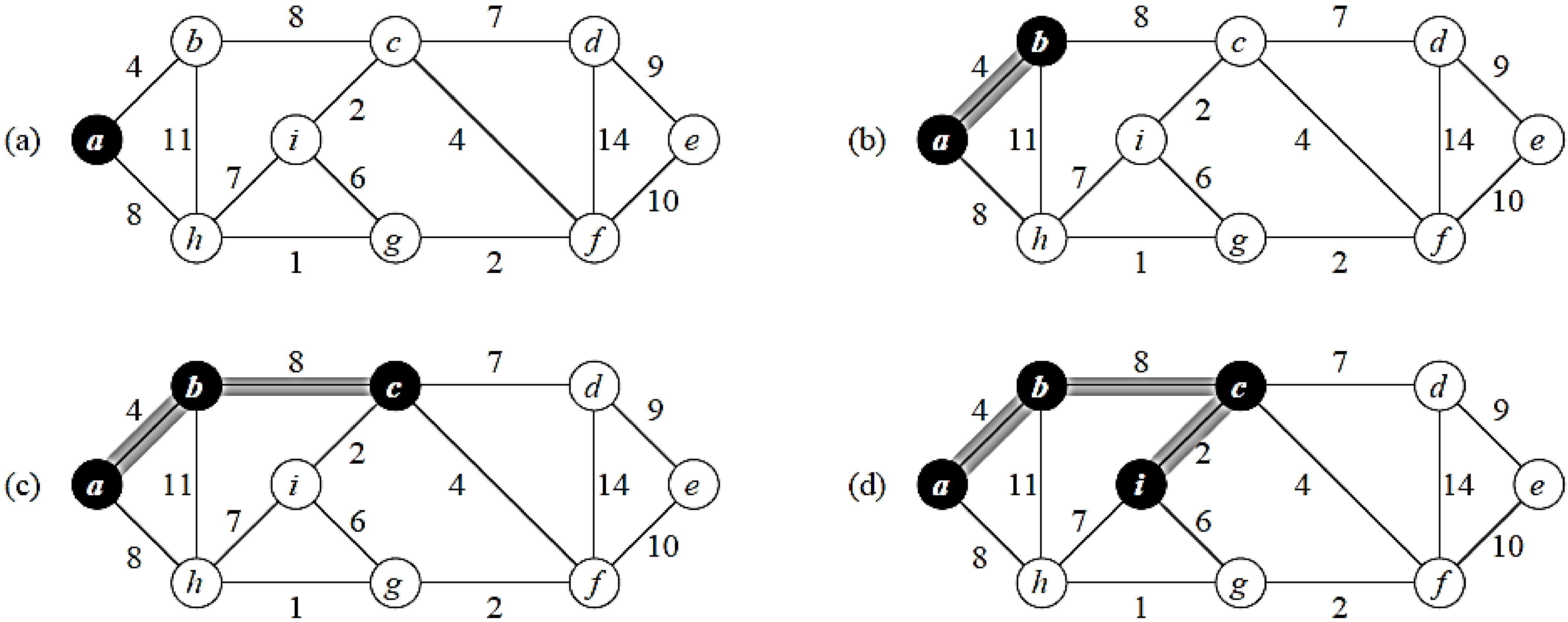
Prim's Algorithm: pseudocode

```
MST-PRIM(G, w, r)
    for each u ∈ G.V
        u.key = ∞
        u.π = NIL
    r.key = 0
    Q = G.V
    while Q ≠ ∅
        u = EXTRACT-MIN(Q)
        for each v ∈ G.Adj[u]
            if v ∈ Q and w(u, v) < v.key
                v.π = u
                v.key = w(u, v)
```

