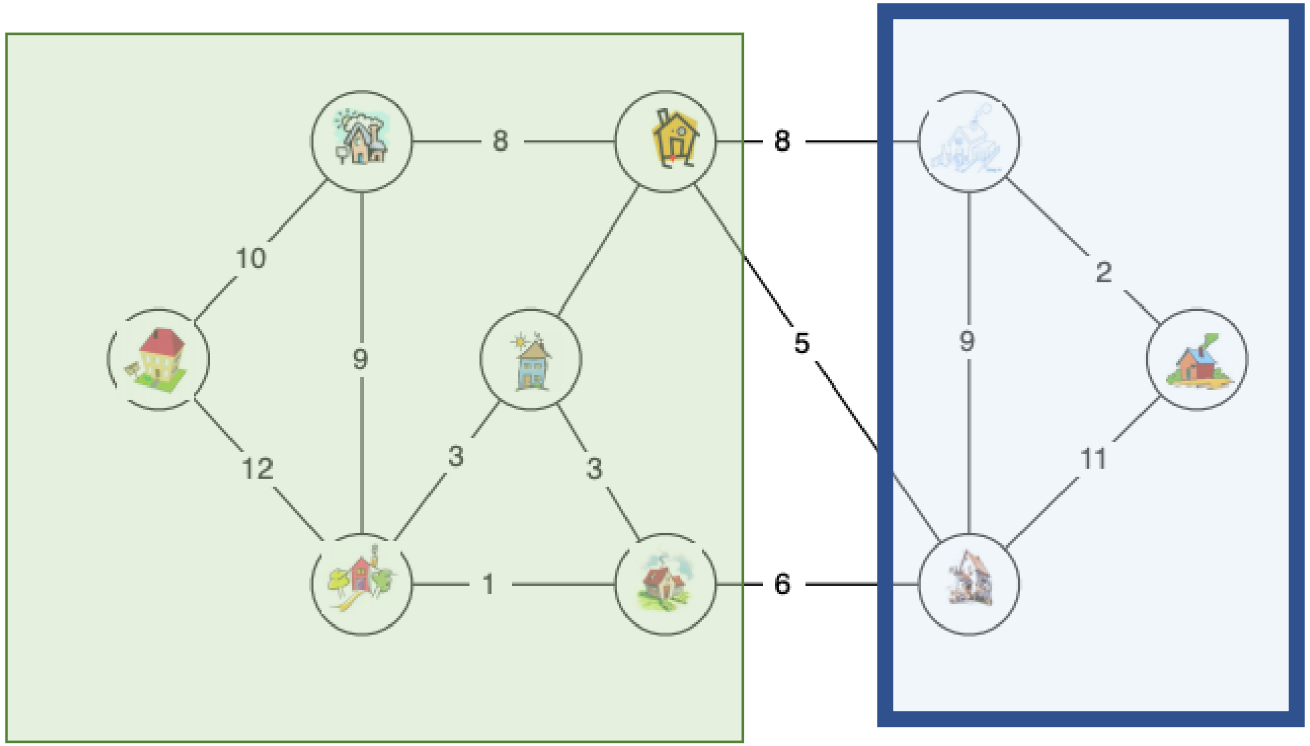
Greedy

Lecture 4

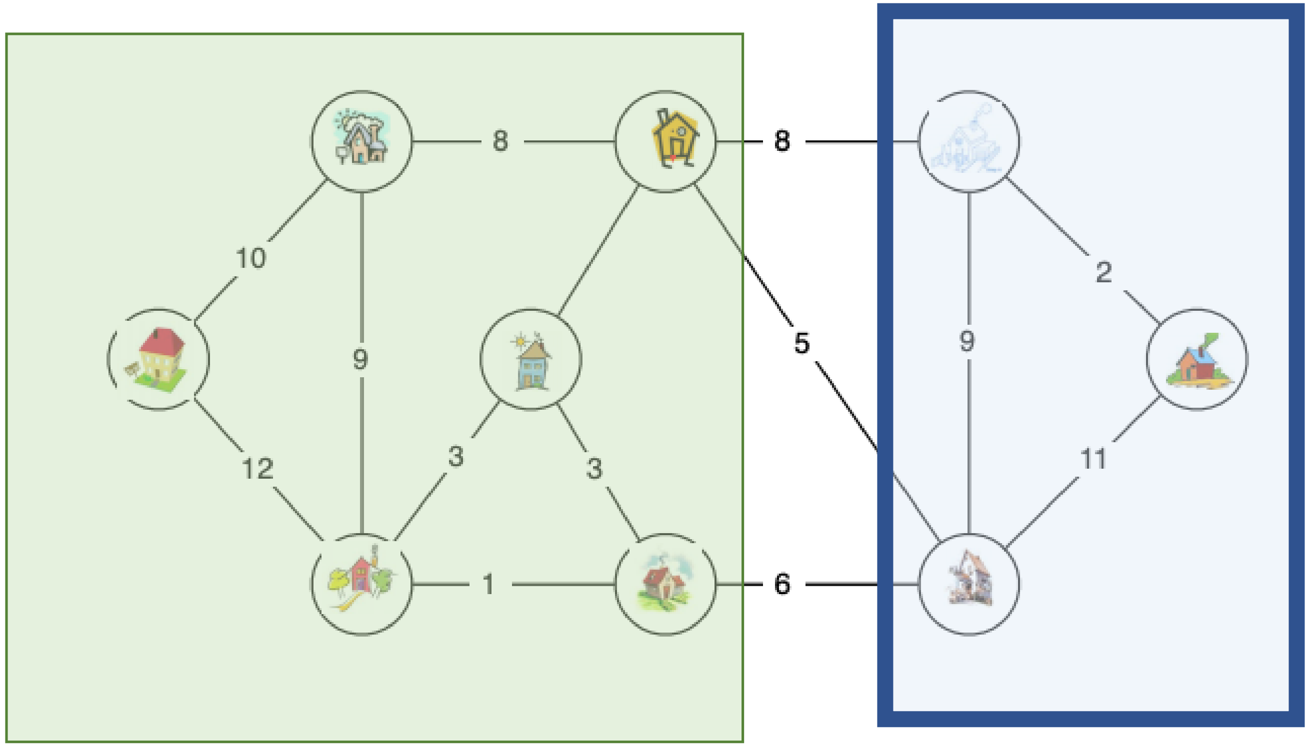
Definition: CUT

- CUT: a cut of G=(V,E) is a partition of V into 2 sets (S, V-S)



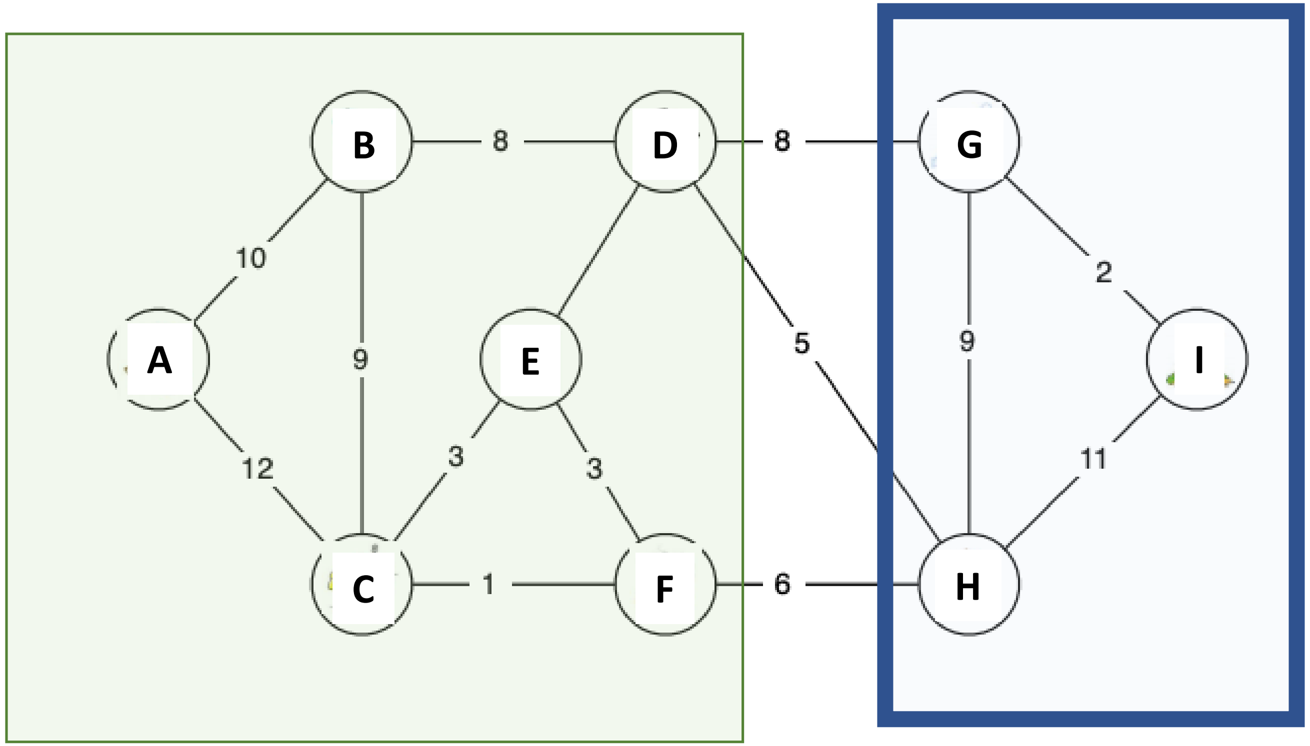
Definition: Crossing a CUT

- An edge **e=(u,v)** crosses a cut (S,V-S) if **u ∈ S**, and **u ∈ V-S**



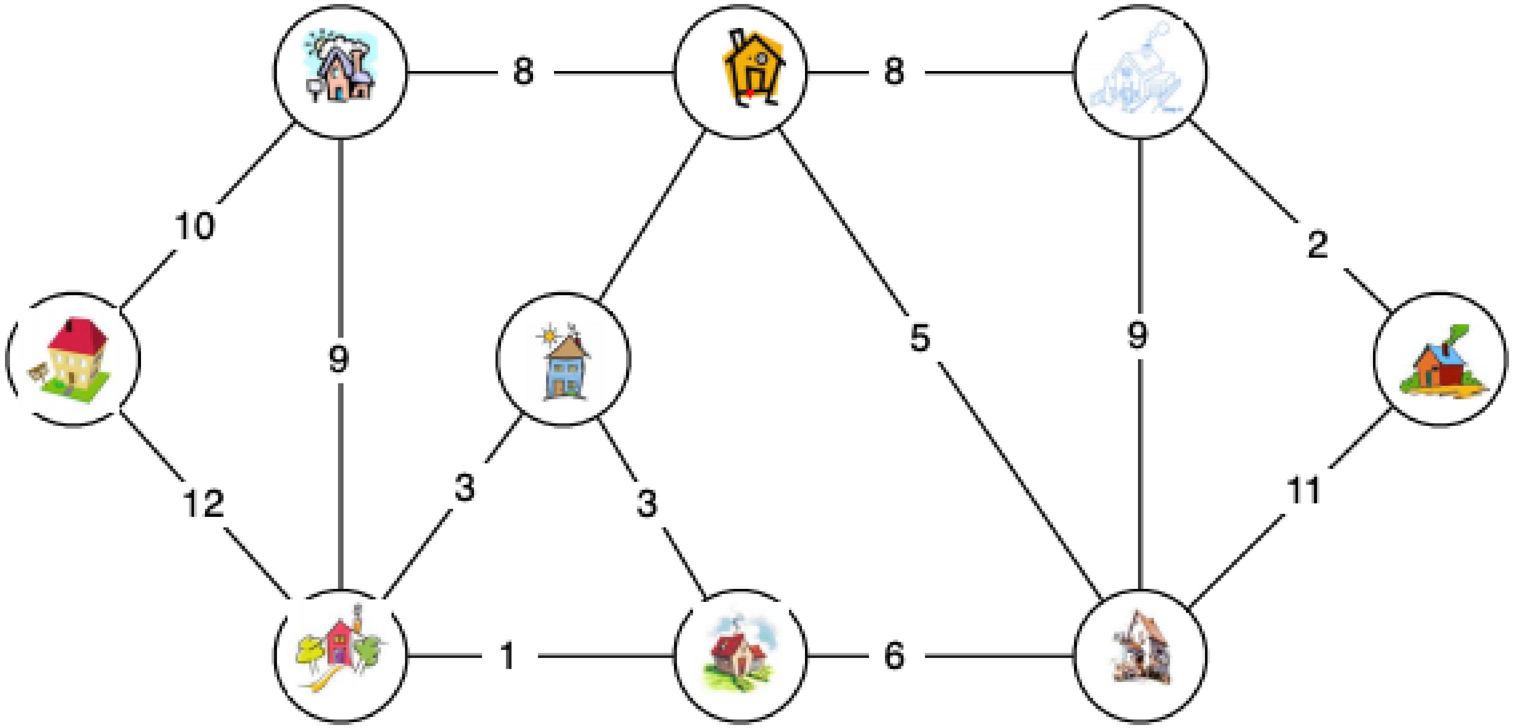
Definition: Respect

- A set A respects the cut (S, V-S) if no edge e **∈ A, crosses (S,V-S)**



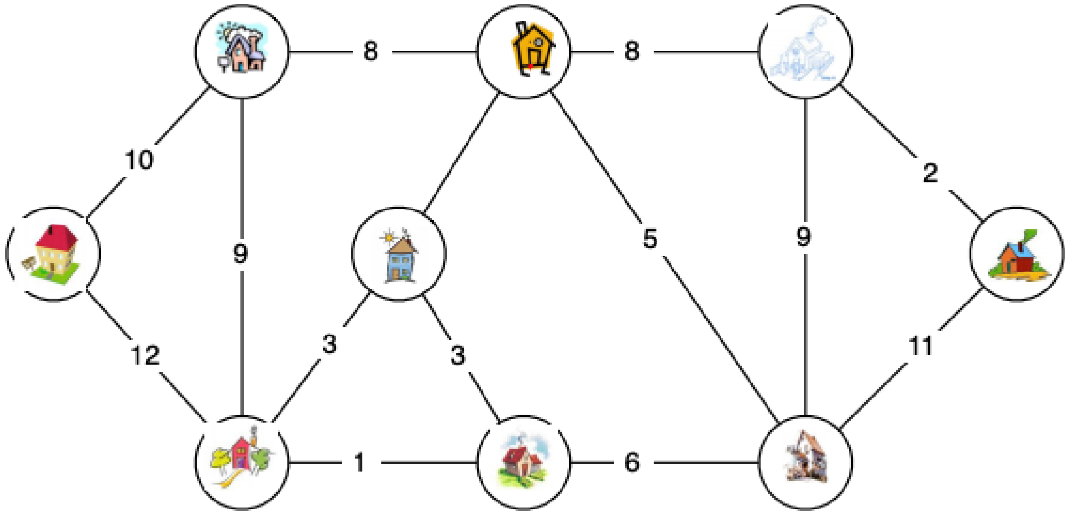
List of A

Minimum Spaning Tree



We want a tree that connects all **V**, of graph **G**, and has **minimum cost**

Minimum Spaning Tree



Looking for a set of edges such that T ⊆ E

1. Connects all vertices (V)
2. Has the least cost:

$$\text{Min} \sum_{(u,v)\in T} w(u,v)$$

How many edges does the solution have?

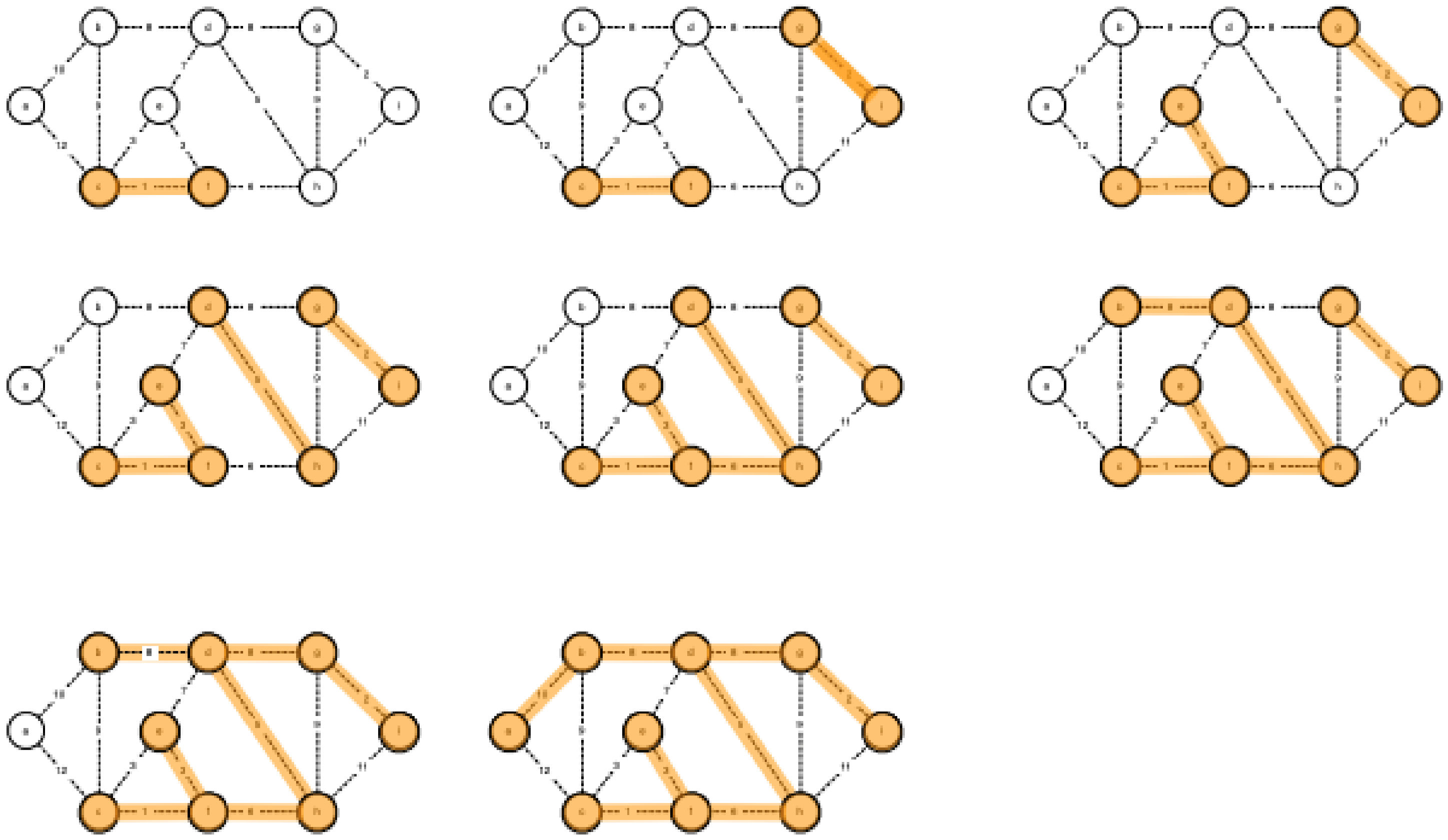
V-1

Does the solution have a cycle?

Strategy

- Start with an empty set of edges A
- Repeat for v-1 times:
  - Add lightest edge that does not create a cycle

Kruskal’s algorithm



Why does this work?

$T \leftarrow \emptyset$

repeat $V - 1$ times:

  add to T the lightest edge $e \in E$ that does not create a cycle

Cut theorem

Suppose the set of edges A is part of an m.s.t.

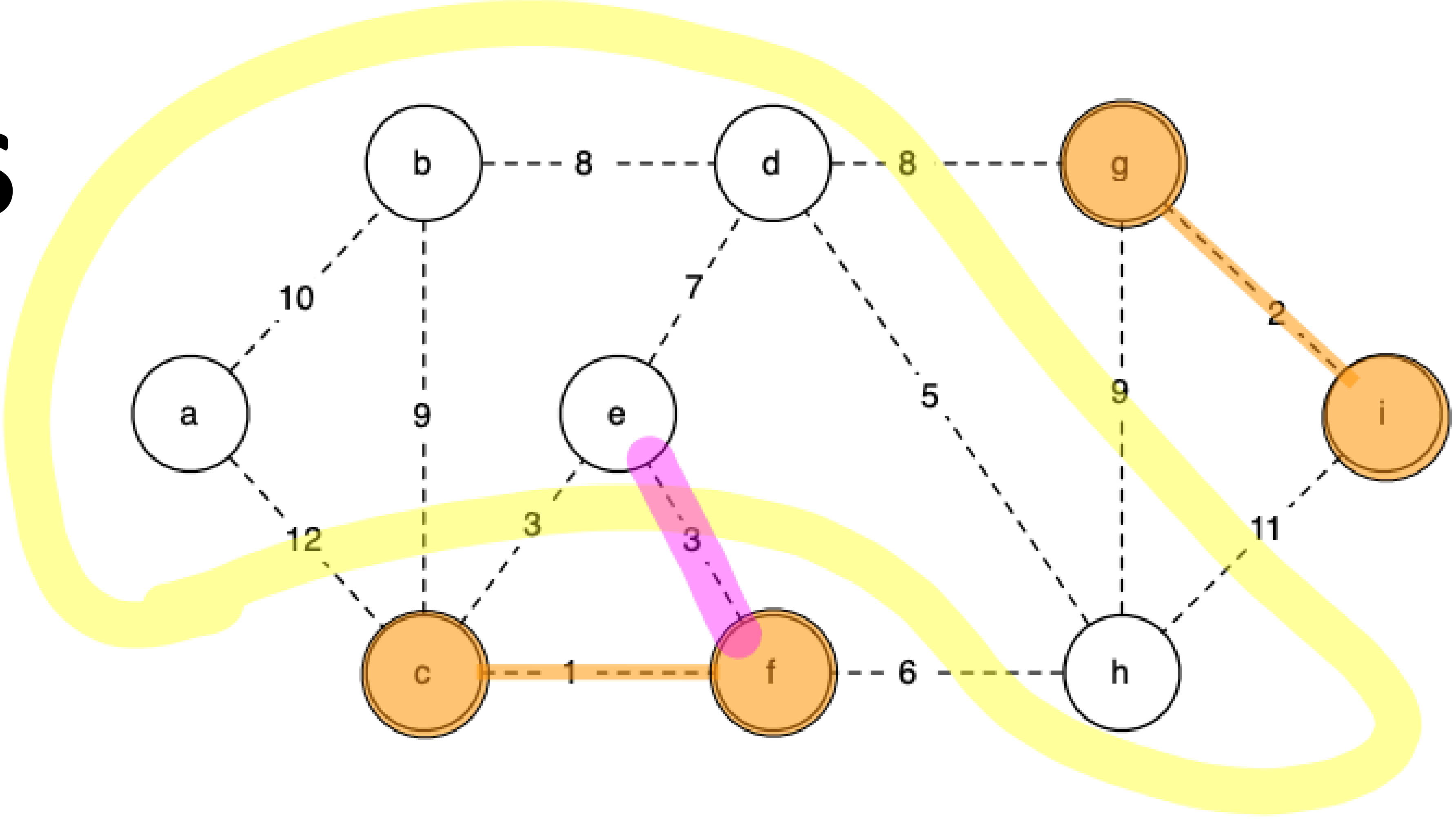
Let $(S, V - S)$ be any cut that respects A.

Let edge e be the min-weight edge across $(S, V - S)$

Then: $A \cup \{e\}$ is part of an m.s.t.

example of theorem

S



A is the set of orange edges.

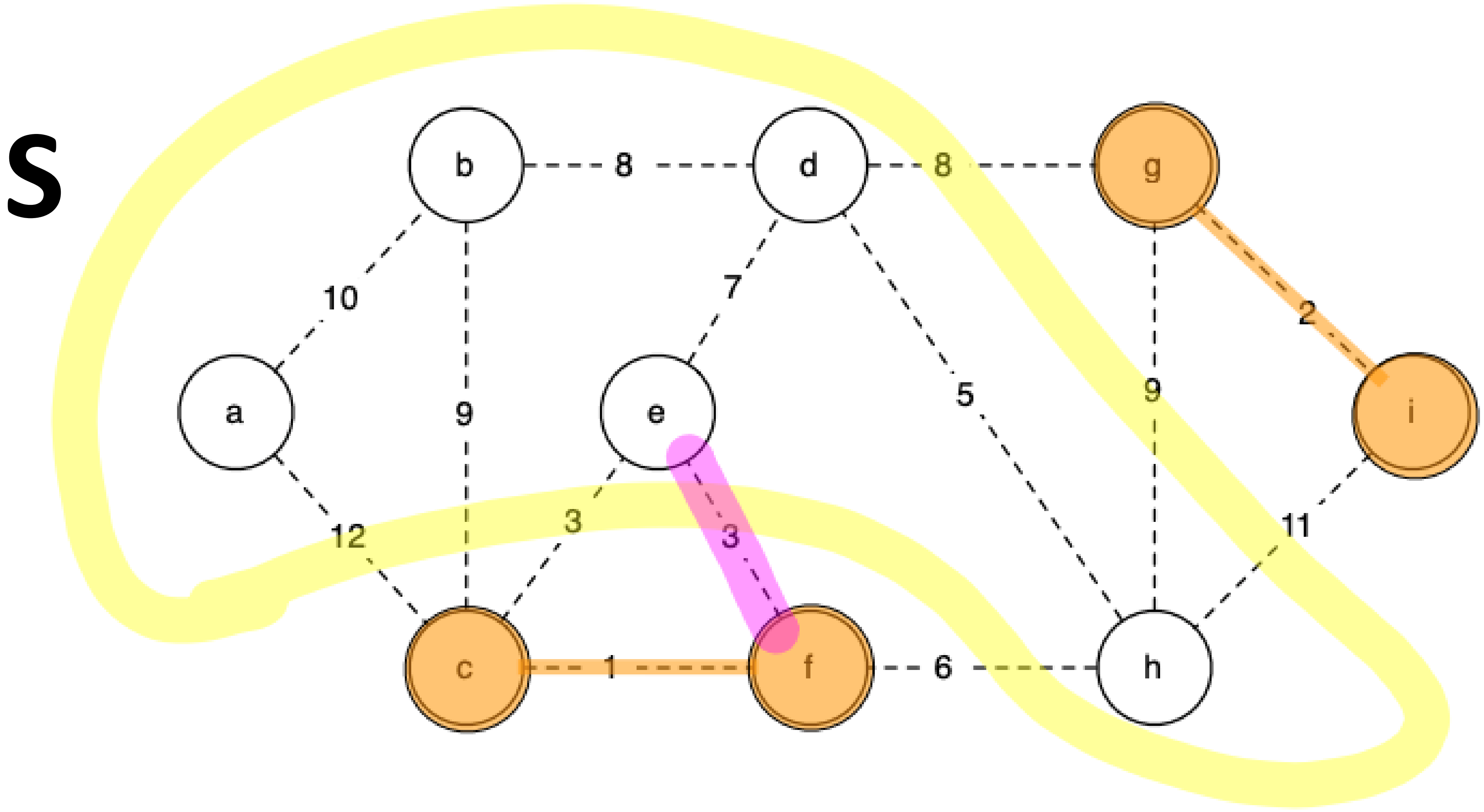
S is the cut.

A respects S.

e is the least weighted edge that crosses S.

A ∪ {e} is part of some minimum spanning tree (MST)

example of theorem



A is the set of orange edges.

S is the cut.

A respects S.

e is the least weighted edge that crosses S.

A ∪ {e} is part of some minimum spanning tree (MST)

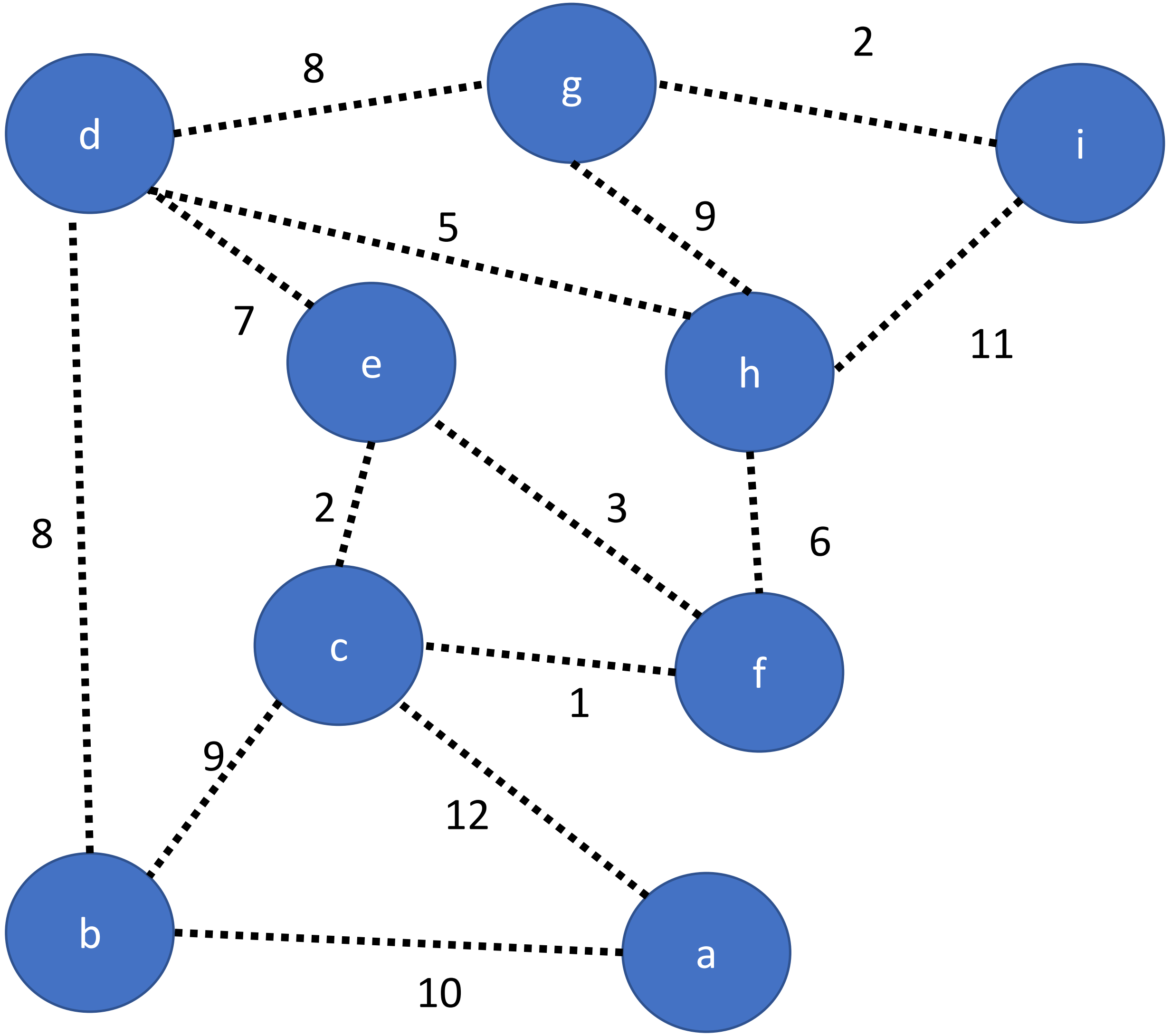
GENERAL-MST-STRATEGY($G = (V, E)$)

$A \leftarrow \emptyset$
repeat $V - 1$ times:
  Pick a cut $(S, V - S)$ that respects A
  Let e be min-weight edge over cut $(S, V - S)$
  $A \leftarrow A \cup \{e\}$

Modification: Cut S consists of all edges, and vertices of A

Algorithm

A= ∅

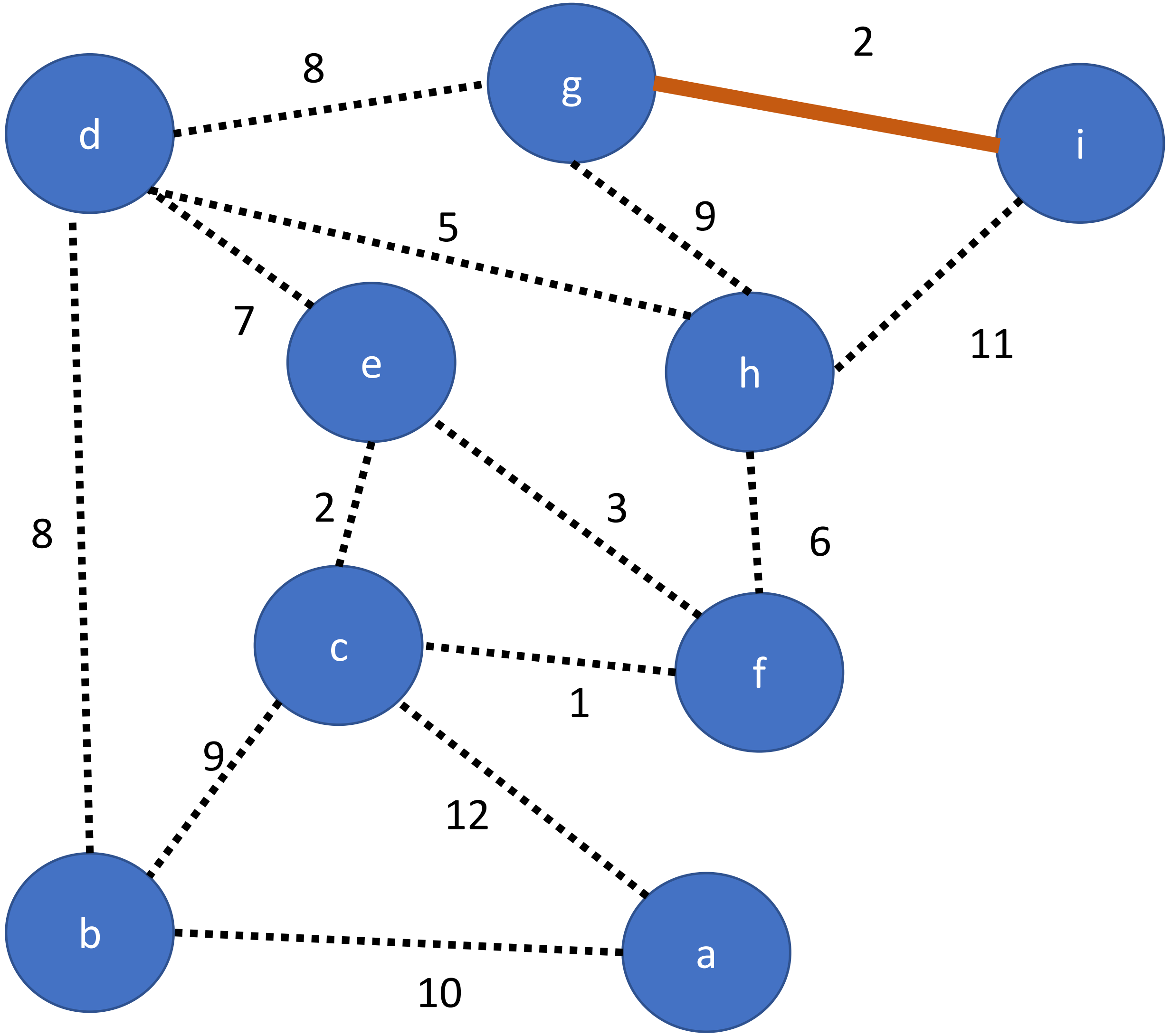


GENERAL-MST-STRATEGY($G = (V, E)$)

$A \leftarrow \emptyset$
repeat $V - 1$ times:
    Pick a cut $(S, V - S)$ that respects A
    Let e be min-weight edge over cut $(S, V - S)$
    $A \leftarrow A \cup \{e\}$

Algorithm

A= {(g,i)}

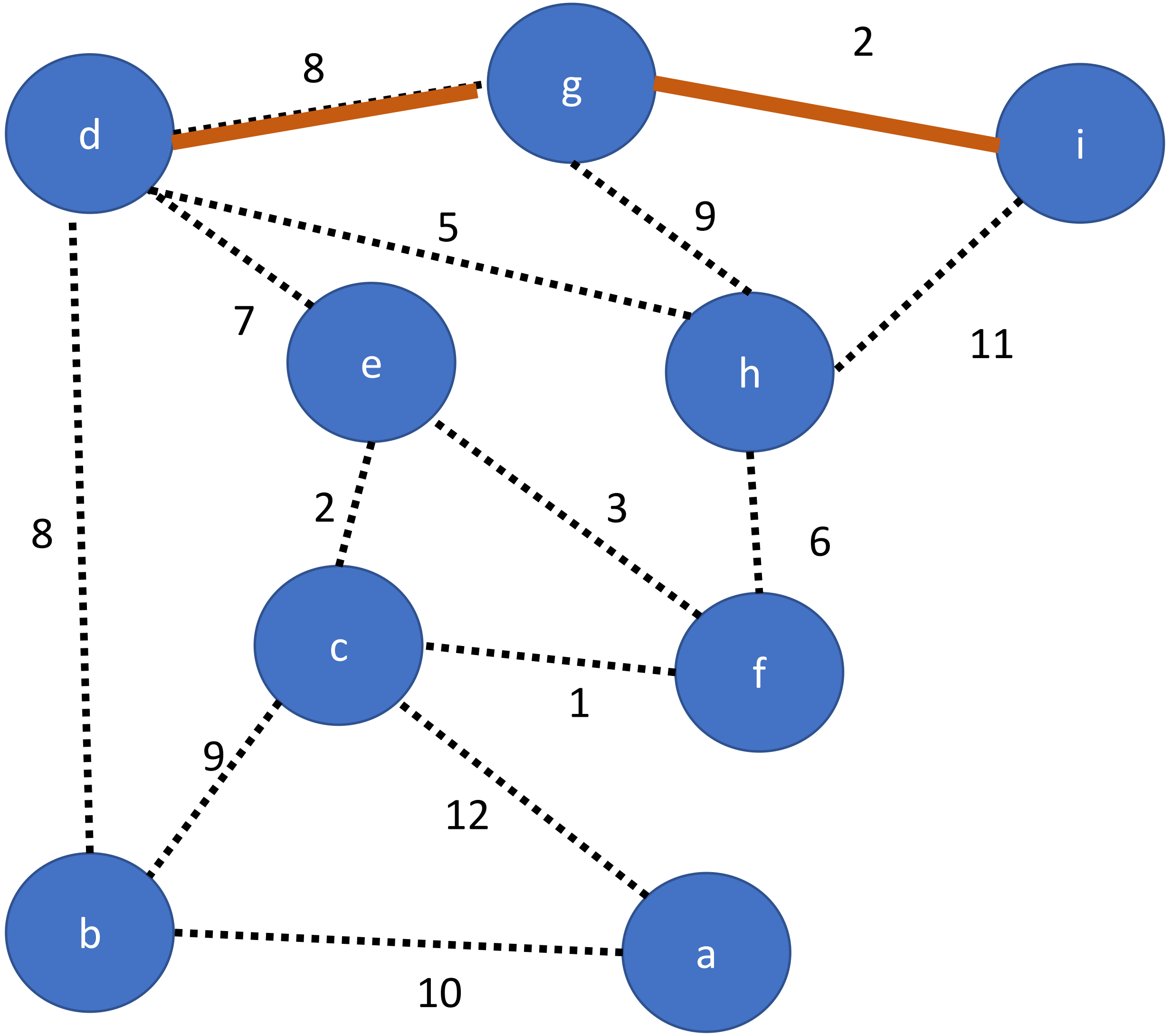


GENERAL-MST-STRATEGY$(G = (V, E))$

```
A ← ∅
repeat  V − 1 times:
        Pick a cut (S, V − S) that respects A
        Let e be min-weight edge over cut (S, V − S)
        A ← A ∪ {e}
```

