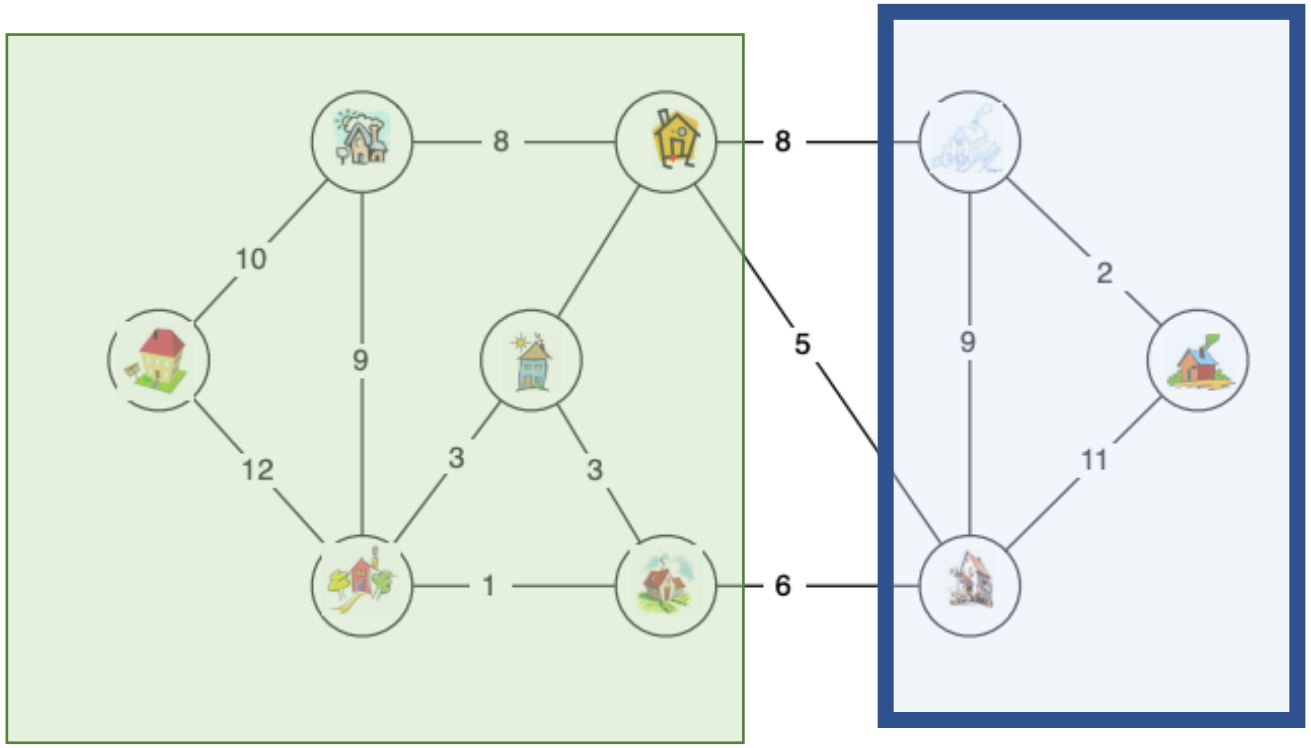
Greedy

Lecture 4

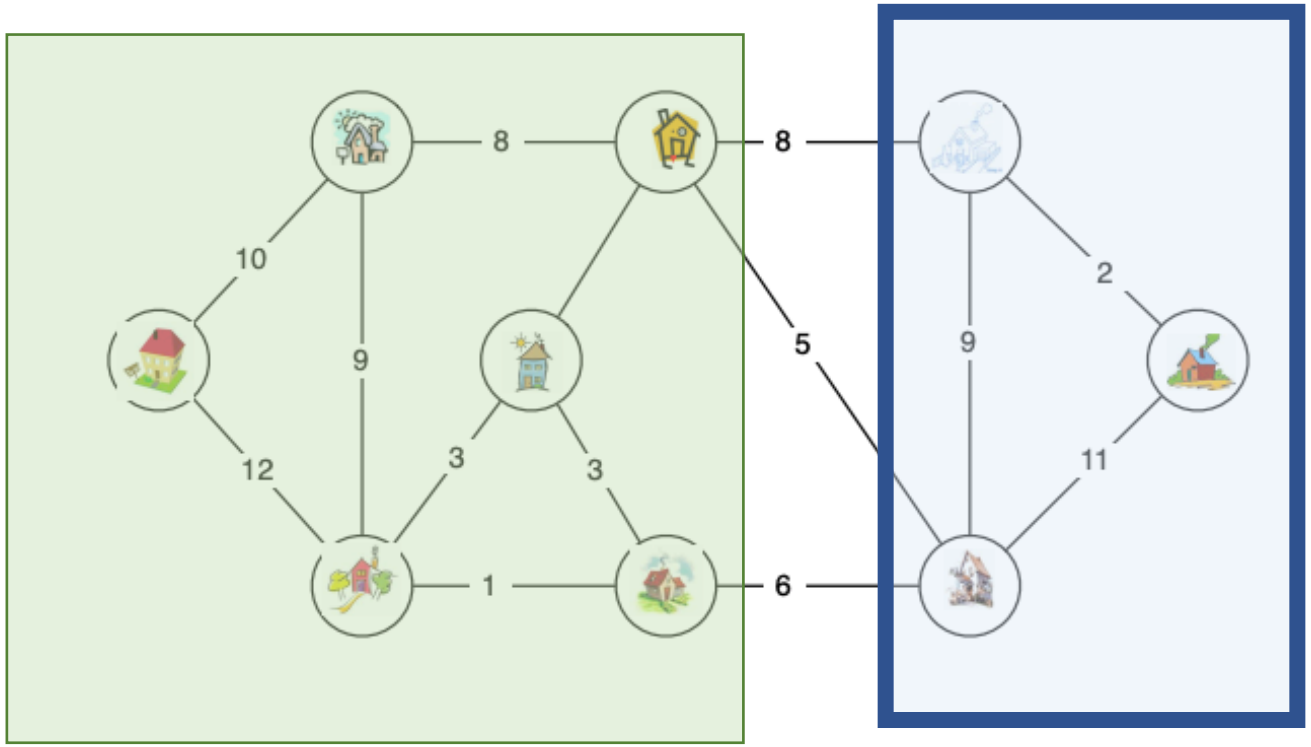
Definition: CUT

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



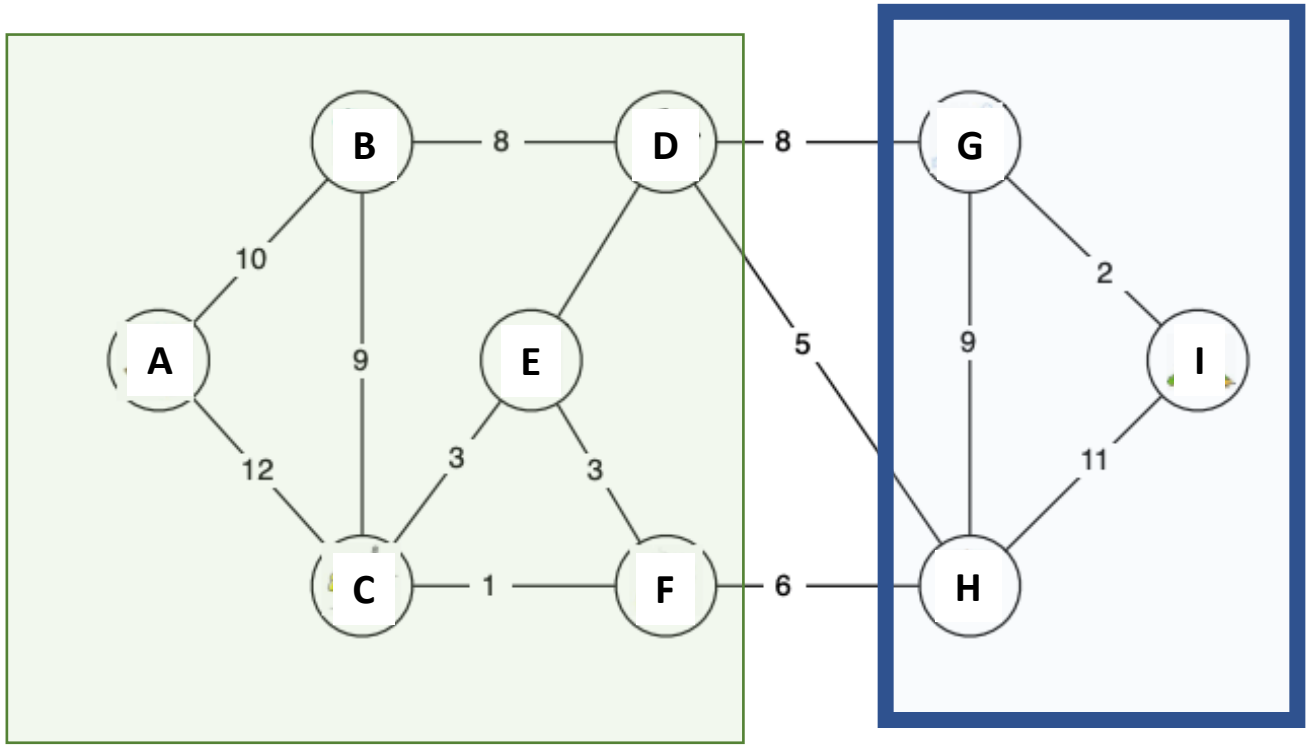
Definition: Crossing a CUT

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



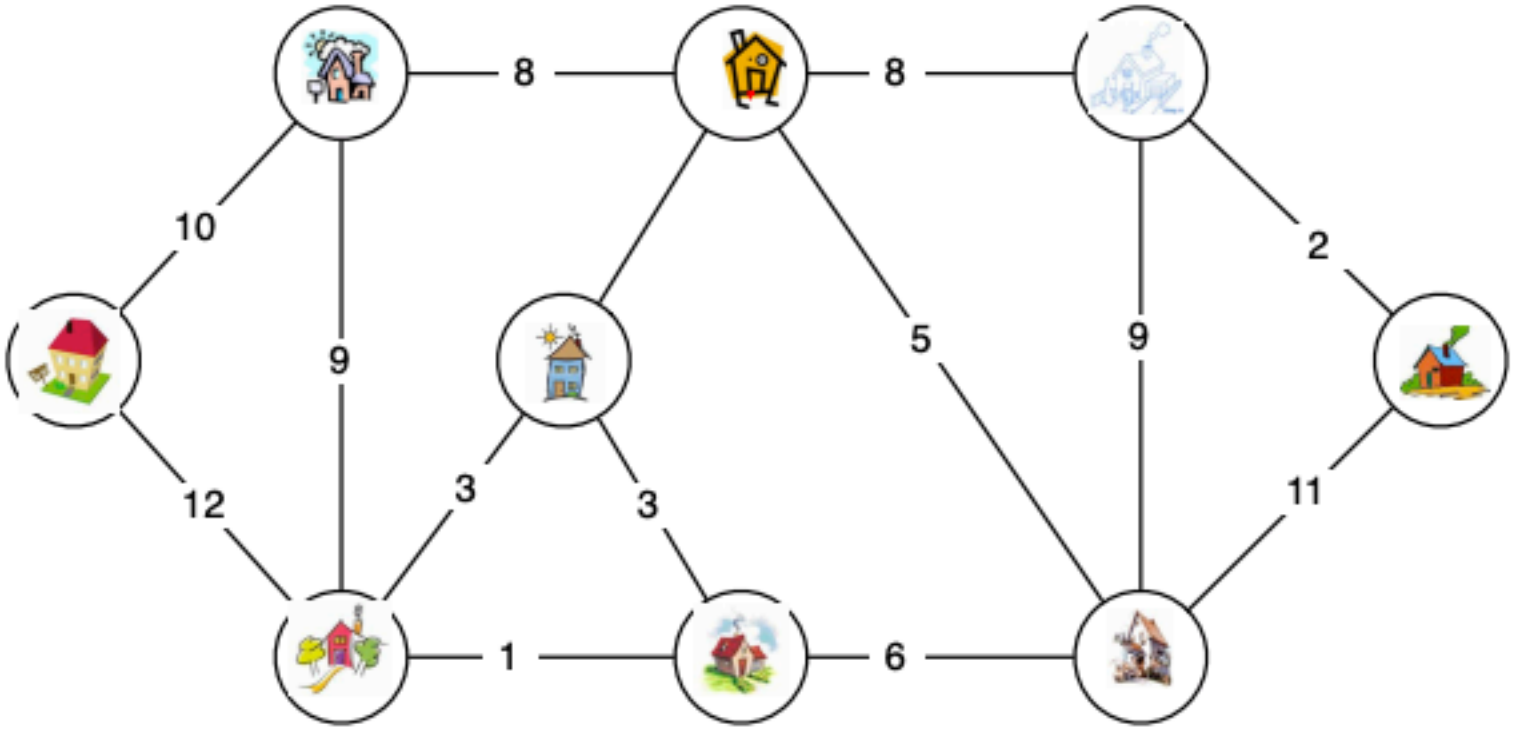
Definition: Respect