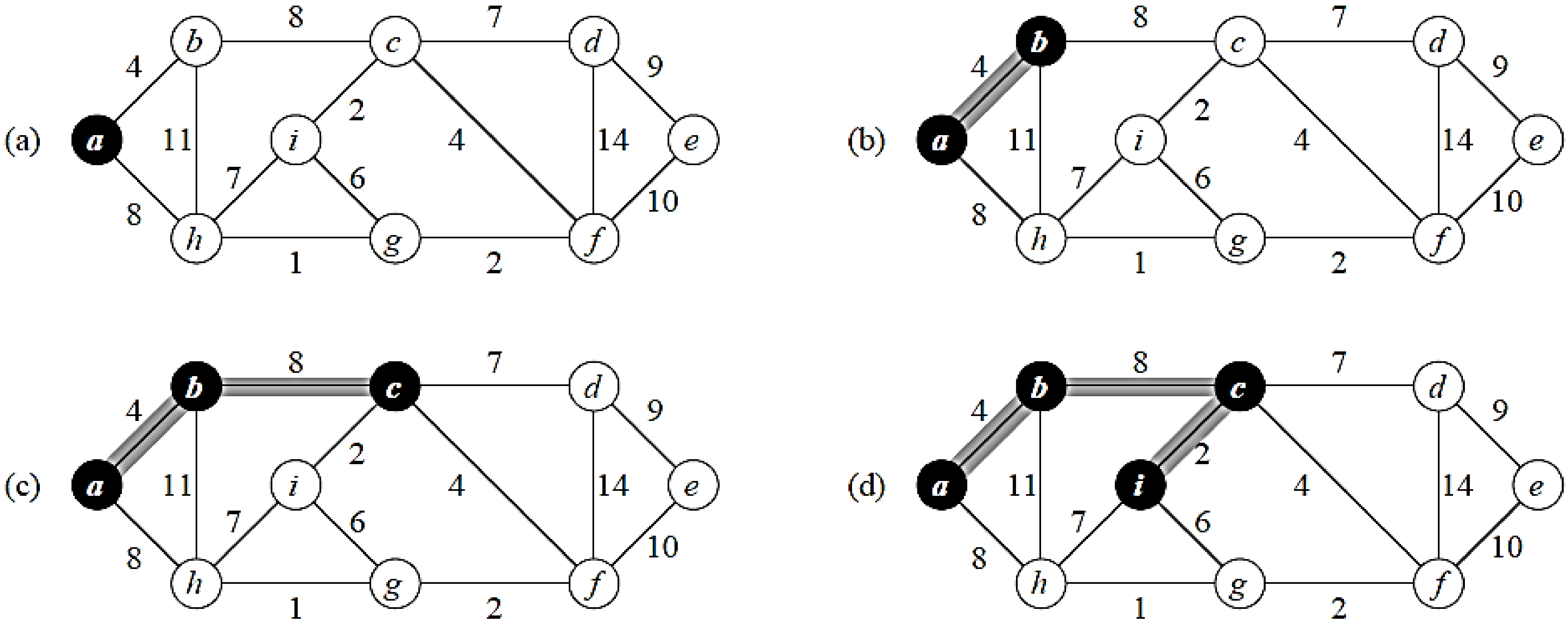
(i)