Algorithm

A= {(g,i), (d,g)}

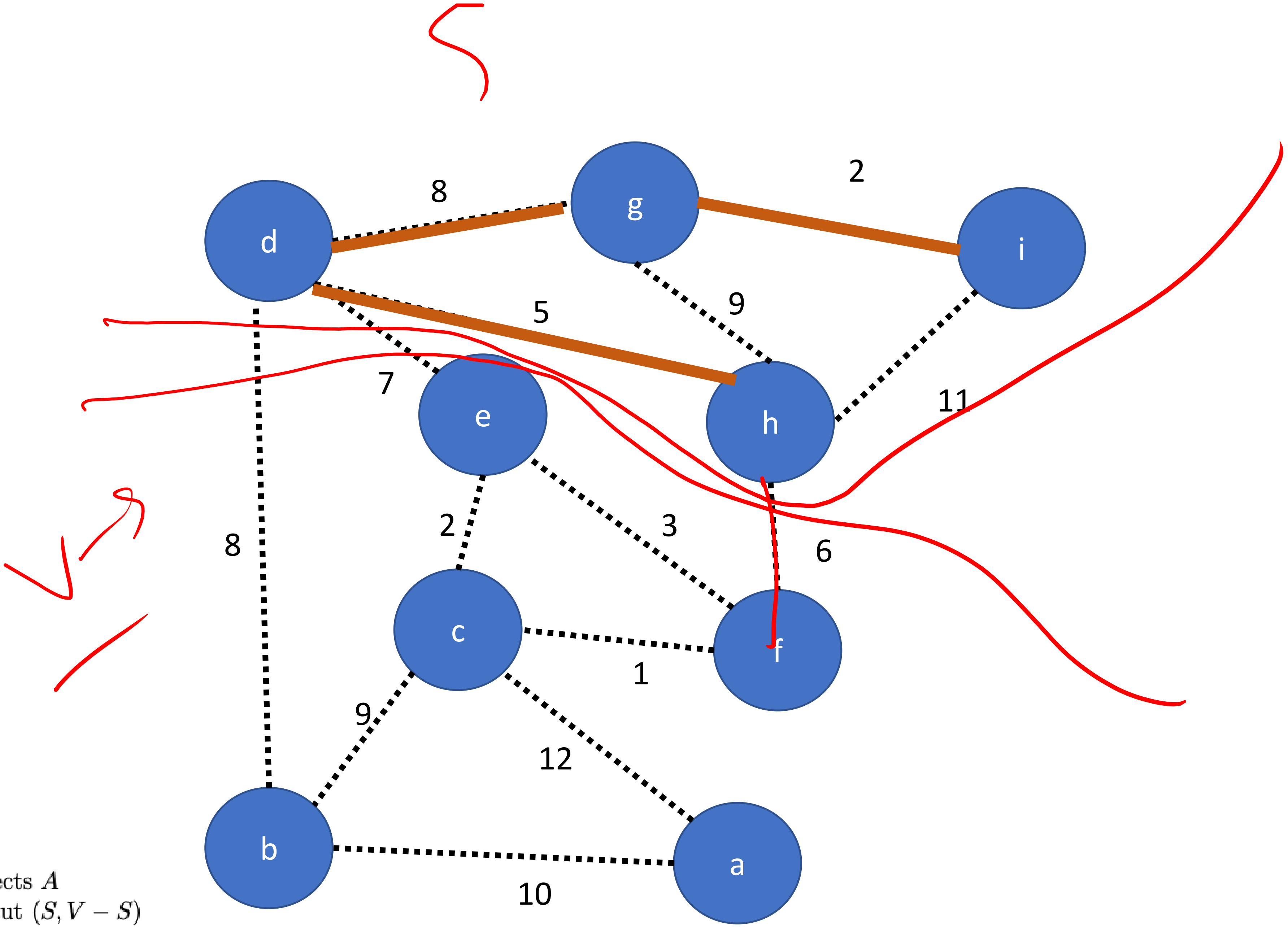


GENERAL-MST-STRATEGY$(G = (V, E))$

```
A ← ∅
repeat  V − 1 times:
        Pick a cut (S, V − S) that respects A
        Let e be min-weight edge over cut (S, V − S)
        A ← A ∪ {e}
```

Algorithm

A= {(g,i), (d,g),(d,h)}

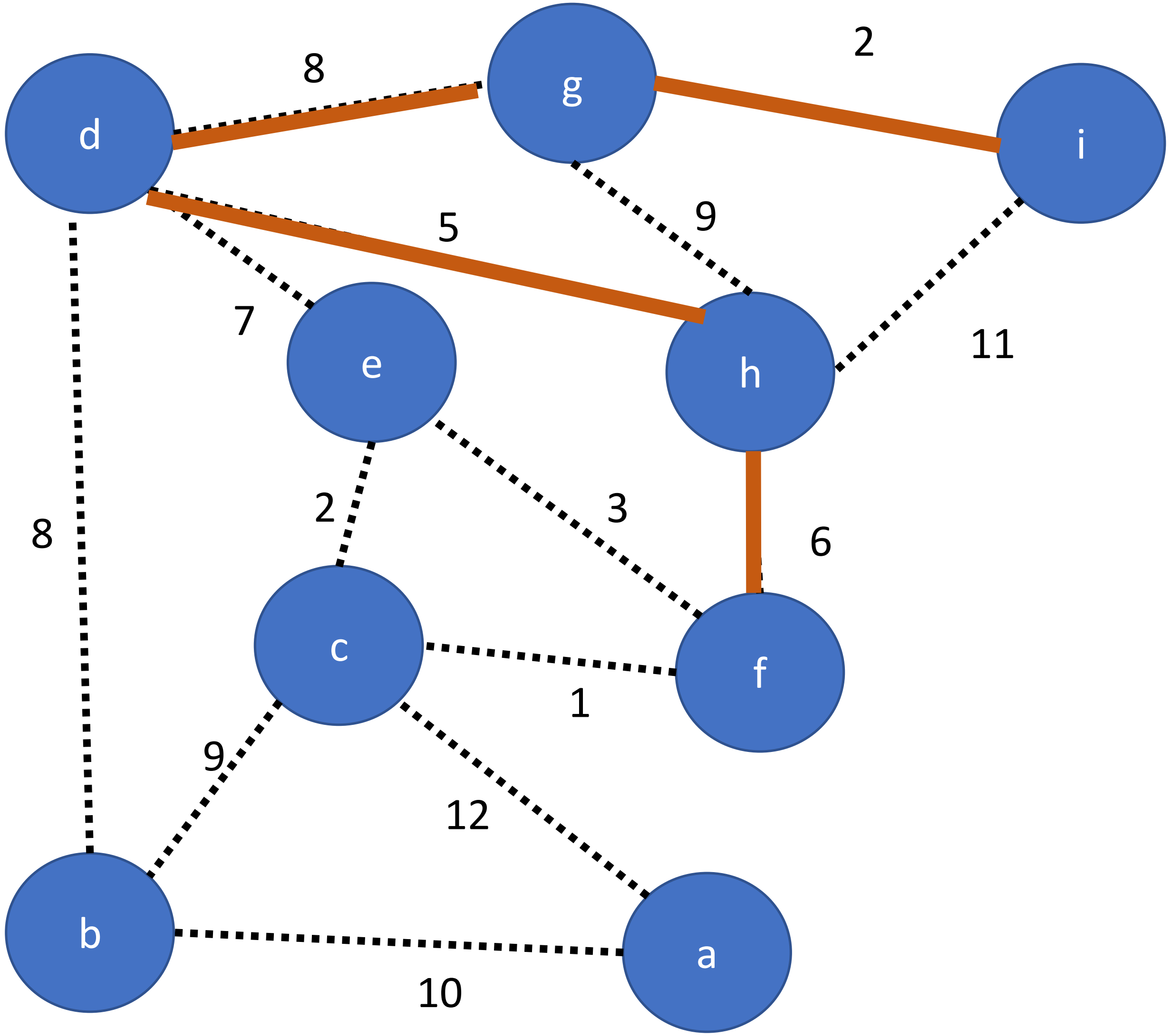


GENERAL-MST-STRATEGY$(G = (V, E))$

1 $A \leftarrow \emptyset$
2 **repeat** $V - 1$ times:
3 Pick a cut $(S, V - S)$ that respects A
4 Let e be min-weight edge over cut $(S, V - S)$
5 $A \leftarrow A \cup \{e\}$

Algorithm

A= {(g,i), (d,g),(d,h), (h,f)}

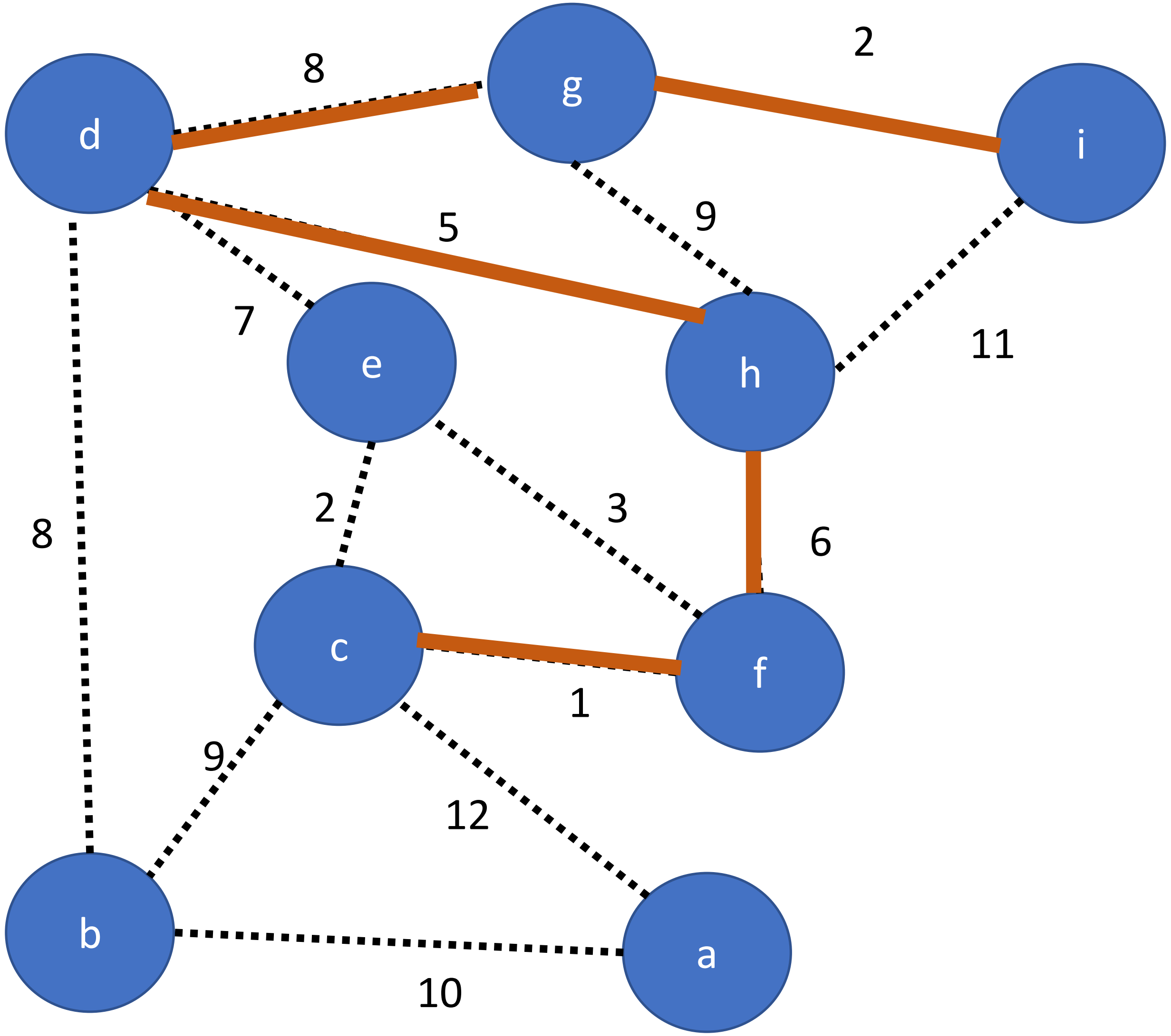


GENERAL-MST-STRATEGY$(G = (V, E))$

```
A ← ∅
repeat V − 1 times:
    Pick a cut (S, V − S) that respects A
    Let e be min-weight edge over cut (S, V − S)
    A ← A ∪ {e}
```

Algorithm

A= {(g,i), (d,g),(d,h), (h,f),(f,c)}



$\textsc{General-MST-Strategy}(G = (V, E))$

$A \leftarrow \emptyset$

repeat $V - 1$ times:

  Pick a cut $(S, V - S)$ that respects A

  Let e be min-weight edge over cut $(S, V - S)$

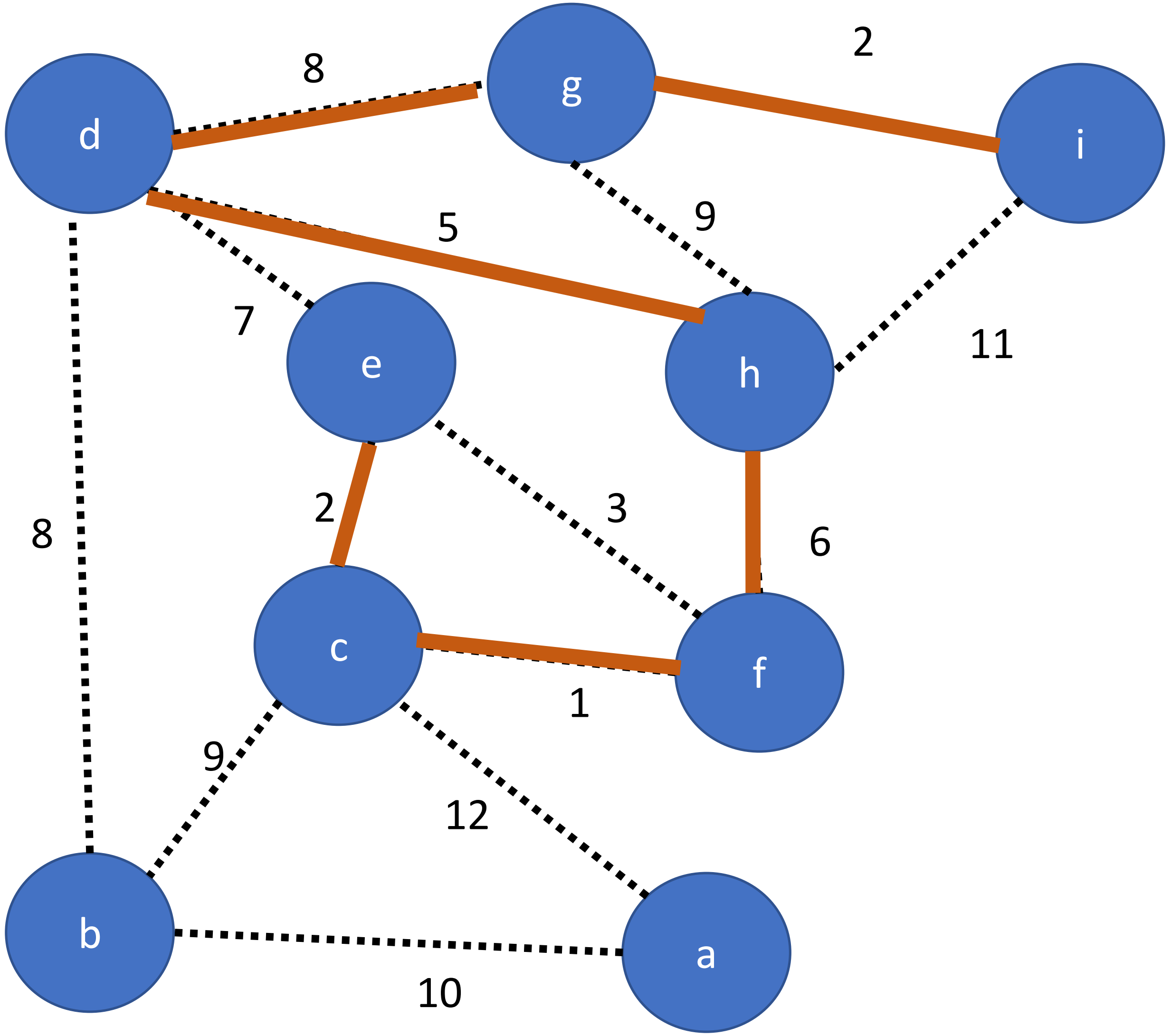
  $A \leftarrow A \cup \{e\}$

Algorithm

A= {(g,i), (d,g),(d,h), (h,f),(f,c), (c,e)}

GENERAL-MST-STRATEGY$(G = (V, E))$

```
A ← ∅
repeat  V − 1 times:
        Pick a cut (S, V − S) that respects A
        Let e be min-weight edge over cut (S, V − S)
        A ← A ∪ {e}
```

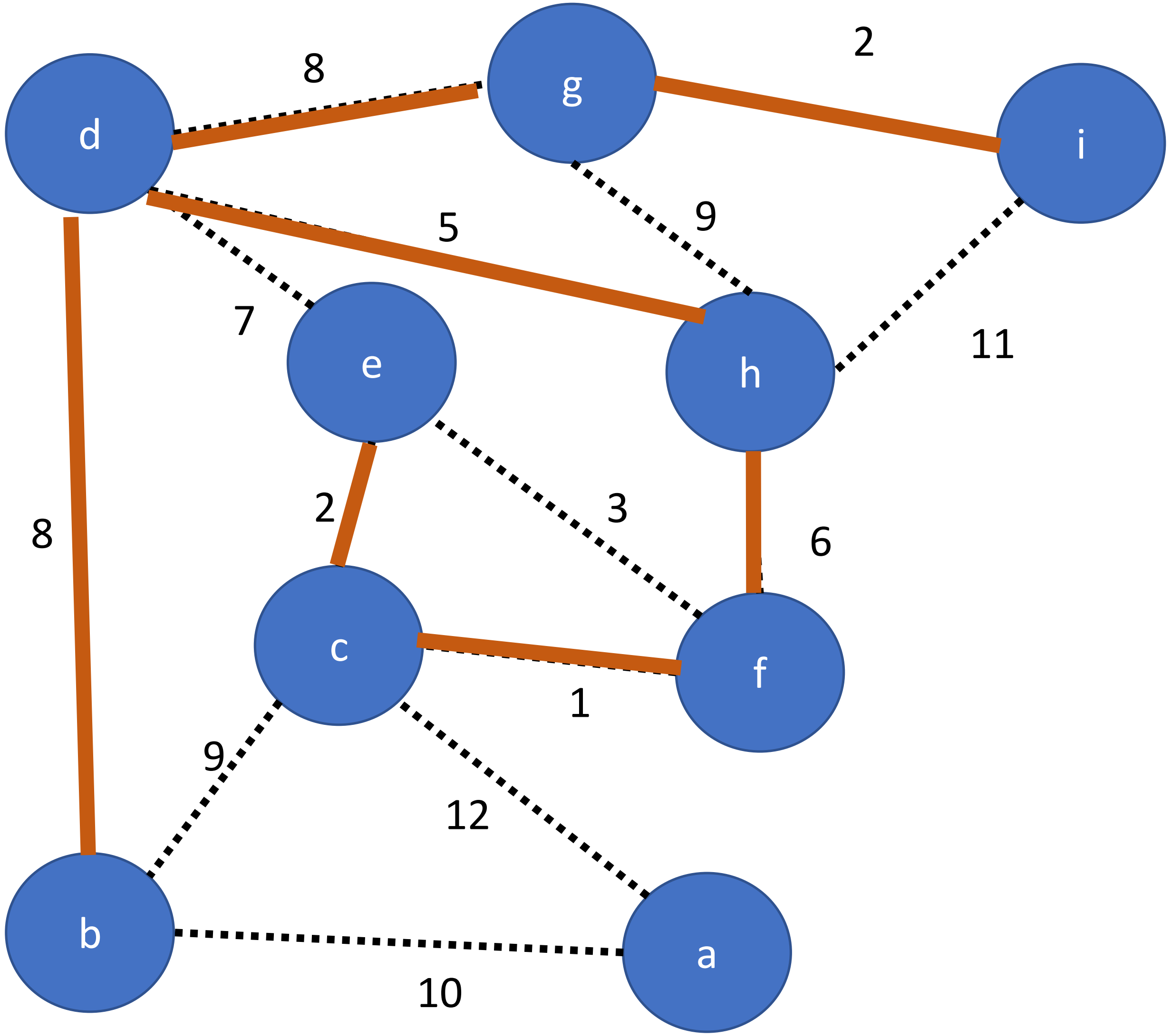


Algorithm

A= {(g,i), (d,g),(d,h), (h,f),(f,c), (c,e), (d,b)}

$\textsc{General-MST-Strategy}(G = (V, E))$

1 $A \leftarrow \emptyset$
2 **repeat** $V - 1$ times:
3 Pick a cut $(S, V - S)$ that respects A
4 Let e be min-weight edge over cut $(S, V - S)$
5 $A \leftarrow A \cup \{e\}$

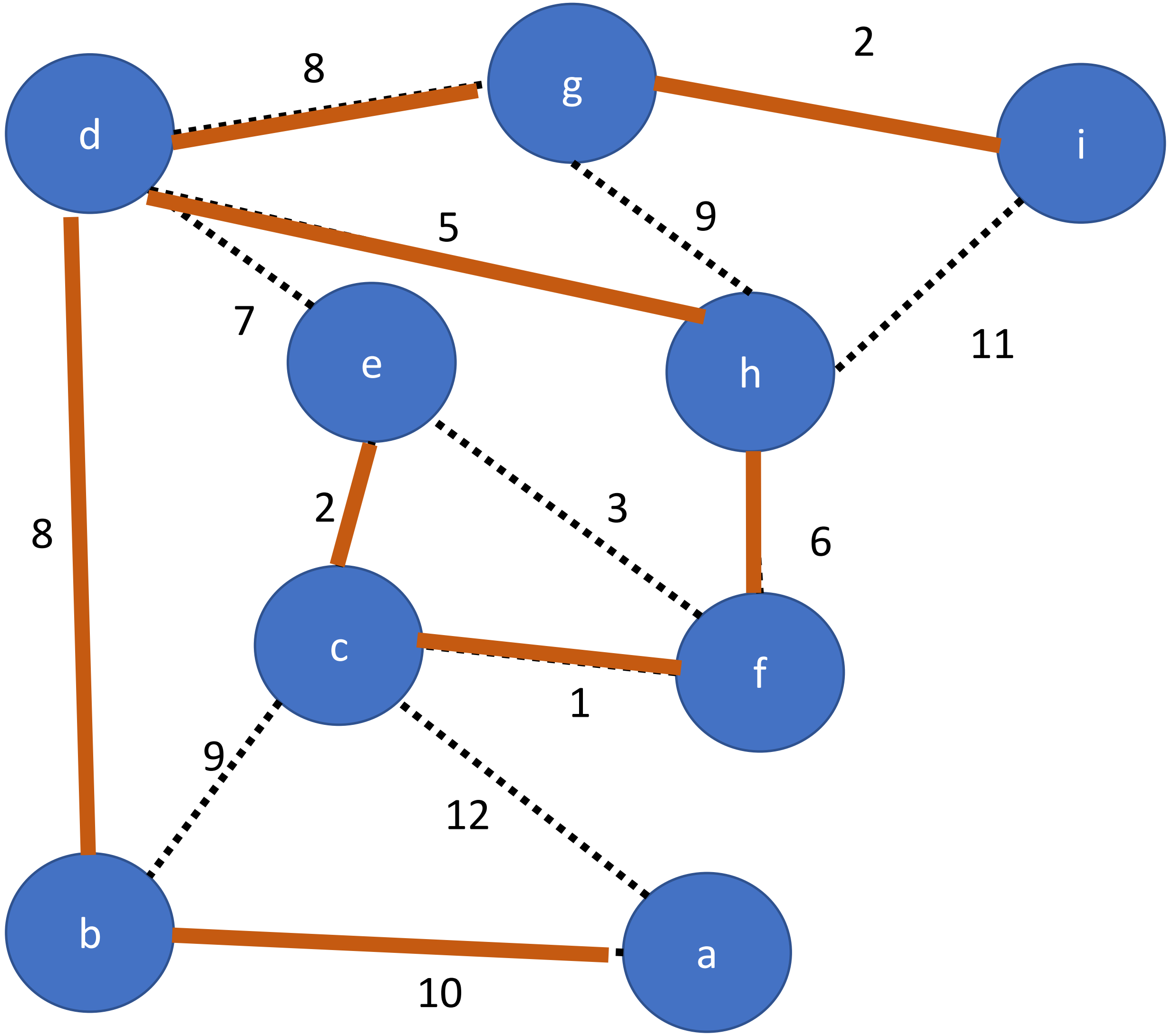


Algorithm

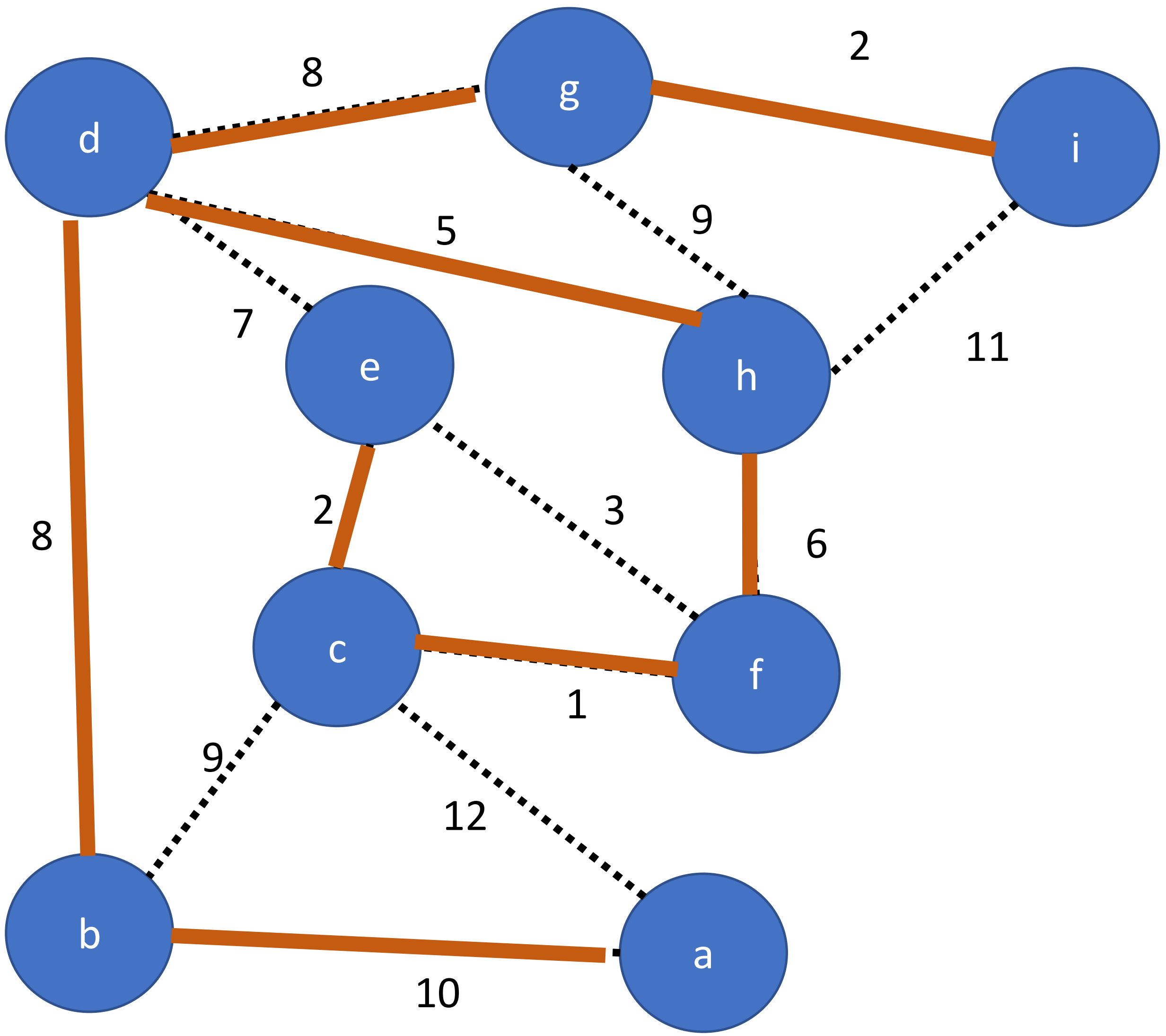
A= {(g,i), (d,g),(d,h), (h,f),(f,c), (c,e), (d,b),(b,a)}

GENERAL-MST-STRATEGY$(G = (V, E))$

```
A ← ∅
repeat  V − 1 times:
        Pick a cut (S, V − S) that respects A
        Let e be min-weight edge over cut (S, V − S)
        A ← A ∪ {e}
```



MST



GENERAL-MST-STRATEGY$(G = (V, E))$

1 $A \leftarrow \emptyset$
2 **repeat** $V - 1$ times:
3 Pick a cut $(S, V - S)$ that respects A
4 Let e be min-weight edge over cut $(S, V - S)$
5 $A \leftarrow A \cup \{e\}$

Prim's Algorithm

GENERAL-MST-STRATEGY$(G = (V, E))$

$A \leftarrow \emptyset$
repeat $V - 1$ times:
  Pick a cut $(S, V - S)$ that respects A
  Let e be min-weight edge over cut $(S, V - S)$
  $A \leftarrow A \cup \{e\}$

Implementation

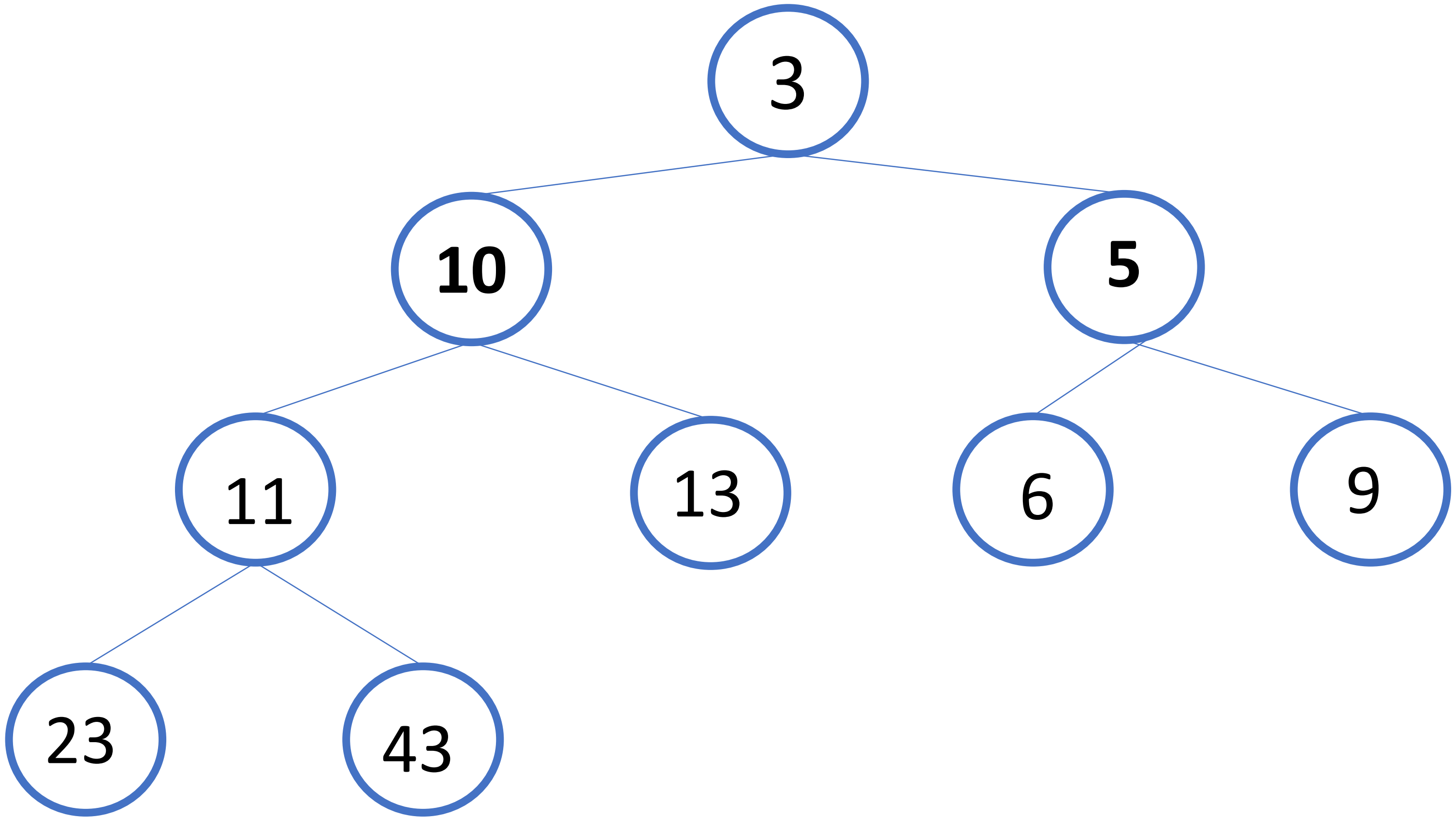
- Idea:
- Keep a data structure which identifies the ‘lightest edge’ that crosses the cuts (A=S, V-S)
  - Priority queue

Implementation

- **Makequeue:**
  - Insert a list of (n1,k1), (n2,k2),…. Into the Queue.
- **Exchange Operation:**
  - Remove the node (ni,ki) where ki is the smallest key in the Queue.
- **Decrease Key operation:**
  - Given a key(ni,ki), and a new value ki* (ki*<ki), decrease the value ki to ki*.