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



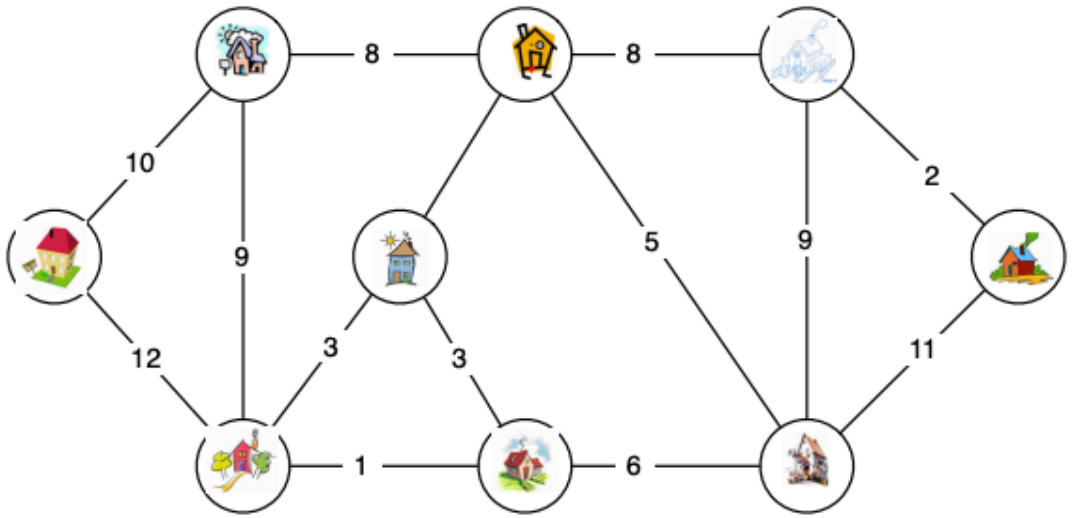
List of A

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?

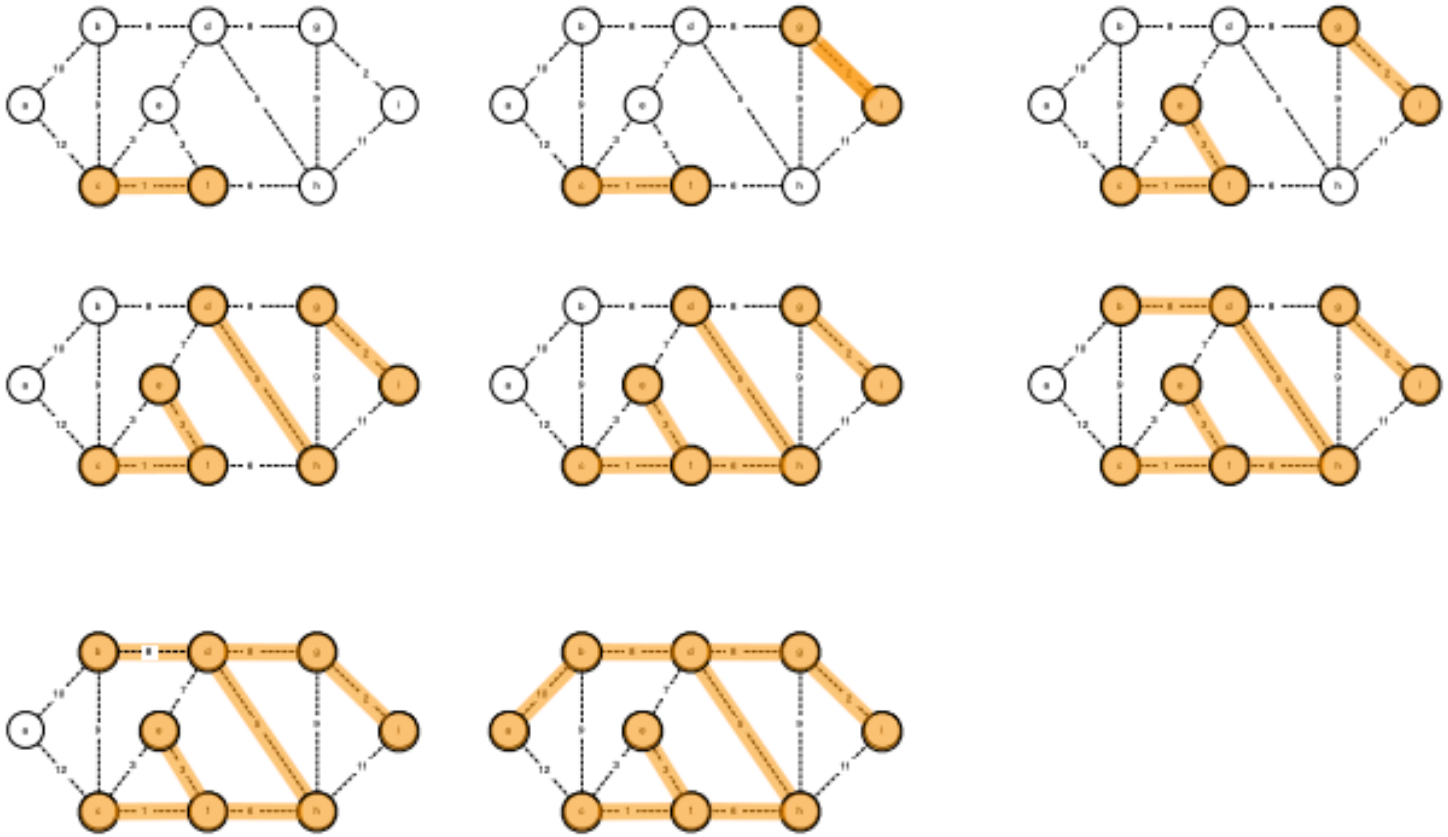
V-1

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.

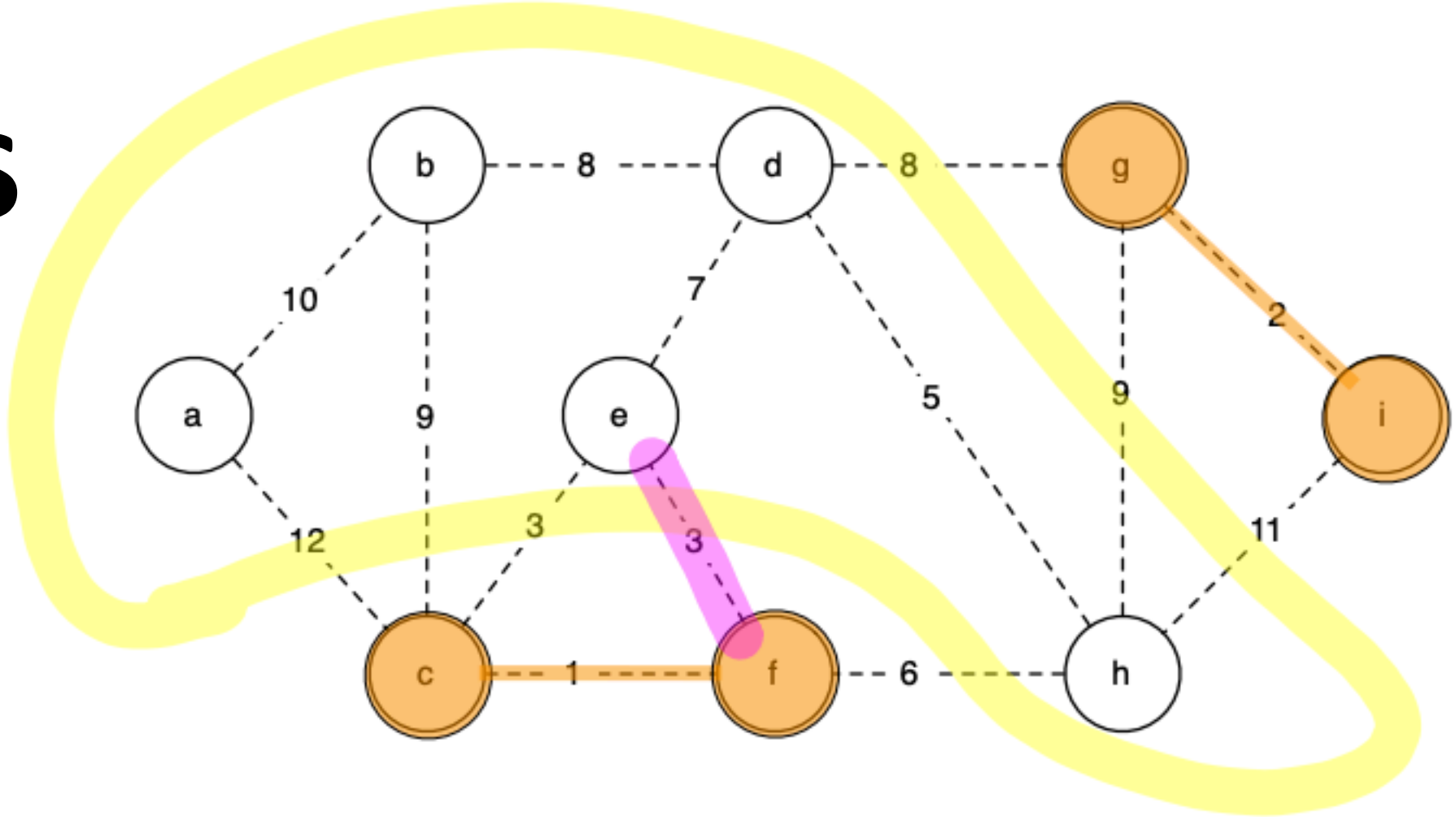
Let $(S, V - S)$ be any cut that respects A.

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.