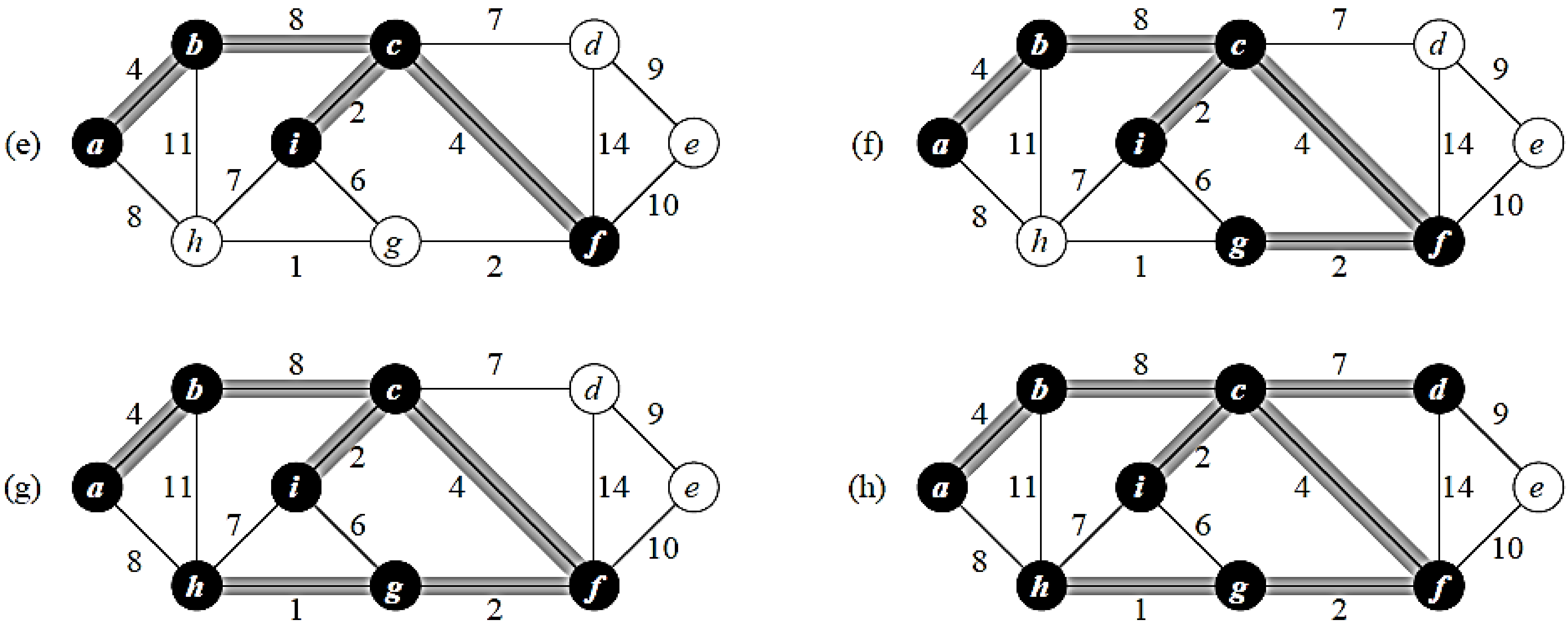
Prim’s Algorithm: an example



Prim’s Algorithm: an example

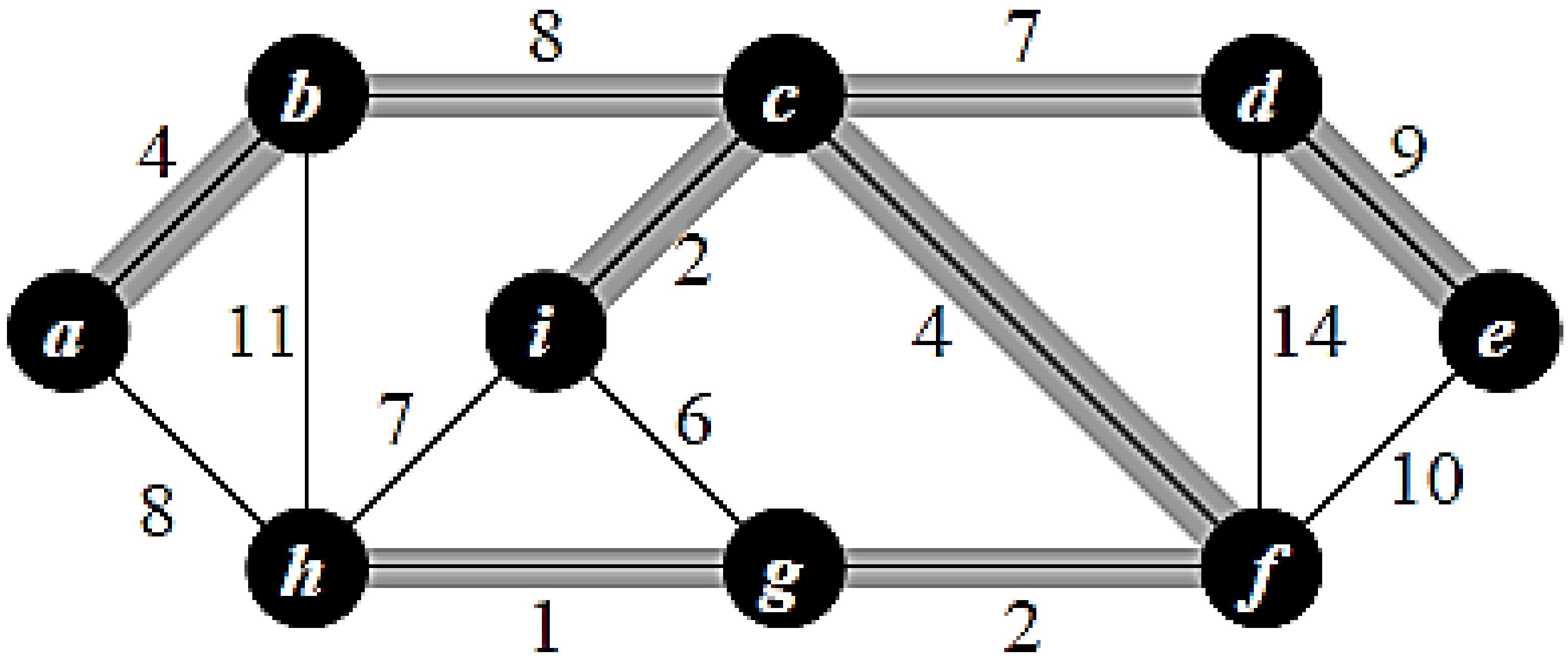