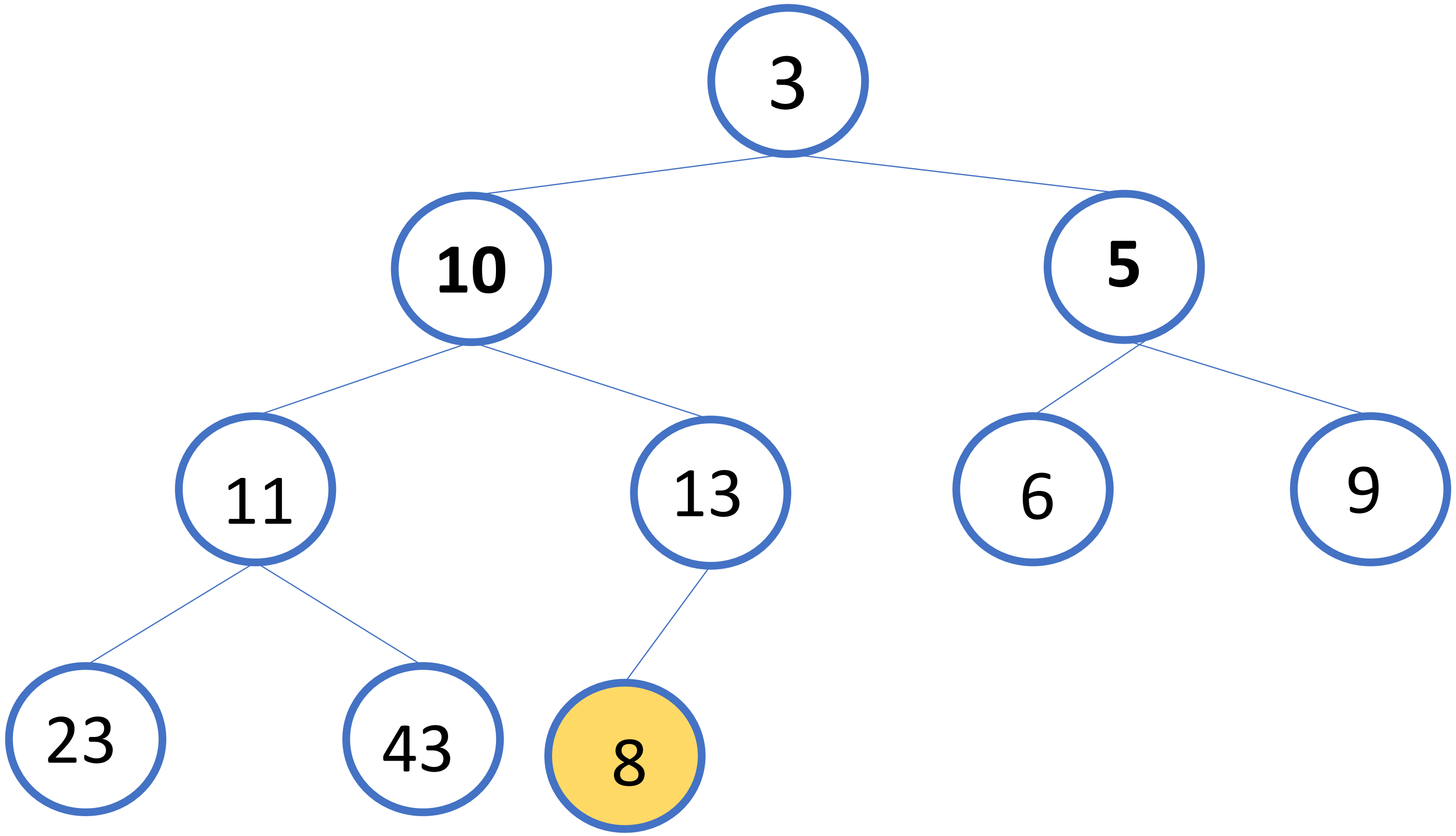
Binary heap

- A Full tree
- Key value of a node <= key value of it’s children



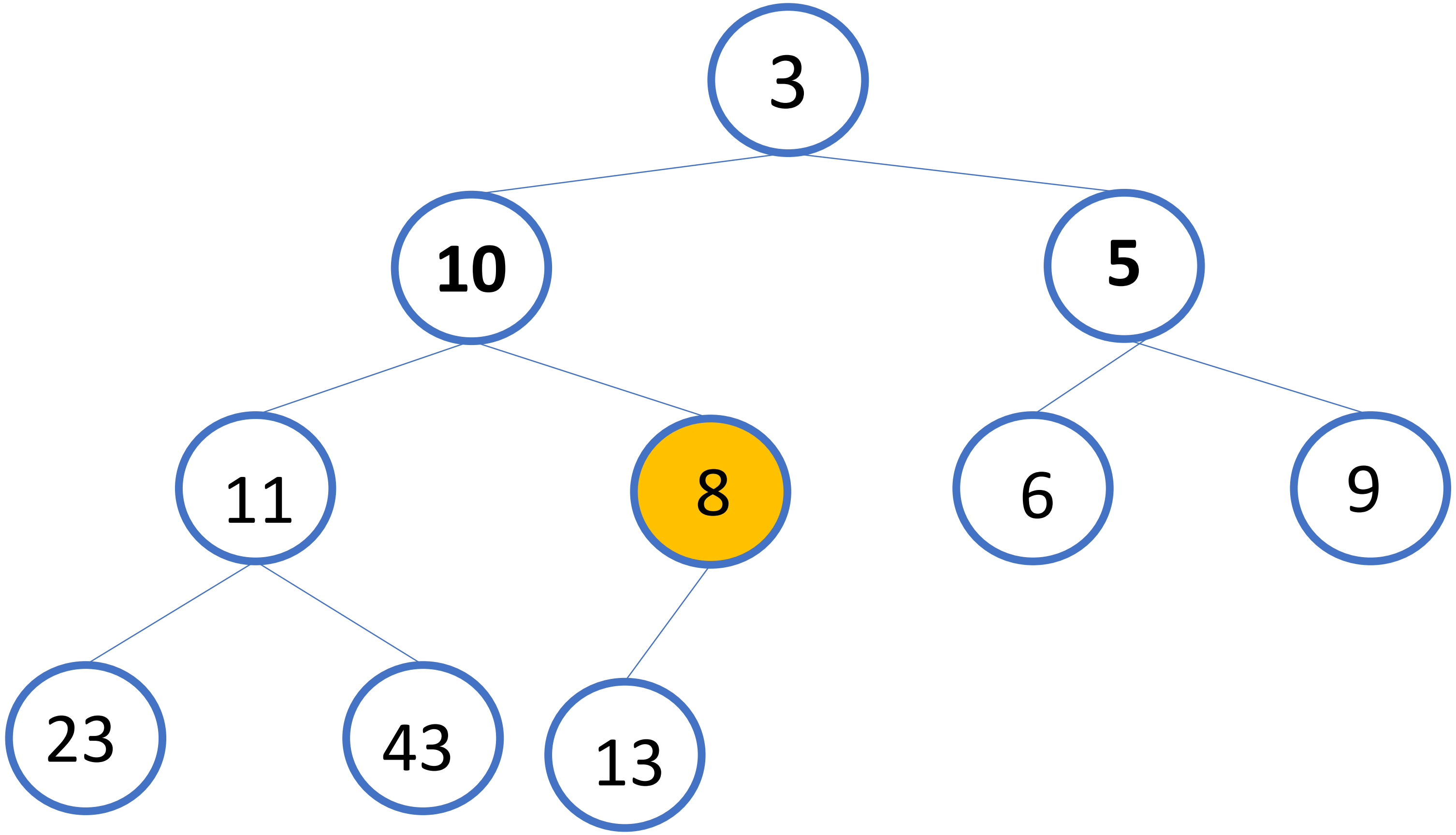
Binary heap

- **Insert:**



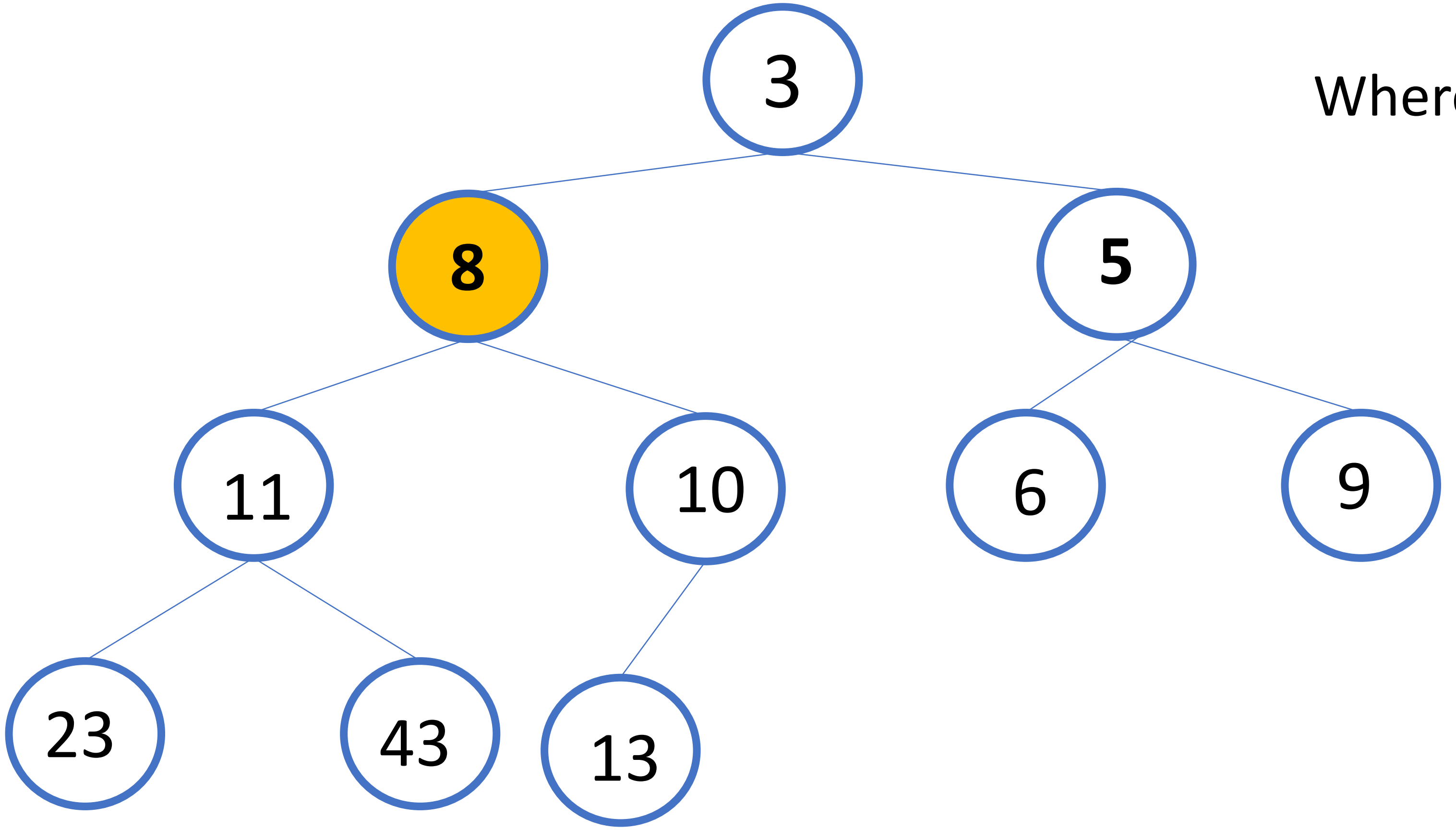
Binary heap

- **Insert:**



Binary heap

- **Insert:**



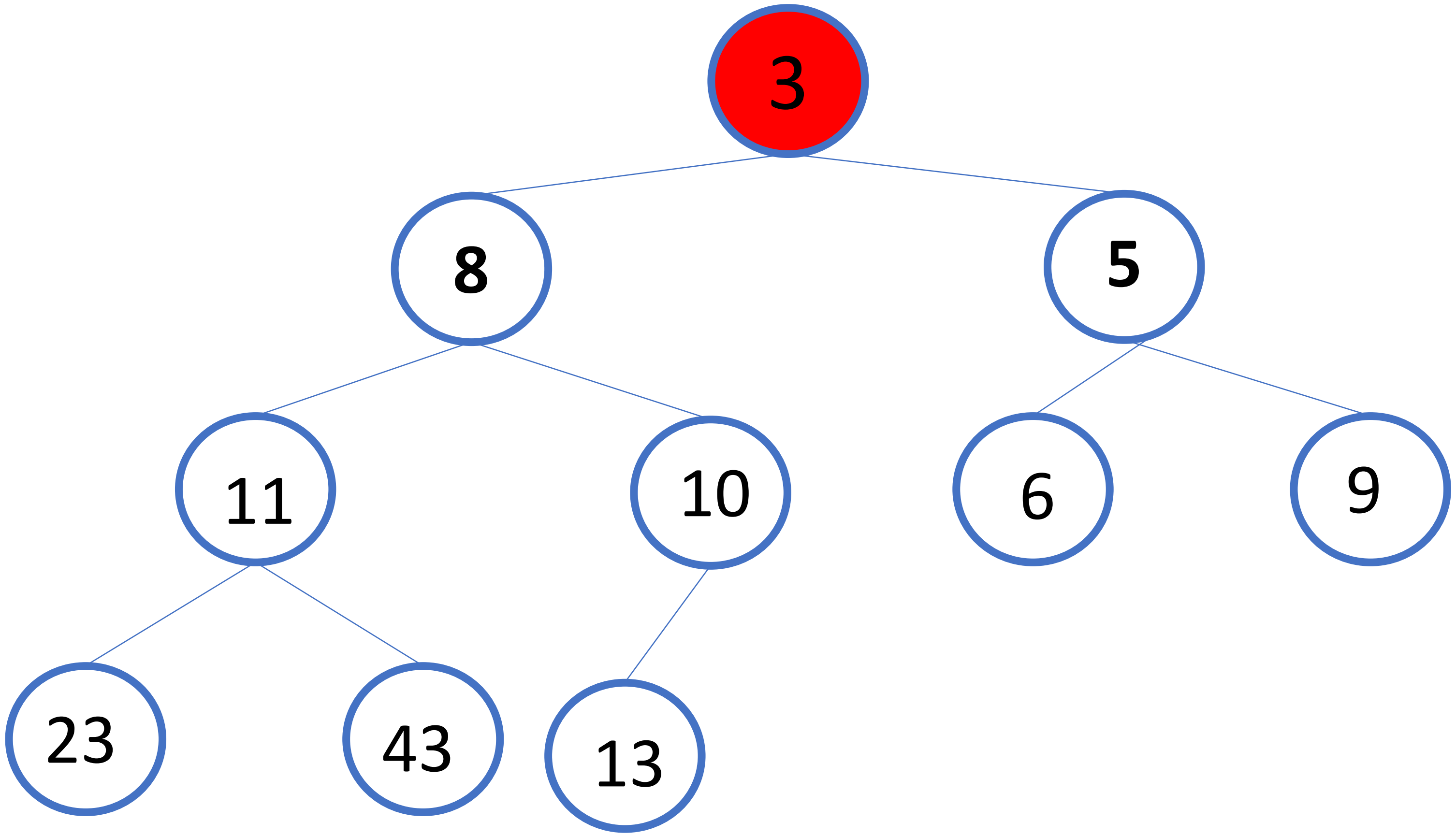
Insert time:

O(logn)

Where n is the size of the heap

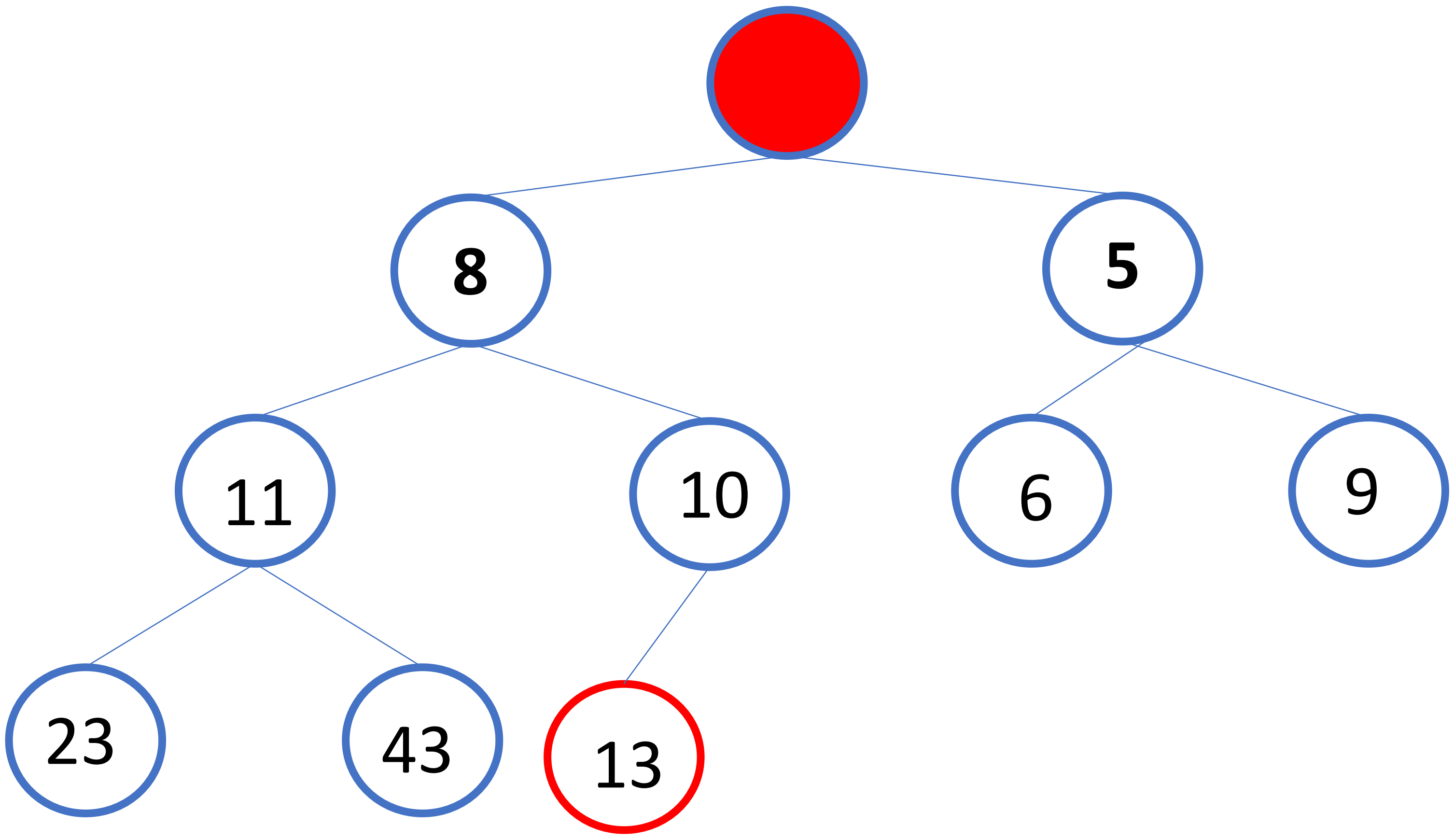
Binary heap

- **Extractmin:**



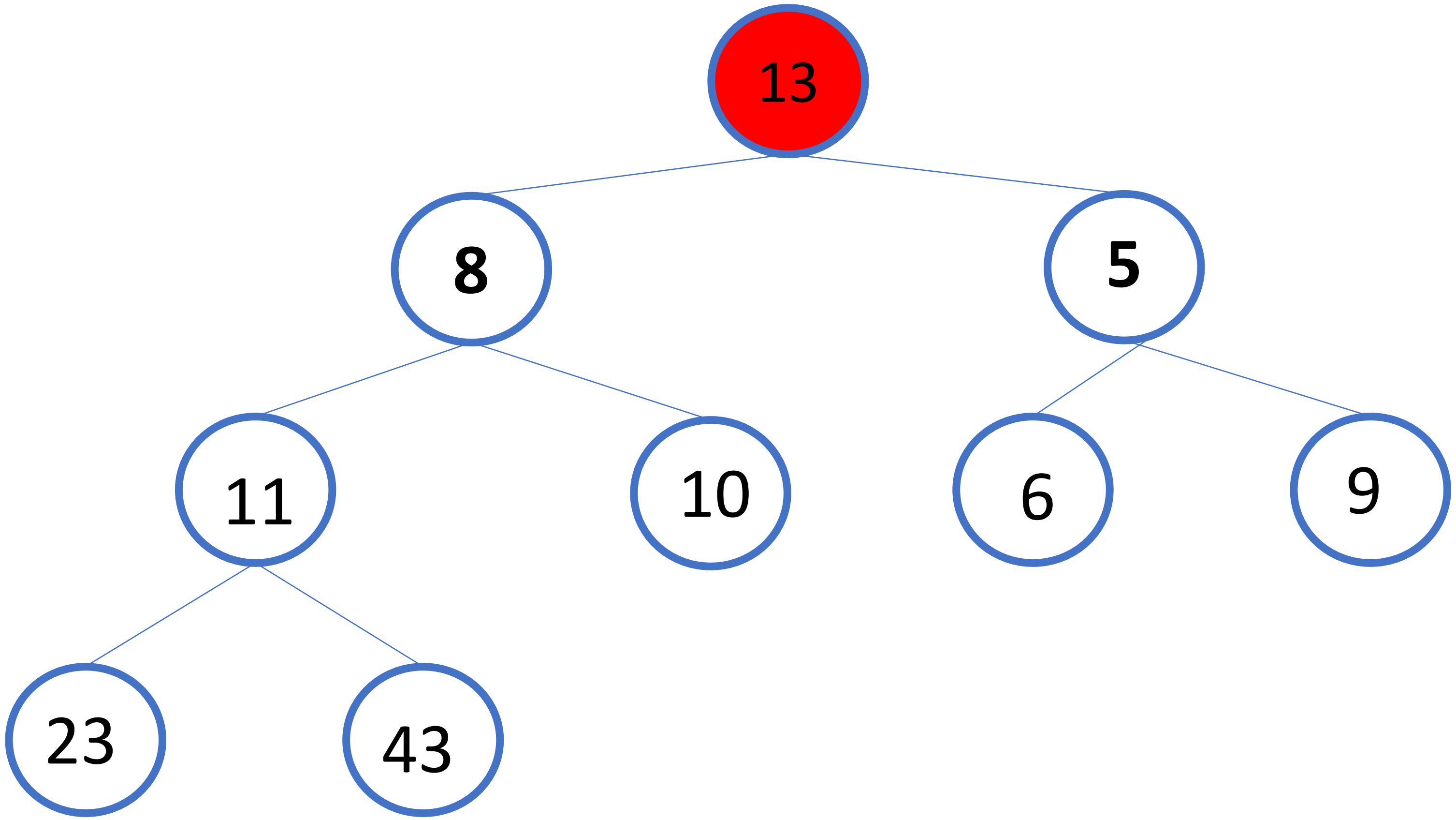
Binary heap

- **Extractmin:**



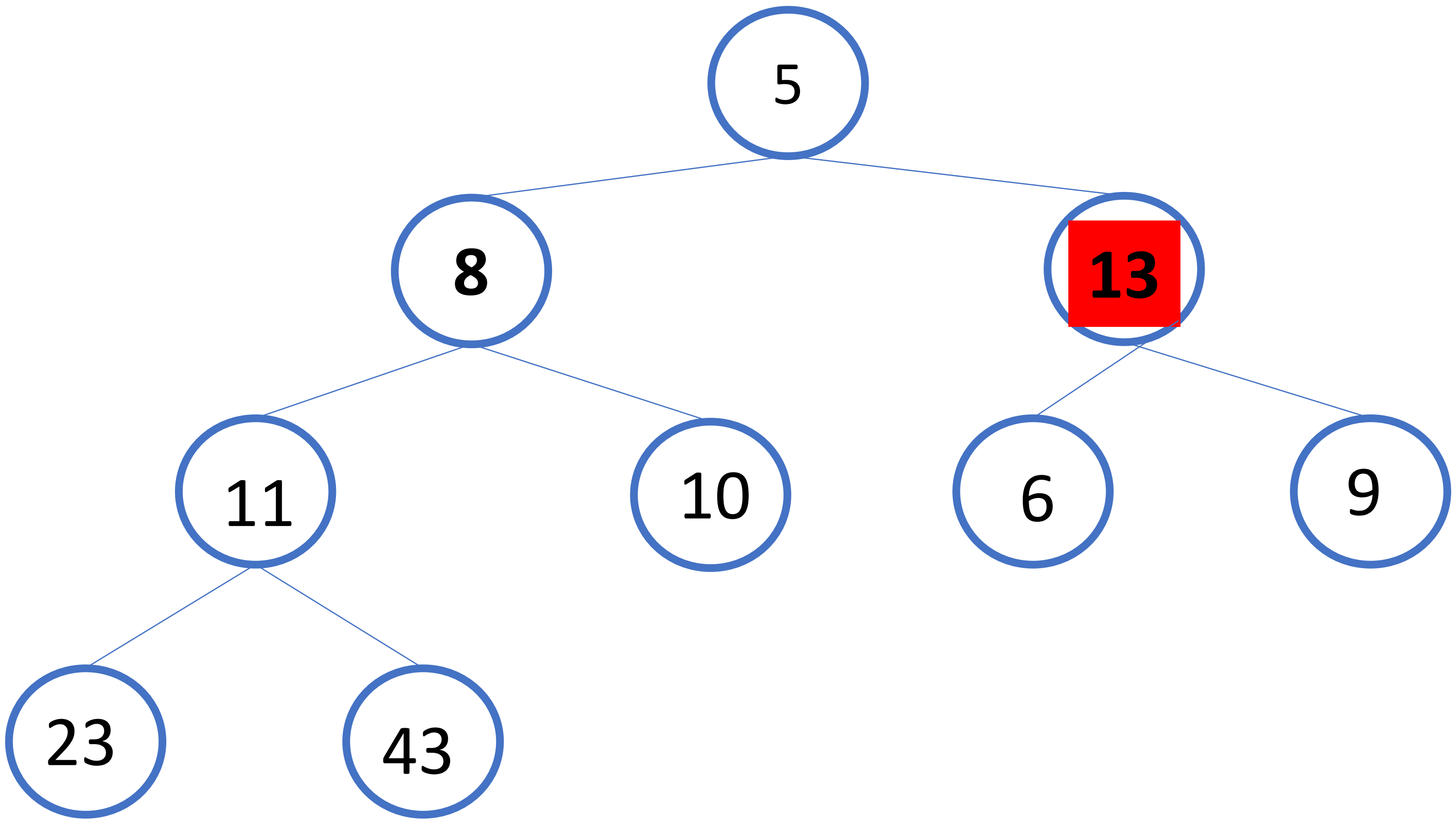
Binary heap

- **Extractmin:**



Binary heap

- **Extractmin:**



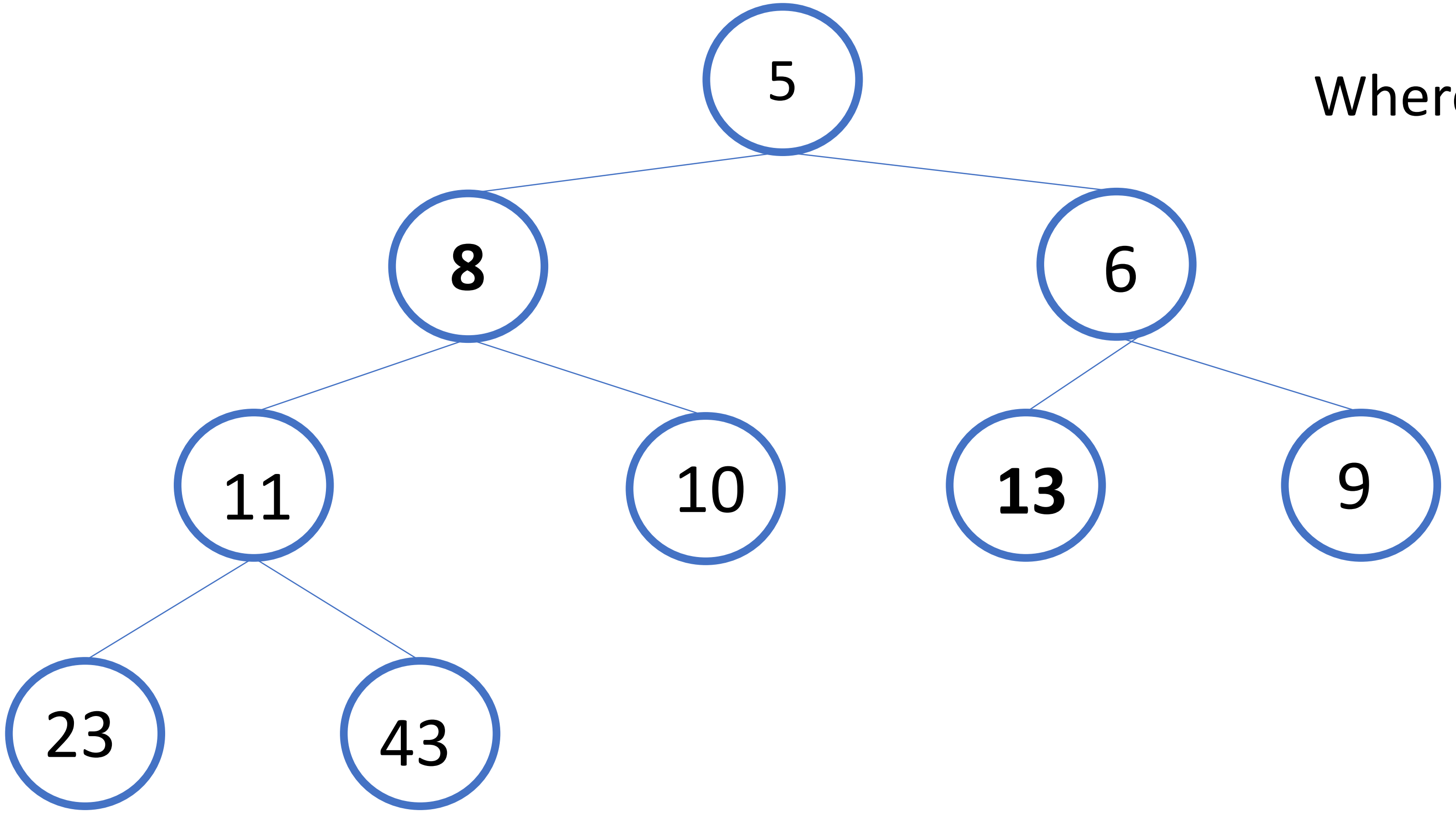
Binary heap

- **Extractmin:**

time:

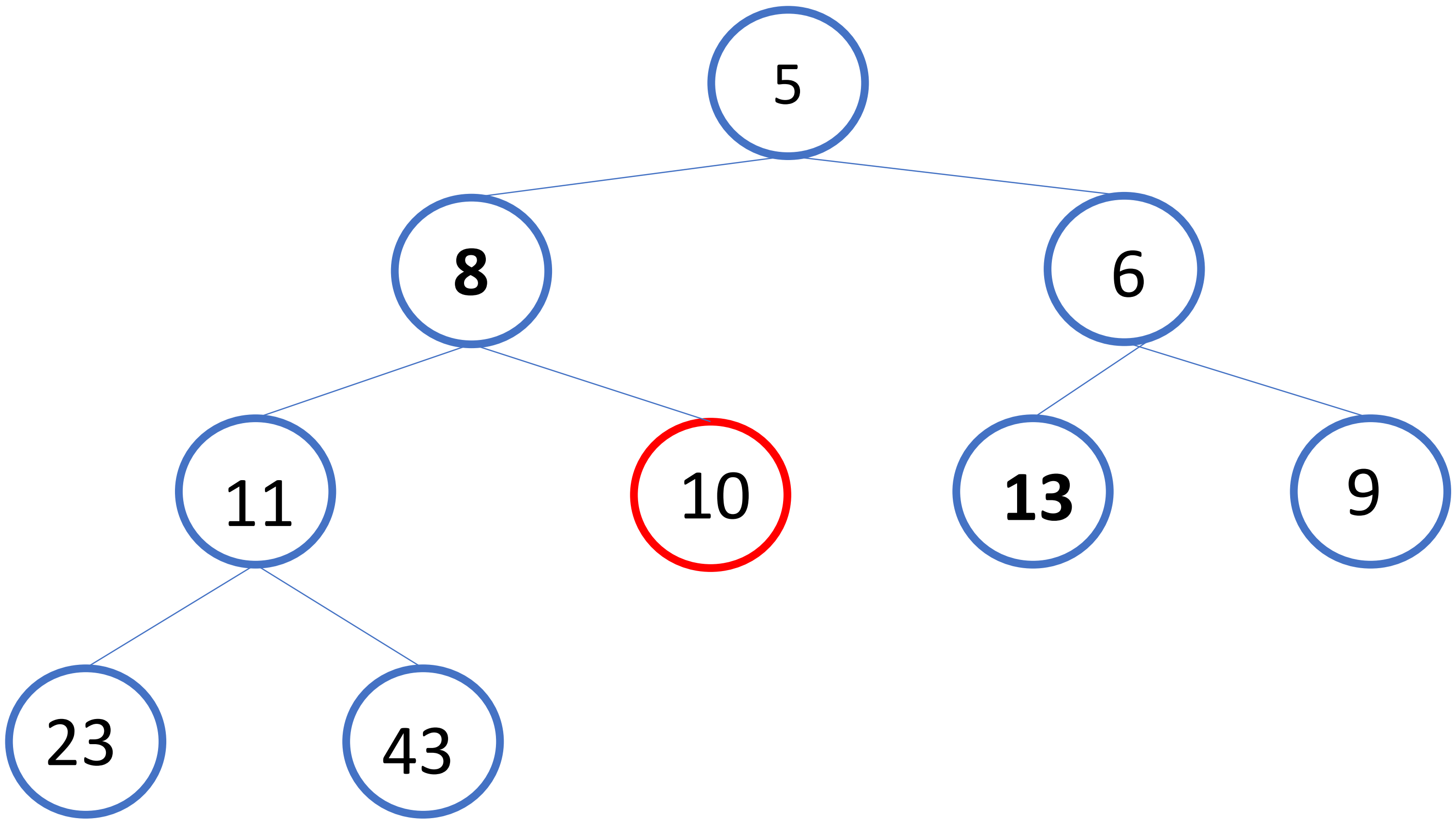
O(logn)

Where n is the size of the heap



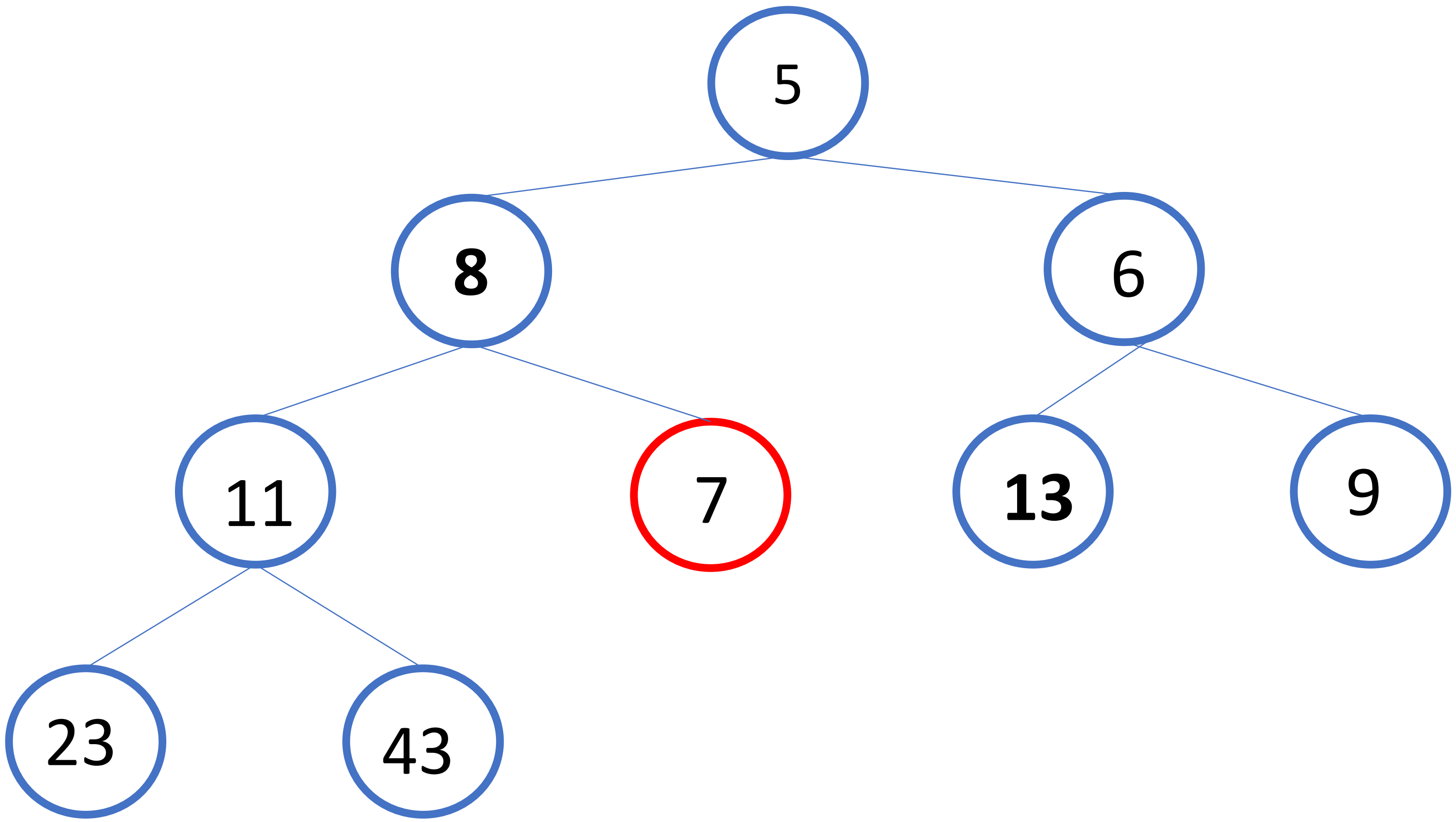
Binary heap

- **Decrease Key operation:**



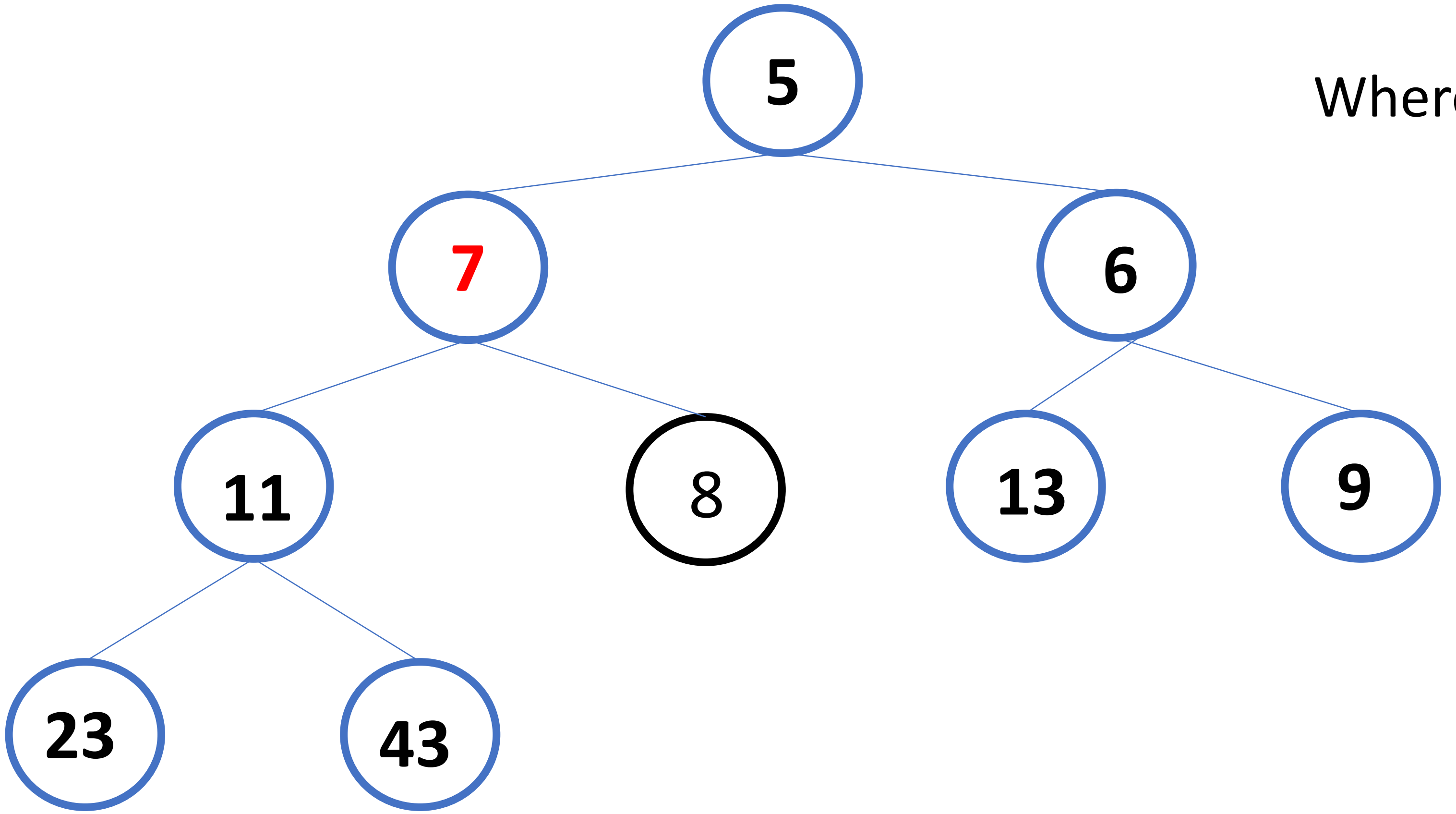
Binary heap

- **Decrease Key operation:**



Binary heap

- **Decrease Key operation:**



time:

O(logn)

Where n is the size of the heap

Implementation

- **Makequeue:**
  - O(n)
- **Exchange Operation:**
  - O(logn)
- **Decrease Key operation:**
  - O(logn)

Implementation

Prim (G= (V,E))

Q = Ø   // Q is the Priority Queue

Initialize each v ∈ V with key $K_v \leftarrow \infty$, $\prod_v \leftarrow$ NIL

Pick a starting node r and set $K_r \leftarrow 0$

Insert all nodes into Q with key K_v

While Q ≠ Ø :

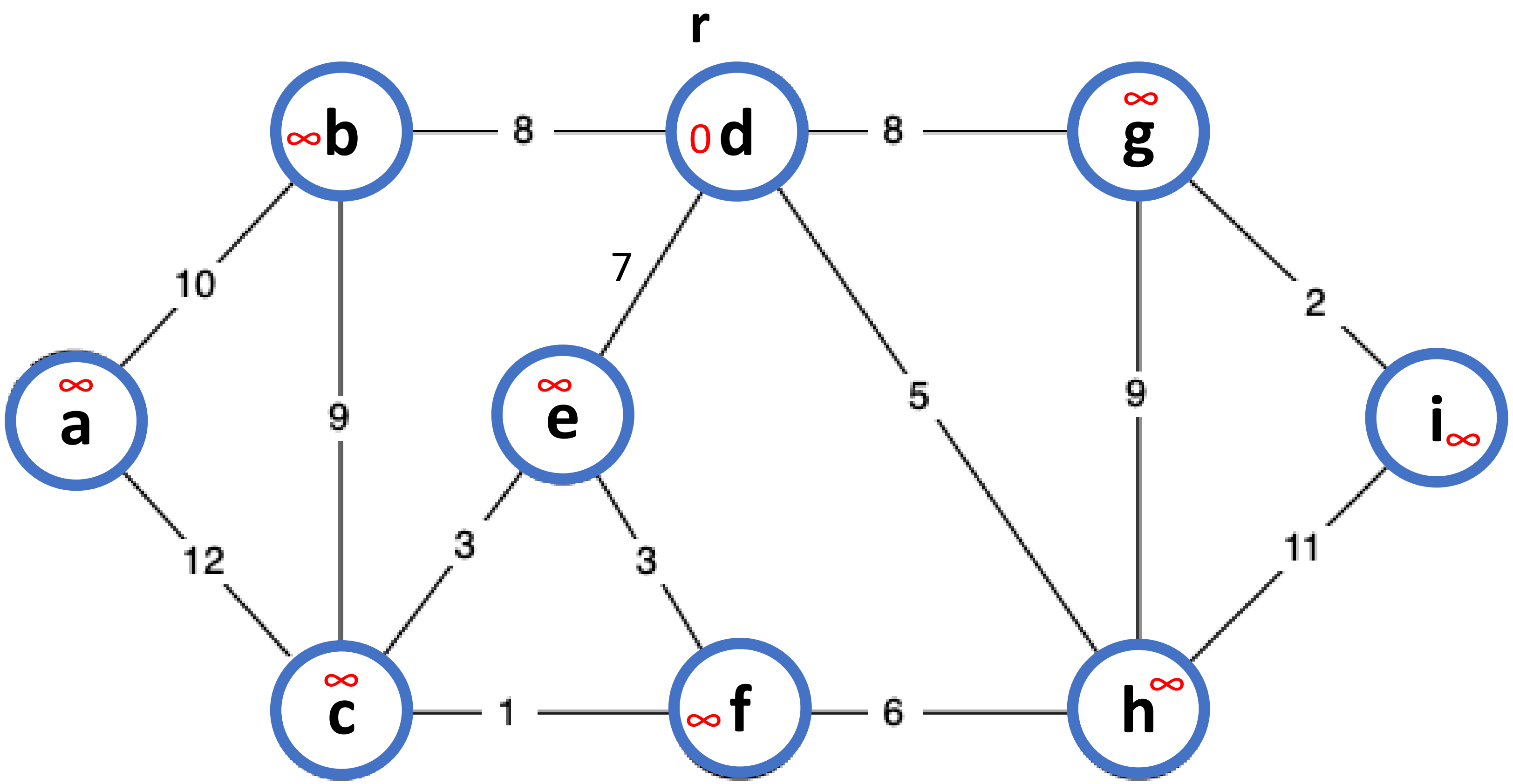
        Do u ← Extract-MIN(Q)

                for each v ∈ Adj(u):

                        do if v ∈ Q and w(u,v) < k_v:

                                then $\prod_v \leftarrow u$

                                DECREASE-KEY(Q, v, w(u,v))   //setting $k_v \leftarrow w(u,v)$



r
∞ b
0 d
∞ g
∞ a
∞ e
i ∞
∞ c
∞ f
h ∞
8
8
10
9
7
5
9
2
12
3
3
11
1
6

Implementation

Prim (G= (V,E))

Q = Ø   // Q is the Priority Queue

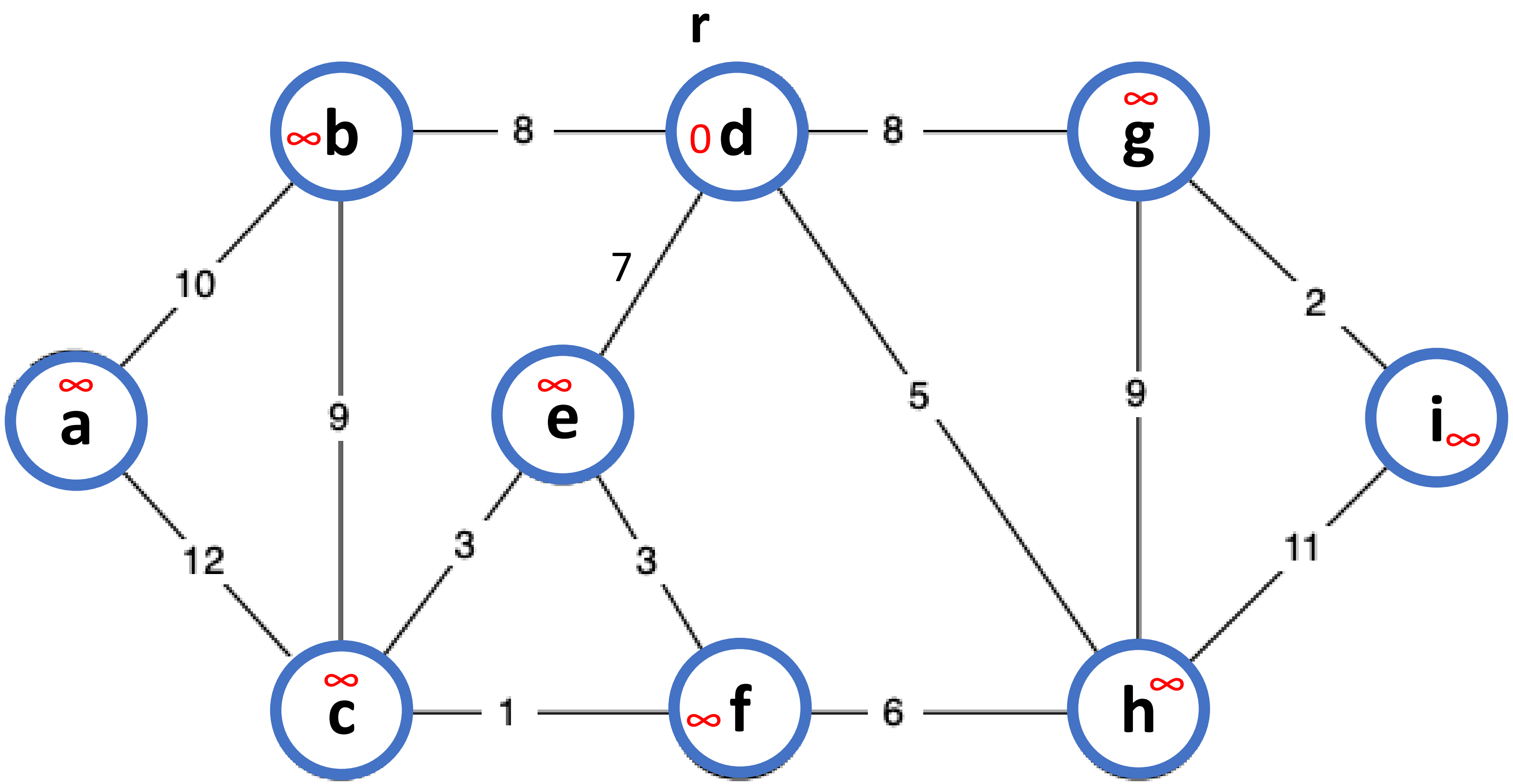
Initialize each v ∈ V with key $K_v \leftarrow \infty$, $\prod_v \leftarrow$ NIL