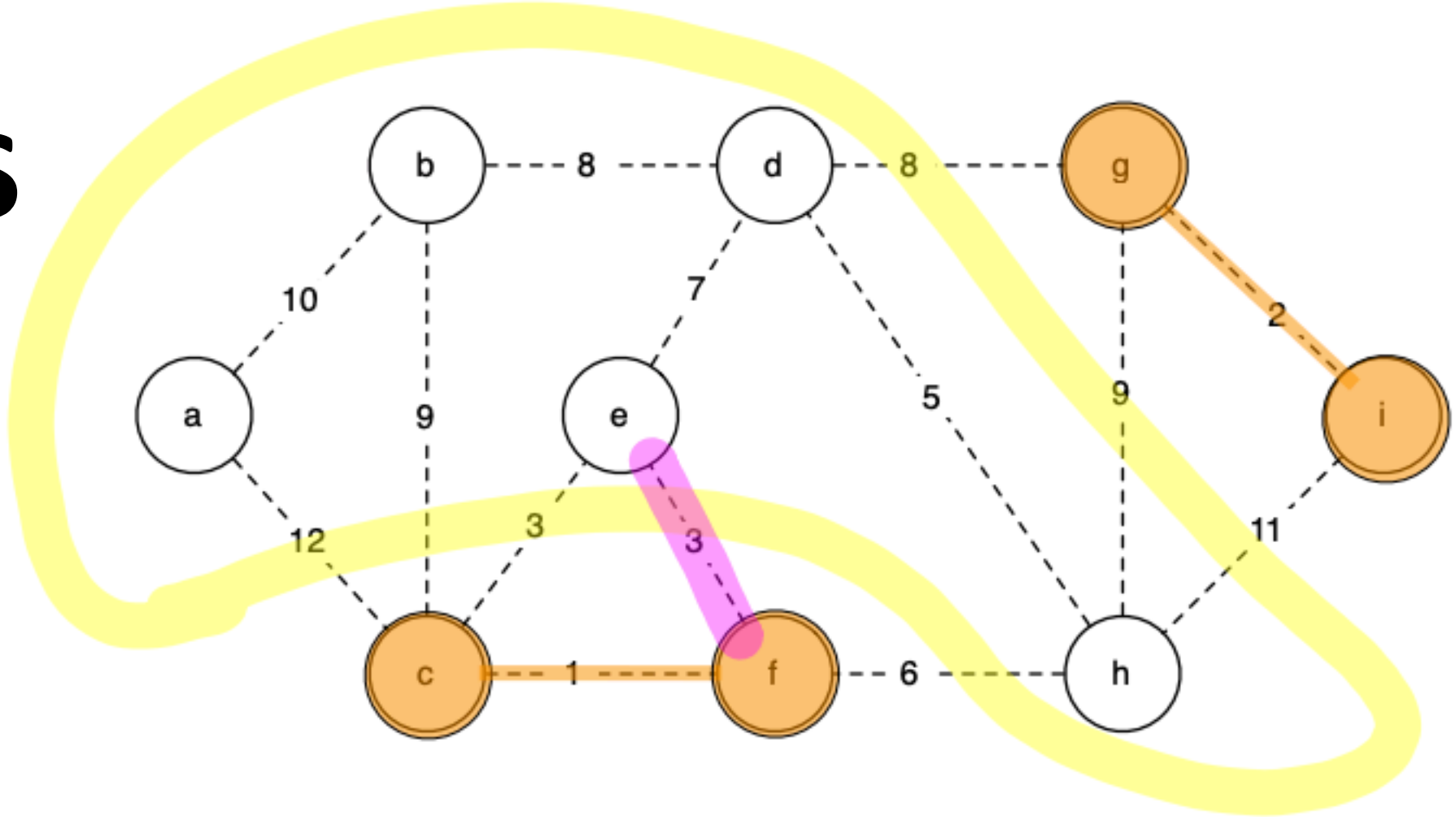
A respects S.

e is the least weighted edge that crosses S.

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

example of theorem

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)

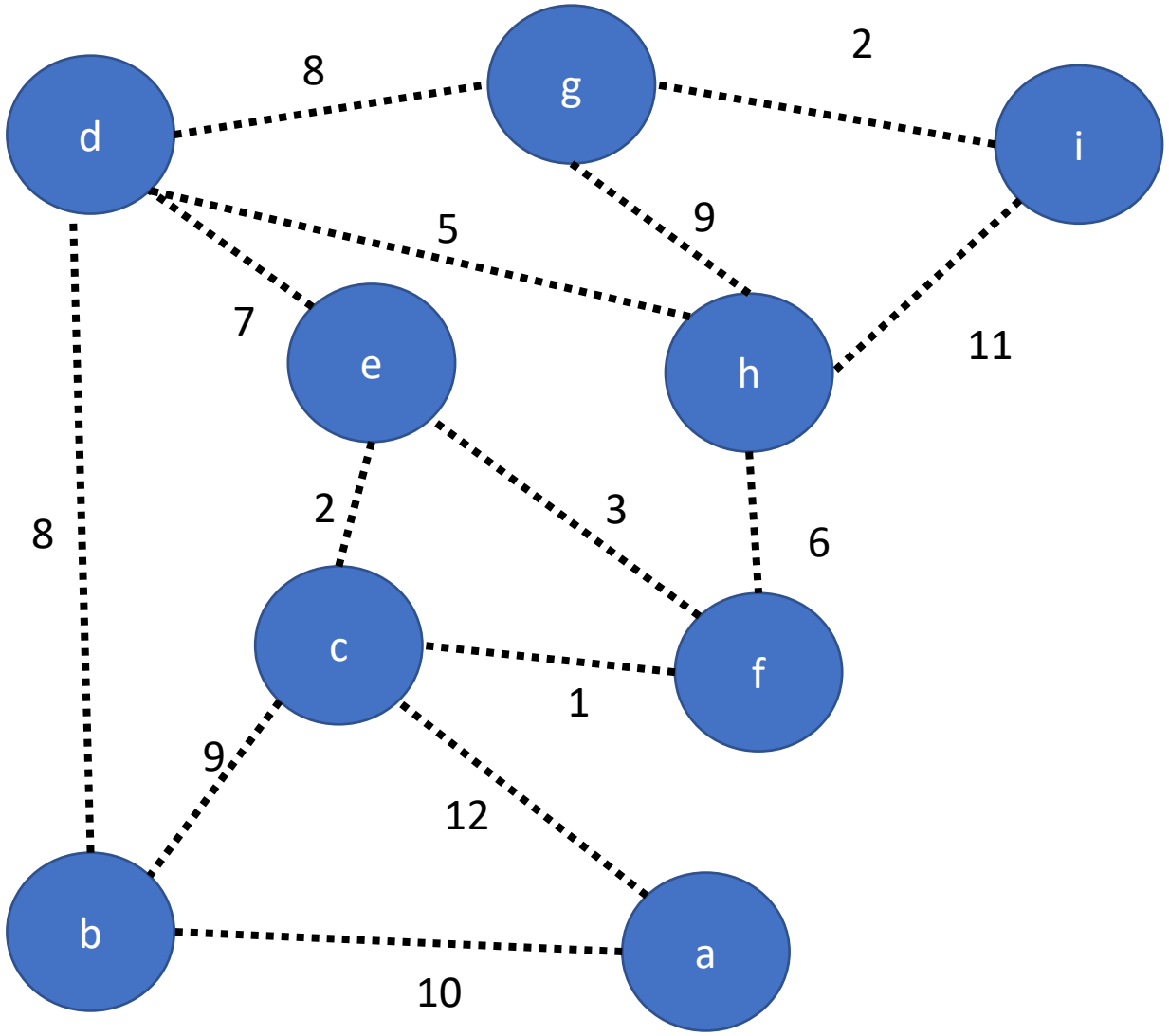
GENERAL-MST-STRATEGY($G = (V, E)$)

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

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

Algorithm

A= Ø

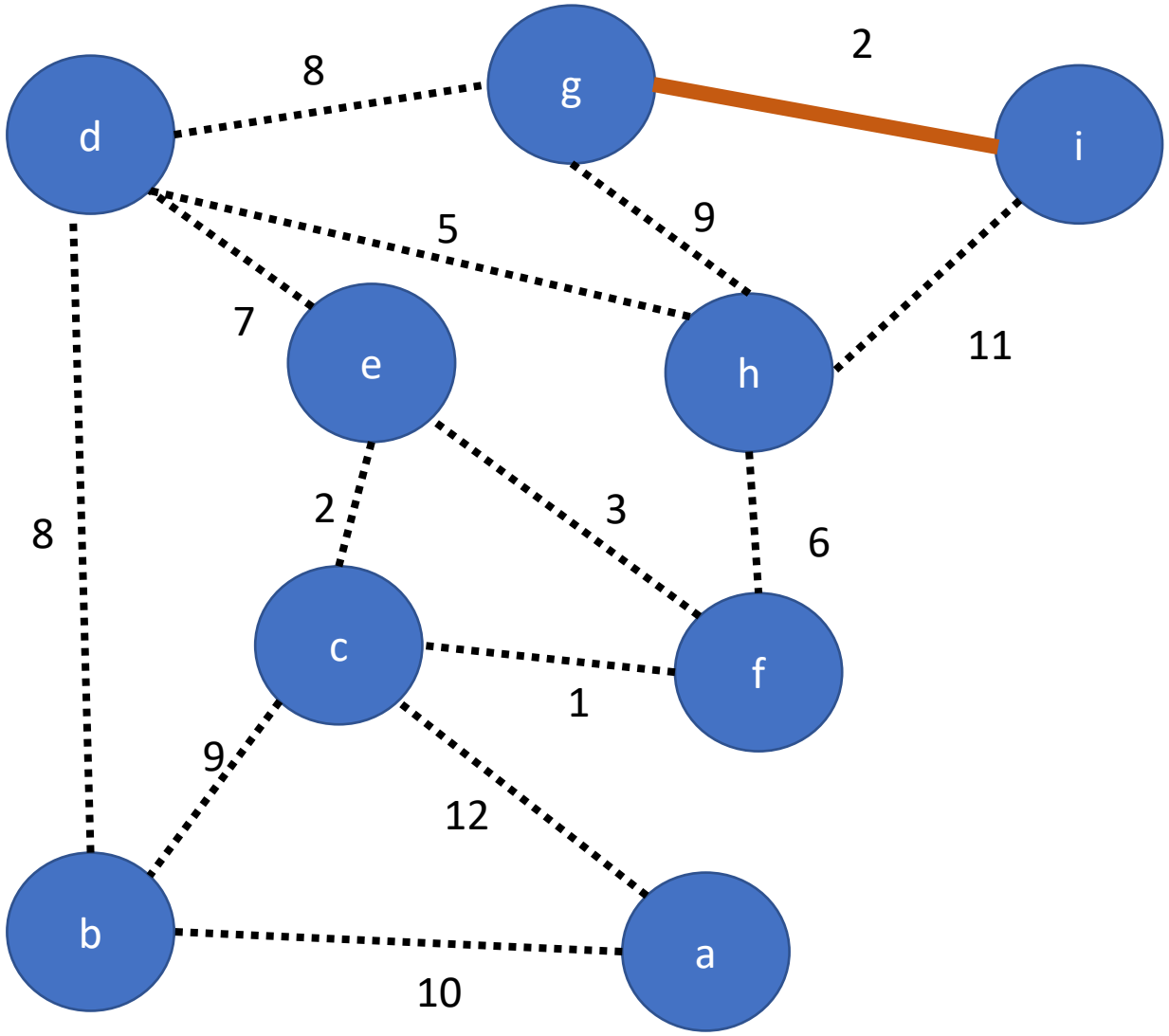


GENERAL-MST-STRATEGY($G = (V, E)$)