Prim’s Algorithm: an example



Prim’s Algorithm: an example

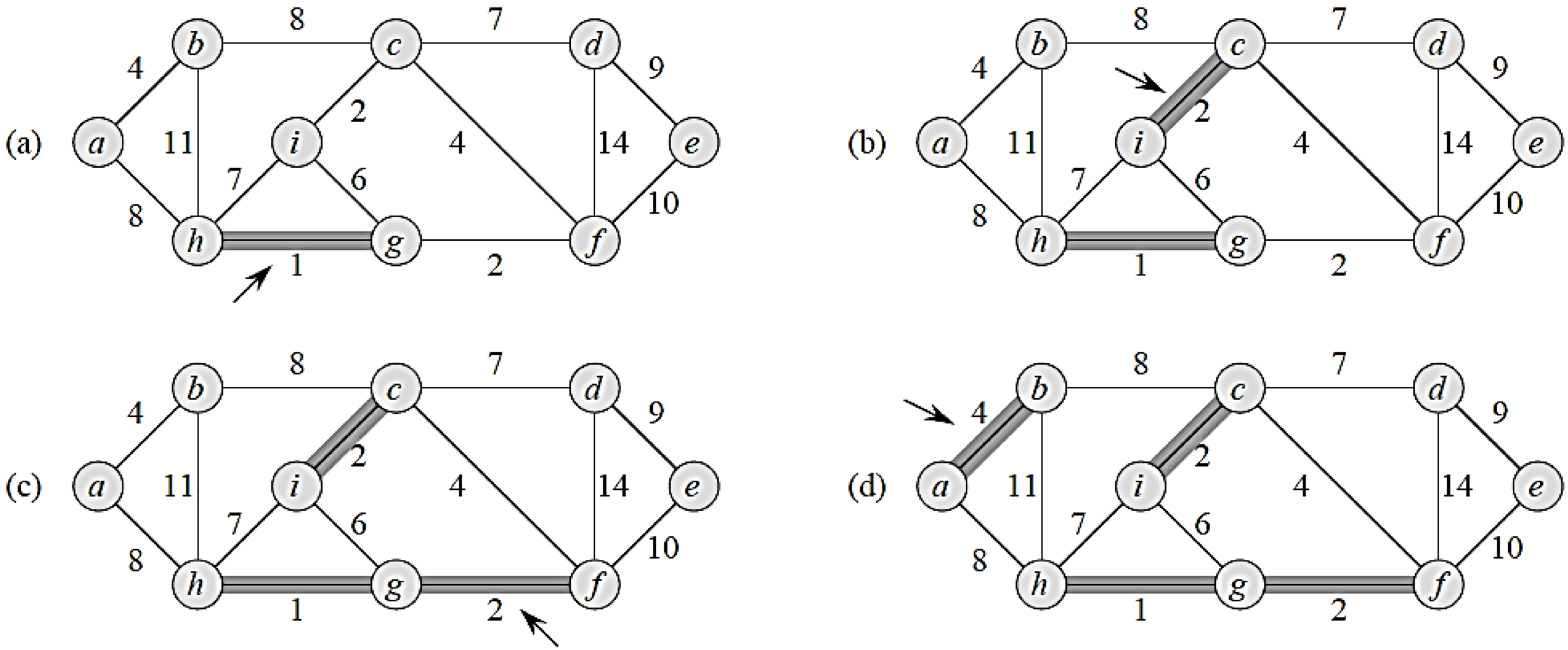
(i)