Pick a starting node r and set $K_r \leftarrow 0$

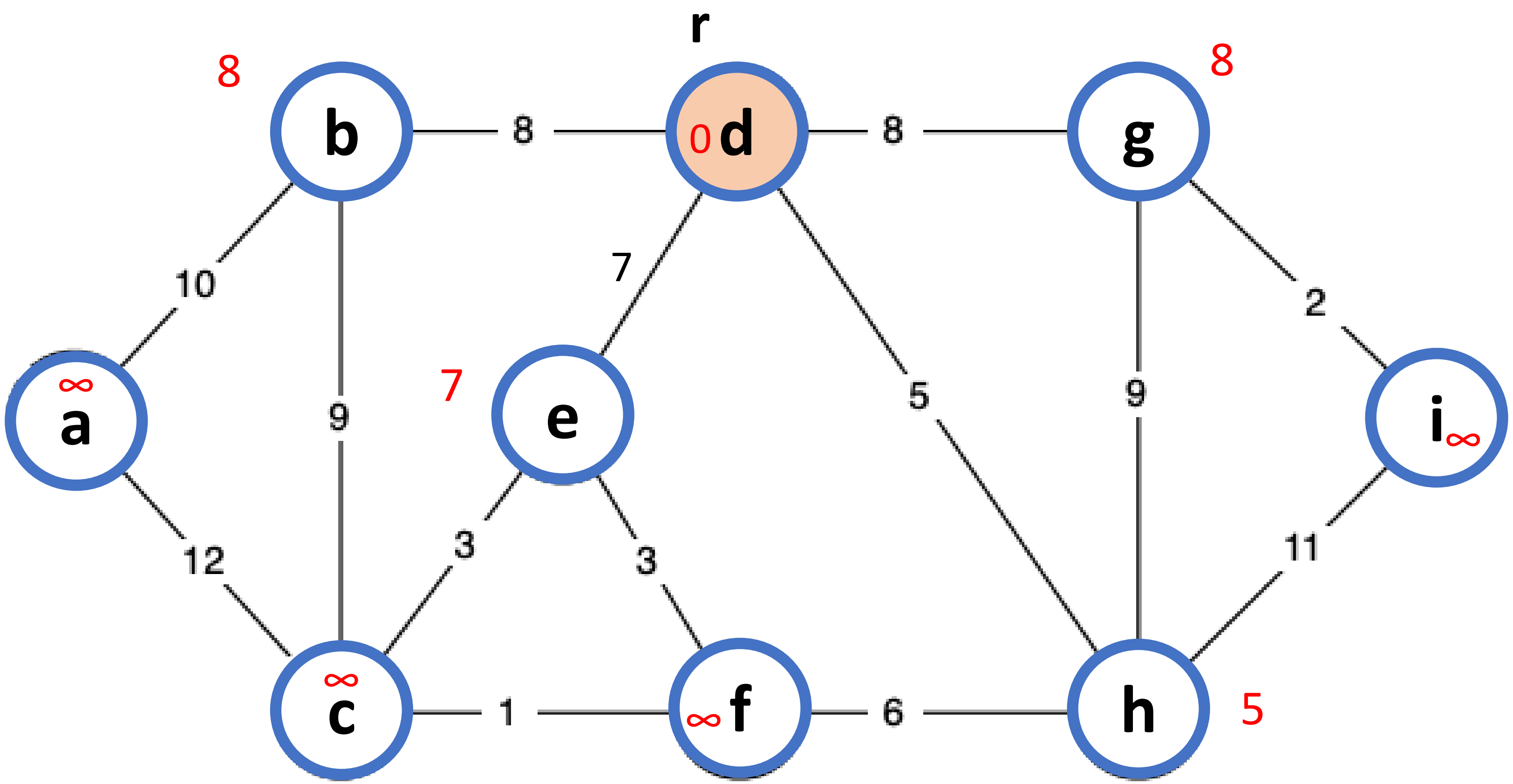
Insert all nodes into Q with key K_v

While Q ≠ Ø :

```
        Do u ← Extract-MIN(Q)
                for each v ∈ Adj(u):
                        do if v ∈ Q and w(u,v) < k_v:
                                then ∏_v ← u
                                DECREASE-KEY(Q, v, w(u,v))   //setting k_v ← w(u,v)
```



r
∞b
0d
∞
g
∞
a
∞
e
i∞
∞
c
∞f
h∞
8
8
10
9
7
5
9
2
12
3
3
11
1
6


r
b
d
g
a
e
i
c
f
h
8
0
8
∞
7
∞
∞
∞
5
8
8
10
9
7
5
9
2
12
3
3
11
1
6

Implementation

Prim (G= (V,E))

Q = ∅   // Q is the Priority Queue

Initialize each v ∈ V with key $K_v \leftarrow \infty$, $\prod_v \leftarrow$ NIL

Pick a starting node r and set $K_r \leftarrow 0$

Insert all nodes into Q with key K_v

While Q ≠ ∅ :

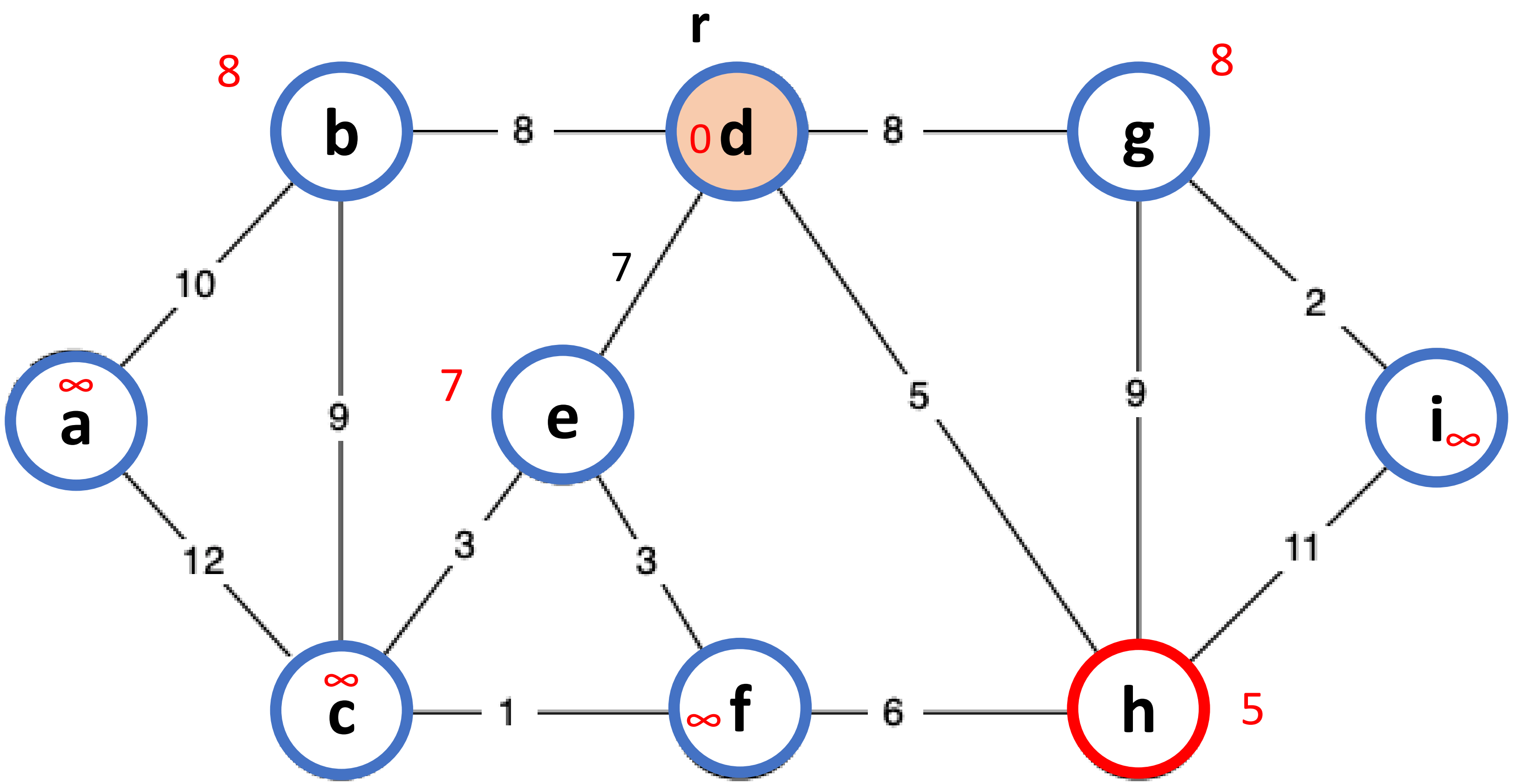
Do u ← Extract-MIN(Q)

for each v ∈ Adj(u):

do if v ∈ Q and w(u,v) < k_v:

then $\prod_v \leftarrow$ u

DECREASE-KEY(Q, v, w(u,v))   //setting $k_v \leftarrow$ w(u,v)



r
b
d
g
a
e
i
c
f
h
8
0
8
7
5
∞
∞
∞
∞
10
9
7
5
9
2
12
3
3
11
1
6
8
8

Implementation

Prim (G= (V,E))

Q = Ø // Q is the Priority Queue

Initialize each v ∊ V with key $K_v \leftarrow \infty$, $\prod_v \leftarrow$ NIL

Pick a starting node r and set $K_r \leftarrow 0$

Insert all nodes into Q with key K_v

While Q ≠ Ø :

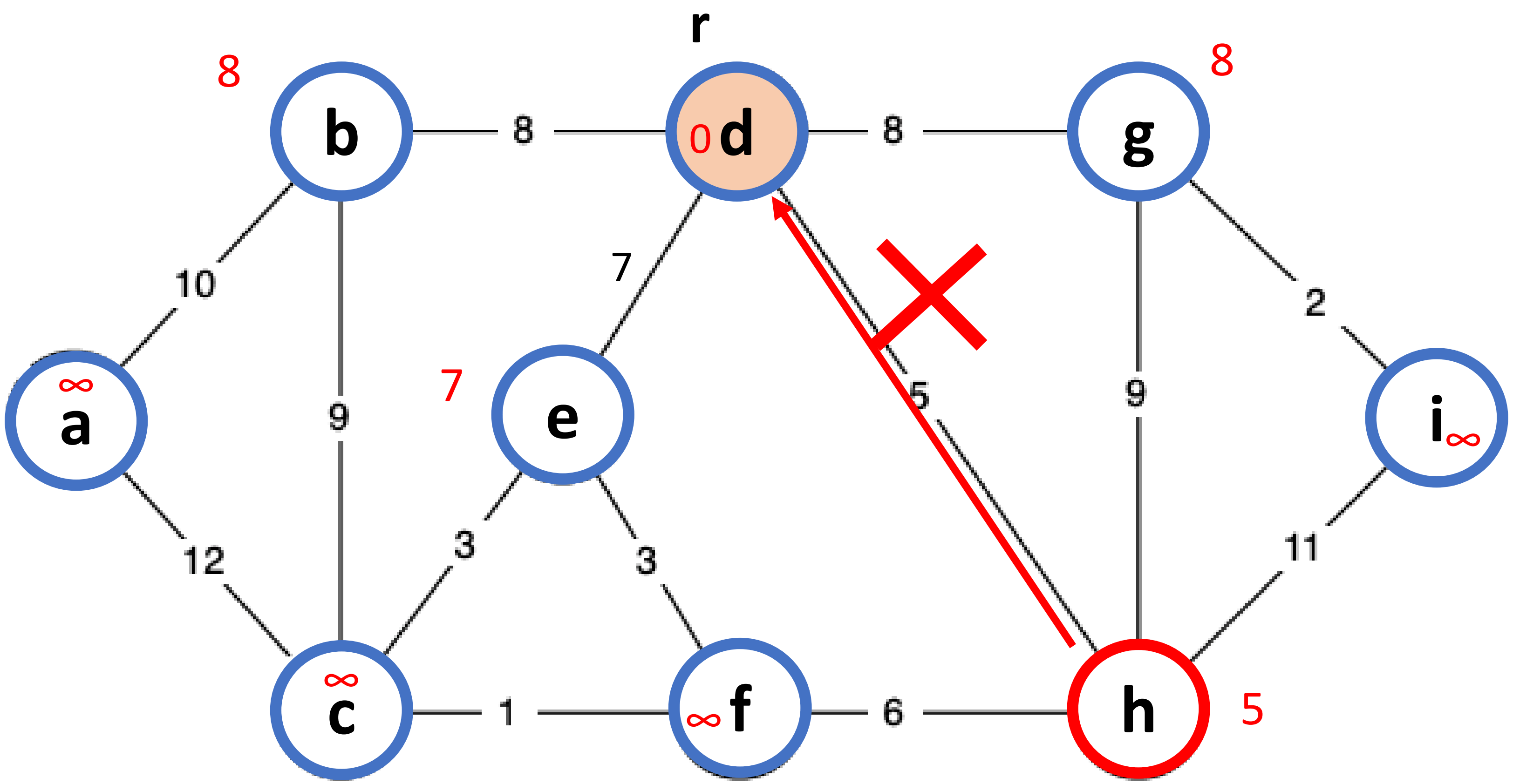
Do u ← Extract-MIN(Q)

for each v ∊ Adj(u):

do if v ∊ Q and $w(u,v) < k_v$:

then $\prod_v \leftarrow u$

DECREASE-KEY(Q, v, w(u,v)) //setting $k_v \leftarrow w(u,v)$


r
8
b
8
0 d
8
g
8
10
7
2
a
∞
9
7
e
5
9
i
∞
12
3
3
11
c
∞
1
∞ f
6
h
5

Implementation

Prim (G= (V,E))

Q = Ø   // Q is the Priority Queue

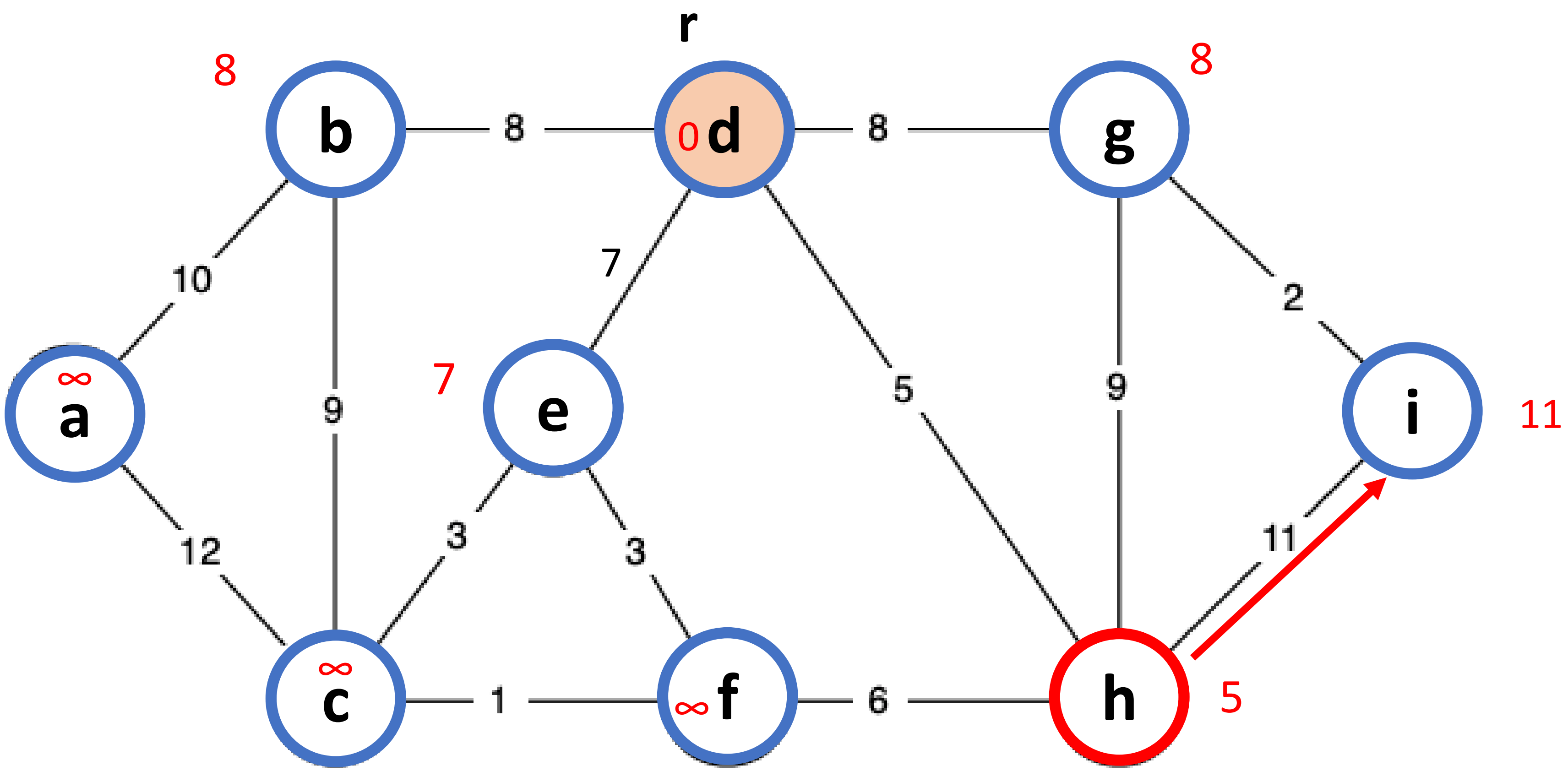
Initialize each v ∊ V with key $K_v \leftarrow \infty$, $\prod_v \leftarrow$ NIL

Pick a starting node r and set $K_r \leftarrow 0$

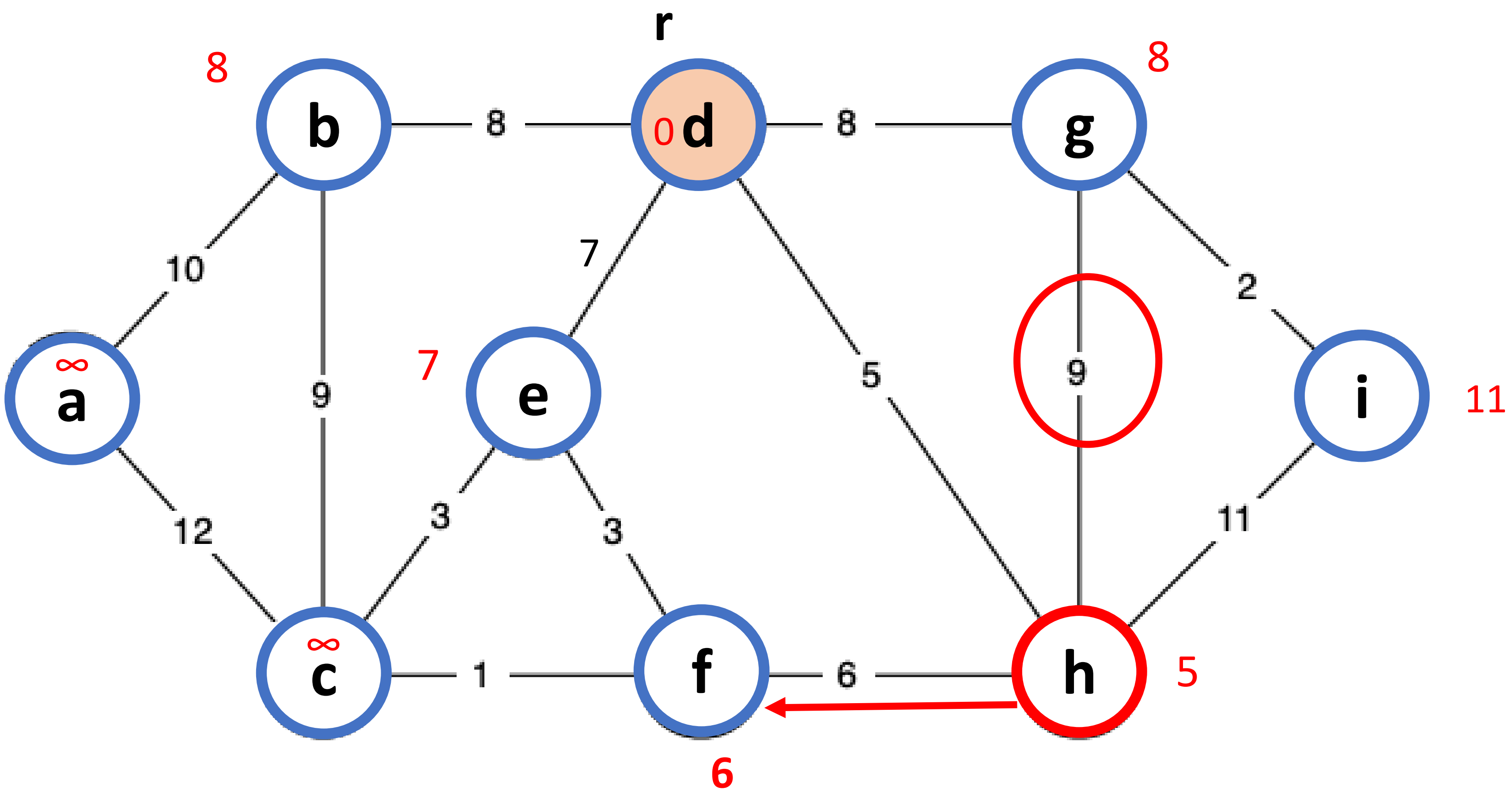
Insert all nodes into Q with key K_v

While Q ≠ Ø :

```
        Do u ← Extract-MIN(Q)
                for each v ∊ Adj(u):
                        do if v ∊ Q and w(u,v) < k_v:
                                then ∏_v ← u
                                DECREASE-KEY(Q, v, w(u,v))   //setting k_v ← w(u,v)
```



r
8
b
8
0 d
8
g
8
10
9
7
5
9
2
∞
a
7
e
i
11
12
3
3
11
∞
c
1
∞ f
6
h
5



r
8
b
8
0 d
8
g
8
10
7
2
5
9
9
∞
a
7
e
i
11
12
3
3
11
∞
c
1
f
6
h
5
6

Implementation

Prim (G= (V,E))

Q = ∅   // Q is the Priority Queue

Initialize each v ∈ V with key $K_v \leftarrow \infty$, $\prod_v \leftarrow$ NIL

Pick a starting node r and set $K_r \leftarrow 0$

Insert all nodes into Q with key K_v

While Q ≠ ∅ :

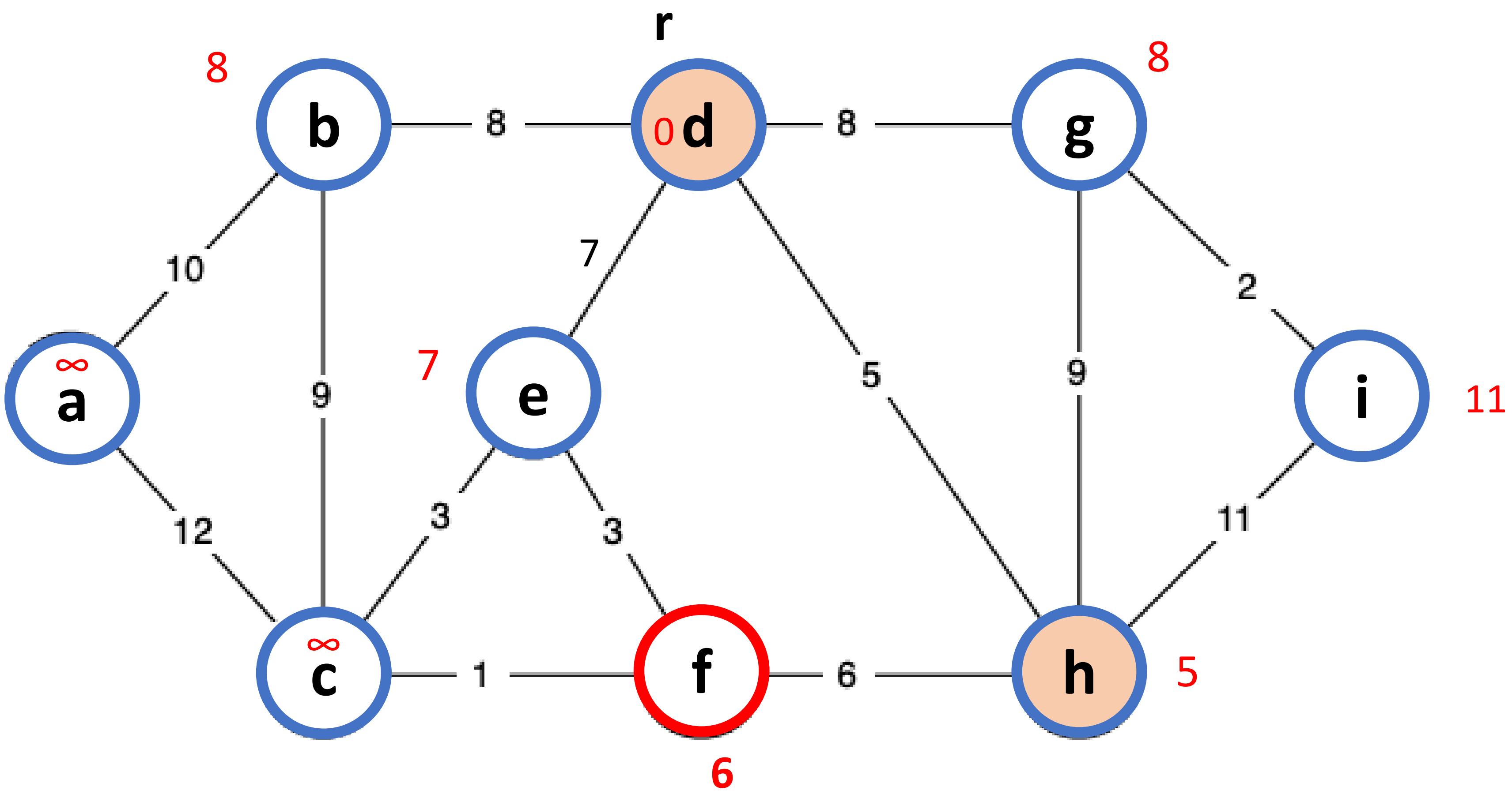
Do u ← Extract-MIN(Q)

for each v ∈ Adj(u):

do if v ∈ Q and w(u,v) < k_v:

then $\prod_v \leftarrow$ u

DECREASE-KEY(Q, v, w(u,v))   //setting $k_v \leftarrow$ w(u,v)



r
b
d
g
a
e
i
c
f
h
8
0
8
∞
7
11
∞
6
5
10
9
12
3
3
1
6
7
5
9
2
11
8
8

Implementation

Prim (G= (V,E))

Q = ∅   // Q is the Priority Queue

Initialize each $v \in V$ with key $K_v \leftarrow \infty$, $\prod_v \leftarrow$ NIL

Pick a starting node r and set $K_r \leftarrow 0$

Insert all nodes into Q with key K_v

While $Q \neq \emptyset$:

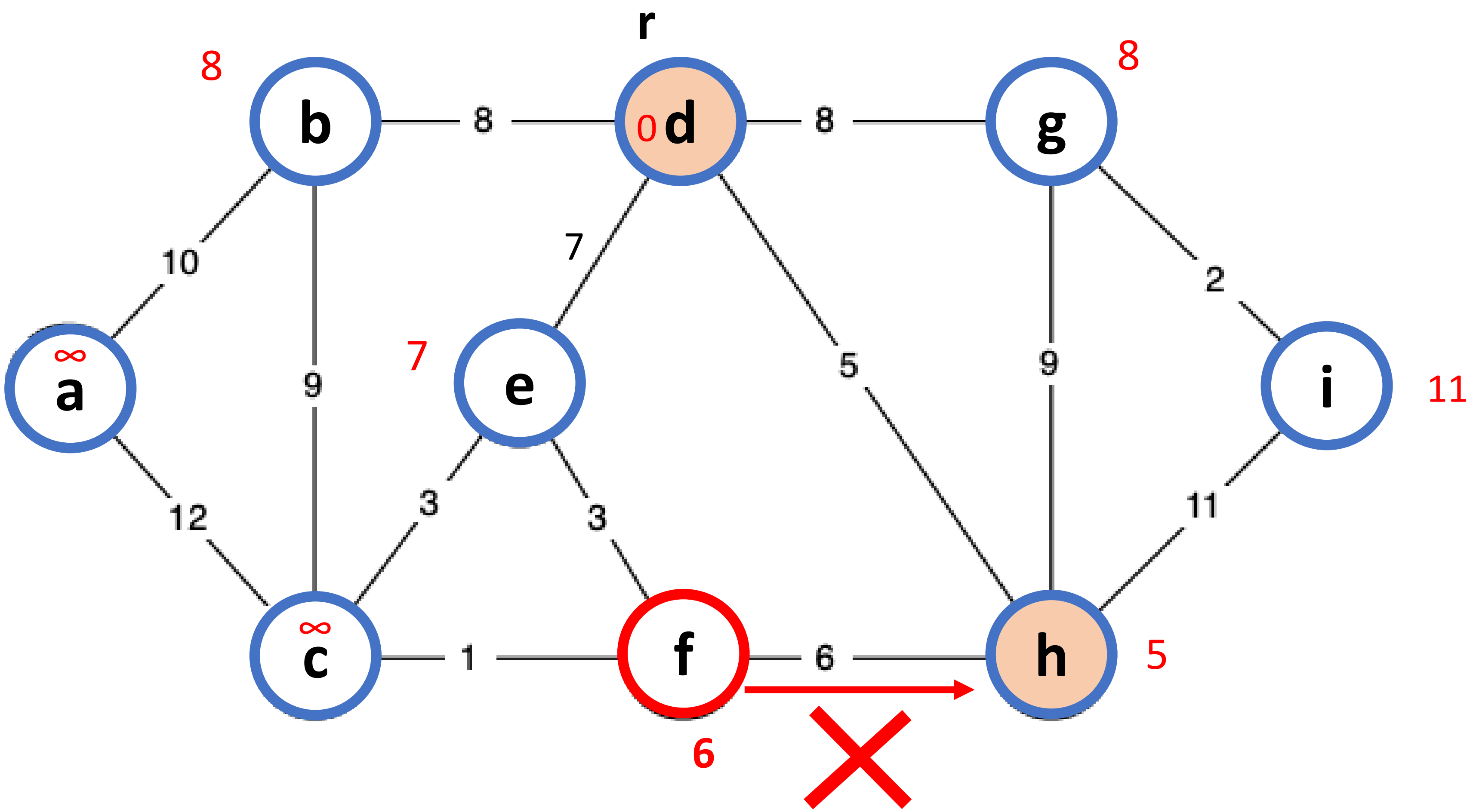
        Do $u \leftarrow$ Extract-MIN(Q)

                for each $v \in Adj(u)$:

                        do if $v \in Q$ and $w(u,v) < k_v$:

                                then $\prod_v \leftarrow u$

                                DECREASE-KEY(Q, v, w(u,v))   //setting $k_v \leftarrow w(u,v)$

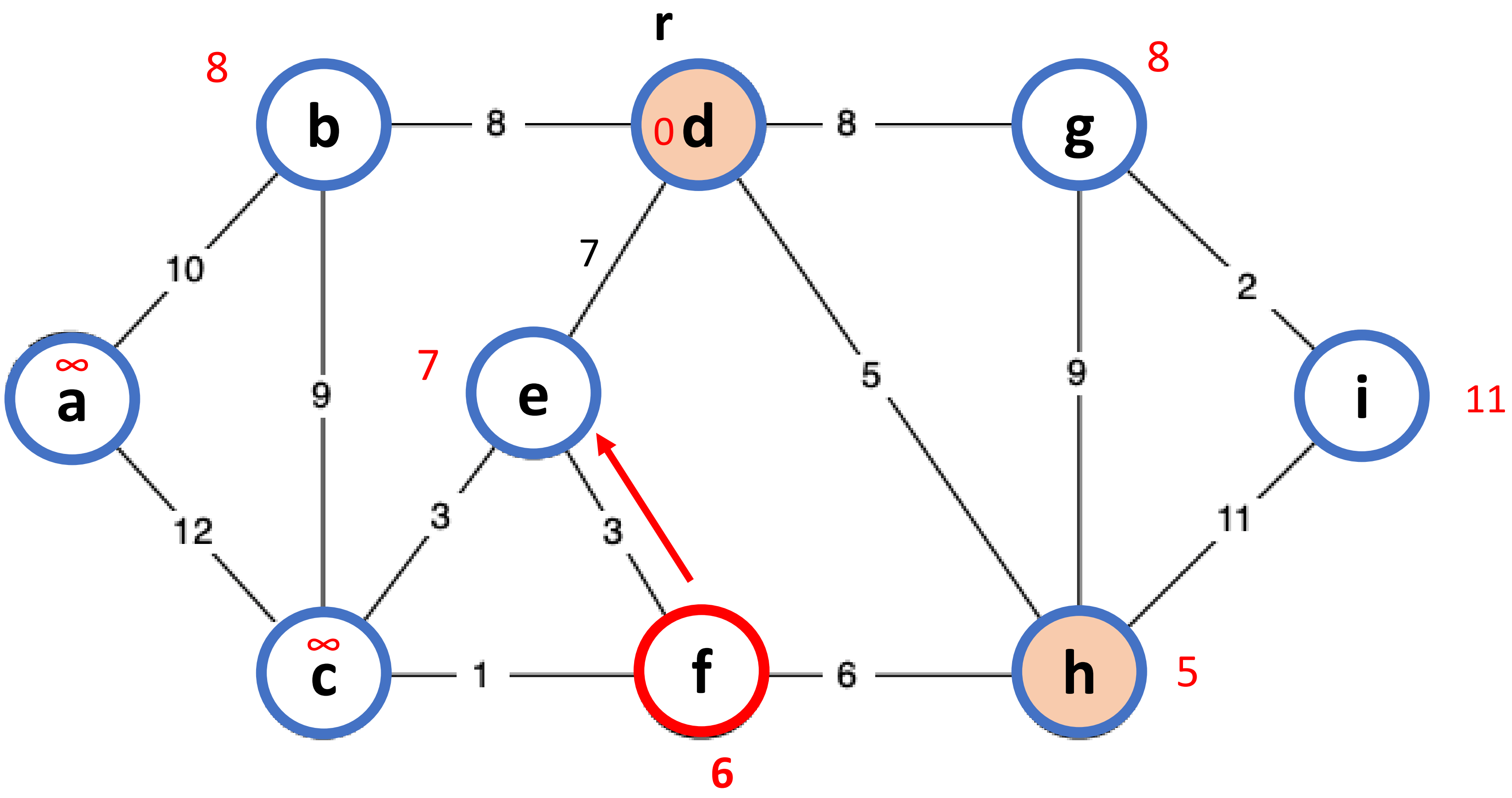

r
8
b
8
0 d
8
g
8
10
7
5
2
9
9
∞
a
7
e
i
11
12
3
3
11
∞
c
1
f
6
h
5
6

Implementation

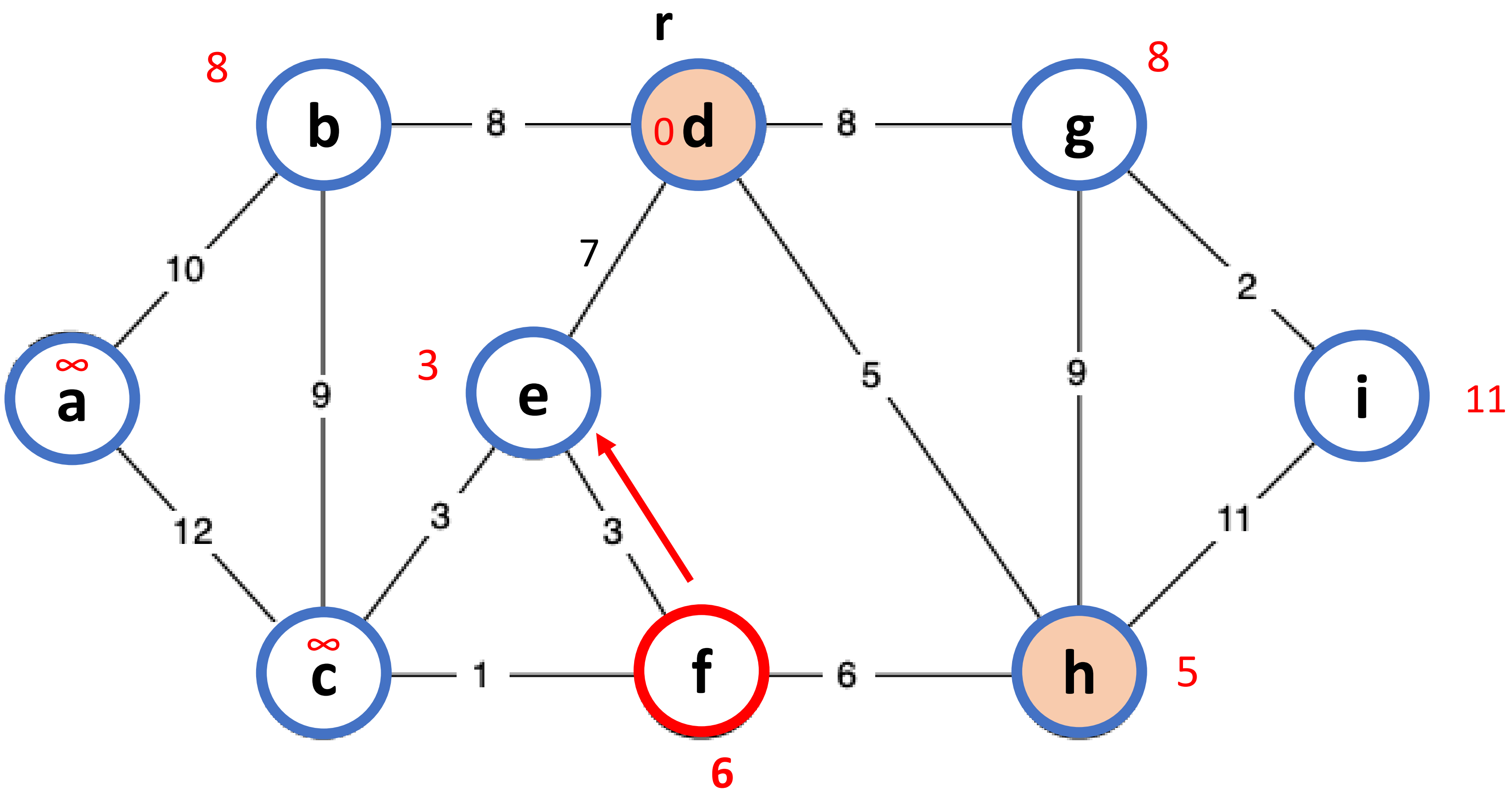
Prim (G= (V,E))
Q = ∅ // Q is the Priority Queue
Initialize each $v \in V$ with key $K_v \leftarrow \infty$, $\Pi_v \leftarrow$ NIL
Pick a starting node r and set $K_r \leftarrow 0$
Insert all nodes into Q with key K_v