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)}

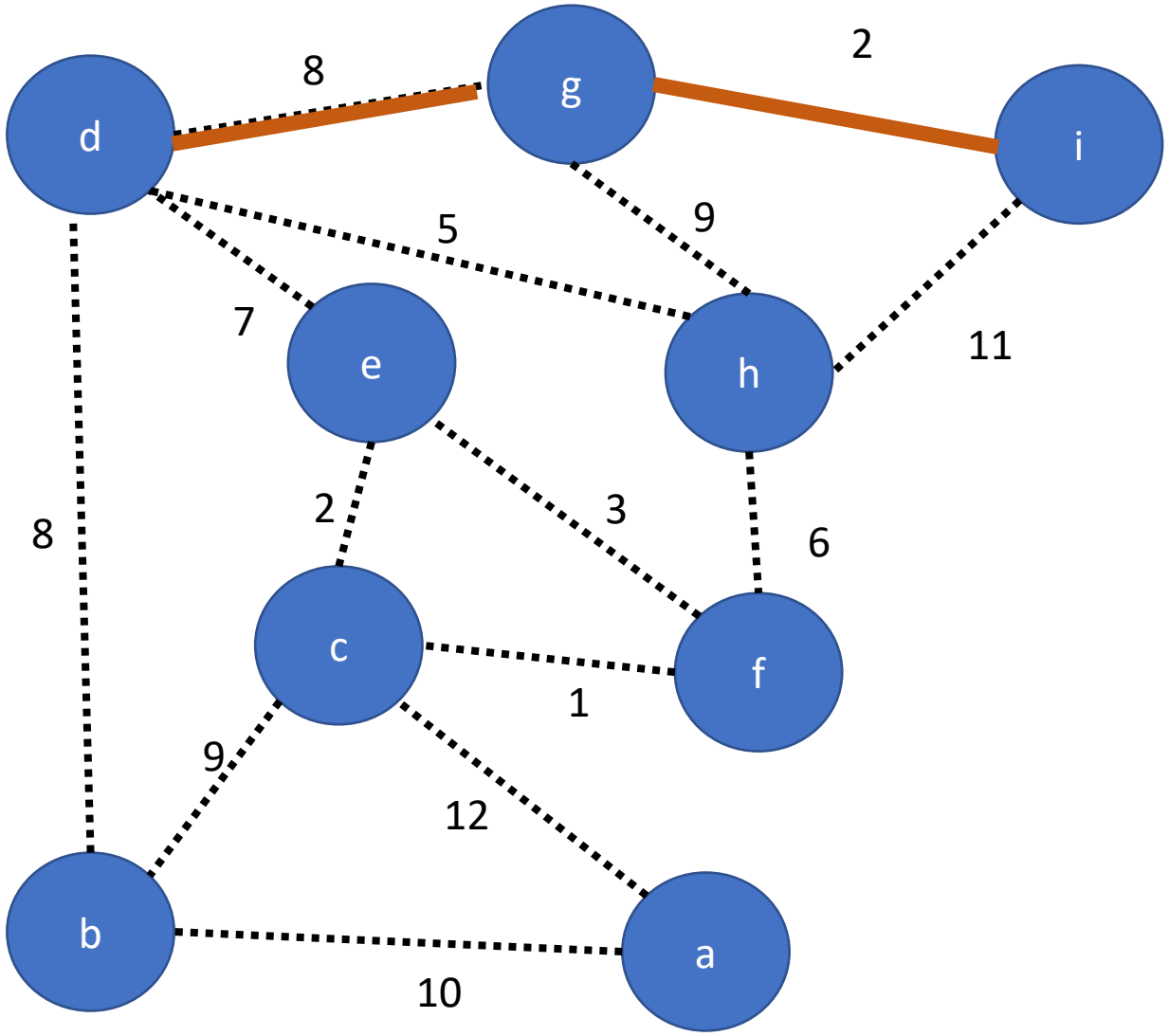


GENERAL-MST-STRATEGY($G = (V, E)$)

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)}

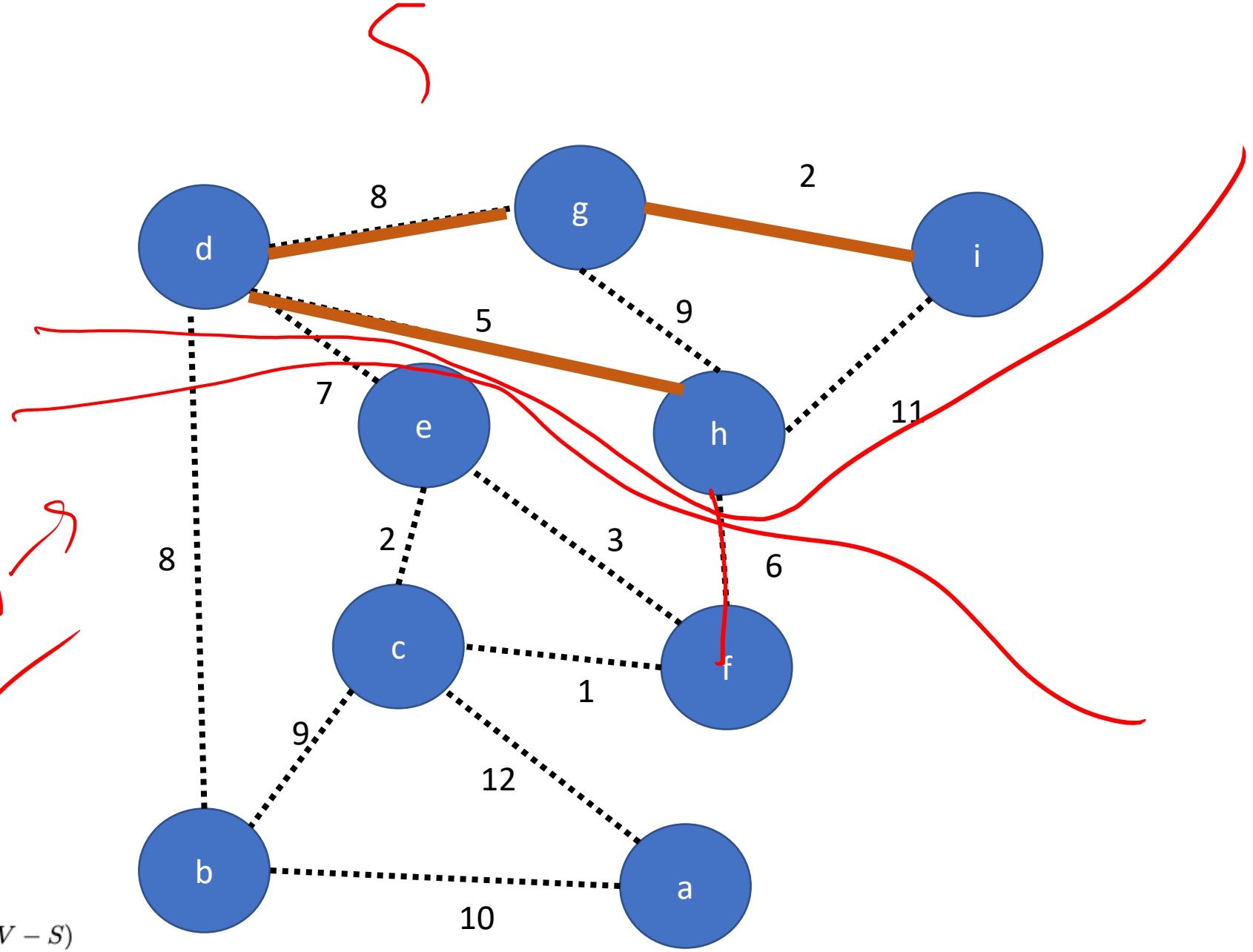


GENERAL-MST-STRATEGY($G = (V, E)$)

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)}

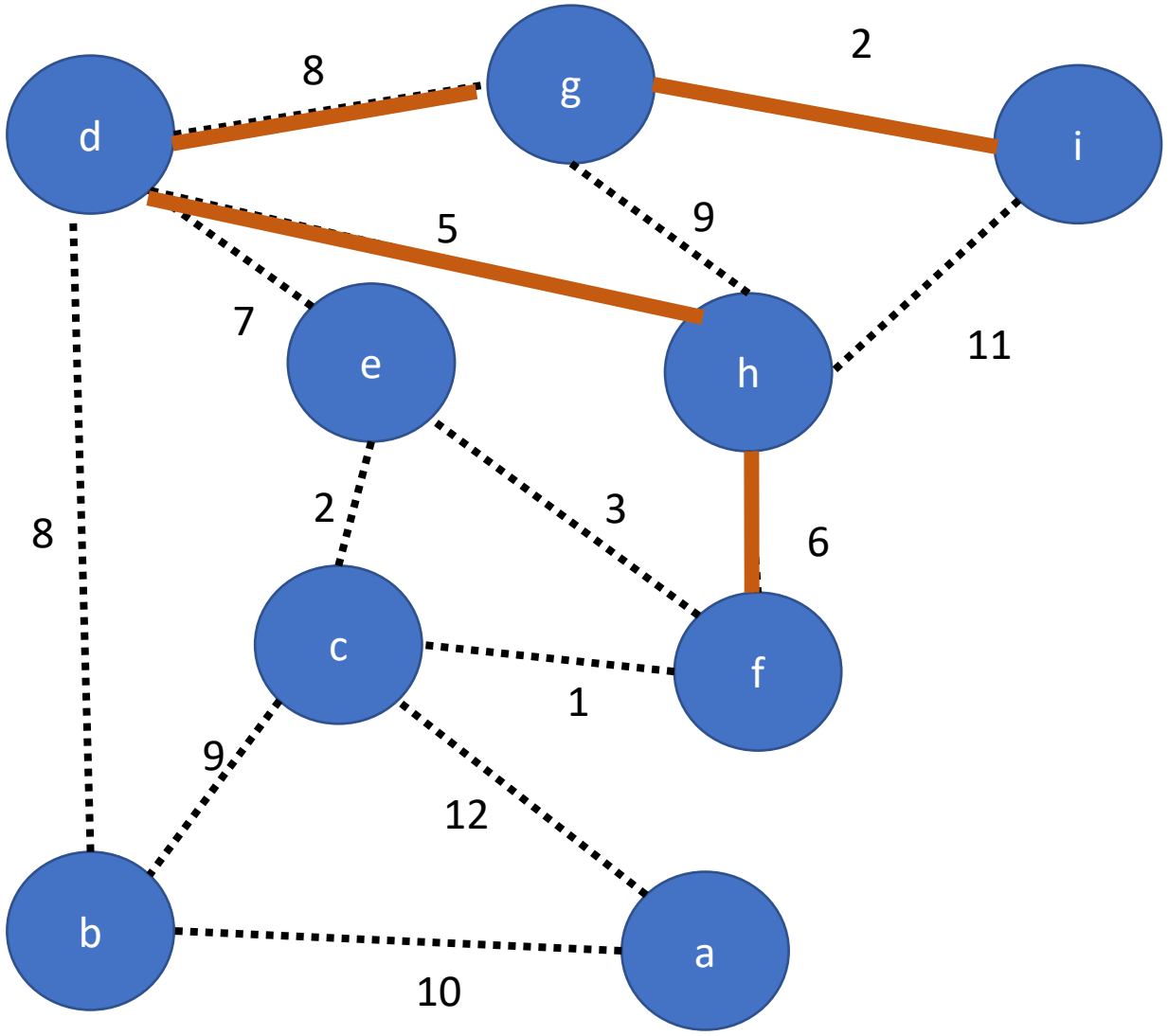


GENERAL-MST-STRATEGY($G = (V, E)$)

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($G = (V, E)$)

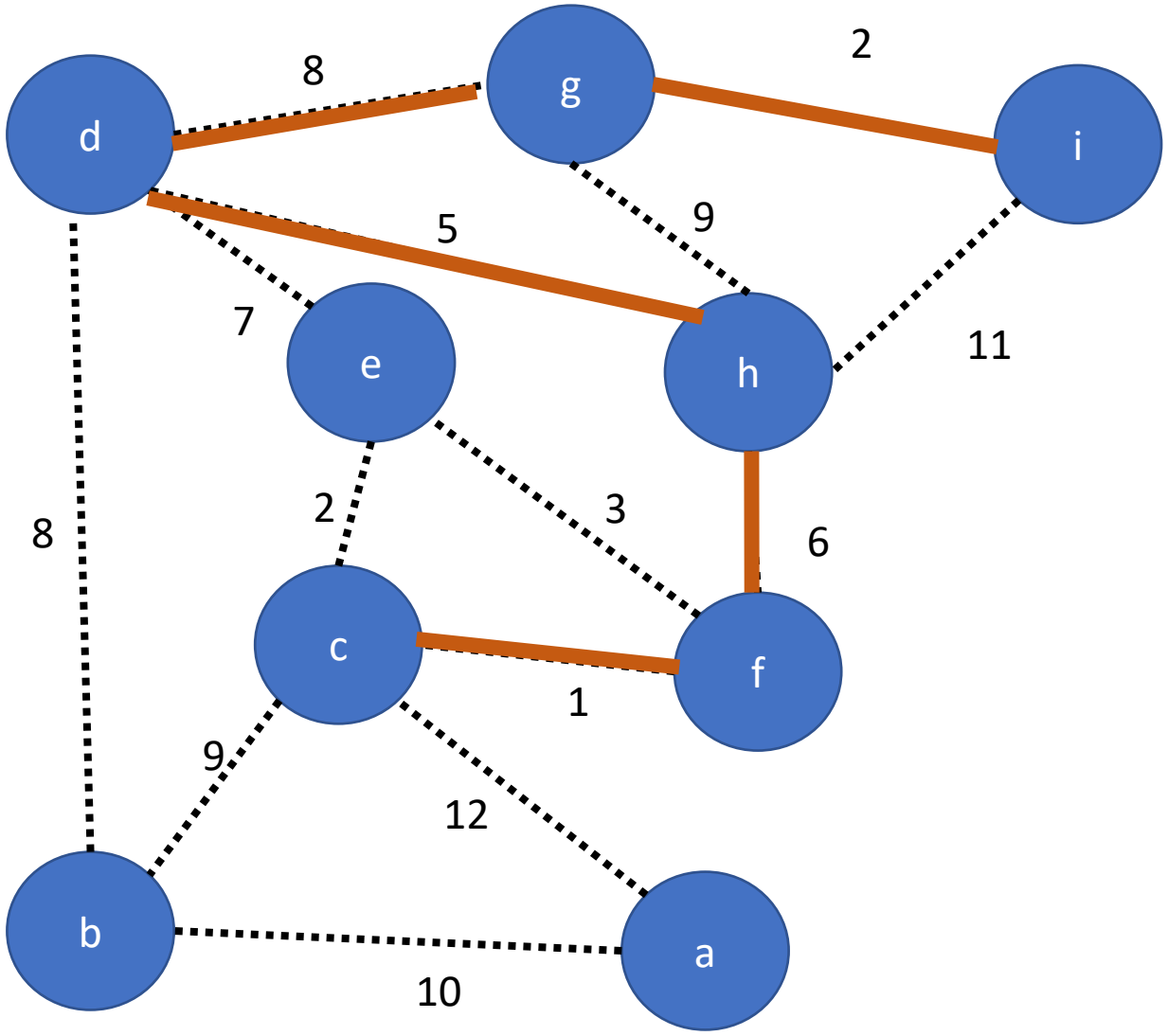
1 $A \leftarrow \emptyset$
2 **repeat** $V - 1$ times:
3 Pick a cut $(S, V - S)$ that respects A
4 Let e be min-weight edge over cut $(S, V - S)$
5 $A \leftarrow A \cup \{e\}$