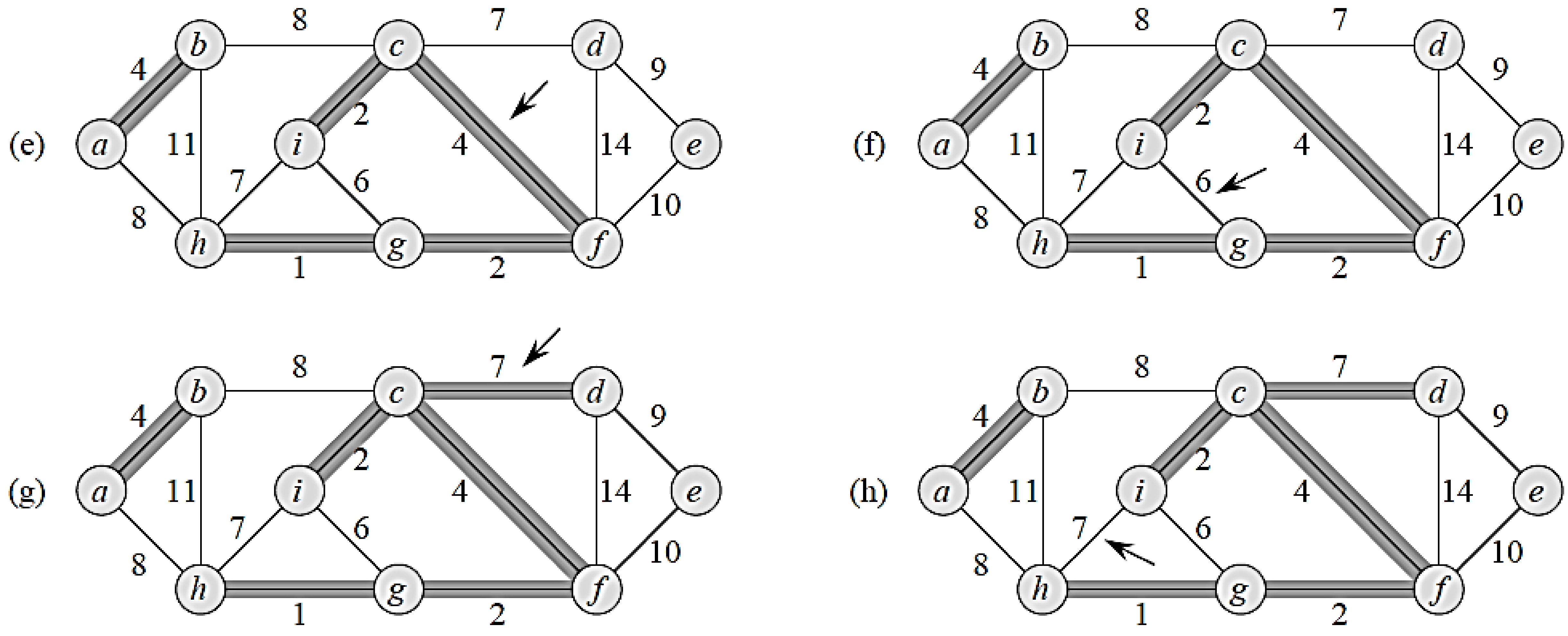
Kruskal’s Algorithm: pseudocode

```
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
```