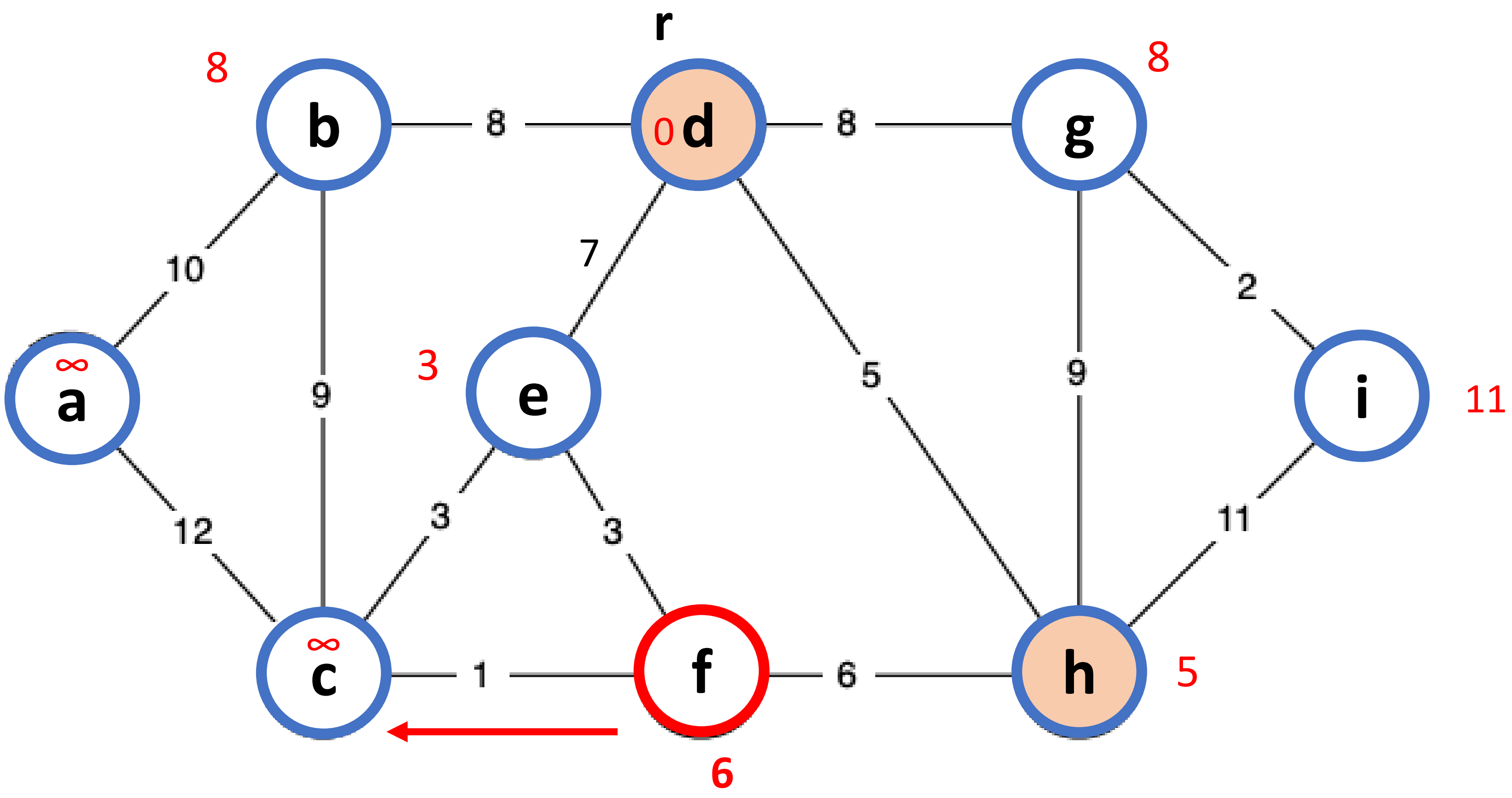
While Q ≠ ∅ :

    Do u ← Extract-MIN(Q)
        for each $v \in Adj(u)$:
            do if $v \in Q$ and $w(u,v) < k_v$:
                then $\Pi_v \leftarrow u$
                DECREASE-KEY(Q, v, w(u,v)) //setting $k_v \leftarrow w(u,v)$

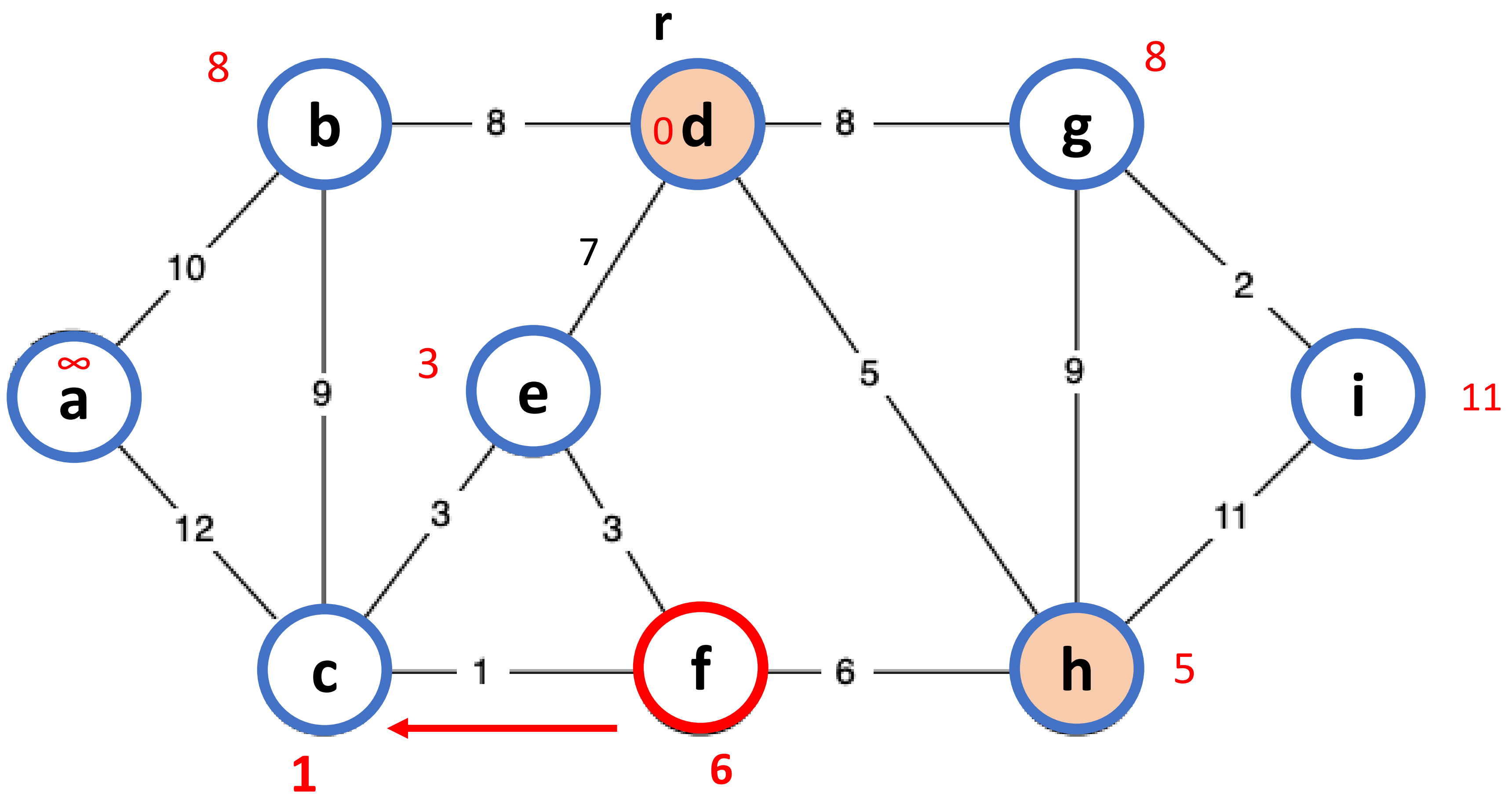


r
b
d
g
a
e
i
c
f
h
8
0
8
∞
7
11
∞
6
5
10
9
12
7
3
3
1
6
5
9
2
11
8
8


r
8
b
8
0 d
8
g
8
10
7
2
∞
a
9
3
e
5
9
i
11
12
3
3
11
∞
c
1
f
6
h
5
6

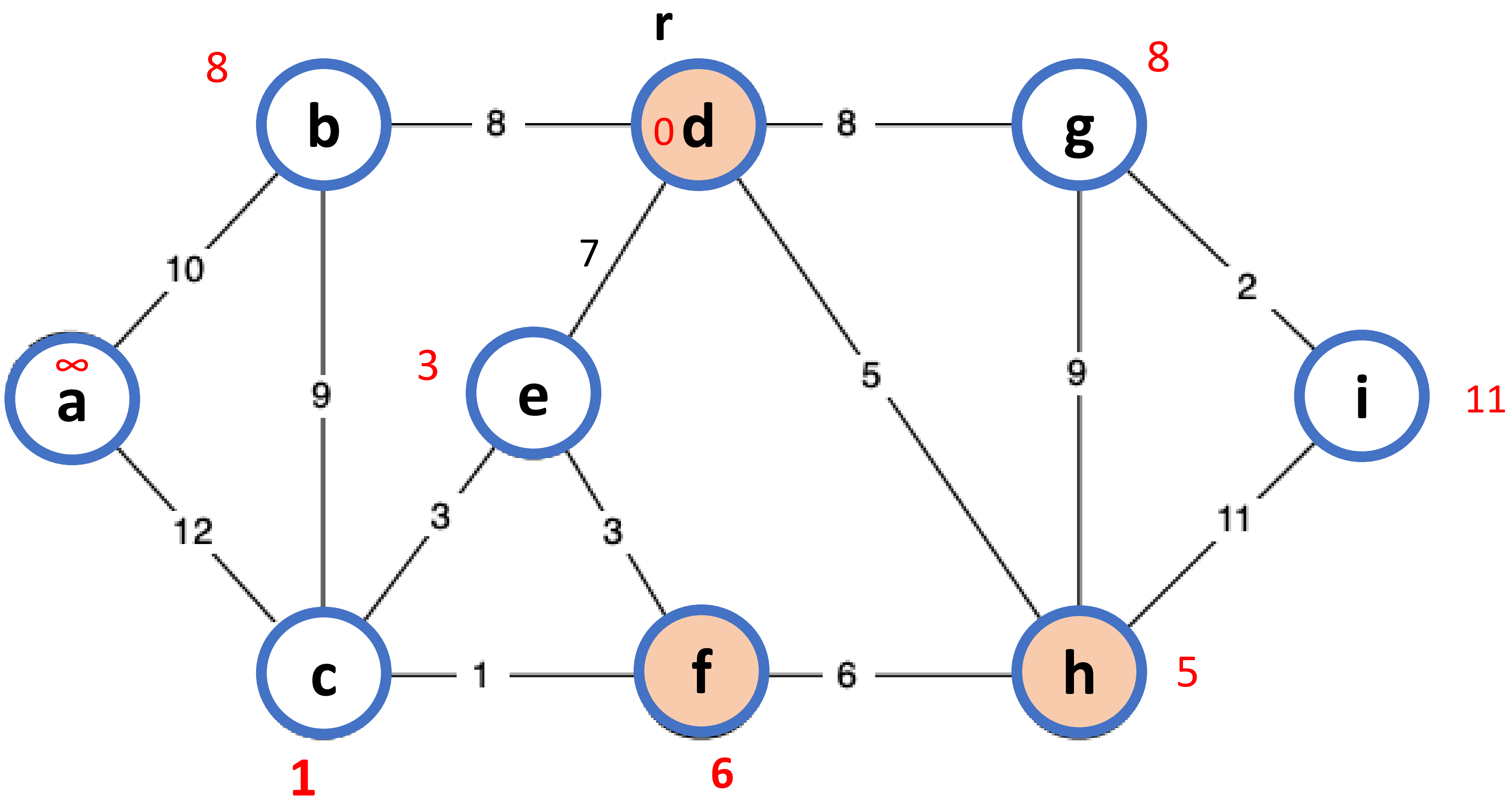


r
8
b
8
0
d
8
g
8
10
9
7
5
9
2
∞
a
3
e
i
11
12
3
3
11
∞
c
1
f
6
h
5
6



r
8
b
8
0 d
8
g
8
10
7
2
a
∞
9
3
e
5
9
i
11
12
3
3
11
c
1
f
6
h
5
1
6

Number of call of : DECREASE-KEY ?



Implementation

Prim (G= (V,E))

Q = Ø   // Q is the Priority Queue

Initialize each v ∈ V with key $K_v \leftarrow \infty$, $\prod_v \leftarrow$ NIL

Pick a starting node r and set $K_r \leftarrow 0$

Insert all nodes into Q with key K_v

While Q ≠ Ø :

    Do u ← Extract-MIN(Q)

        for each v ∈ Adj(u):

            do if v ∈ Q and w(u,v) < k_v:

                then $\prod_v \leftarrow u$

                DECREASE-KEY(Q, v, w(u,v))   //setting $k_v \leftarrow$ w(u,v)

Implementation

Prim (G= (V,E))
$Q = \emptyset$ // Q is the Priority Queue
Initialize each $v \in V$ with key $K_v \leftarrow \infty$, $\prod_v \leftarrow NIL$
Pick a starting node r and set $K_r \leftarrow 0$
Insert all nodes into Q with key K_v

O(vlogv)

While $Q \neq \emptyset$:

Do $u \leftarrow$ Extract-MIN(Q)

for each $v \in Adj(u)$:

do if $v \in Q$ and $w(u,v) < k_v$:

then $\prod_v \leftarrow u$

DECREASE-KEY(Q, v, w(u,v)) //setting $k_v \leftarrow w(u,v)$

Implementation

- **Makequeue:**
  - O(n), one insert O(logn)
- **Exchange Operation:**
  - O(logn)
- **Decrease Key operation:**
  - O(logn)

Implementation

Prim (G= (V,E))

Q = Ø   // Q is the Priority Queue

Initialize each $v \in V$ with key $K_v \leftarrow \infty$, $\prod_v \leftarrow$ NIL

Pick a starting node r and set $K_r \leftarrow 0$

Insert all nodes into Q with key K_v

O(vlogv)

While Q ≠ Ø :

Do $u \leftarrow$ Extract-MIN(Q) → O(vlogv)

for each $v \in Adj(u)$:

do if $v \in Q$ and $w(u,v) < k_v$:

then $\prod_v \leftarrow u$

DECREASE-KEY(Q, v, w(u,v))   //setting $k_v \leftarrow w(u,v)$

Implementation

Prim (G= (V,E))

Q = ∅   // Q is the Priority Queue

Initialize each $v \in V$ with key $K_v \leftarrow \infty$, $\prod_v \leftarrow$ NIL

Pick a starting node r and set $K_r \leftarrow 0$

Insert all nodes into Q with key K_v

O(vlogv)

While Q $\neq$ ∅ :

Do u $\leftarrow$ Extract-MIN(Q) $\longrightarrow$ O(vlogv)

for each $v \in$ Adj(u):

do if $v \in Q$ and $w(u,v) < k_v$:

then $\prod_v \leftarrow u$

DECREASE-KEY(Q, v, w(u,v))   //setting $k_v \leftarrow w(u,v)$

E

Implementation

Prim (G= (V,E))
Q = ∅ // Q is the Priority Queue
Initialize each $v \in V$ with key $K_v \leftarrow \infty$, $\Pi_v \leftarrow$ NIL
Pick a starting node r and set $K_r \leftarrow 0$
Insert all nodes into Q with key K_v

O(vlogv)

While Q ≠ ∅ :

Do u ← Extract-MIN(Q) → O(vlogv)

for each $v \in$ Adj(u):
do if $v \in Q$ and $w(u,v) < k_v$:
then $\Pi_v \leftarrow u$
DECREASE-KEY(Q, v, w(u,v)) //setting $k_v \leftarrow w(u,v)$

E

O(Elogv)

Greedy

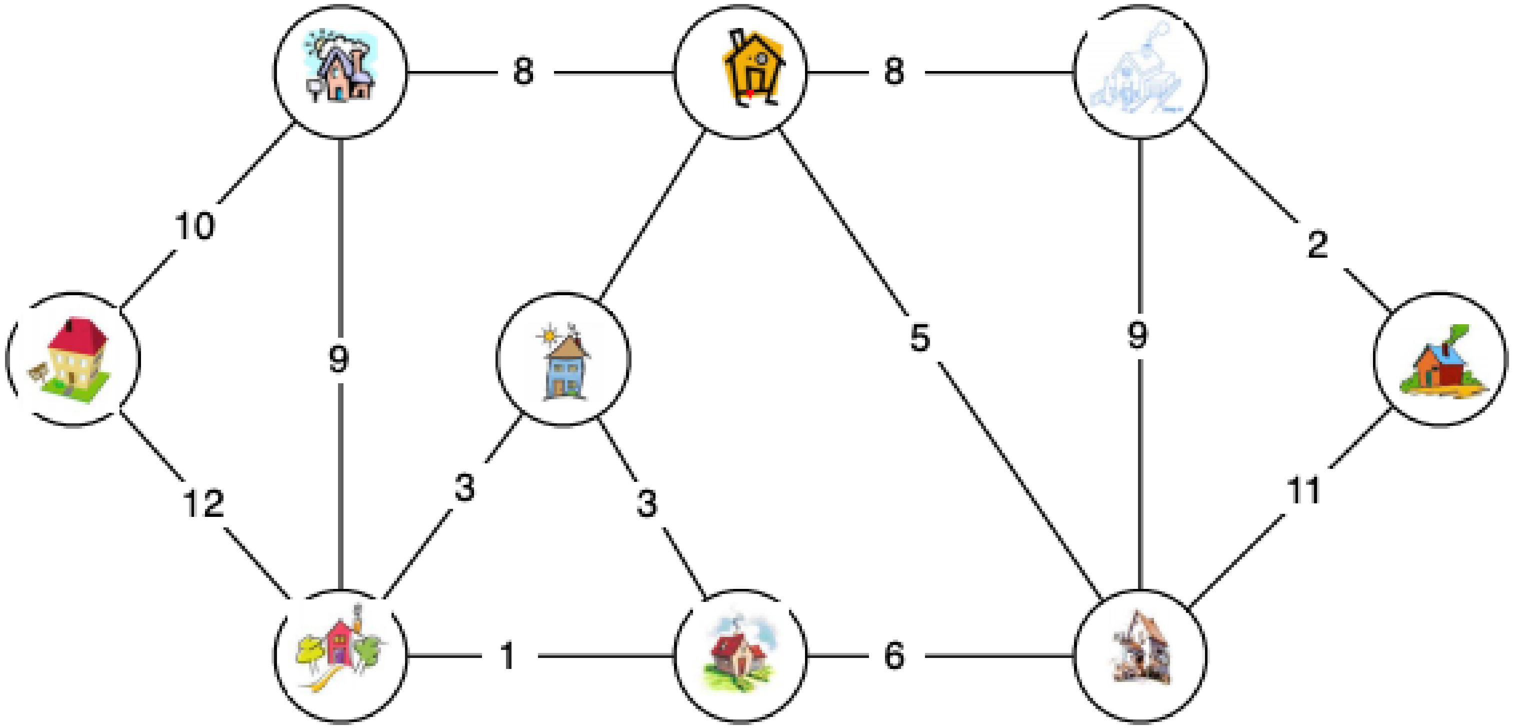
Lecture 5

Shortest Path Property

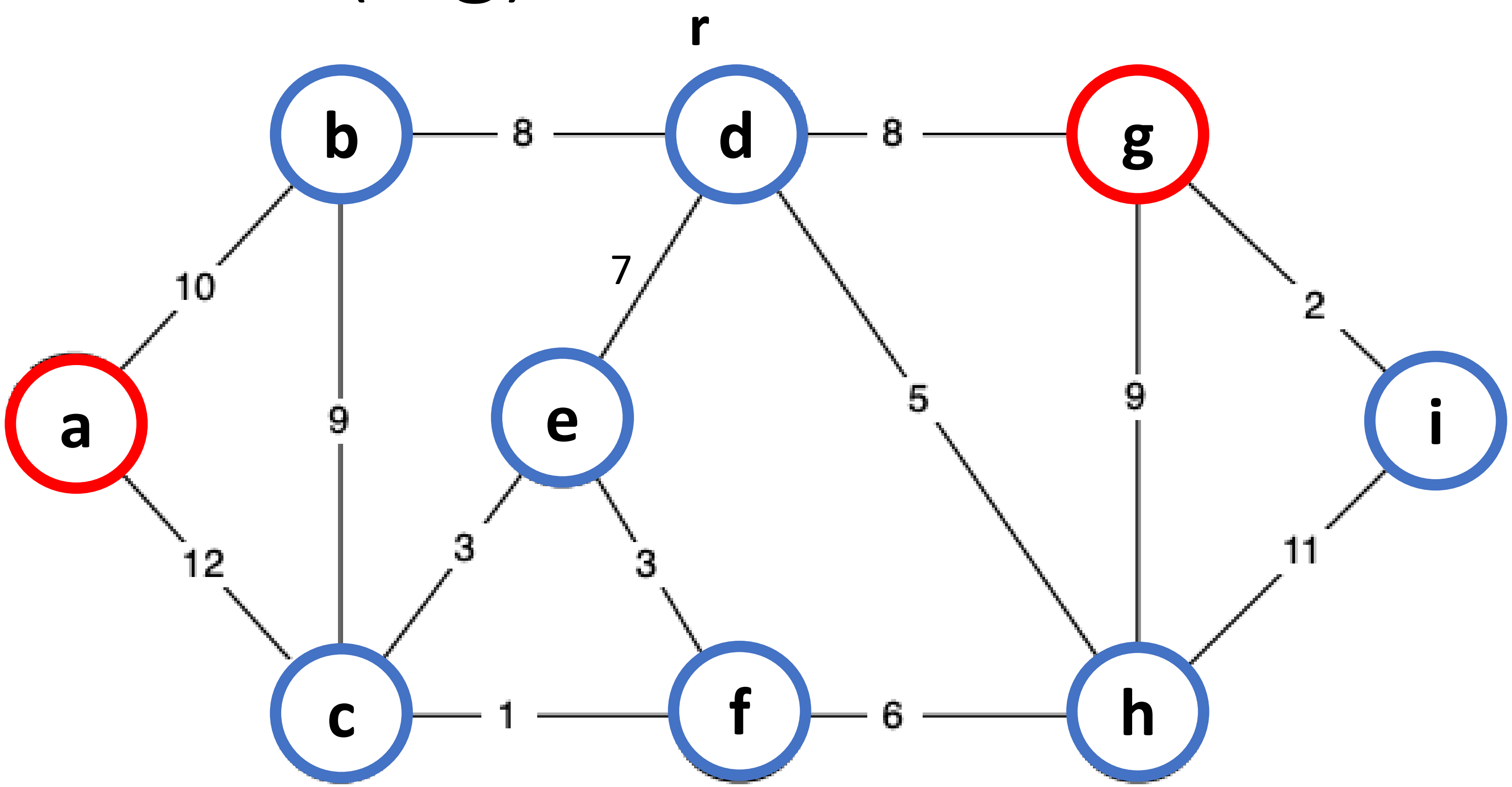
- Definition:
  - δ (s,v) = length of the shortest path from s to v in G

Shortest Path Property

- Definition:
  - δ (s,v) = length of the shortest path from s to v in G

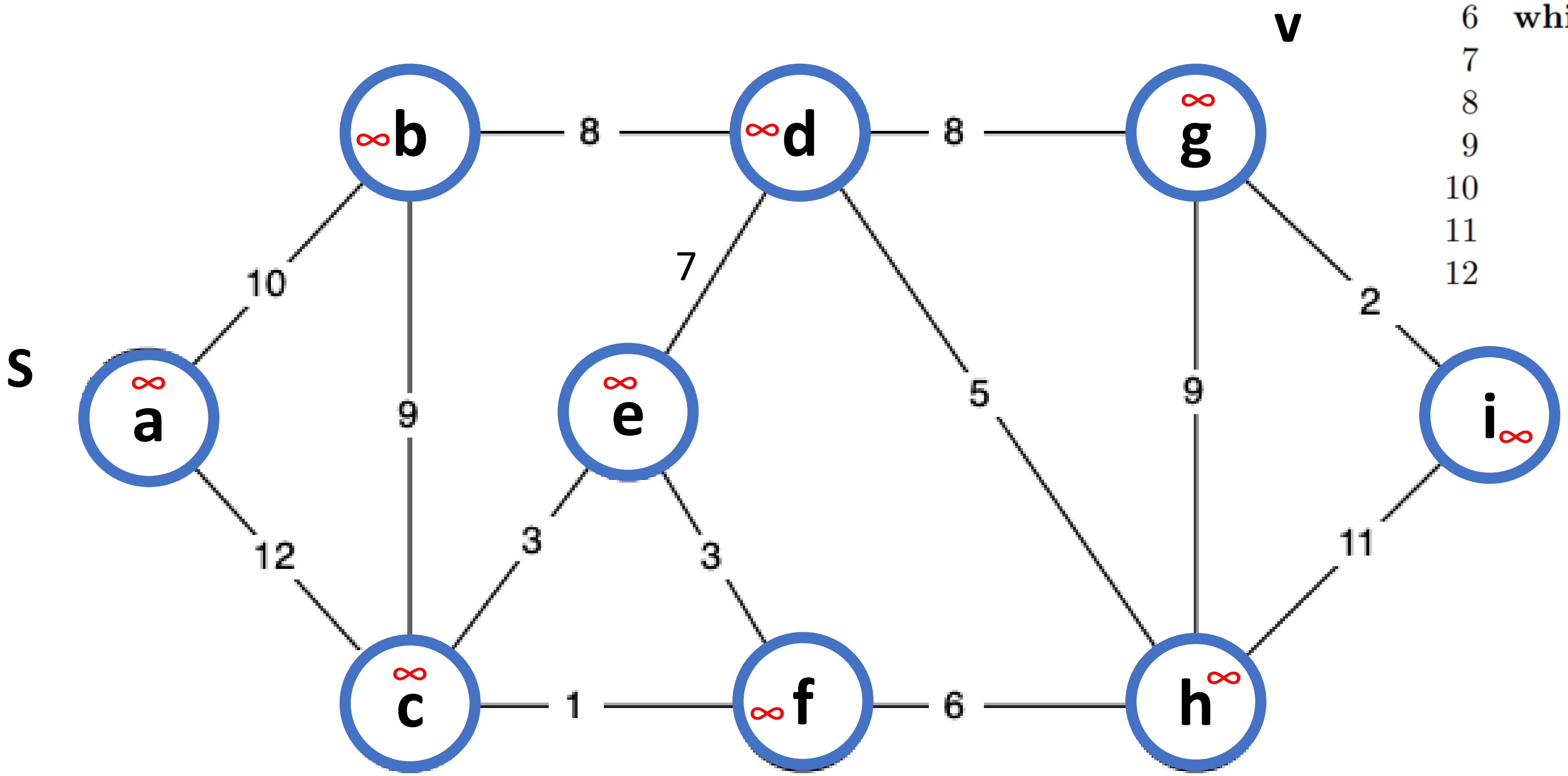


Shortest Path (a,g)



```
DIJKSTRA(G = (V, E), s)
for all v ∈ V
        do d_u ← ∞
           π_u ← NIL
d_s ← 0
Q ← MAKEQUEUE(V)      ▷ use d_u as key
while Q ≠ ∅
        do u ← EXTRACTMIN(Q)
           for each v ∈ Adj(u)
               do if d_v > d_u + w(u, v)
                     then d_v ← d_u + w(u, v)
                          π_v ← u
                          DECREASEKEY(Q, v)
```

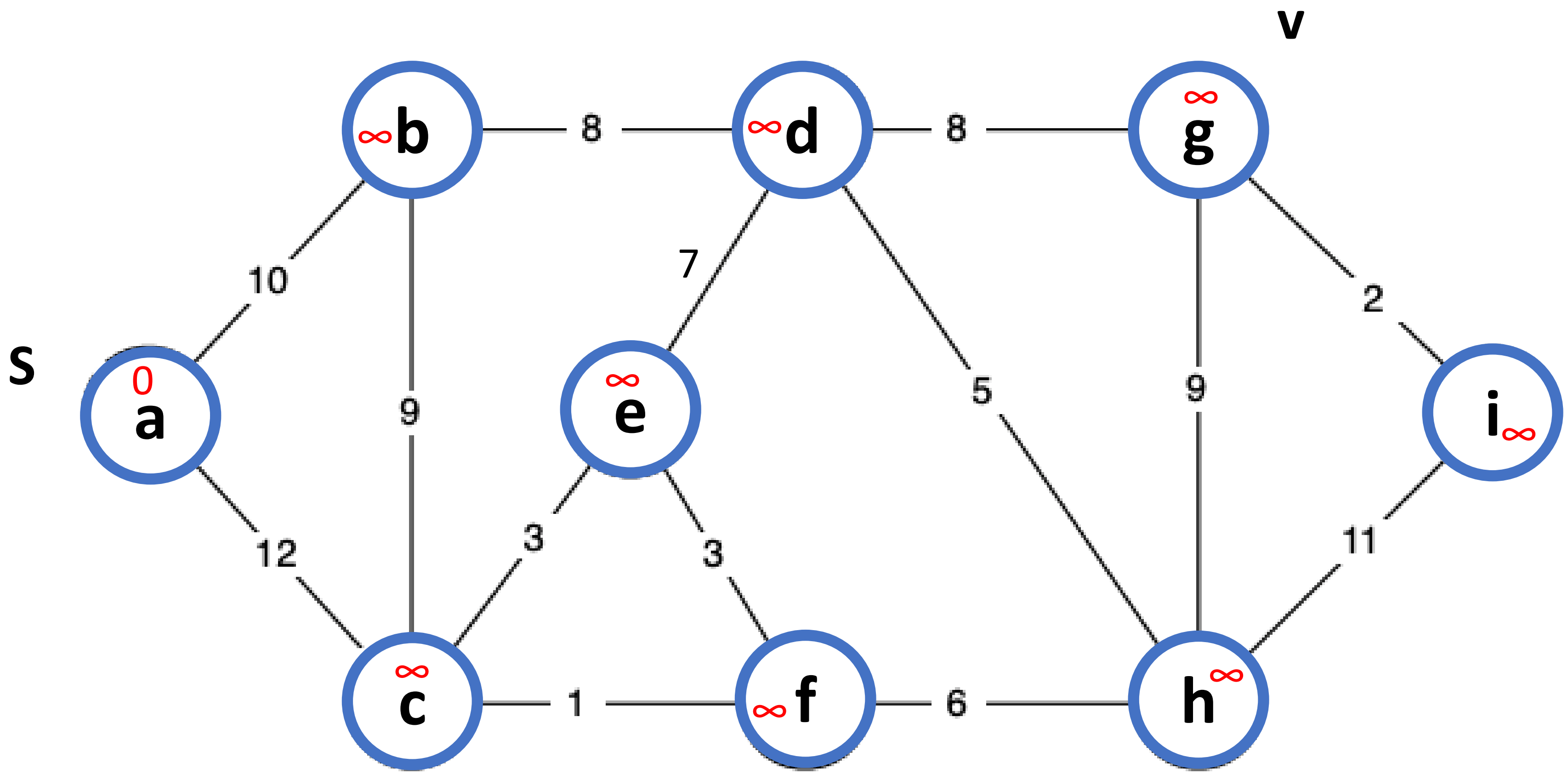
- Definition: δ (s,v) = length of the shortest path from s to v in G



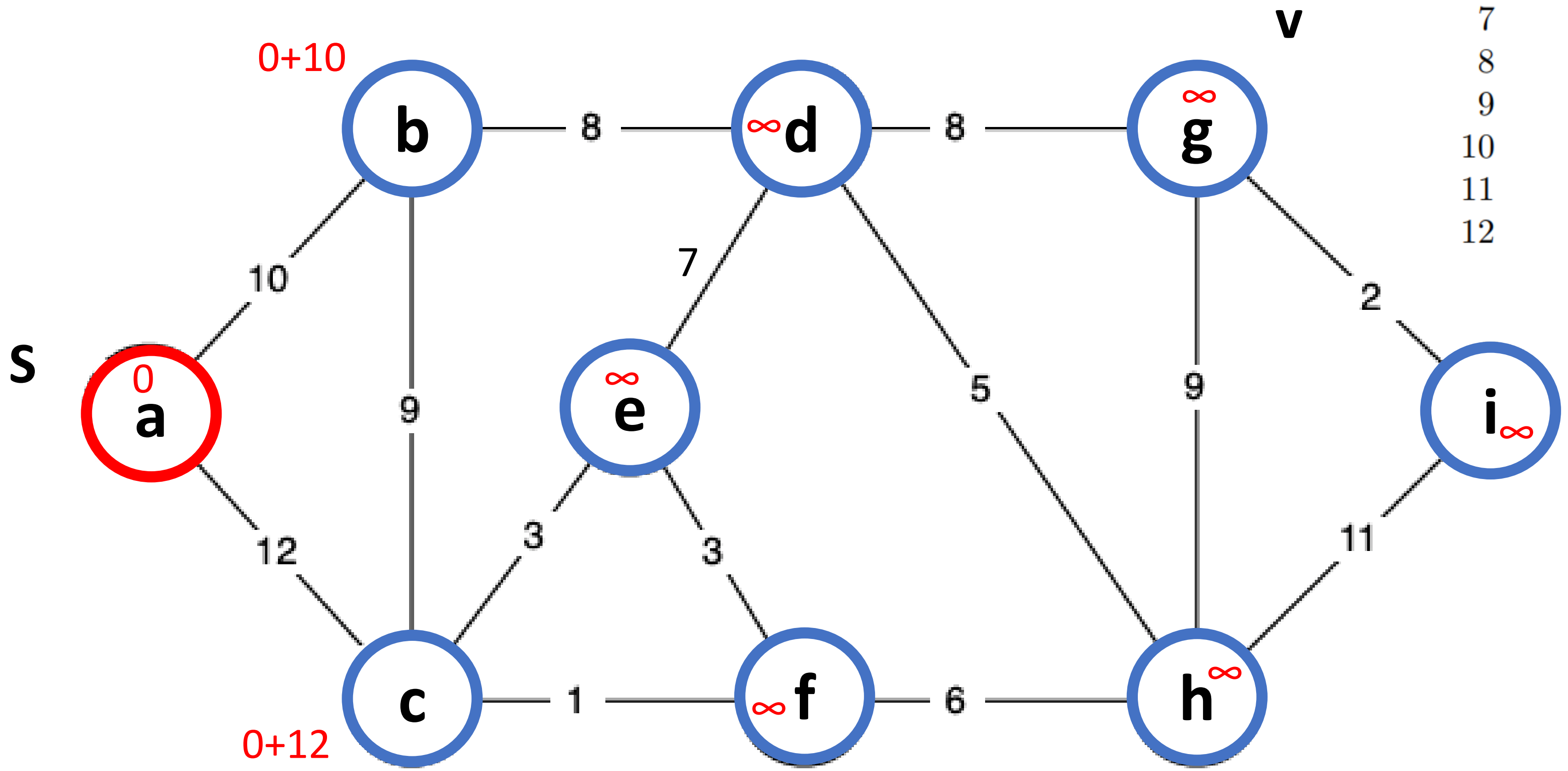
$\textsc{Dijkstra}(G = (V, E), s)$

```
for all v ∈ V
        do d_u ← ∞
           π_u ← NIL
d_s ← 0
Q ← MAKEQUEUE(V)      ▷ use d_u as key
while Q ≠ ∅
        do u ← EXTRACTMIN(Q)
           for each v ∈ Adj(u)
                do if d_v > d_u + w(u, v)
                      then d_v ← d_u + w(u, v)
                           π_v ← u
                           DECREASEKEY(Q, v)
```

- Definition: δ (s,v) = length of the shortest path from s to v in G



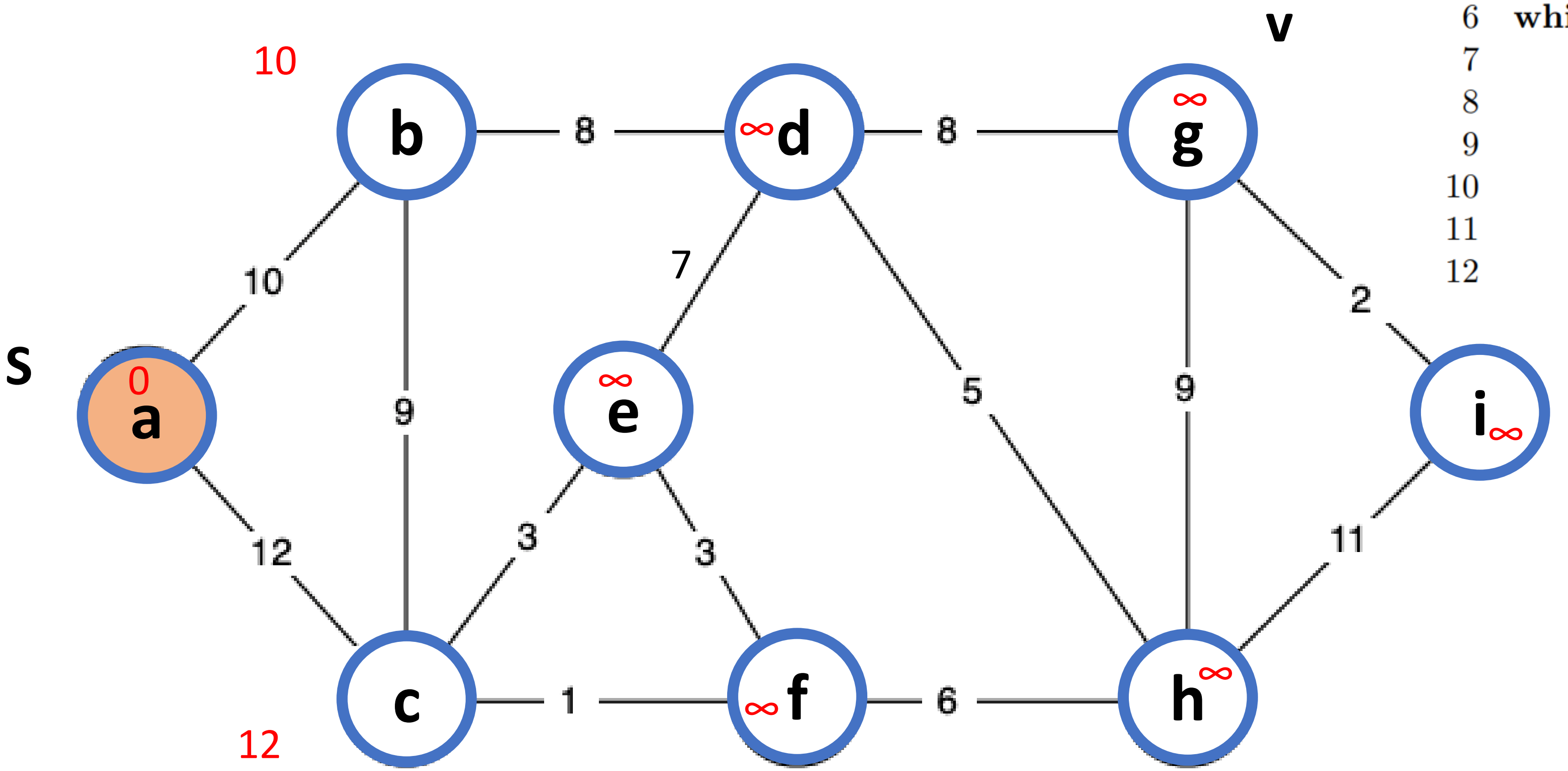
- Definition: δ (s,v) = length of the shortest path from s to v in G



DIJKSTRA$(G = (V, E), s)$

```
for all v ∈ V
        do d_u ← ∞
           π_u ← NIL
d_s ← 0
Q ← MAKEQUEUE(V)     ▷ use d_u as key
while Q ≠ ∅
        do u ← EXTRACTMIN(Q)
           for each v ∈ Adj(u)
               do if d_v > d_u + w(u, v)
                      then d_v ← d_u + w(u, v)
                           π_v ← u
                           DECREASEKEY(Q, v)
```

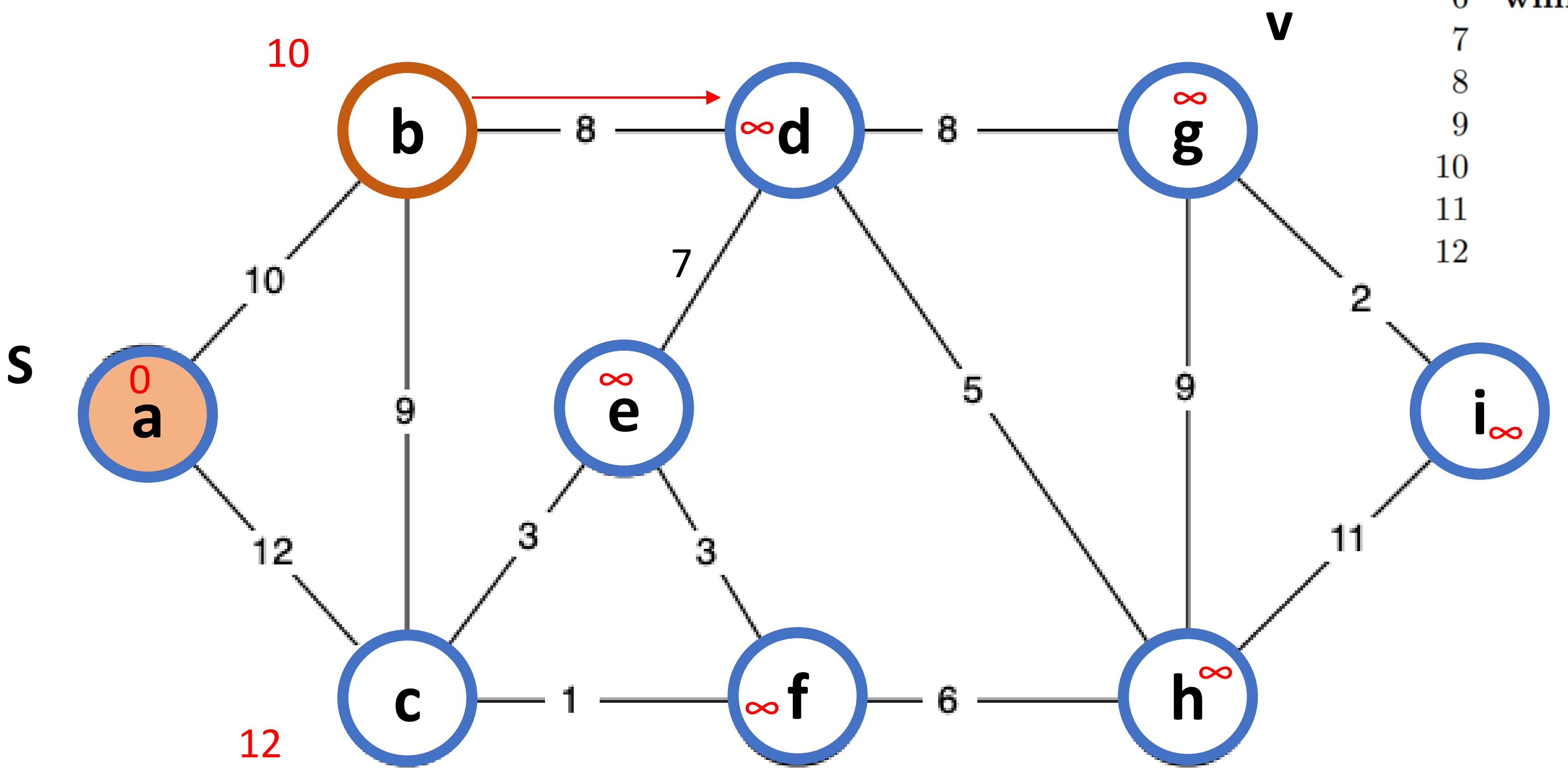
- Definition: δ (s,v) = length of the shortest path from s to v in G