Algorithm

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

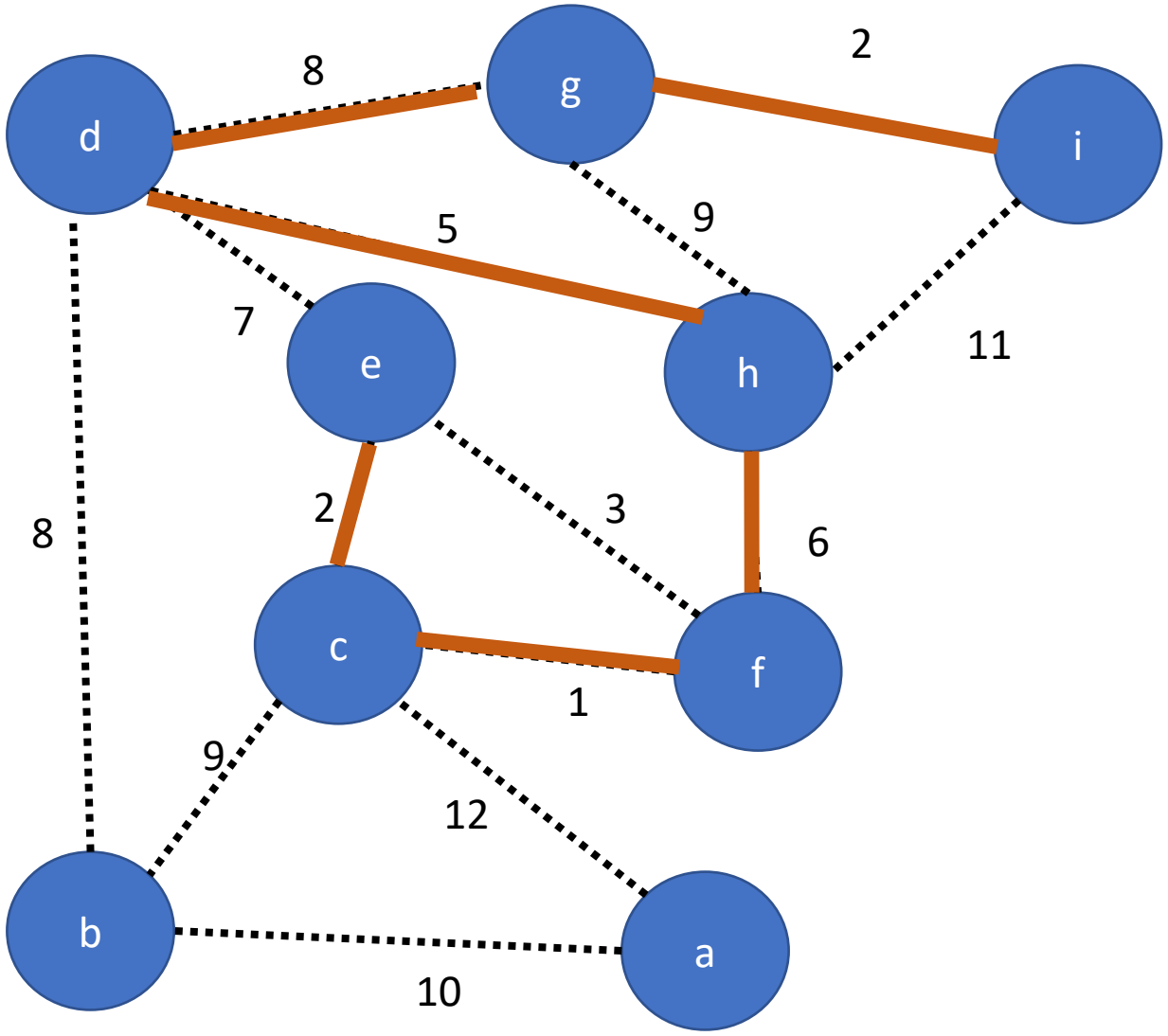
GENERAL-MST-STRATEGY$(G = (V, E))$

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)}



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

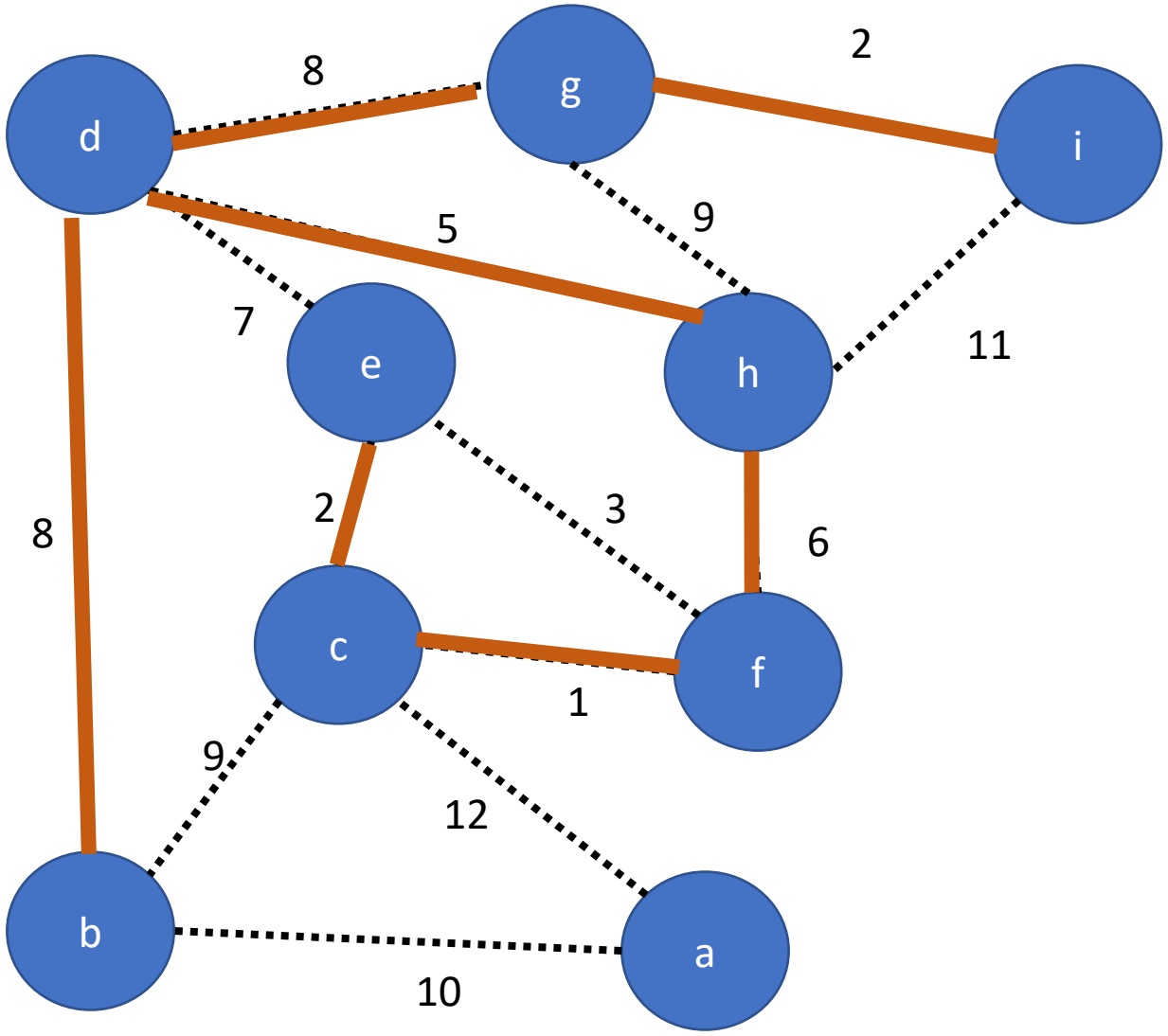
1 $A \leftarrow \emptyset$
2 **repeat** $V - 1$ times:
3 Pick a cut $(S, V - S)$ that respects A
4 Let e be min-weight edge over cut $(S, V - S)$
5 $A \leftarrow A \cup \{e\}$