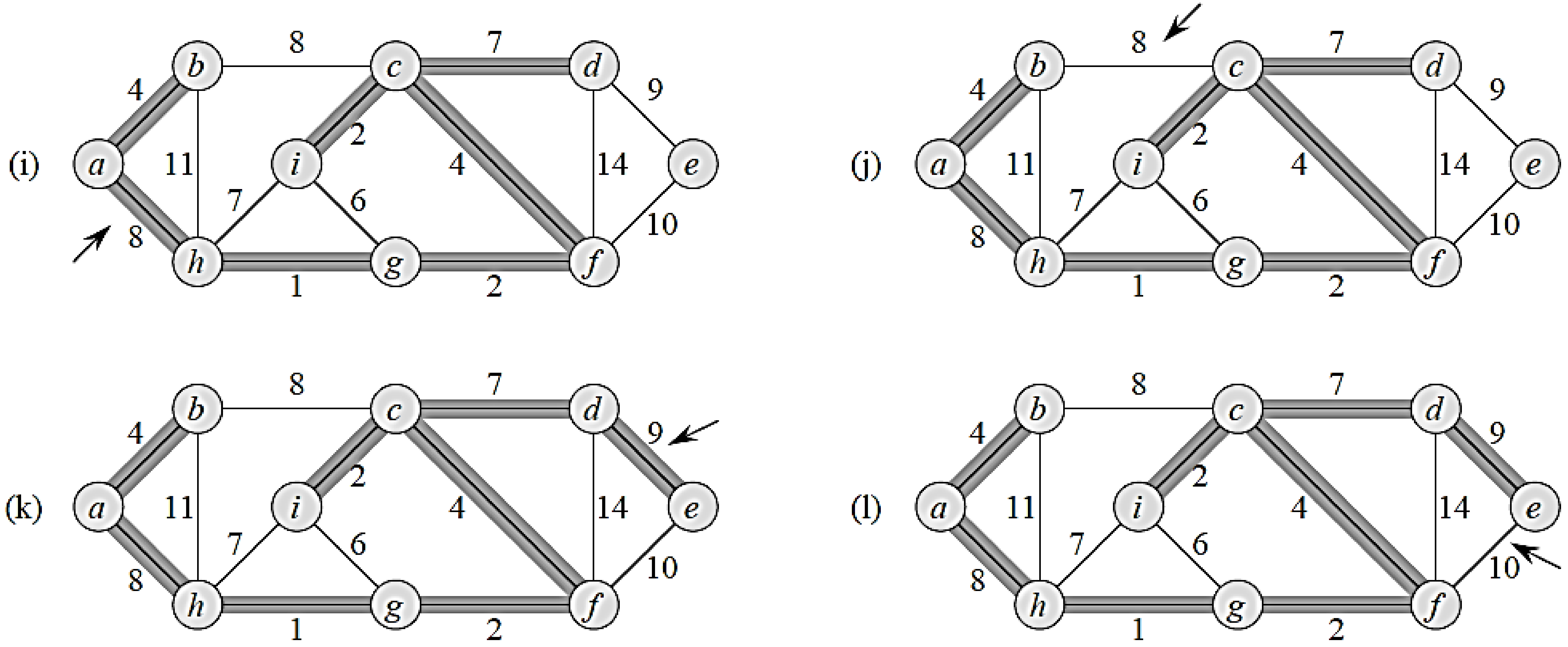
Kruskal’s Algorithm: an example



Kruskal’s Algorithm: an example



Kruskal’s Algorithm: an example



Kruskal’s Algorithm: an example