$\textsc{Dijkstra}(G = (V, E), s)$

```
for all v ∈ V
        do d_u ← ∞
           π_u ← NIL
d_s ← 0
Q ← MAKEQUEUE(V)      ▷ use d_u as key
while Q ≠ ∅
        do u ← EXTRACTMIN(Q)
           for each v ∈ Adj(u)
                do if d_v > d_u + w(u, v)
                     then d_v ← d_u + w(u, v)
                          π_v ← u
                          DECREASEKEY(Q, v)
```

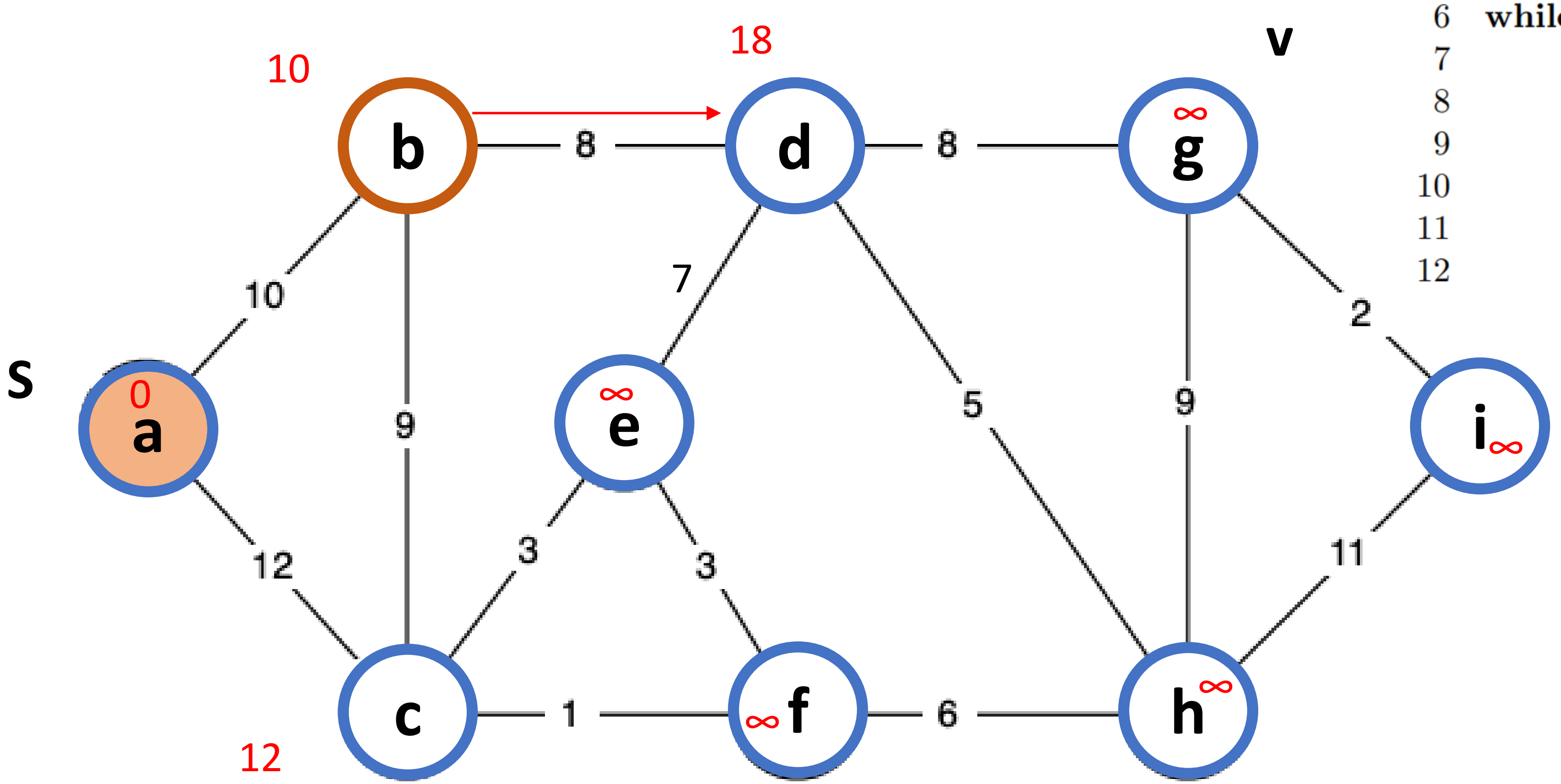
- Definition: δ (s,v) = length of the shortest path from s to v in G



$\text{DIJKSTRA}(G = (V, E), s)$

$$
\begin{array}{l}
\textbf{for } \text{all } v \in V \\
\quad \textbf{do } d_u \leftarrow \infty \\
\quad\quad \pi_u \leftarrow \text{NIL} \\
d_s \leftarrow 0 \\
Q \leftarrow \text{MAKEQUEUE}(V) \quad \triangleright \text{ use } d_u \text{ as key} \\
\textbf{while } Q \neq \emptyset \\
\quad \textbf{do } u \leftarrow \text{EXTRACTMIN}(Q) \\
\quad\quad \textbf{for } \text{each } v \in Adj(u) \\
\quad\quad\quad \textbf{do if } d_v > d_u + w(u, v) \\
\quad\quad\quad\quad \textbf{then } d_v \leftarrow d_u + w(u, v) \\
\quad\quad\quad\quad\quad \pi_v \leftarrow u \\
\quad\quad\quad\quad\quad \text{DECREASEKEY}(Q, v)
\end{array}
$$

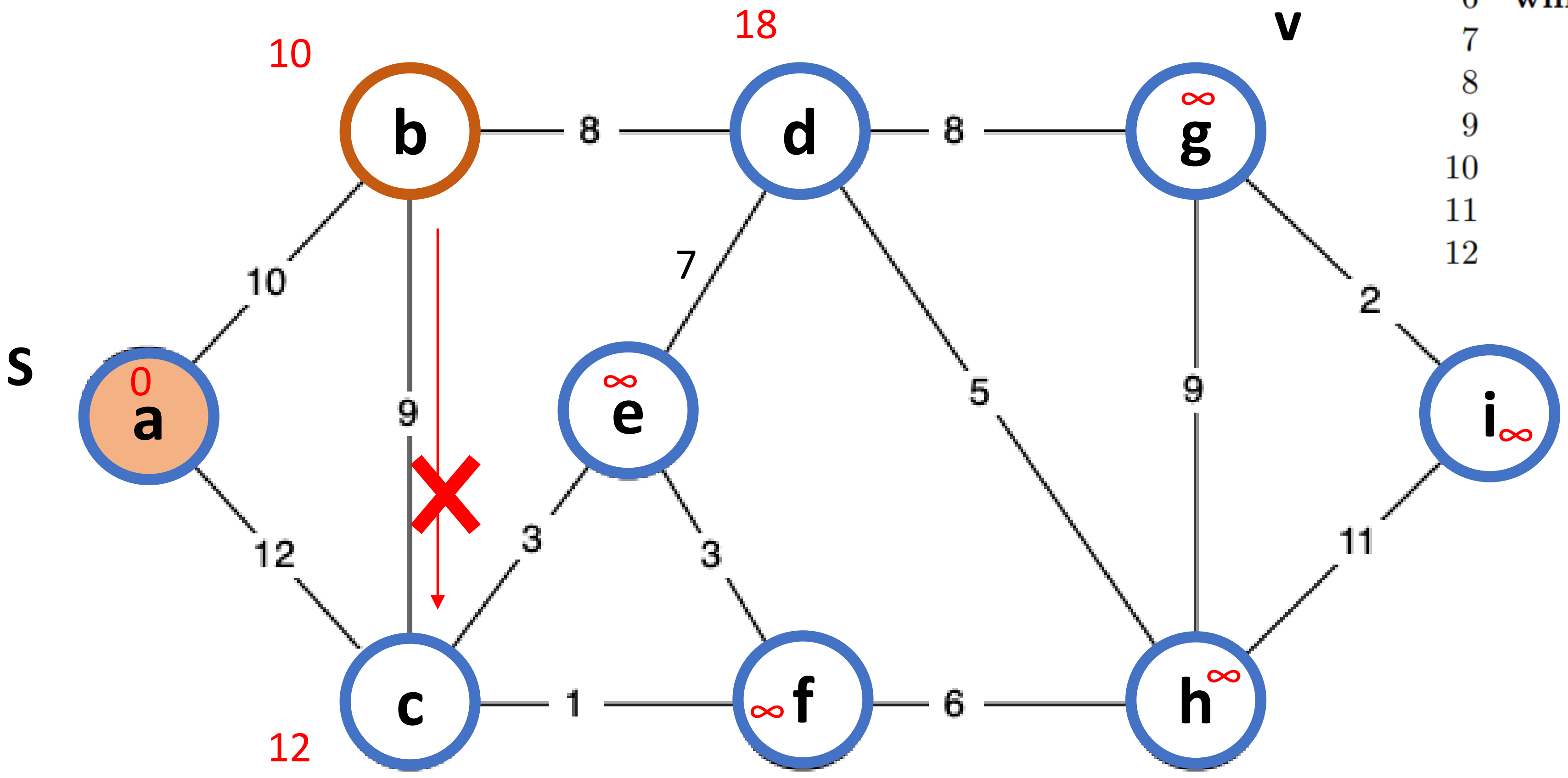
- Definition: δ (s,v) = length of the shortest path from s to v in G



DIJKSTRA$(G = (V, E), s)$

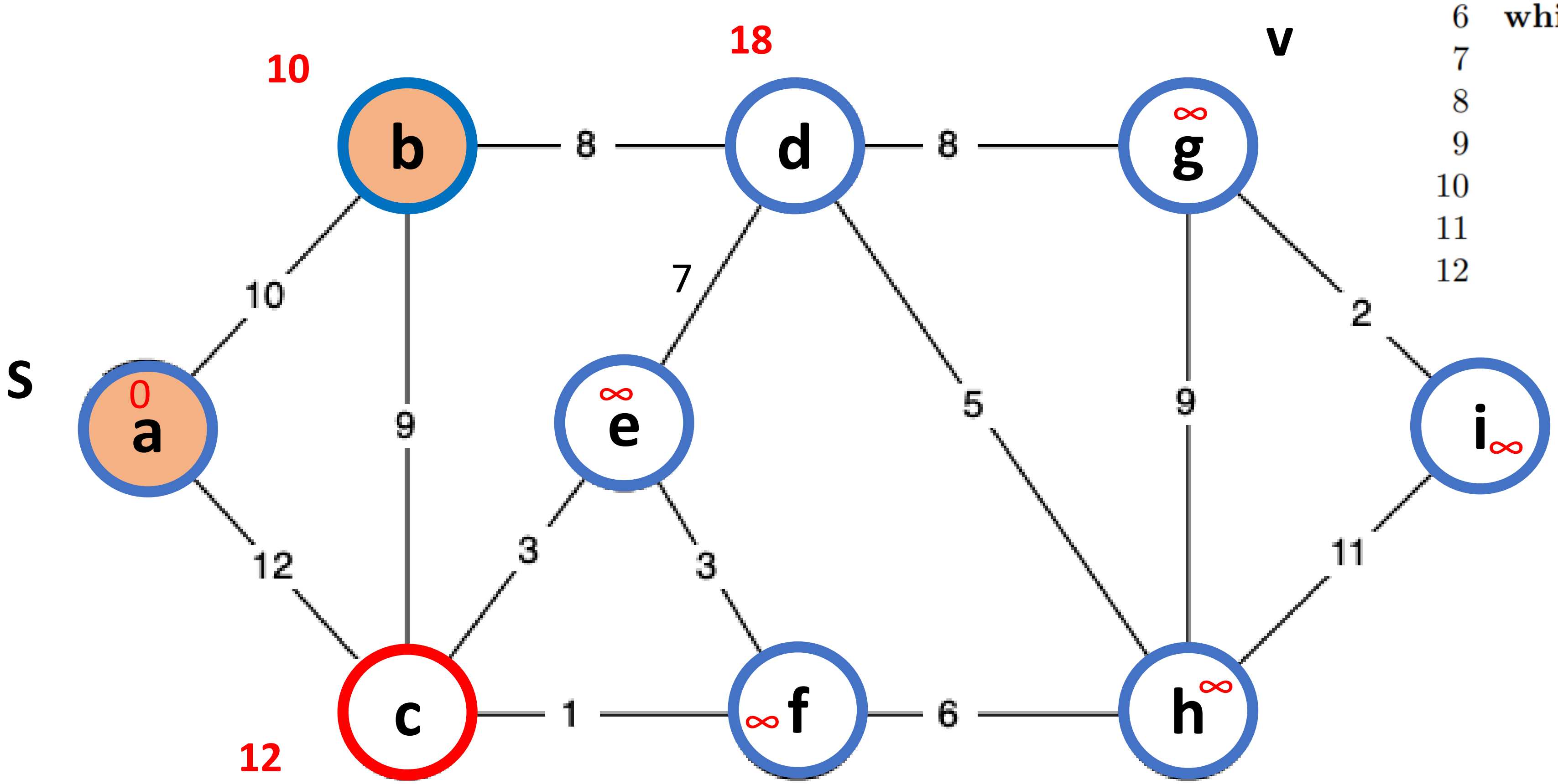
for all $v \in V$
  do $d_u \leftarrow \infty$
    $\pi_u \leftarrow$ NIL
$d_s \leftarrow 0$
$Q \leftarrow$ MAKEQUEUE(V) $\triangleright$ use d_u as key
while $Q \neq \emptyset$
  do $u \leftarrow$ EXTRACTMIN(Q)
    for each $v \in Adj(u)$
      do if $d_v > d_u + w(u, v)$
        then $d_v \leftarrow d_u + w(u, v)$
          $\pi_v \leftarrow u$
          DECREASEKEY(Q, v)

- Definition: δ (s,v) = length of the shortest path from s to v in G



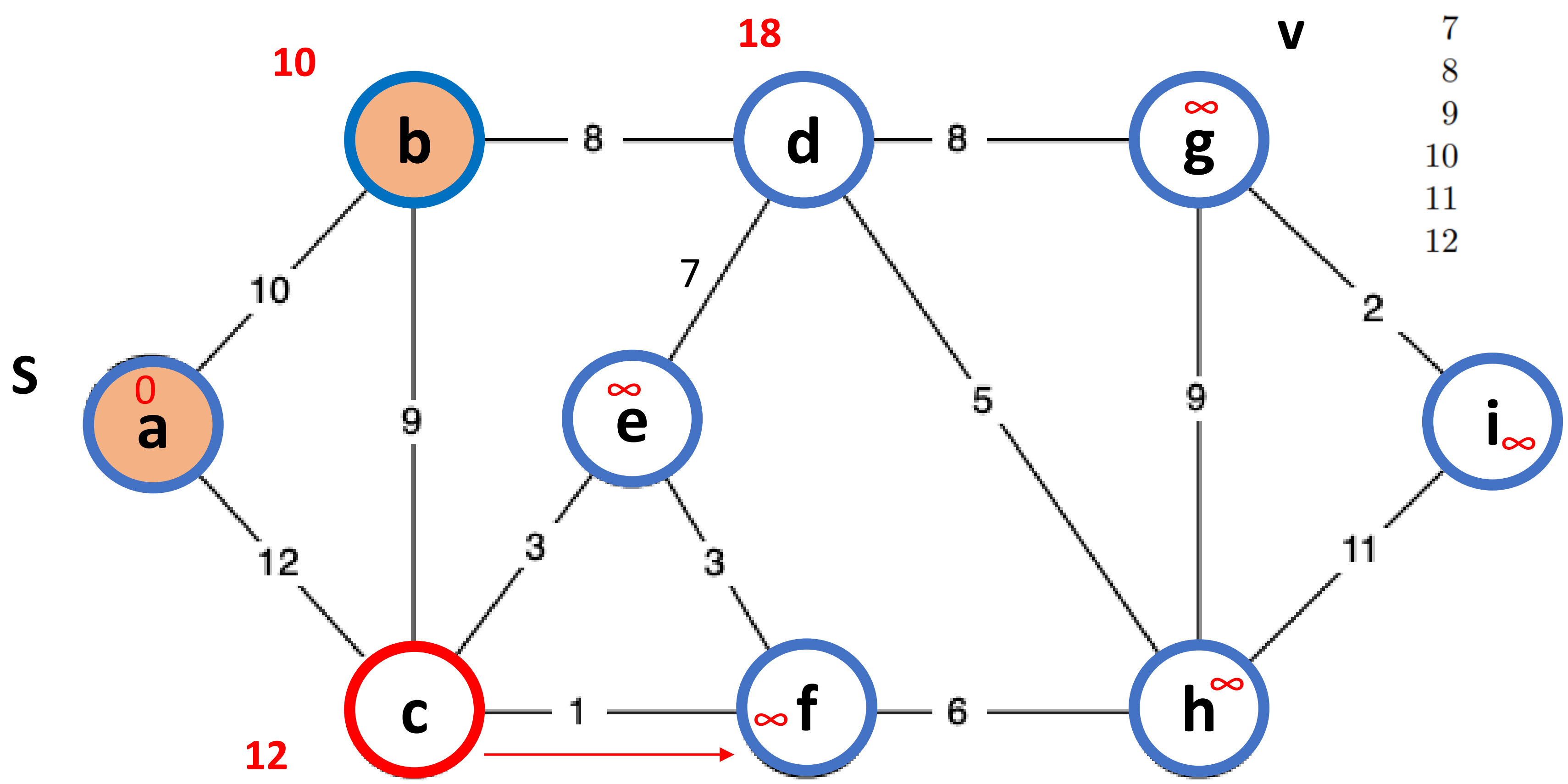

```
DIJKSTRA(G = (V, E), s)
for all v ∈ V
        do d_u ← ∞
           π_u ← NIL
d_s ← 0
Q ← MAKEQUEUE(V)    ▷ use d_u as key
while Q ≠ ∅
        do u ← EXTRACTMIN(Q)
           for each v ∈ Adj(u)
                do if d_v > d_u + w(u, v)
                      then d_v ← d_u + w(u, v)
                           π_v ← u
                           DECREASEKEY(Q, v)
```

- Definition: δ (s,v) = length of the shortest path from s to v in G




```
DIJKSTRA(G = (V, E), s)
for all v ∈ V
        do d_u ← ∞
            π_u ← NIL
d_s ← 0
Q ← MAKEQUEUE(V)      ▷ use d_u as key
while Q ≠ ∅
        do u ← EXTRACTMIN(Q)
            for each v ∈ Adj(u)
                do if d_v > d_u + w(u, v)
                    then d_v ← d_u + w(u, v)
                         π_v ← u
                         DECREASEKEY(Q, v)
```

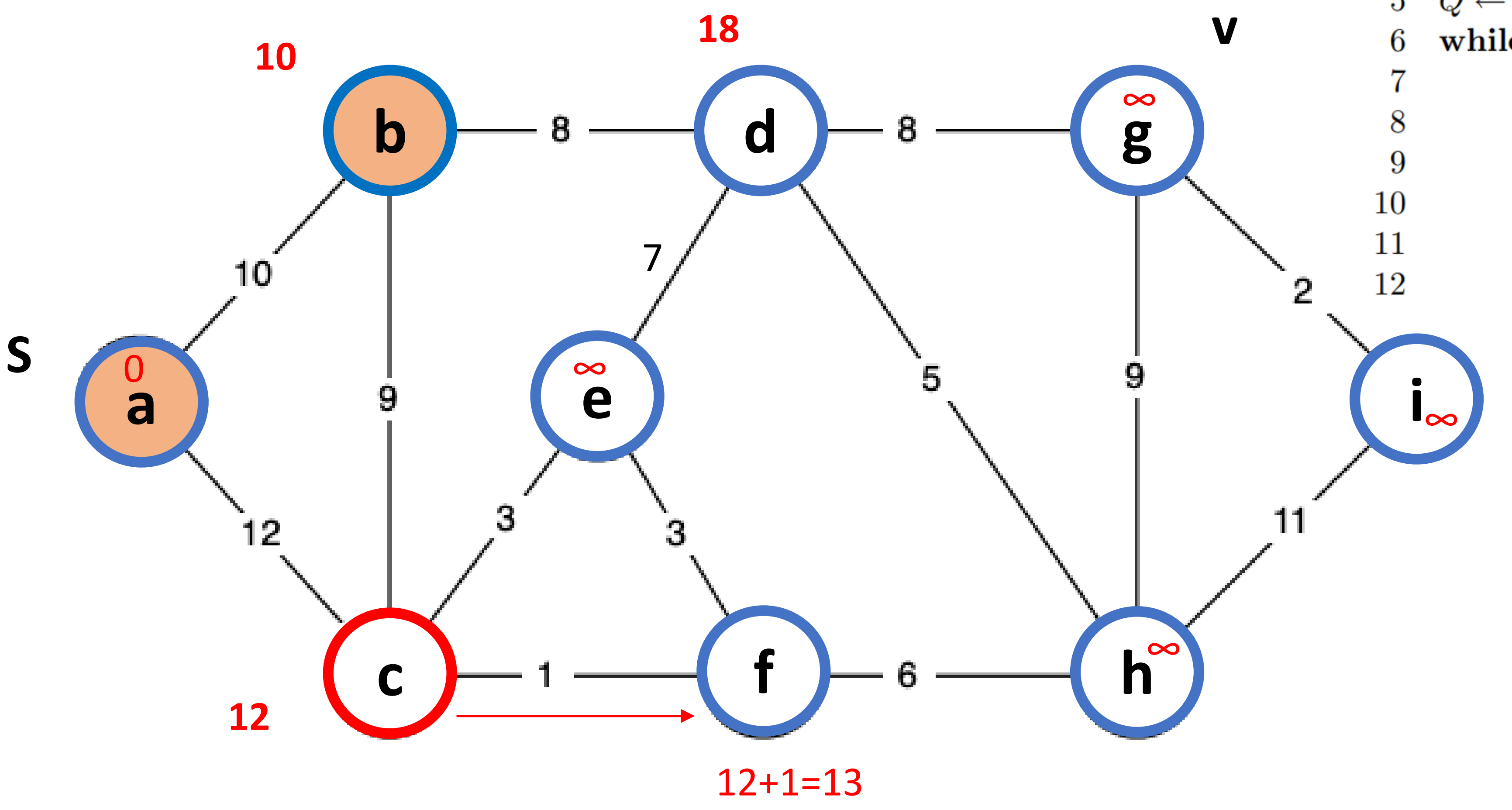
- Definition: δ (s,v) = length of the shortest path from s to v in G



DIJKSTRA$(G = (V, E), s)$

```
for all v ∈ V
        do d_u ← ∞
           π_u ← NIL
d_s ← 0
Q ← MAKEQUEUE(V)      ▷ use d_u as key
while Q ≠ ∅
        do u ← EXTRACTMIN(Q)
           for each v ∈ Adj(u)
                do if d_v > d_u + w(u, v)
                      then d_v ← d_u + w(u, v)
                           π_v ← u
                           DECREASEKEY(Q, v)
```

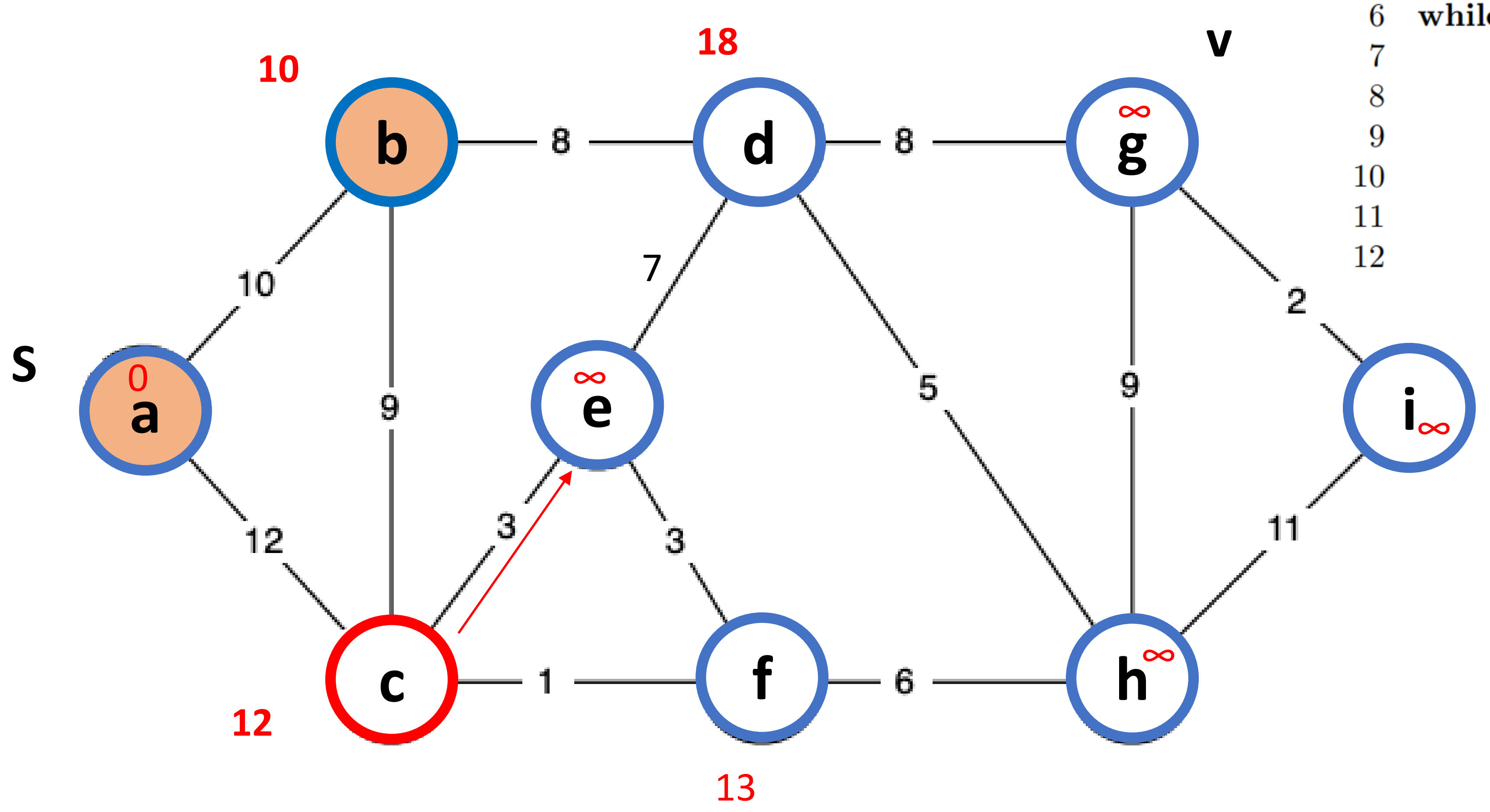
- Definition: δ (s,v) = length of the shortest path from s to v in G



$\text{DIJKSTRA}(G = (V, E), s)$

for all $v \in V$
  do $d_u \leftarrow \infty$
    $\pi_u \leftarrow \text{NIL}$
$d_s \leftarrow 0$
$Q \leftarrow \text{MAKEQUEUE}(V)$ $\quad \triangleright$ use d_u as key
while $Q \neq \emptyset$
  do $u \leftarrow \text{EXTRACTMIN}(Q)$
    for each $v \in Adj(u)$
      do if $d_v > d_u + w(u, v)$
        then $d_v \leftarrow d_u + w(u, v)$
          $\pi_v \leftarrow u$
          $\text{DECREASEKEY}(Q, v)$

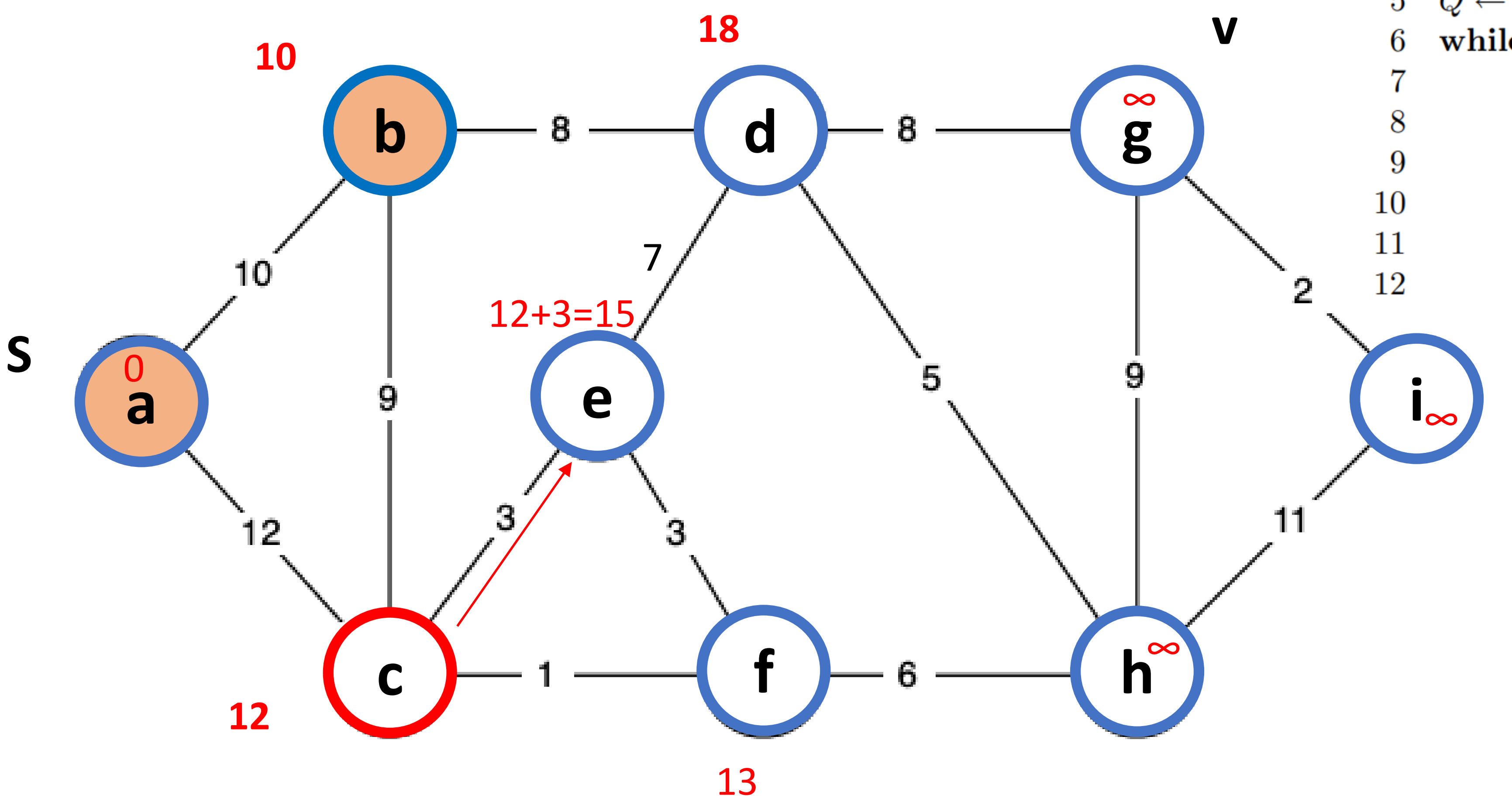
- Definition: δ (s,v) = length of the shortest path from s to v in G



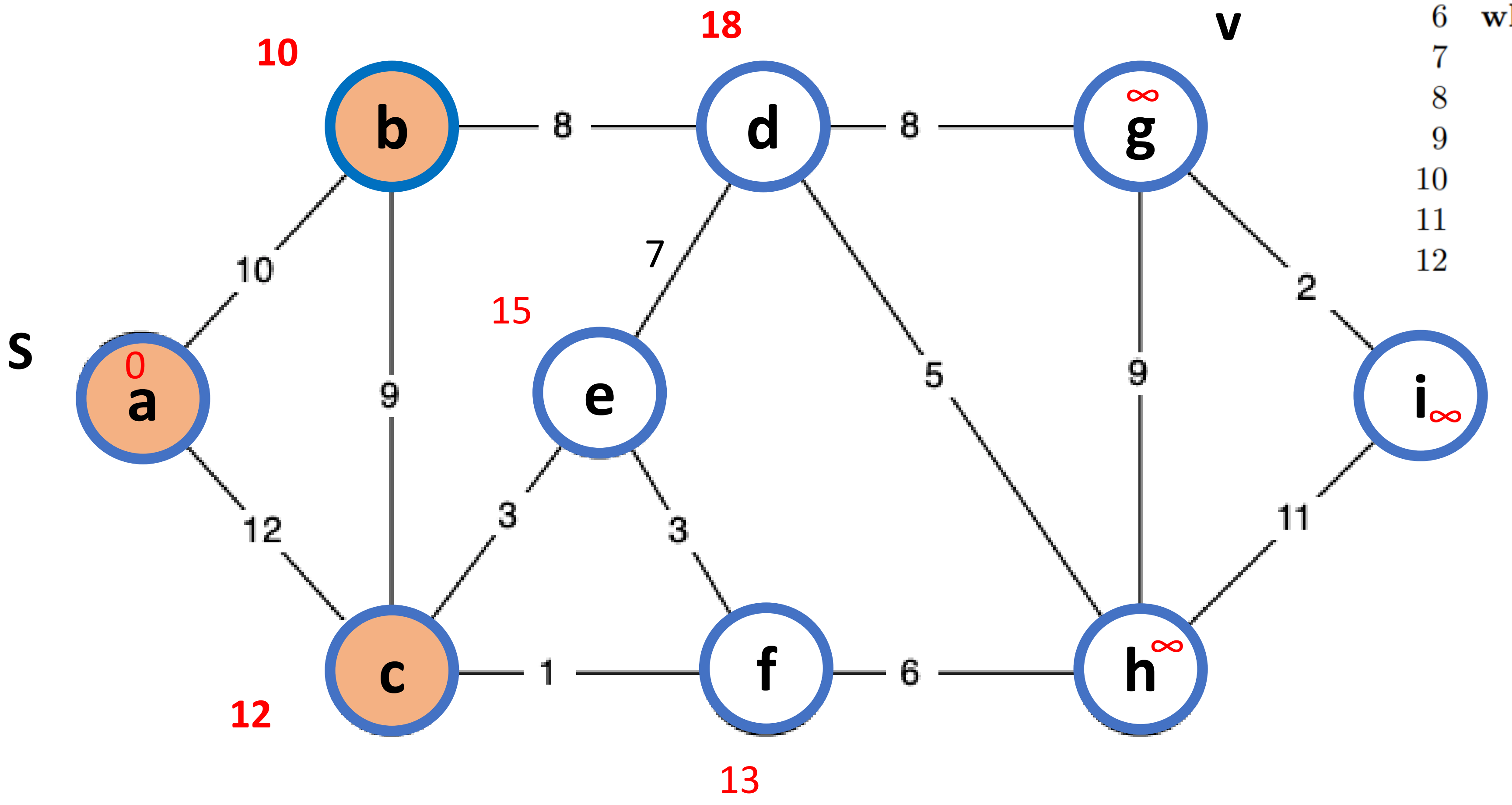

```
DIJKSTRA(G = (V, E), s)
for all v ∈ V
        do d_u ← ∞
           π_u ← NIL
d_s ← 0
Q ← MAKEQUEUE(V)    ▷ use d_u as key
while Q ≠ ∅
        do u ← EXTRACTMIN(Q)
           for each v ∈ Adj(u)
               do if d_v > d_u + w(u, v)
                     then d_v ← d_u + w(u, v)
                          π_v ← u
                          DECREASEKEY(Q, v)
```

- Definition: δ (s,v) = length of the shortest path from s to v in G

```
DIJKSTRA(G = (V, E), s)
for all v ∈ V
        do d_u ← ∞
           π_u ← NIL
d_s ← 0
Q ← MAKEQUEUE(V)      ▷ use d_u as key
while Q ≠ ∅
        do u ← EXTRACTMIN(Q)
           for each v ∈ Adj(u)
               do if d_v > d_u + w(u, v)
                     then d_v ← d_u + w(u, v)
                          π_v ← u
                          DECREASEKEY(Q, v)
```



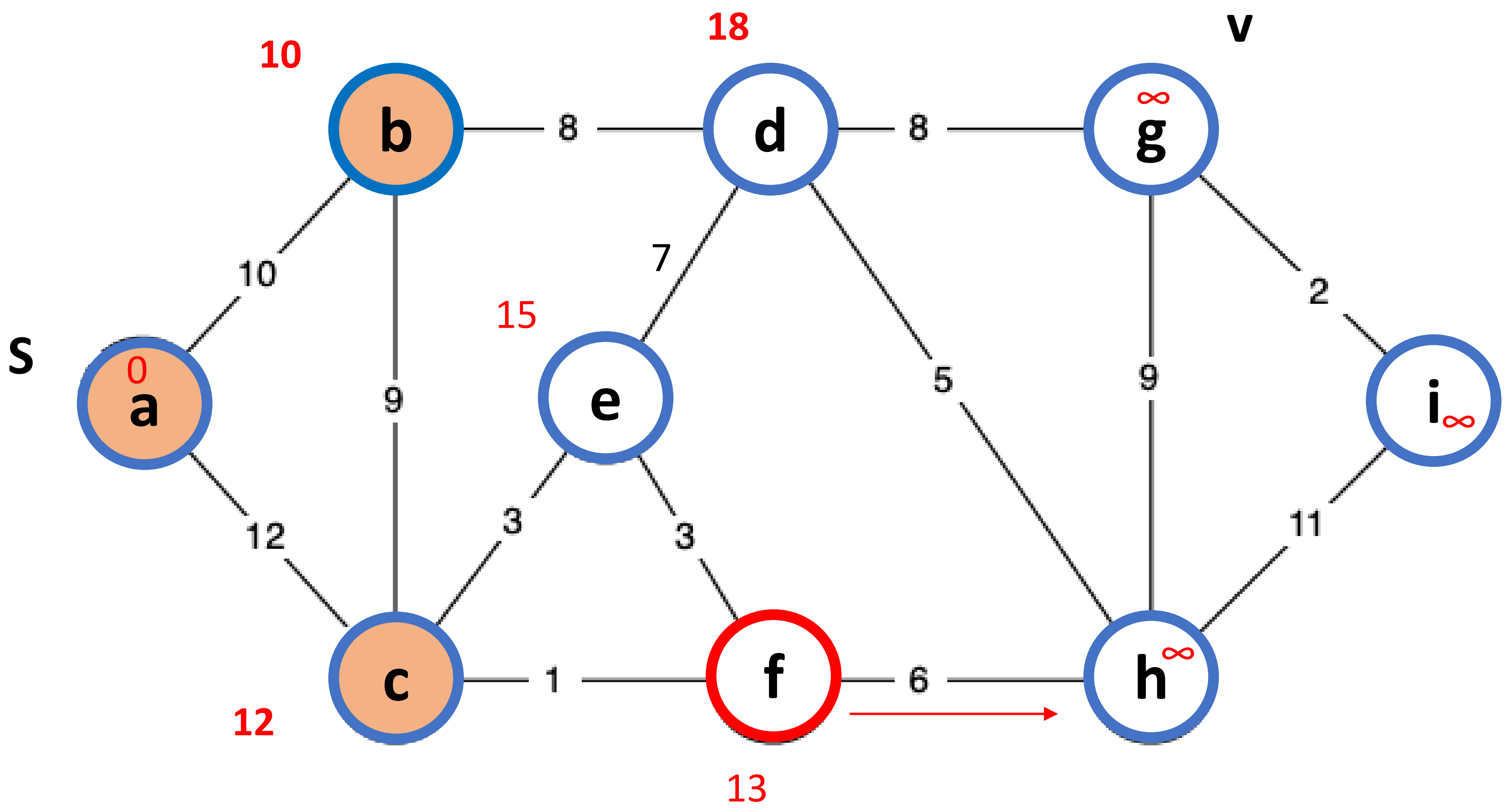
- Definition: δ (s,v) = length of the shortest path from s to v in G