Algorithm

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

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

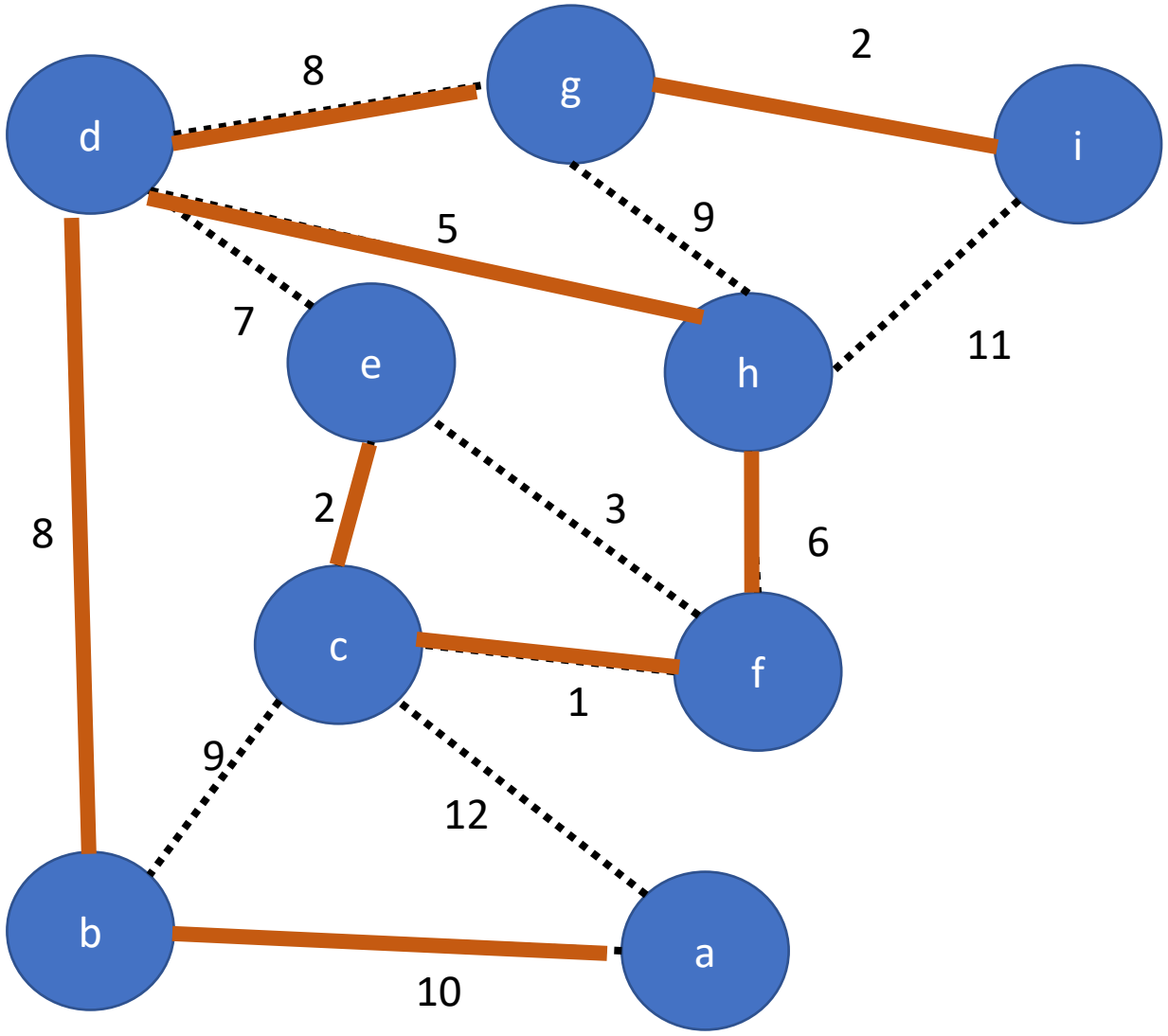


Algorithm

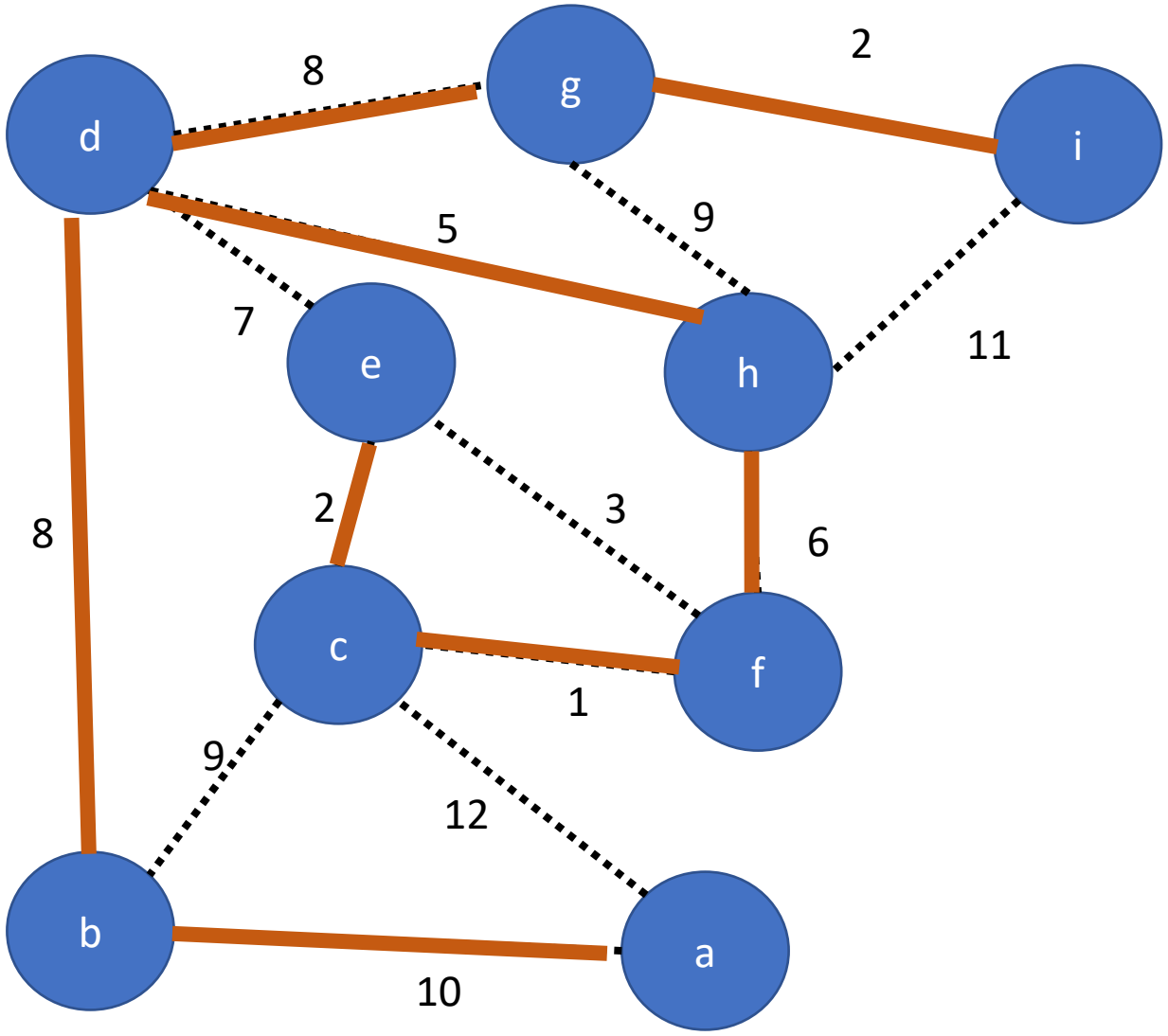
A= {(g,i), (d,g),(d,h), (h,f),(f,c), (c,e), (d,b),(b,a)}

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

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



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($G = (V, E)$)

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

Implementation

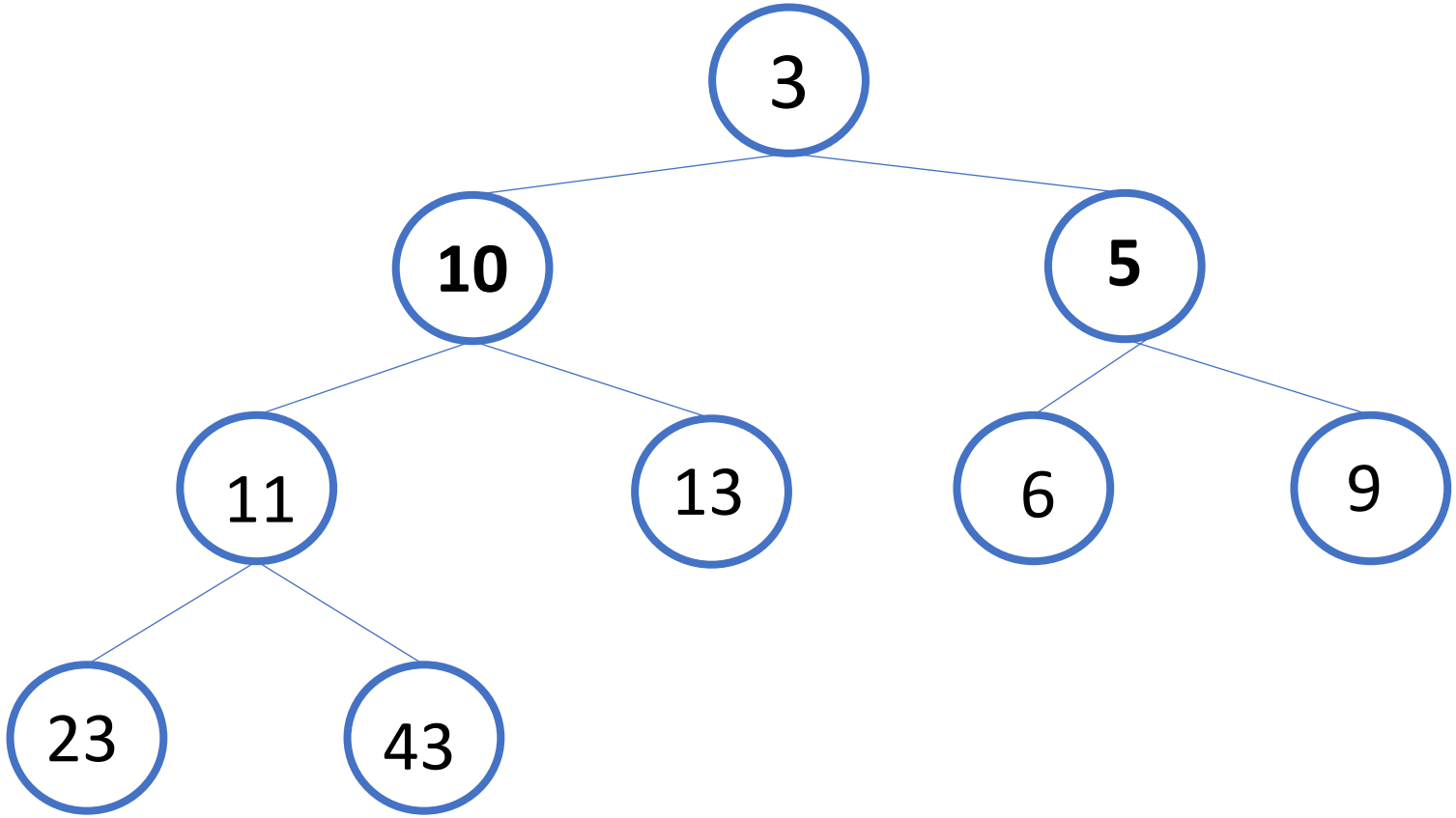
- Idea:
- Keep a data structure which identifies the ‘lightest edge’ that crosses the cuts (A=S, V-S)
  - Priority queue

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*.