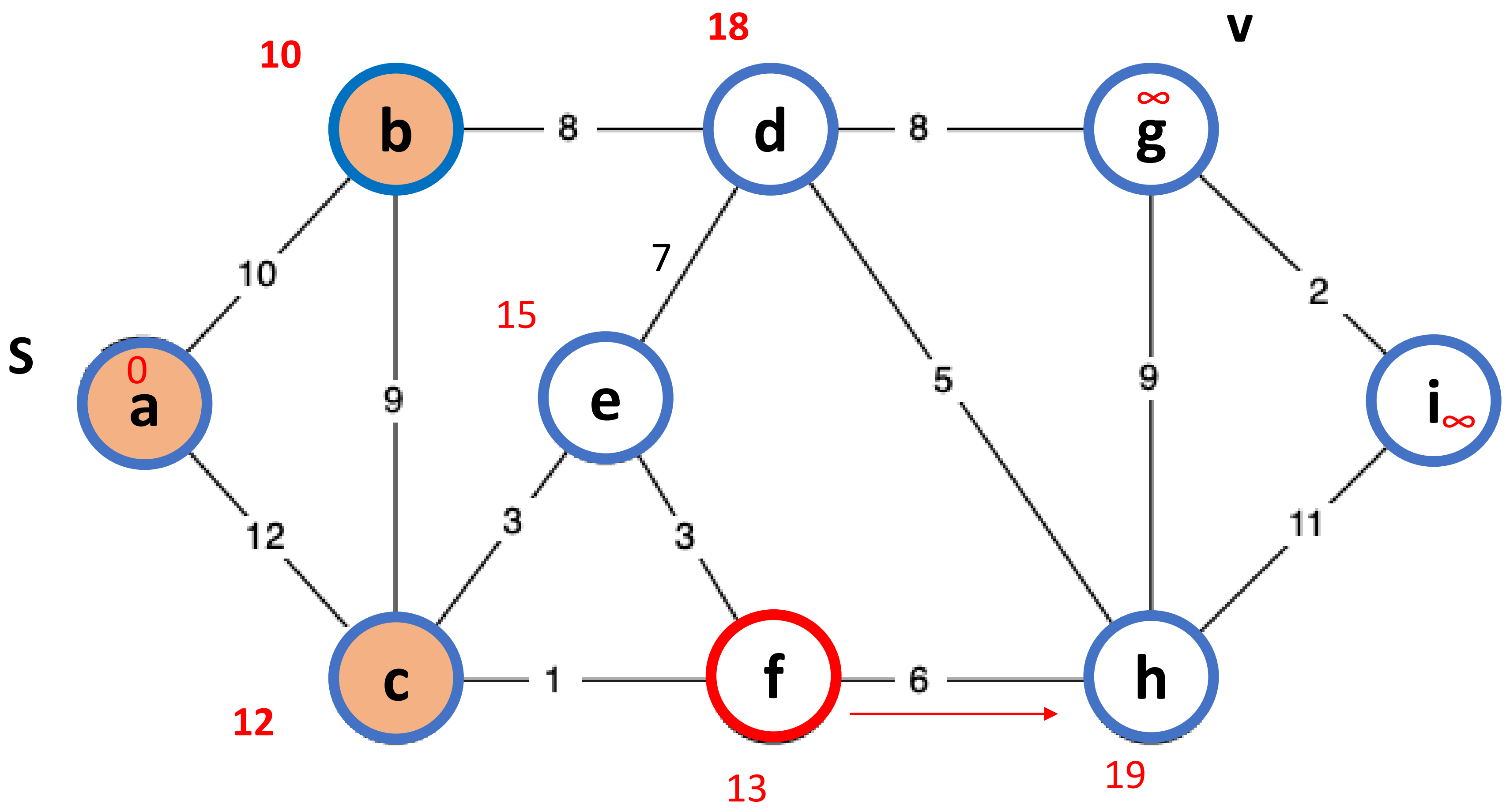
$\text{DIJKSTRA}(G = (V, E), s)$

```
for all v ∈ V
        do d_u ← ∞
           π_u ← NIL
d_s ← 0
Q ← MAKEQUEUE(V)      ▷ use d_u as key
while Q ≠ ∅
        do u ← EXTRACTMIN(Q)
           for each v ∈ Adj(u)
                do if d_v > d_u + w(u, v)
                      then d_v ← d_u + w(u, v)
                           π_v ← u
                           DECREASEKEY(Q, v)
```

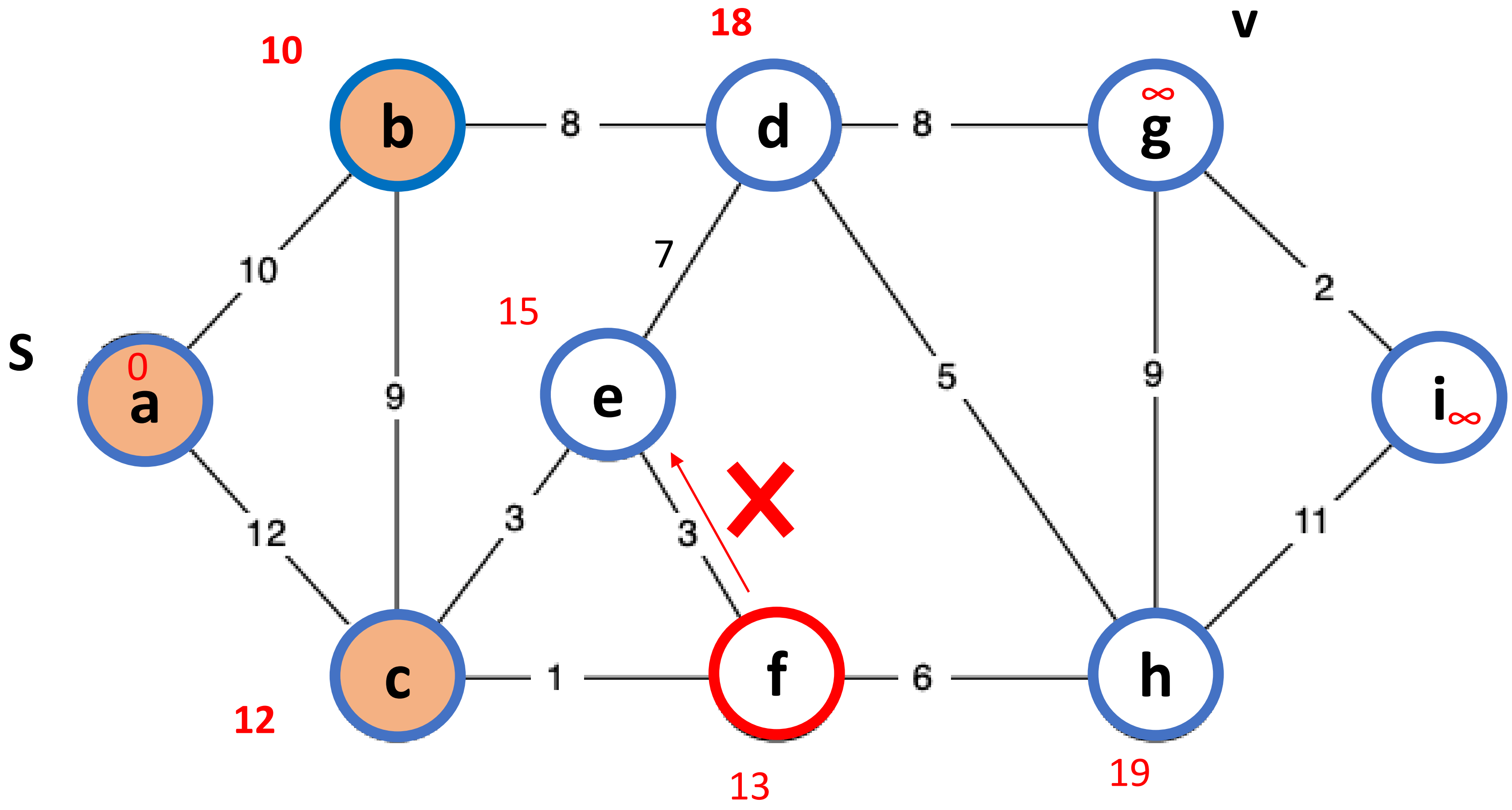
- Definition: δ (s,v) = length of the shortest path from s to v in G



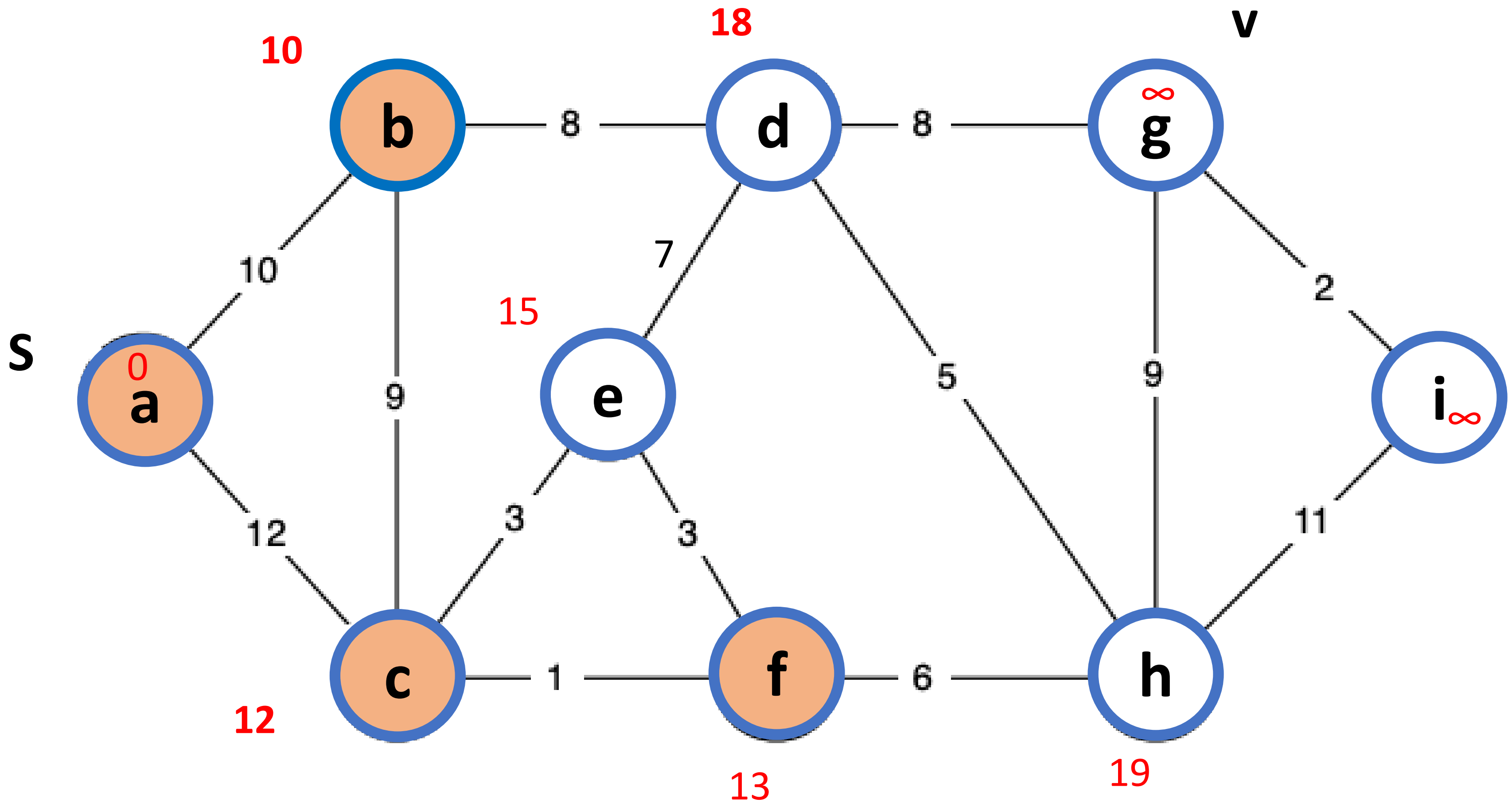
- Definition: $\delta(s,v)$ = length of the shortest path from s to v in G



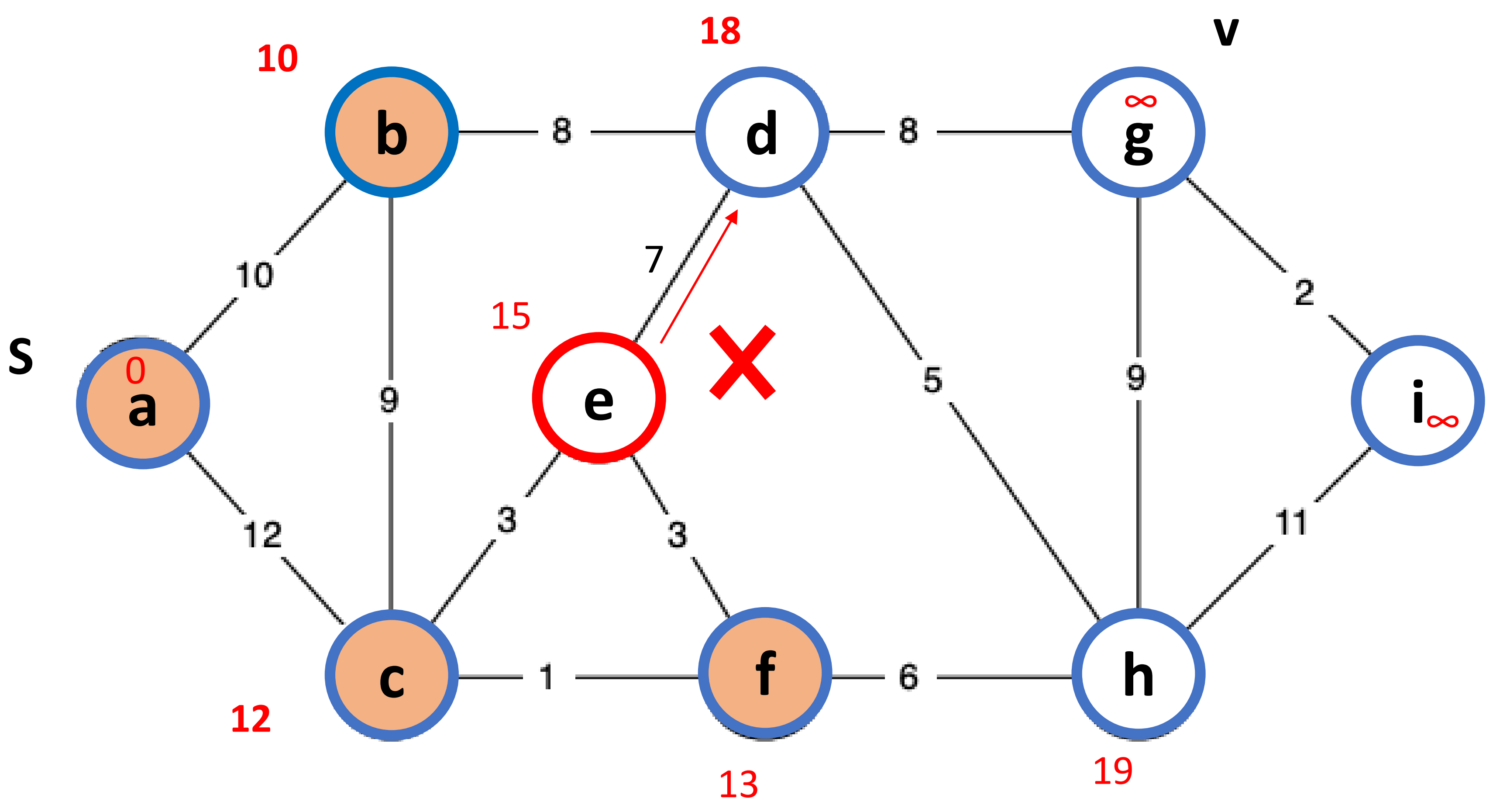
- Definition: δ (s,v) = length of the shortest path from s to v in G



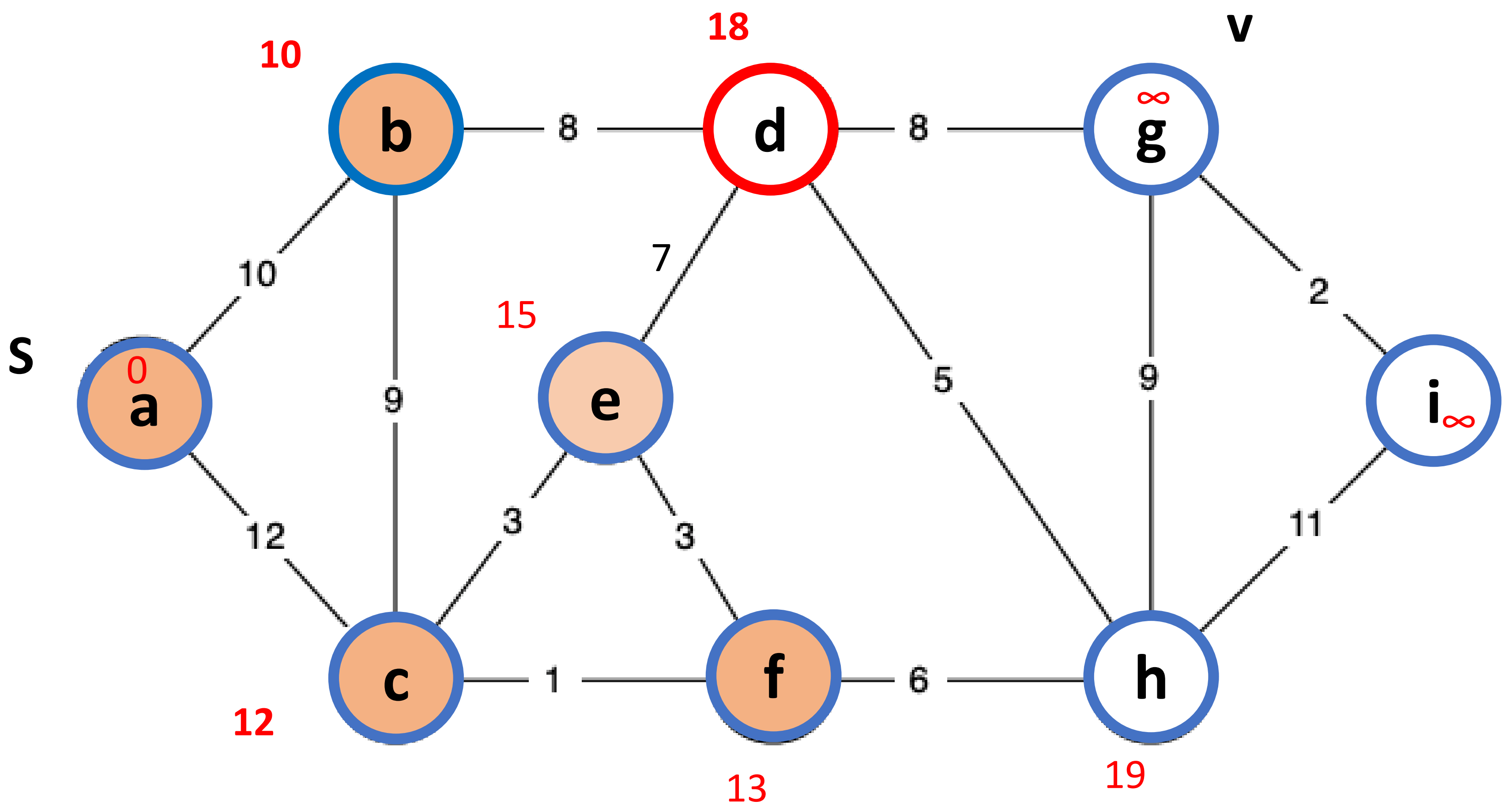
- Definition: δ (s,v) = length of the shortest path from s to v in G



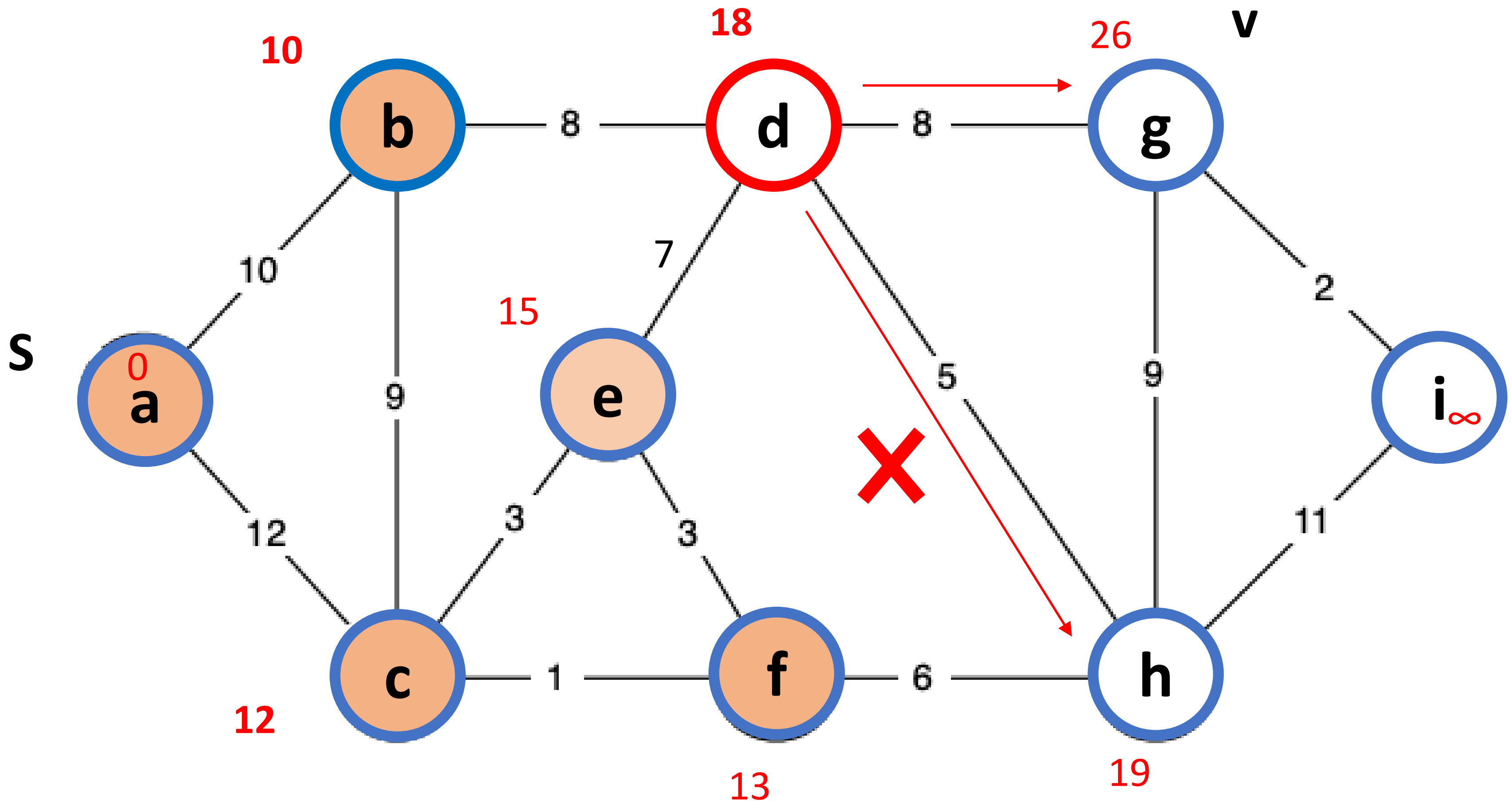
- Definition: δ (s,v) = length of the shortest path from s to v in G



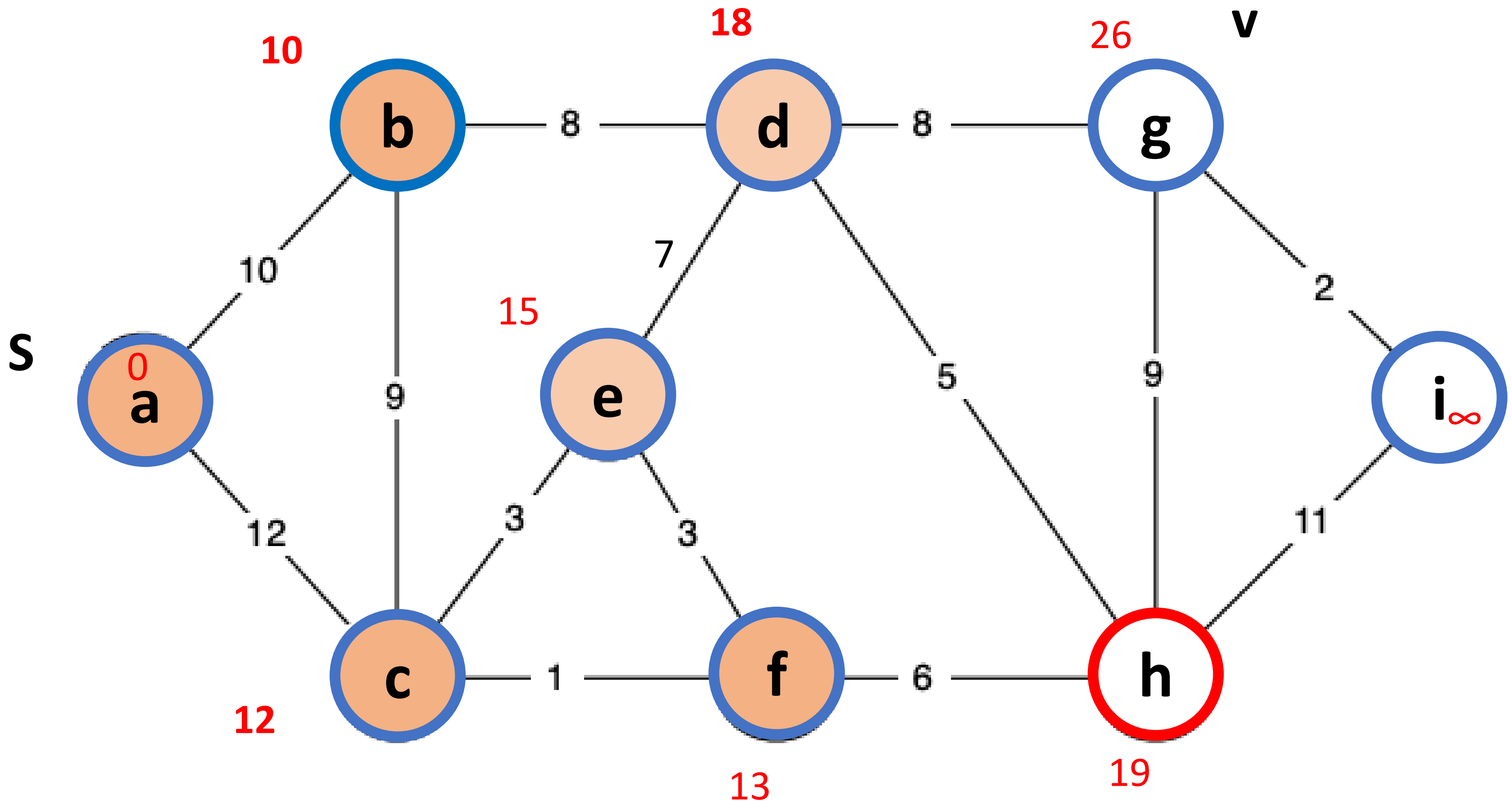
- Definition: δ (s,v) = length of the shortest path from s to v in G



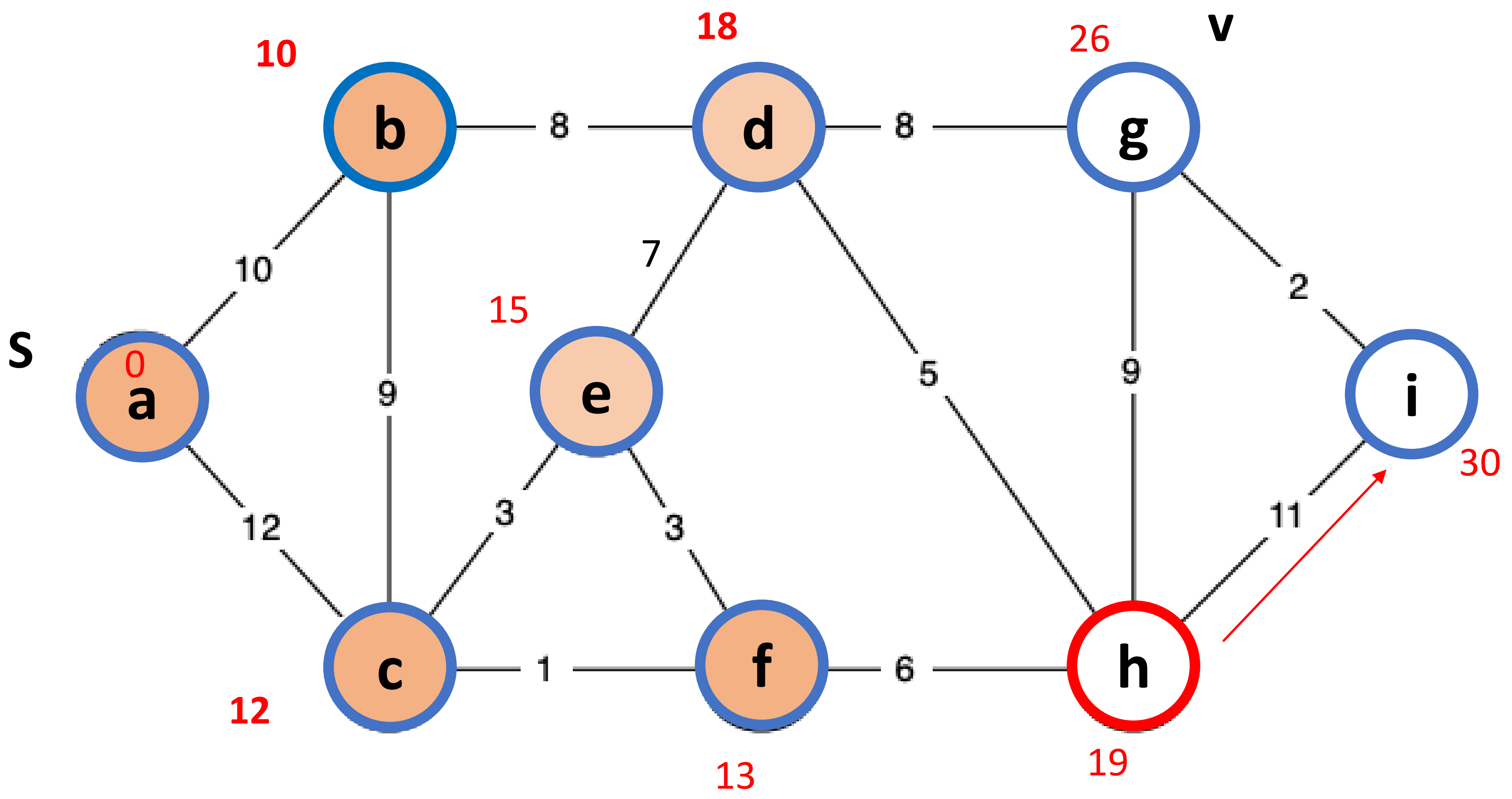
- Definition: δ (s,v) = length of the shortest path from s to v in G



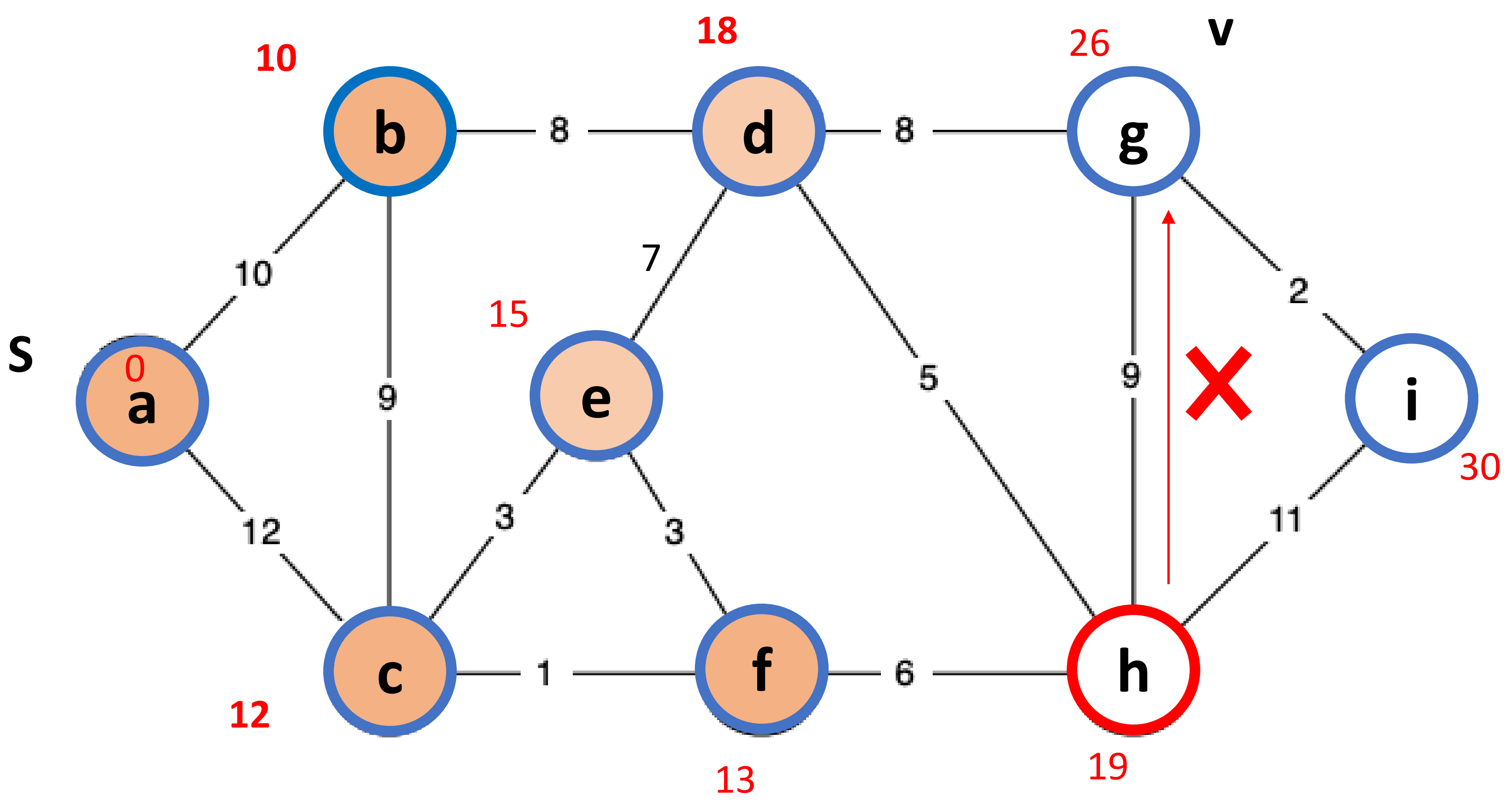
- Definition: δ (s,v) = length of the shortest path from s to v in G



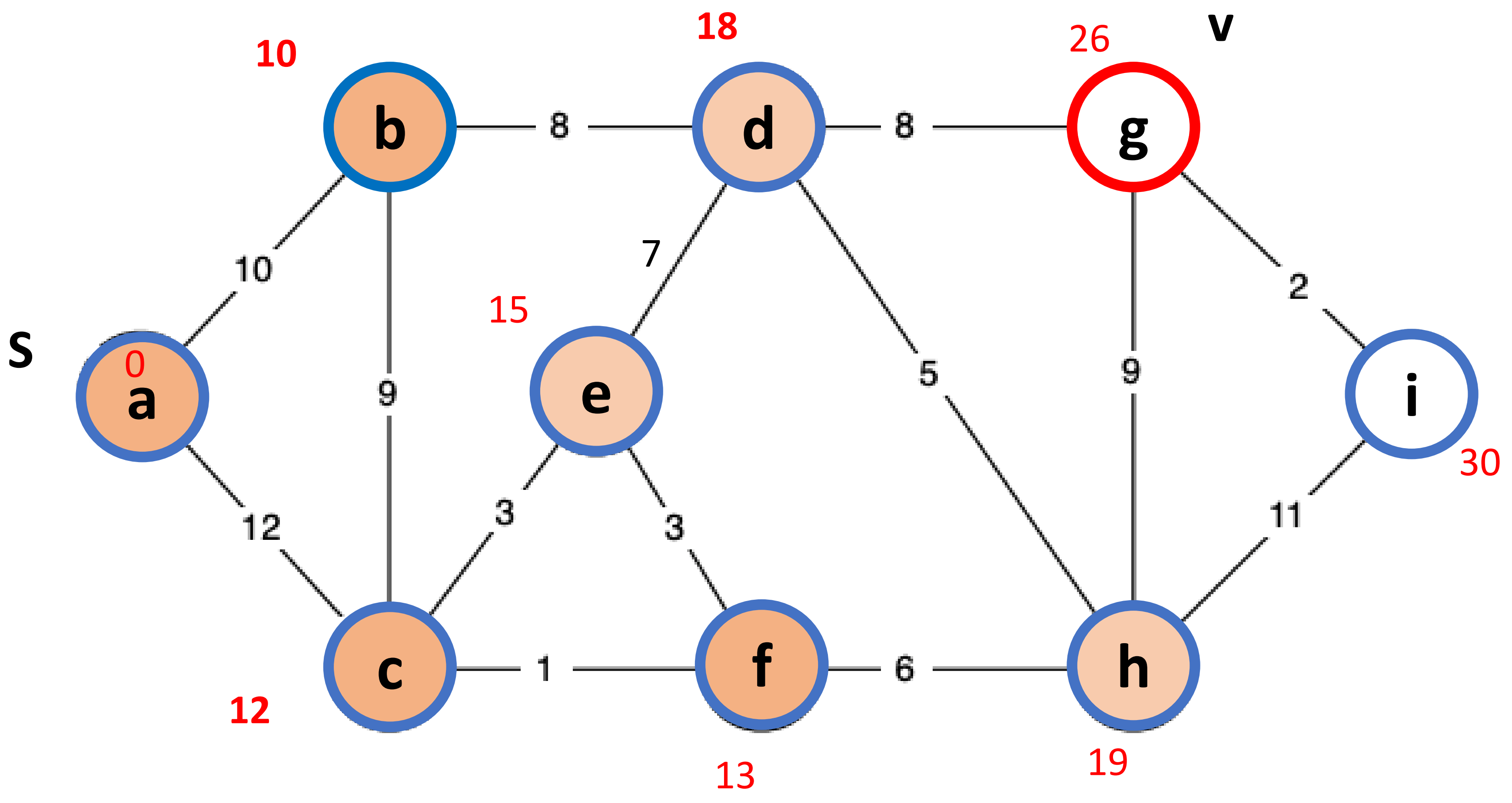
- Definition: δ (s,v) = length of the shortest path from s to v in G



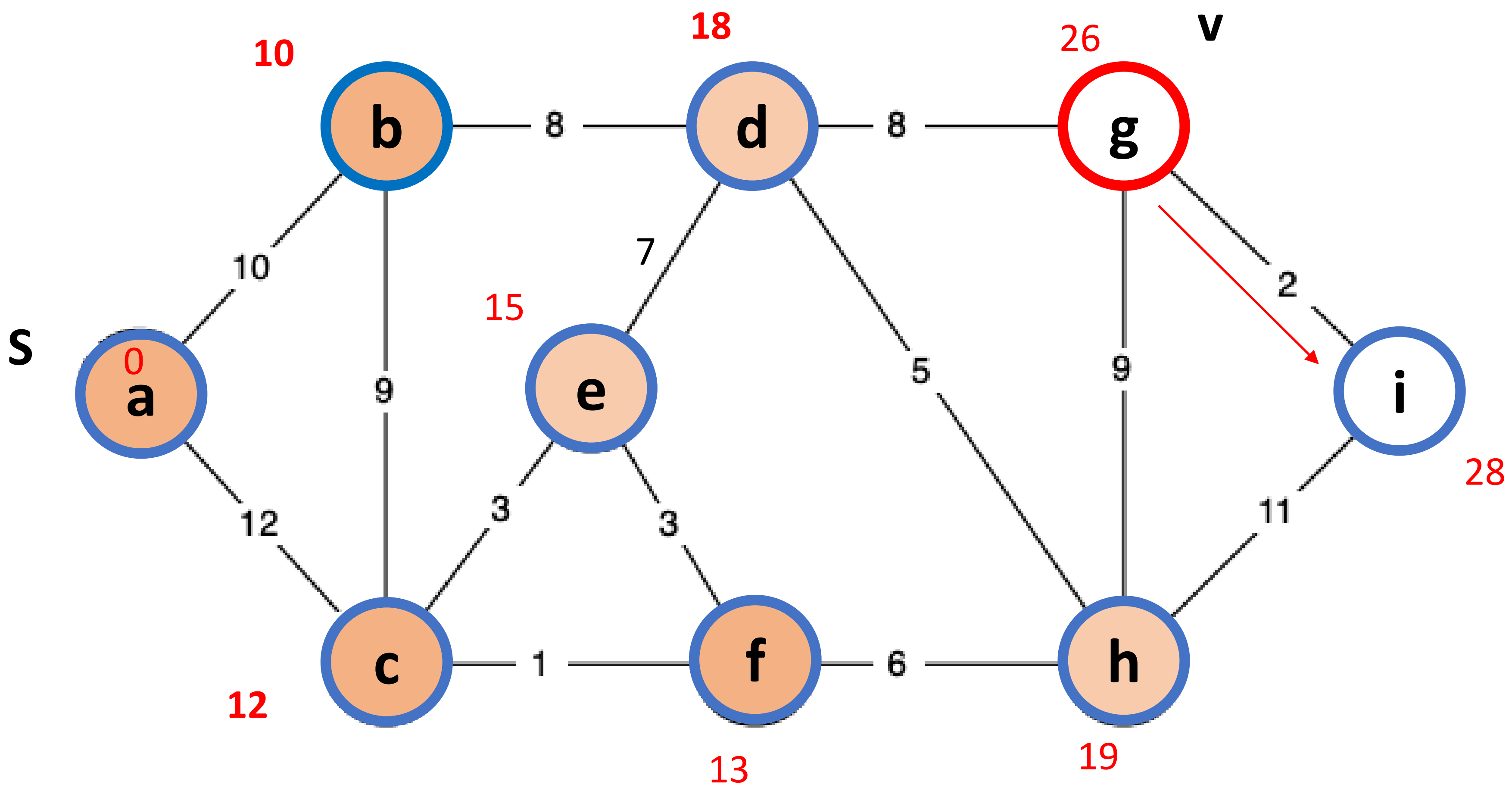
- Definition: δ (s,v) = length of the shortest path from s to v in G



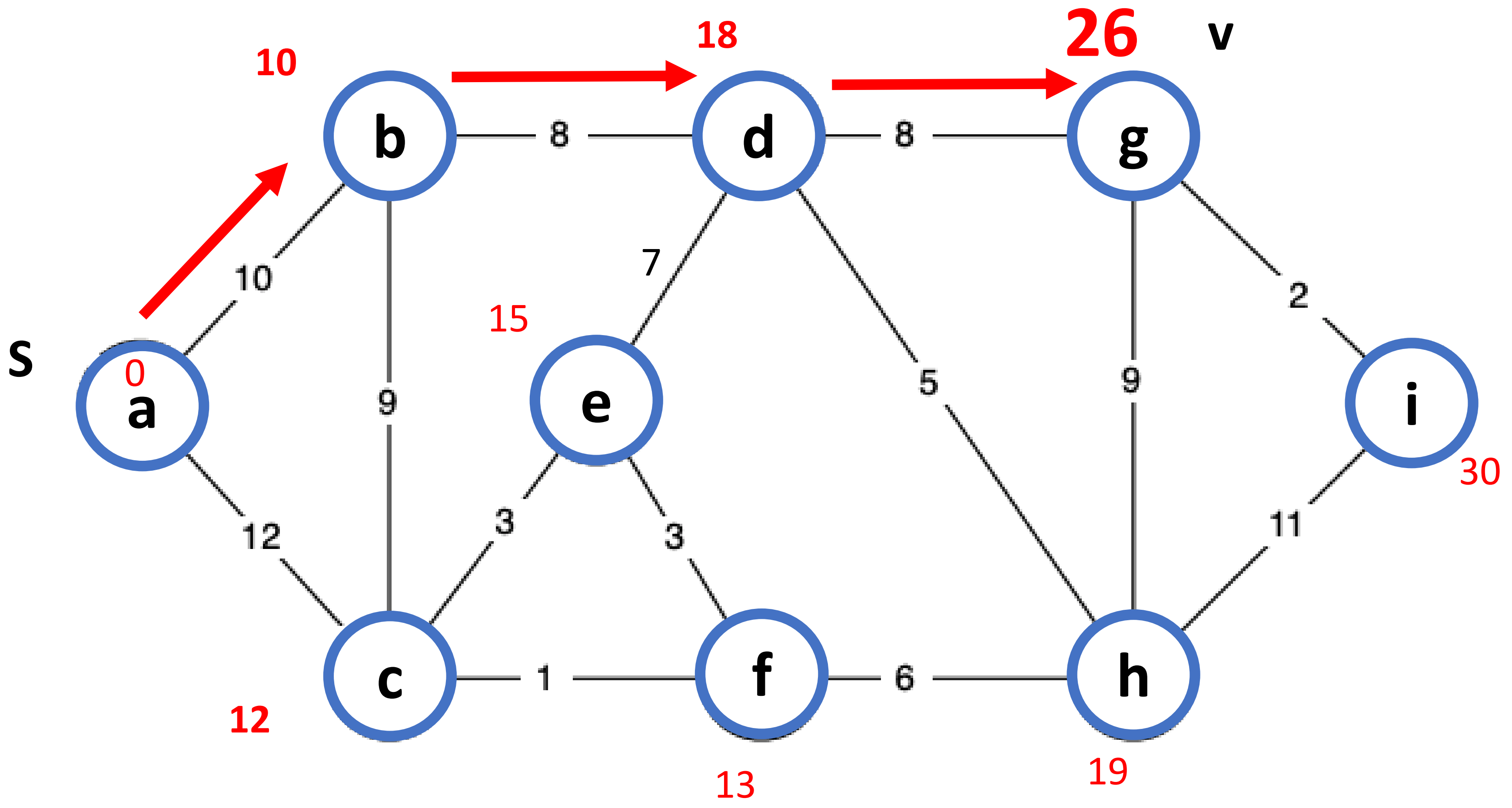
- Definition: δ (s,v) = length of the shortest path from s to v in G



- Definition: δ (s,v) = length of the shortest path from s to v in G



- Definition: $\delta(s,v)$ = length of the shortest path from s to v in G



$\textsc{Dijkstra}(G = (V, E), s)$

for all $v \in V$
    do $d_u \leftarrow \infty$
        $\pi_u \leftarrow \text{NIL}$
$d_s \leftarrow 0$
$Q \leftarrow \textsc{MakeQueue}(V)$ $\triangleright$ use d_u as key
while $Q \neq \emptyset$
    do $u \leftarrow \textsc{ExtractMin}(Q)$
        for each $v \in Adj(u)$
            do if $d_v > d_u + w(u, v)$
                then $d_v \leftarrow d_u + w(u, v)$
                    $\pi_v \leftarrow u$
                    $\textsc{DecreaseKey}(Q, v)$

DIJKSTRA$(G = (V, E), s)$

for all $v \in V$
  do $d_u \leftarrow \infty$
    $\pi_u \leftarrow$ NIL
$d_s \leftarrow 0$
$Q \leftarrow$ MAKEQUEUE(V) $\quad \triangleright$ use d_u as key
while $Q \neq \emptyset$
  do $u \leftarrow$ EXTRACTMIN(Q)
    for each $v \in Adj(u)$
      do if $d_v > d_u + w(u, v)$
        then $d_v \leftarrow d_u + w(u, v)$
          $\pi_v \leftarrow u$
          DECREASEKEY(Q, v)

PRIM$(G = (V, E))$

$Q \leftarrow \emptyset \quad \triangleright$ Q is a Priority Queue
Initialize each $v \in V$ with key $k_v \leftarrow \infty$, $\pi_v \leftarrow$ NIL
Pick a starting node r and set $k_r \leftarrow 0$
Insert all nodes into Q with key k_v.
while $Q \neq \emptyset$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    for each $v \in Adj(u)$
      do if $v \in Q$ and $w(u, v) < k_v$
        then $\pi_v \leftarrow u$
          DECREASE-KEY$(Q, v, w(u, v))$ $\quad \triangleright$ Sets $k_v \leftarrow w(u, v)$

$\text{DIJKSTRA}(G = (V, E), s)$

for all $v \in V$
  do $d_u \leftarrow \infty$
    $\pi_u \leftarrow \text{NIL}$
$d_s \leftarrow 0$
$Q \leftarrow \text{MAKEQUEUE}(V)$ $\triangleright$ use d_u as key
while $Q \neq \emptyset$
  do $u \leftarrow \text{EXTRACTMIN}(Q)$ → **(V-1) times**
    for each $v \in Adj(u)$
      do if $d_v > d_u + w(u, v)$
        then $d_v \leftarrow d_u + w(u, v)$
          $\pi_v \leftarrow u$
          $\text{DECREASEKEY}(Q, v)$

E times

0(ElogV + VlogV)

$\textsc{Dijkstra}(G = (V, E), s)$

for all $v \in V$
    do $d_u \leftarrow \infty$
        $\pi_u \leftarrow \text{NIL}$
$d_s \leftarrow 0$
$Q \leftarrow \text{MAKEQUEUE}(V)$ $\quad \triangleright$ use d_u as key
while $Q \neq \emptyset$
    do $u \leftarrow \text{EXTRACTMIN}(Q)$
        for each $v \in Adj(u)$
            do if $d_v > d_u + w(u, v)$
                then $d_v \leftarrow d_u + w(u, v)$
                    $\pi_v \leftarrow u$
                    $\text{DECREASEKEY}(Q, v)$

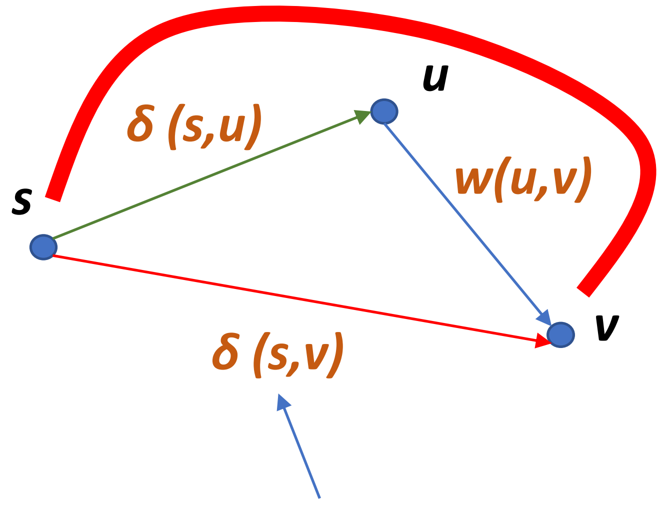
$\text{PRIM}(G = (V, E))$

$Q \leftarrow \emptyset$ $\quad \triangleright$ Q is a Priority Queue
Initialize each $v \in V$ with key $k_v \leftarrow \infty$, $\pi_v \leftarrow \text{NIL}$
Pick a starting node r and set $k_r \leftarrow 0$
Insert all nodes into Q with key k_v.
while $Q \neq \emptyset$
    do $u \leftarrow \text{EXTRACT-MIN}(Q)$
        for each $v \in Adj(u)$
            do if $v \in Q$ and $w(u, v) < k_v$
                then $\pi_v \leftarrow u$
                    $\text{DECREASE-KEY}(Q, v, w(u, v))$ $\quad \triangleright$ Sets $k_v \leftarrow w(u, v)$

Why does Dijkstra work?

Triangle inequality: $\forall (u,v) \in E,\ \delta(s,v) <= \delta(s,u)+w(u,v)$

What if this is the shortest path from s-> v?

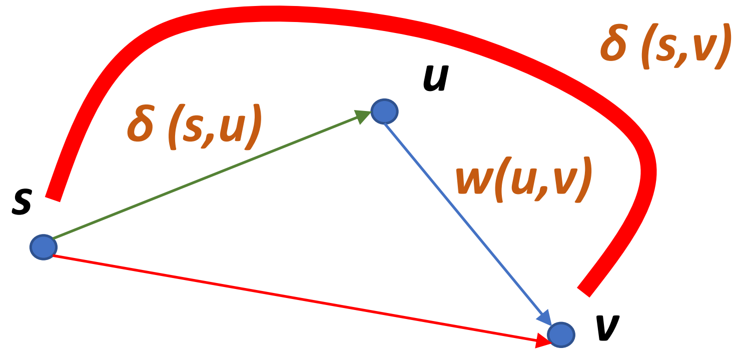


What if this is not the shortest path?

Then! $\delta(s,v) = \delta(s,u)+w(u,v)$

What if this is the shortest path from s-> v?

Why does Dijkstra work?

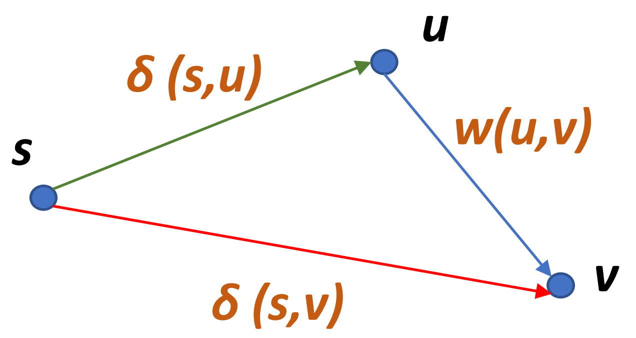


Triangle inequality: $\forall (u,v) \in E,\ \delta(s,v) <= \delta(s,u)+w(u,v)$

Then! $\delta(s,v) = \delta(s,u)+w(u,v)$

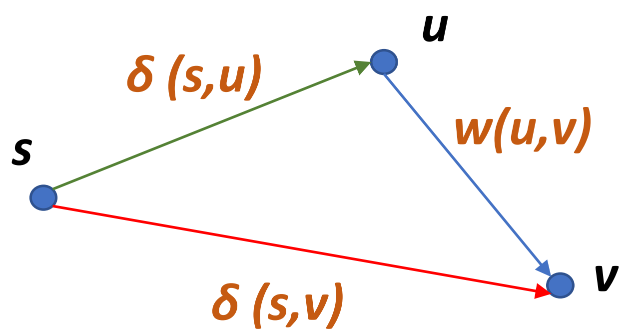
Why does Dijkstra work?

Triangle inequality: $\forall (u,v) \in E, \delta(s,v) <= \delta(s,u)+w(u,v)$



Upper bound: $dv >= \delta(s,v)$

Why does Dijkstra work?



Triangle inequality: $\forall (u,v) \in E,\ \delta(s,v) <= \delta(s,u) + w(u,v)$

Upper bound: $d_v >= \delta(s,v)$

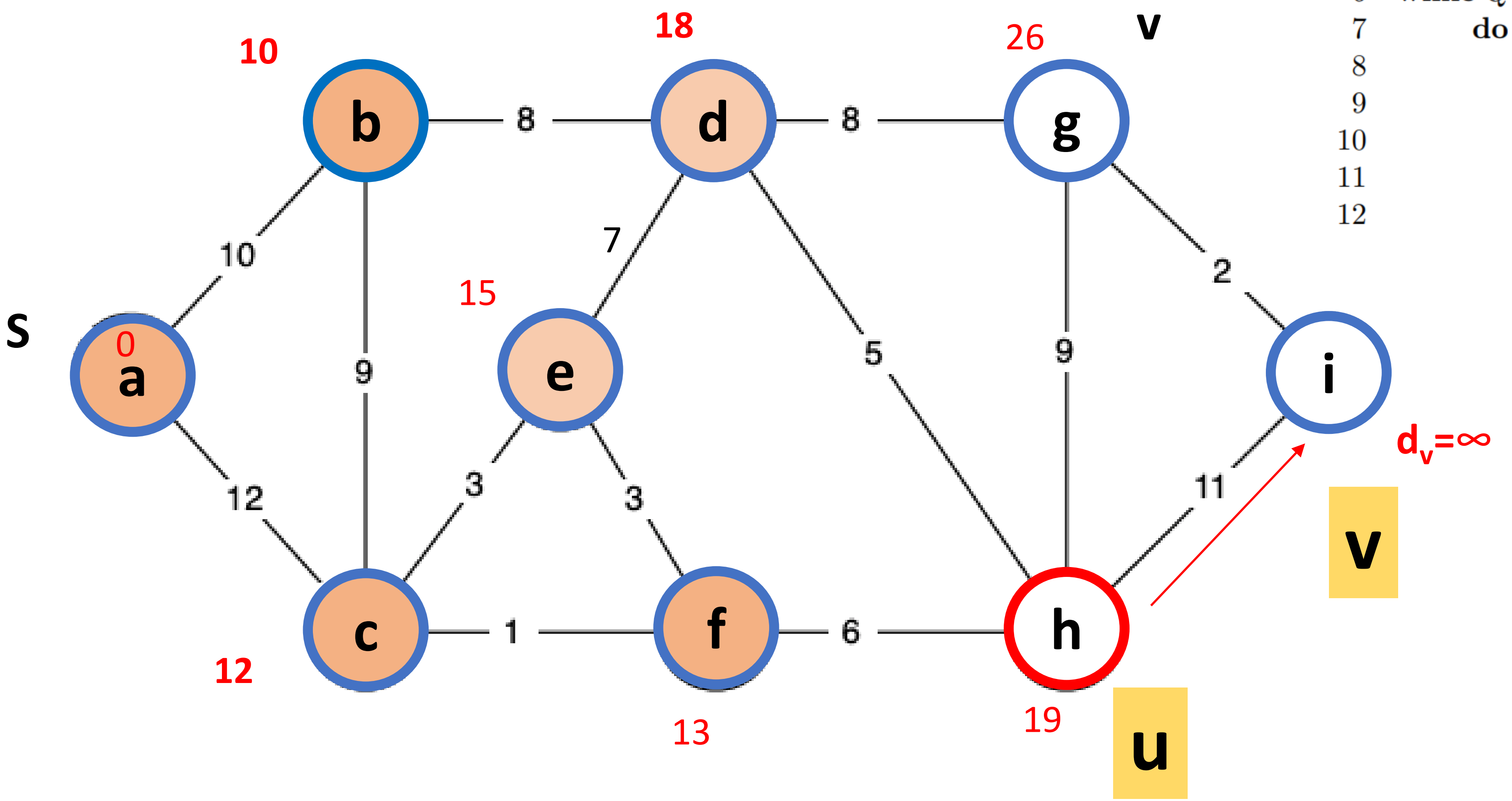
Follows because, we initiate all d_v with ∞

And we only update d_v by using

DIJKSTRA$(G = (V, E), s)$

```
for all v ∈ V
        do d_u ← ∞
           π_u ← NIL
d_s ← 0
Q ← MAKEQUEUE(V)    ▷ use d_u as key
while Q ≠ ∅
        do u ← EXTRACTMIN(Q)
           for each v ∈ Adj(u)
               do if d_v > d_u + w(u,v)
                     then d_v ← d_u + w(u,v)
                          π_v ← u
                          DECREASEKEY(Q, v)
```

Upper bound: $d_v >= \delta\ (s,v)$

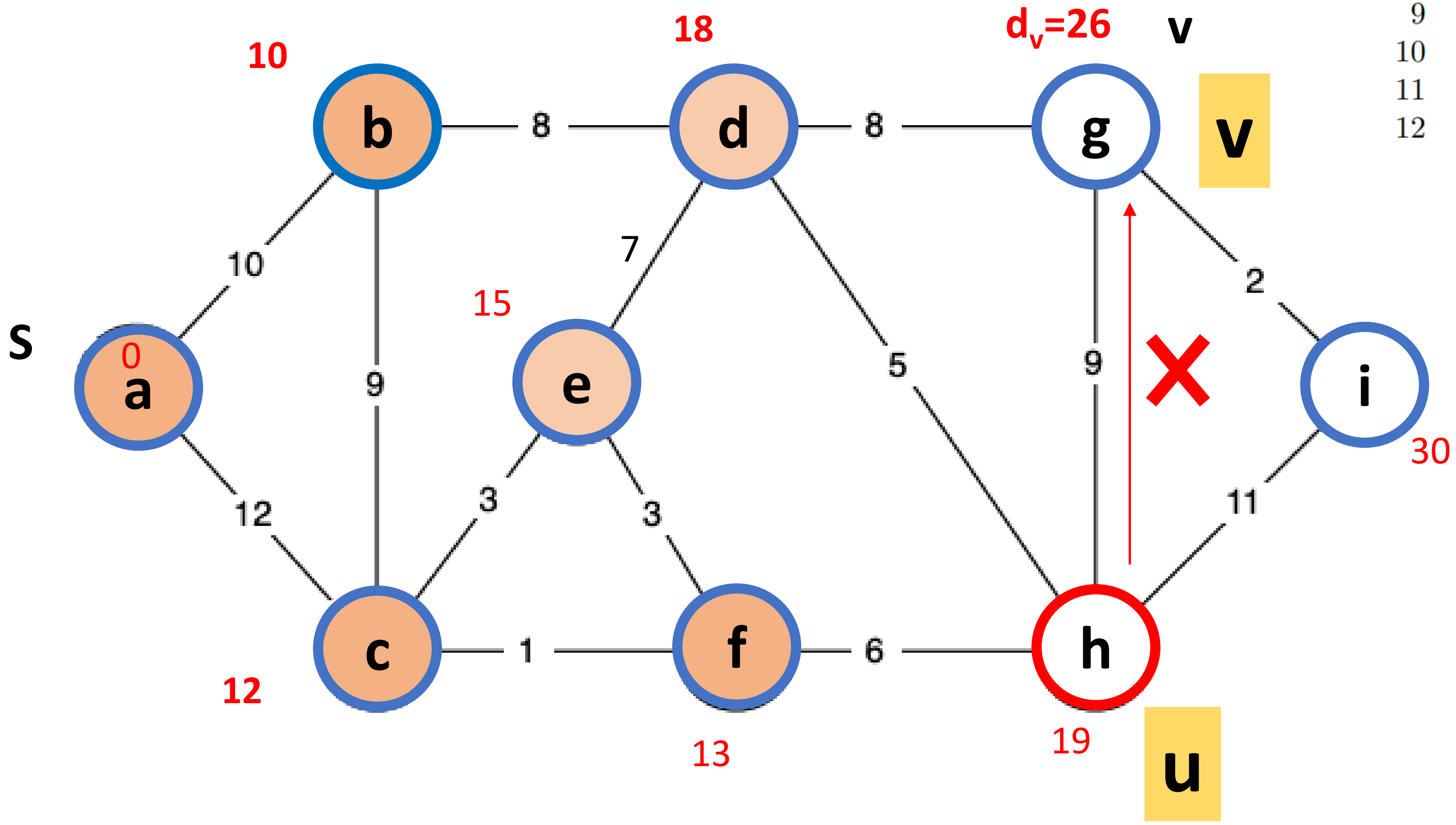



```
DIJKSTRA(G = (V, E), s)
for all v ∈ V
        do d_u ← ∞
            π_u ← NIL
d_s ← 0
Q ← MAKEQUEUE(V)      ▷ use d_u as key
while Q ≠ ∅
        do u ← EXTRACTMIN(Q)
            for each v ∈ Adj(u)
                do if d_v > d_u + w(u, v)
                      then d_v ← d_u + w(u, v)
                           π_v ← u
                           DECREASEKEY(Q, v)
```

$d_v > \delta\ (s,v)$

Upper bound: $d_v >= \delta(s,v)$

$d_v = \delta(s,v)$

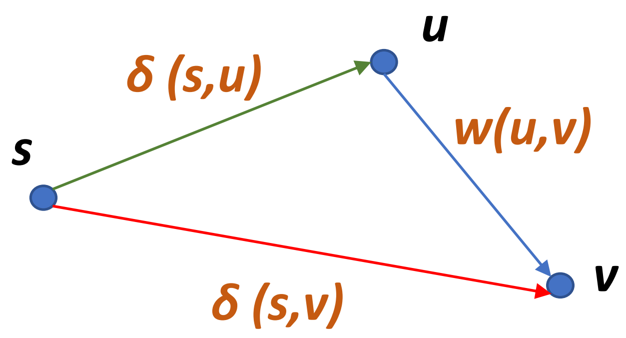


DIJKSTRA$(G = (V, E), s)$

for all $v \in V$
  do $d_u \leftarrow \infty$
    $\pi_u \leftarrow$ NIL
$d_s \leftarrow 0$
$Q \leftarrow$ MAKEQUEUE(V) $\triangleright$ use d_u as key
while $Q \neq \emptyset$
  do $u \leftarrow$ EXTRACTMIN(Q)
    for each $v \in Adj(u)$
      do if $d_v > d_u + w(u, v)$
        then $d_v \leftarrow d_u + w(u, v)$
          $\pi_v \leftarrow u$
          DECREASEKEY(Q, v)

Why does Dijkstra work?

Triangle inequality: $\forall (u,v) \in E,\ \delta(s,v) <= \delta(s,u) + w(u,v)$



Upper bound: $d_v >= \delta(s,v)$

Follows because, we initiate all d_v with ∞

And we only update d_v by using

```
DIJKSTRA(G = (V, E), s)
for all v ∈ V
        do d_u ← ∞
           π_u ← NIL
d_s ← 0
Q ← MAKEQUEUE(V)        ▷ use d_u as key
while Q ≠ ∅
        do u ← EXTRACTMIN(Q)
           for each v ∈ Adj(u)
                do if d_v > d_u + w(u, v)
                      then d_v ← d_u + w(u, v)
                           π_v ← u
                           DECREASEKEY(Q, v)
```

Why does Dijkstra work? *Proof*

```
DIJKSTRA(G = (V, E), s)
1   for all v ∈ V
2       do d_u ← ∞
3          π_u ← NIL
4   d_s ← 0
5   Q ← MAKEQUEUE(V)    ▷ use d_u as key
6   while Q ≠ ∅
7       do u ← EXTRACTMIN(Q)
8          for each v ∈ Adj(u)
9              do if d_v > d_u + w(u, v)
10                 then d_v ← d_u + w(u, v)
11                      π_v ← u
12                      DECREASEKEY(Q, v)
```

Let S be all the nodes, not in Q. At line 1, S is empty.

Property 1: for all v ϵ S, $\mathbf{d_v} = \boldsymbol{\delta}$ ***(s,v)***

We will prove it

Proof by induction: Property 1 holds at the start of the loop.
Suppose it holds at iteration i of the loop. Consider iteration **i+1.**
In line 7, we extract a node u. Only **u** is added to S.
Now lets argue that, $\mathbf{d_u} = \boldsymbol{\delta}$ ***(s,u)***

prove by contradiction

Why does Dijkstra work? *Proof*

Let S be all the nodes, not in Q. At line 1, S is empty.

Property 1: for all v ϵ S, $\mathbf{d_v} = \boldsymbol{\delta(s,v)}$

We will prove it

Proof by induction: Property 1 holds at the start of the loop.
Suppose it holds at iteration i of the loop. Consider iteration i+1.
In line 7, we extract a node u. Only u is added to S.
Now lets argue that, $\mathbf{d_u} = \boldsymbol{\delta(s,u)}$

For the sake of reaching a contradiction, lets consider: $\mathbf{d_u} \neq \boldsymbol{\delta(s,u)}$

DIJKSTRA$(G = (V, E), s)$

1 **for** all $v \in V$
2   **do** $d_u \leftarrow \infty$
3         $\pi_u \leftarrow$ NIL
4 $d_s \leftarrow 0$
5 $Q \leftarrow$ MAKEQUEUE(V) ▷ use d_u as key
6 **while** $Q \neq \emptyset$
7   **do** $u \leftarrow$ EXTRACTMIN(Q)
8   **for** each $v \in Adj(u)$
9   **do if** $d_v > d_u + w(u, v)$
10   **then** $d_v \leftarrow d_u + w(u, v)$
11                     $\pi_v \leftarrow u$
12   DECREASEKEY(Q, v)

Why does Dijkstra work? *Proof*

```
DIJKSTRA(G = (V, E), s)
1   for all v ∈ V
2       do d_u ← ∞
3           π_u ← NIL
4   d_s ← 0
5   Q ← MAKEQUEUE(V)    ▷ use d_u as key
6   while Q ≠ ∅
7       do u ← EXTRACTMIN(Q)
8           for each v ∈ Adj(u)
9               do if d_v > d_u + w(u, v)
10                  then d_v ← d_u + w(u, v)
11                      π_v ← u
12                      DECREASEKEY(Q, v)
```

Let S be all the nodes, not in Q. At line 1, S is empty.

Property 1: for all v ϵ S, $\mathbf{d_v} = \boldsymbol{\delta}$ ***(s,v)***

We will prove it

Proof by induction: Property 1 holds at the start of the loop.
Suppose it holds at iteration i of the loop. Consider iteration i+1.
In line 7, we extract a node u. Only u is added to S.
Now lets argue that, $\mathbf{d_u} = \boldsymbol{\delta}$ ***(s,u)***

For the sake of reaching a contradiction, lets consider: $\mathbf{d_u} \neq \boldsymbol{\delta}$ ***(s,u) (*)***

Claim 1: $\mathbf{d_u} >= \boldsymbol{\delta}$ ***(s,u)*** *(from previously shown upper bound lemma)*

There is some path from s to u. If there are no path, δ(s,u)=∞

By definition ***(*)***, we know that $\mathbf{d_u}$!= *δ(s,u)* but also, $\mathbf{d_u}$ >= *δ(s,u)*

So, δ(s,u)!=∞, δ(s,u) can not be ∞

The point is algorithm will never reach to node u if there were no path from s -> u

Why does Dijkstra work? *Proof*

Let **S** be all the nodes, not in **Q.**

Consider any path from **s** to **u**. **(any include the shortest path too)**

Let **x** be the very last node is **S** along the path from s to u: **path P**

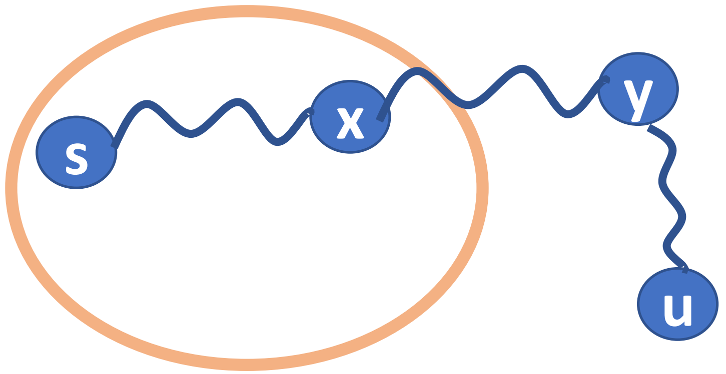
(x,y) is the first edge that cross the cut (S, V-S) where **S** is the Set right before **u** is extracted. **(until iteration i)**

```
DIJKSTRA(G = (V, E), s)
for all v ∈ V
        do d_u ← ∞
           π_u ← NIL
d_s ← 0
Q ← MAKEQUEUE(V)    ▷ use d_u as key
while Q ≠ ∅
        do u ← EXTRACTMIN(Q)
           for each v ∈ Adj(u)
               do if d_v > d_u + w(u, v)
                     then d_v ← d_u + w(u, v)
                          π_v ← u
                          DECREASEKEY(Q, v)
```

$$\mathbf{L(p)} = \boldsymbol{\delta\,(s,x) + w(x,y) + l(y\text{->}u)}$$

Because this happened in iteration i or less? (induction step)

Any path from y to u



Why does Dijkstra work? *Proof*

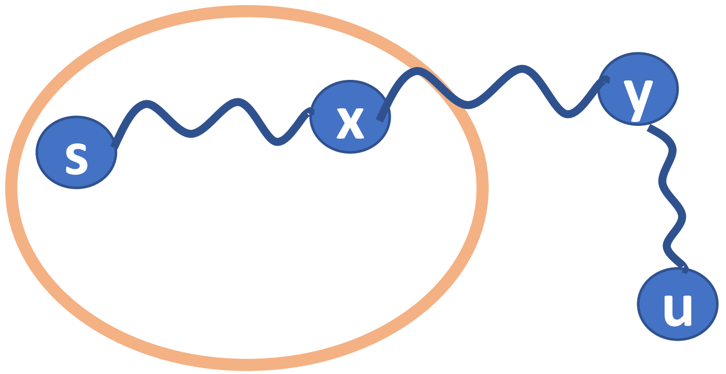
```
DIJKSTRA(G = (V, E), s)
1   for all v ∈ V
2        do d_u ← ∞
3           π_u ← NIL
4   d_s ← 0
5   Q ← MAKEQUEUE(V)    ▷ use d_u as key
6   while Q ≠ ∅
7        do u ← EXTRACTMIN(Q)
8           for each v ∈ Adj(u)
9                do if d_v > d_u + w(u, v)
10                     then d_v ← d_u + w(u, v)
11                          π_v ← u
12                          DECREASEKEY(Q, v)
```

Let **S** be all the nodes, not in **Q.**

Consider any path from **s** to **u**. **(any include the shortest path too)**

Let **x** be the very last node is **S** along the path from s to u: **path P**

(x,y) is the first edge that cross the cut (S, V-S) where **S** is the Set right before **u** is extracted. **(until iteration i)**



$$L(p) = \delta\,(s,x) + w(x,y) + l(y\text{->}u)$$

$$\quad >= dy + l(y\text{->}u)$$

Why? X is already been added to S

When line 10 of the algo was run for x,

Y was neighbor of x (d_v was Y)

So, dy ← δ (s,x) + w(x,y)

Why does Dijkstra work? *Proof*

```
DIJKSTRA(G = (V, E), s)
1   for all v ∈ V
2       do d_u ← ∞
3          π_u ← NIL
4   d_s ← 0
5   Q ← MAKEQUEUE(V)    ▷ use d_u as key
6   while Q ≠ ∅
7       do u ← EXTRACTMIN(Q)
8          for each v ∈ Adj(u)
9              do if d_v > d_u + w(u, v)
10                 then d_v ← d_u + w(u, v)
11                      π_v ← u
12                      DECREASEKEY(Q, v)
```

Let **S** be all the nodes, not in **Q.**

Consider any path from **s** to **u**. **(any include the shortest path too)**

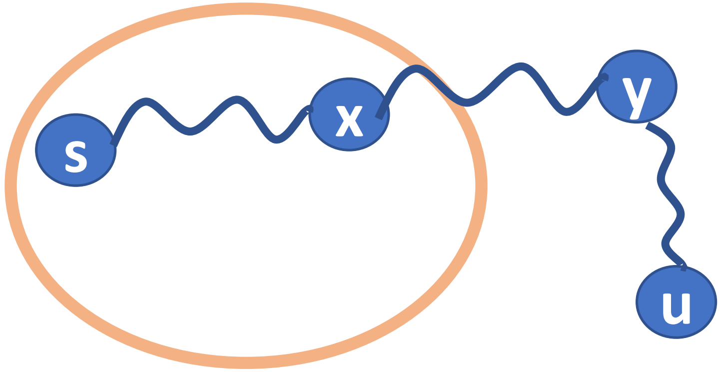
Let **x** be the very last node is **S** along the path from s to u: **path P**

(x,y) is the first edge that cross the cut (S, V-S) where **S** is the Set right before **u** is extracted. **(until iteration i)**

L(p)= *δ (s,x) + w(x,y) + l(y->u)*

>= dy + l(y->u)

>= du + l(y->u)



Why?

Because in this iteration, u was extracted, not y

Meaning!!! (line 7!)

du<=dy

Why does Dijkstra work? *Proof*

```
DIJKSTRA(G = (V, E), s)
for all v ∈ V
    do d_u ← ∞
       π_u ← NIL
d_s ← 0
Q ← MAKEQUEUE(V)    ▷ use d_u as key
while Q ≠ ∅
    do u ← EXTRACTMIN(Q)
       for each v ∈ Adj(u)
           do if d_v > d_u + w(u, v)
                 then d_v ← d_u + w(u, v)
                      π_v ← u
                      DECREASEKEY(Q, v)
```

Let **S** be all the nodes, not in **Q.**

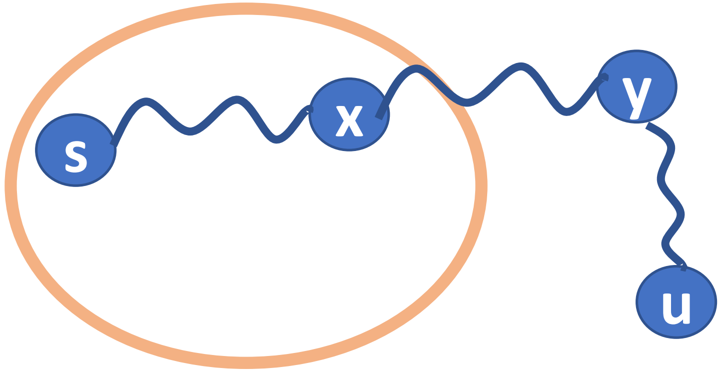
Consider any path from **s** to **u**. **(any include the shortest path too)**

Let **x** be the very last node is **S** along the path from s to u: **path P**

(x,y) is the first edge that cross the cut (S, V-S) where **S** is the Set right before **u** is extracted. **(until iteration i)**

$$L(p) = \delta(s,x) + w(x,y) + l(y\text{->}u)$$
$$>= dy + l(y\text{->}u)$$
$$>= du + l(y\text{->}u)$$
$$>= du$$

It is some positive value!!



Why does Dijkstra work? *Proof*

Let **S** be all the nodes, not in **Q.**

Consider any path from **s** to **u**. **(any include the shortest path too)**

Let **x** be the very last node is **S** along the path from s to u: **path P**

(x,y) is the first edge that cross the cut (S, V-S) where **S** is the Set right before **u** is extracted. **(until iteration i)**

```
DIJKSTRA(G = (V, E), s)
for all v ∈ V
      do d_u ← ∞
            π_u ← NIL
d_s ← 0
Q ← MAKEQUEUE(V)    ▷ use d_u as key
while Q ≠ ∅
      do u ← EXTRACTMIN(Q)
            for each v ∈ Adj(u)
                  do if d_v > d_u + w(u, v)
                        then d_v ← d_u + w(u, v)
                              π_v ← u
                              DECREASEKEY(Q, v)
```

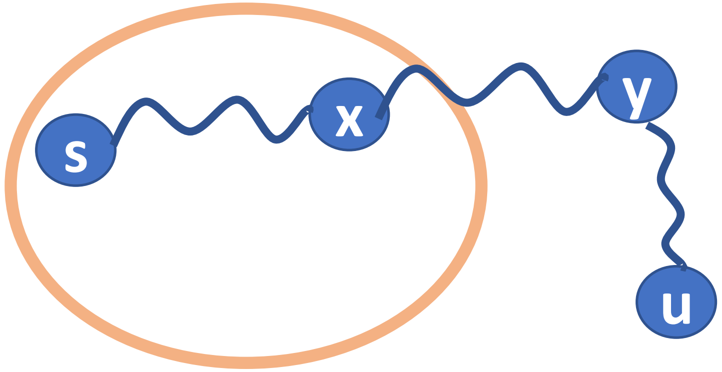
$$\mathbf{L(p)} = \boldsymbol{\delta}\,\boldsymbol{(s,x) + w(x,y) + l(y\text{->}u)}$$

$$\boldsymbol{>= dy + l(y\text{->}u)}$$

$$\boldsymbol{>= du + l(y\text{->}u)}$$

$$\boldsymbol{>= du}$$

It is some positive value!!



So, the length of any path from s-> u, will always be >= d_u

Why does Dijkstra work? *Proof*

Let **S** be all the nodes, not in **Q.**

Consider any path from **s** to **u**. **(any include the shortest path too)**

Let **x** be the very last node is **S** along the path from s to u: **path P**

(x,y) is the first edge that cross the cut (S, V-S) where **S** is the Set right before **u** is extracted. **(until iteration i)**

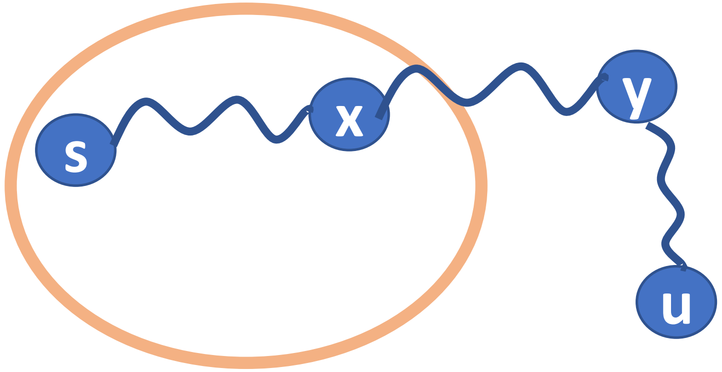
```
DIJKSTRA(G = (V, E), s)
for all v ∈ V
        do d_u ← ∞
           π_u ← NIL
d_s ← 0
Q ← MAKEQUEUE(V)    ▷ use d_u as key
while Q ≠ ∅
        do u ← EXTRACTMIN(Q)
           for each v ∈ Adj(u)
                do if d_v > d_u + w(u, v)
                      then d_v ← d_u + w(u, v)
                           π_v ← u
                           DECREASEKEY(Q, v)
```

$$L(p) = \delta(s,x) + w(x,y) + l(y\to u)$$

$$\geq d_y + l(y\to u)$$

$$\geq d_u + l(y\to u)$$

$$\geq d_u$$



It is some positive value!!

So, the length of any path from s-> u, will always be >= d_u

We also know that: $d_u \geq \delta(s,u)$

So, the length of any path from s-> u, will always be >= d_u

We also know that: $d_u >= \delta\ (s,u)$

$$d_u = \delta\ (s,u)$$

$\textsc{Dijkstra}(G = (V, E), s)$

for all $v \in V$
  do $d_u \leftarrow \infty$
    $\pi_u \leftarrow \text{NIL}$
$d_s \leftarrow 0$
$Q \leftarrow \text{MAKEQUEUE}(V)$ $\quad \triangleright$ use d_u as key
while $Q \neq \emptyset$
  do $u \leftarrow \text{EXTRACTMIN}(Q)$
    for each $v \in Adj(u)$
      do if $d_v > d_u + w(u, v)$
        then $d_v \leftarrow d_u + w(u, v)$
          $\pi_v \leftarrow u$
          $\text{DECREASEKEY}(Q, v)$