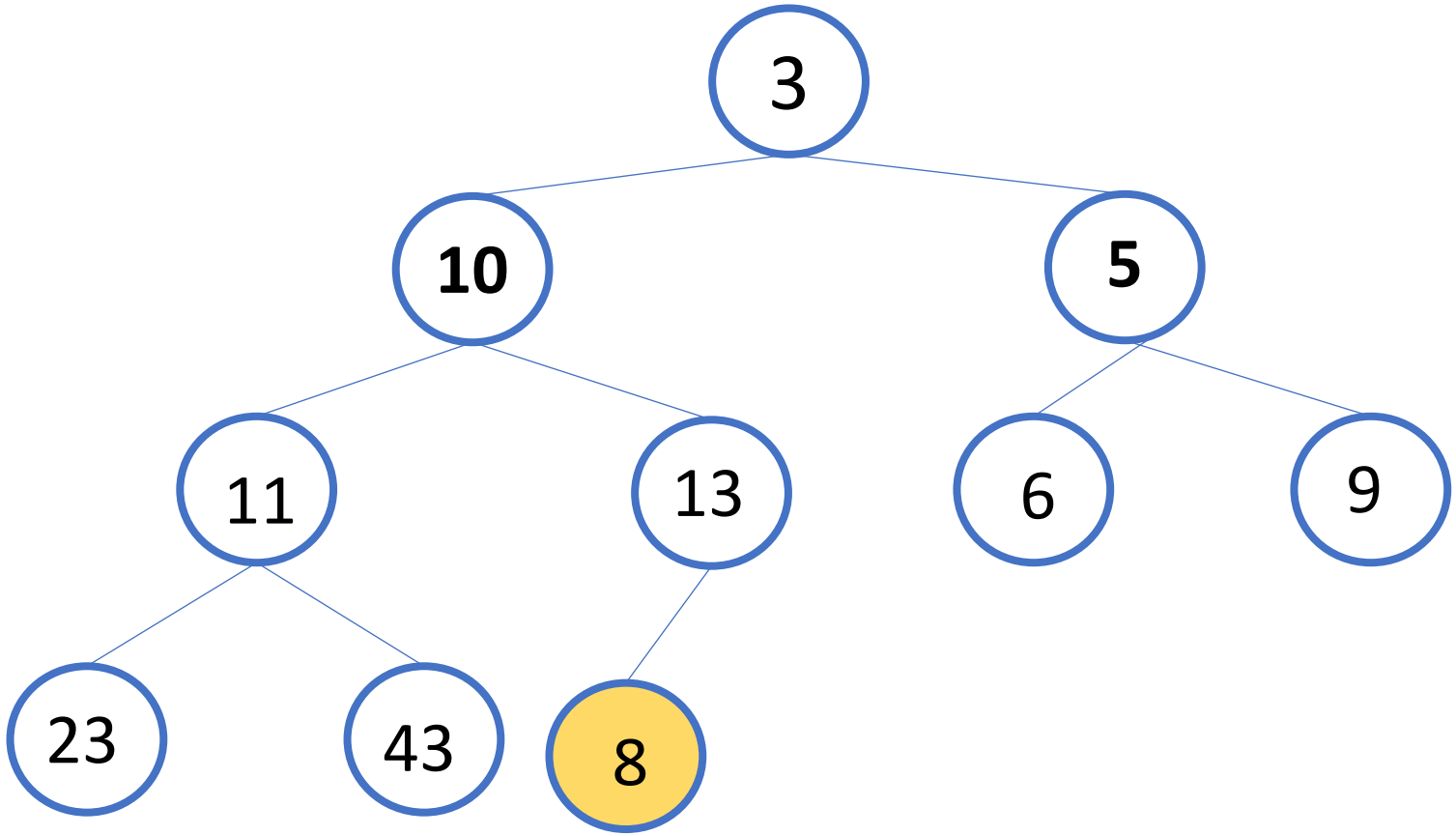
Binary heap

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



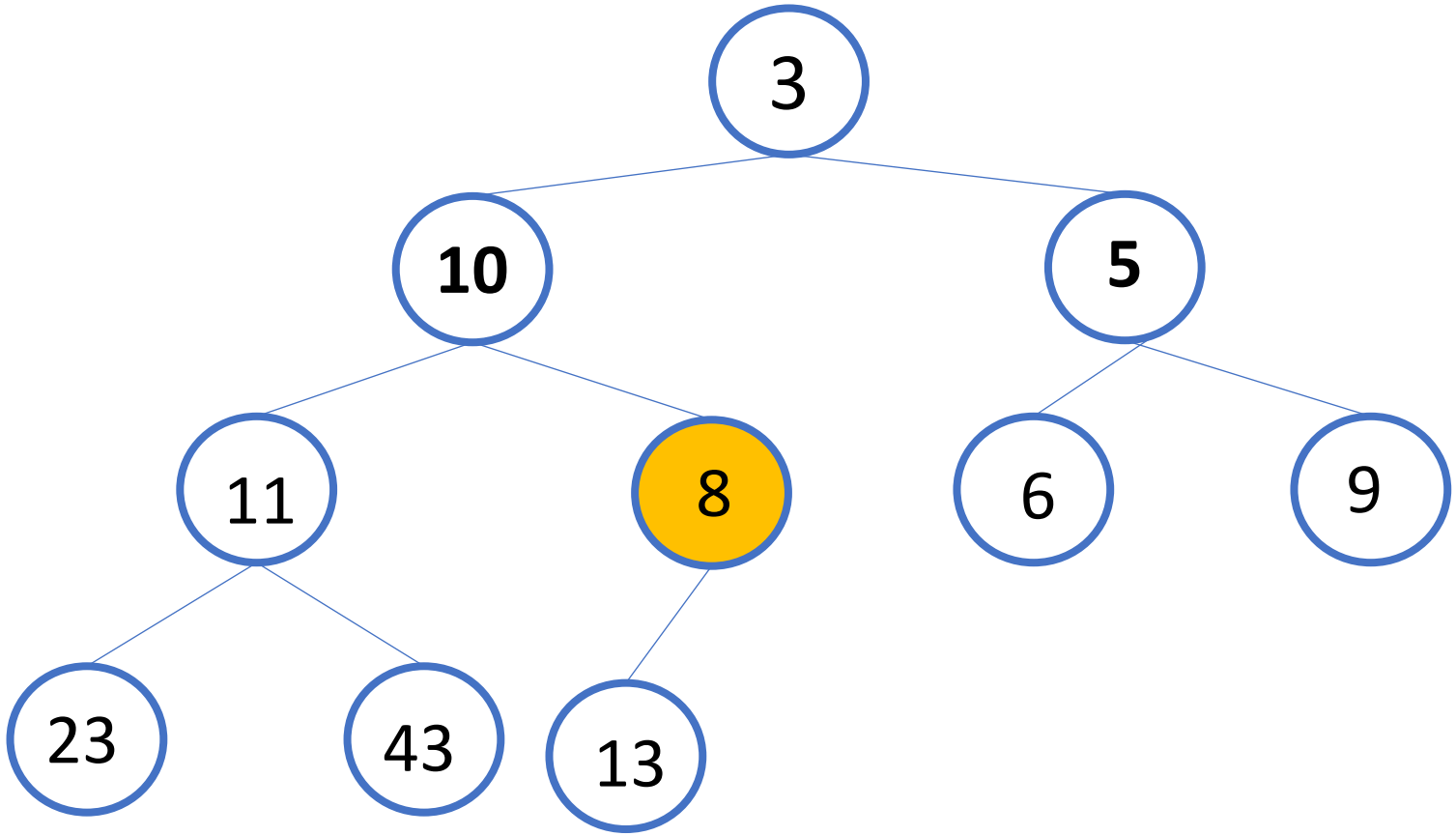
Binary heap

- **Insert:**



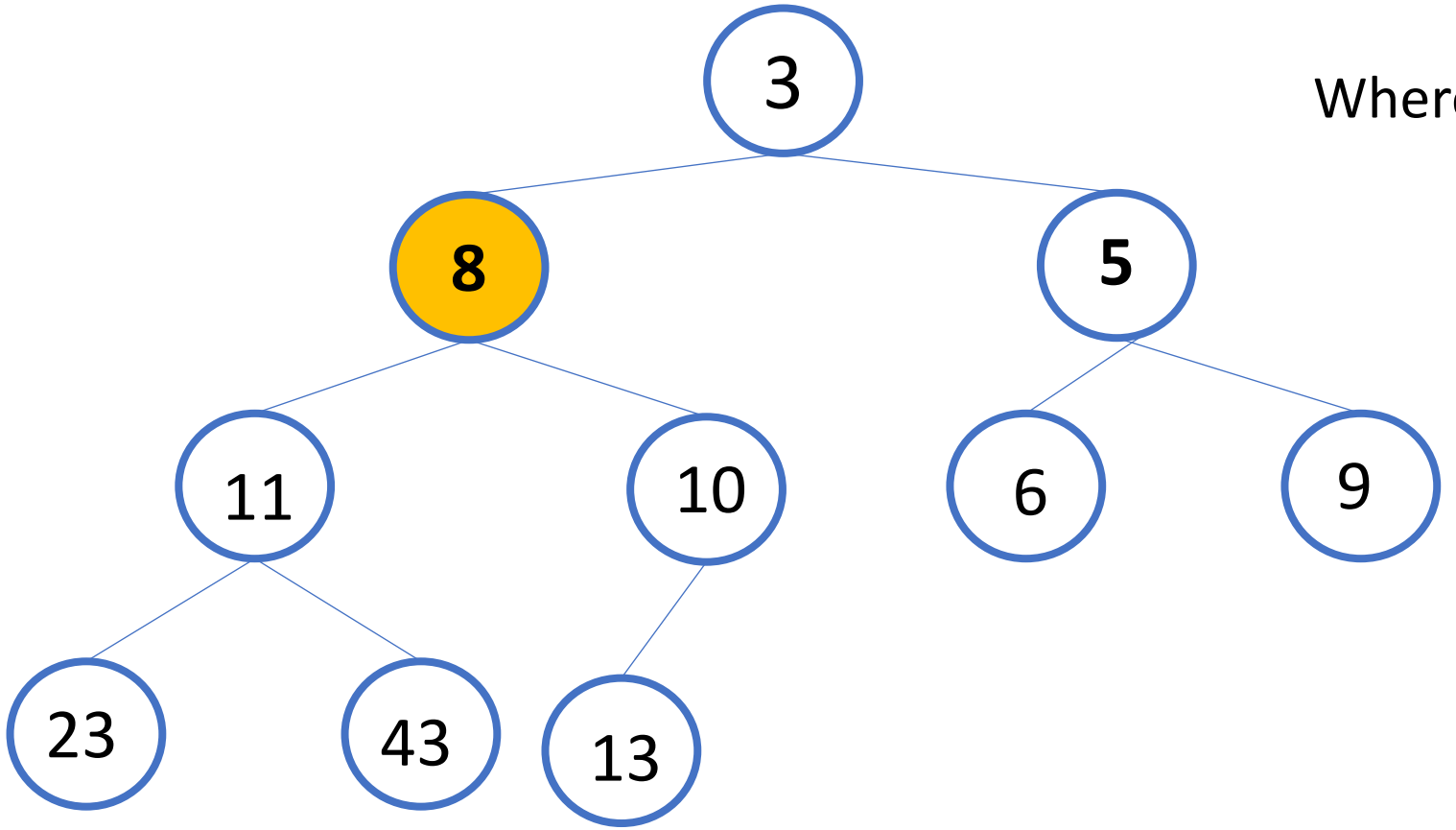
Binary heap

- **Insert:**



Binary heap

- **Insert:**



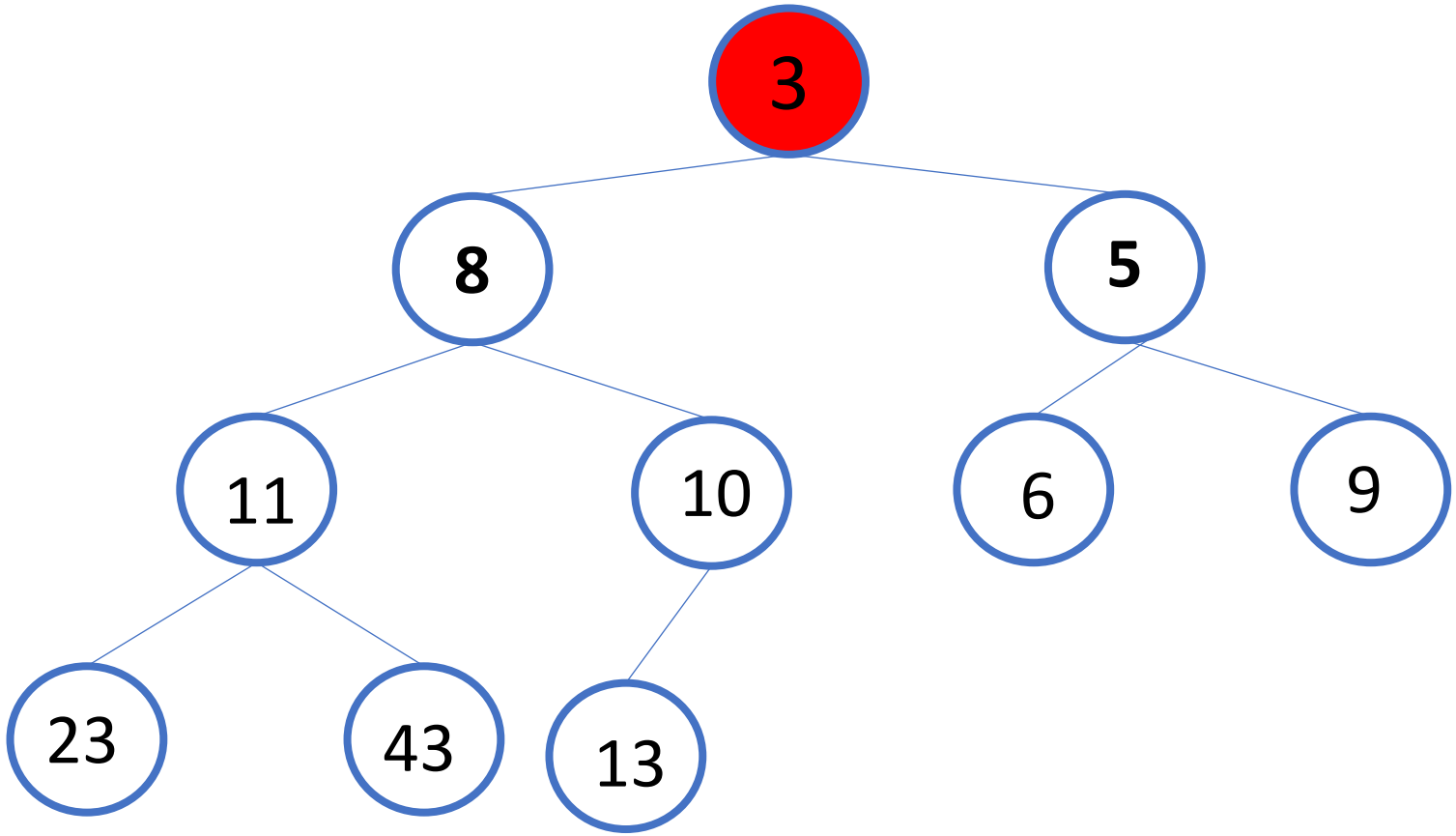
Insert time:

O(logn)

Where n is the size of the heap

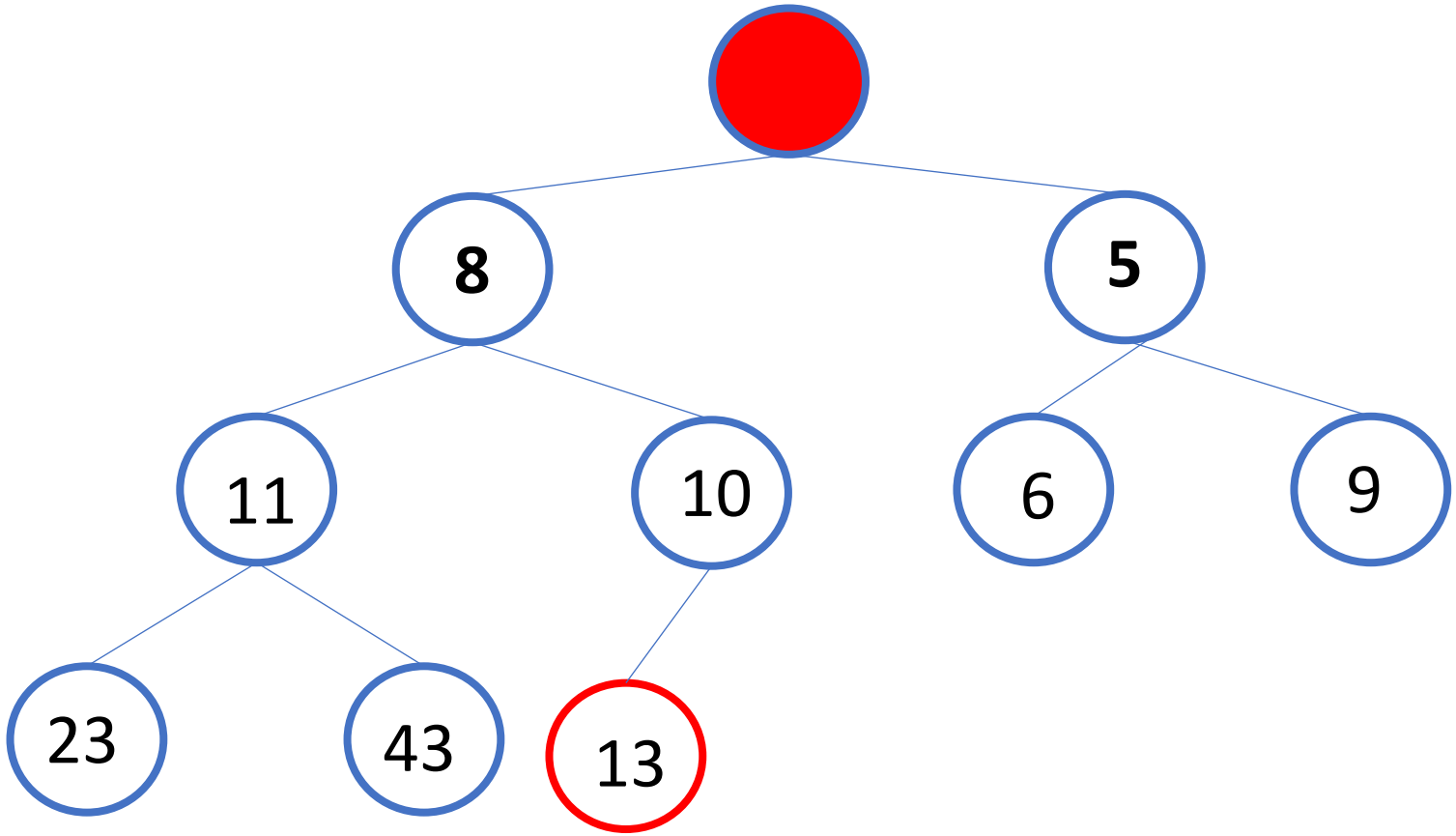
Binary heap

- **Extractmin:**



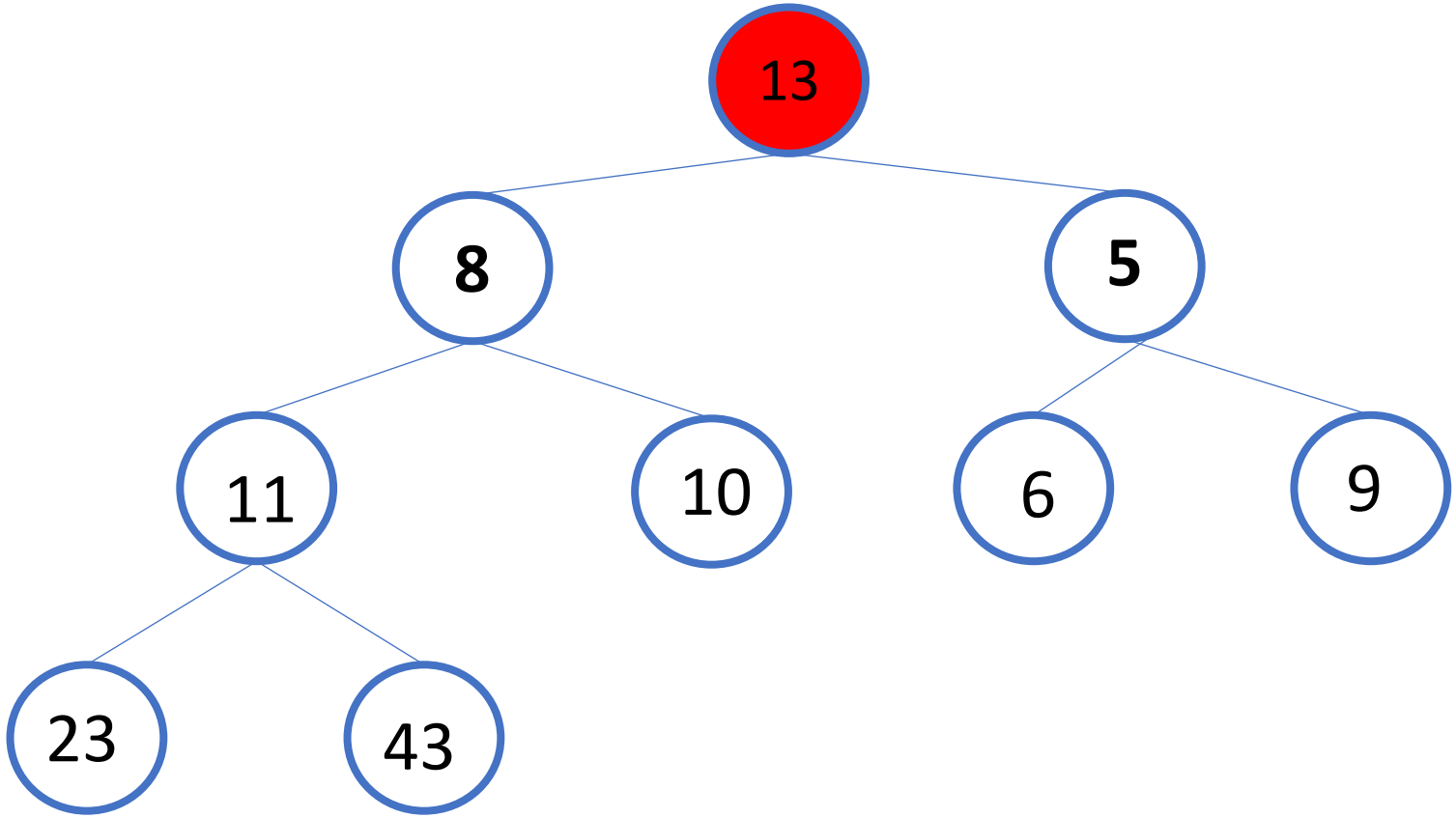
Binary heap

- **Extractmin:**



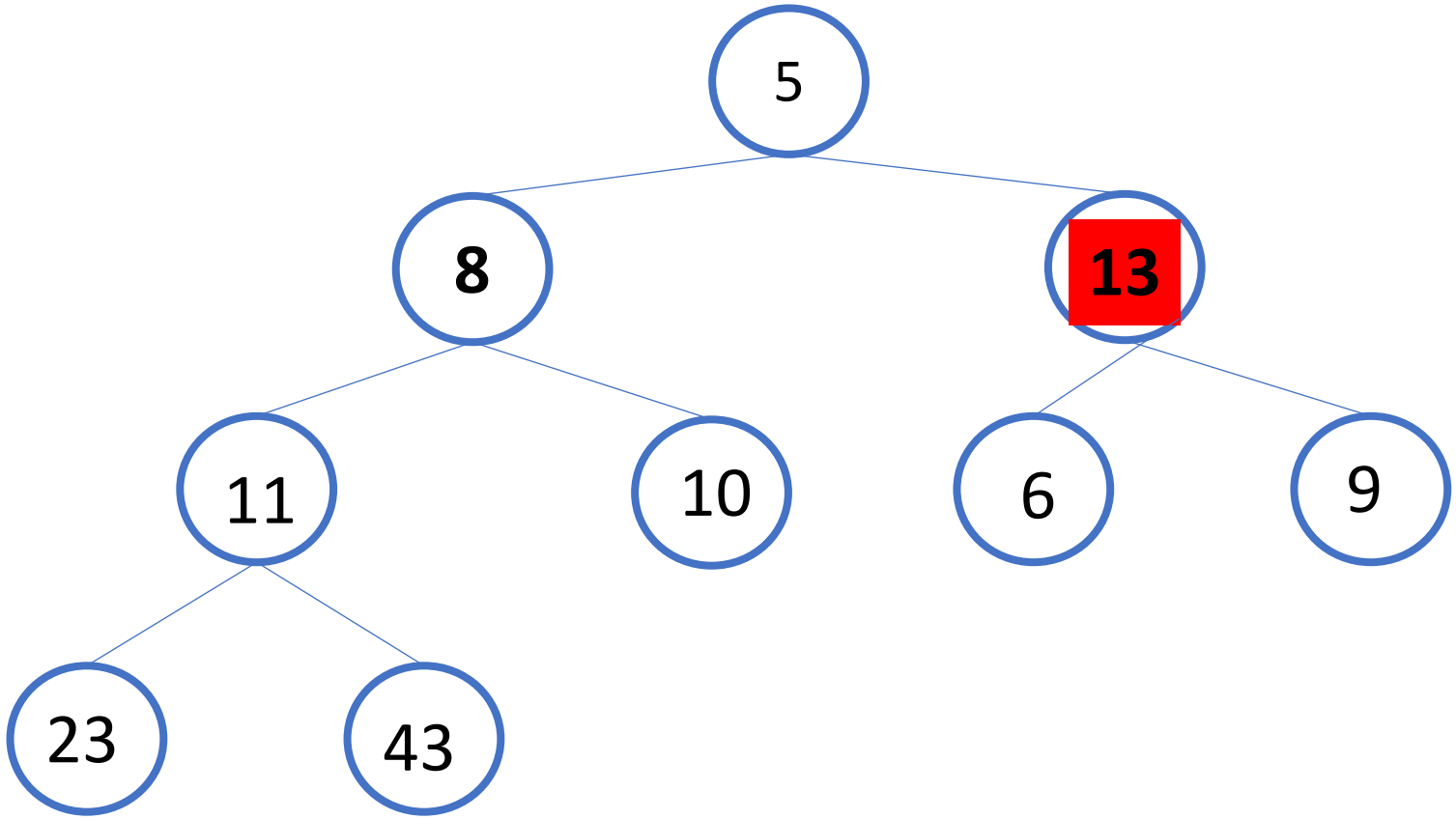
Binary heap

- **Extractmin:**



Binary heap

- **Extractmin:**



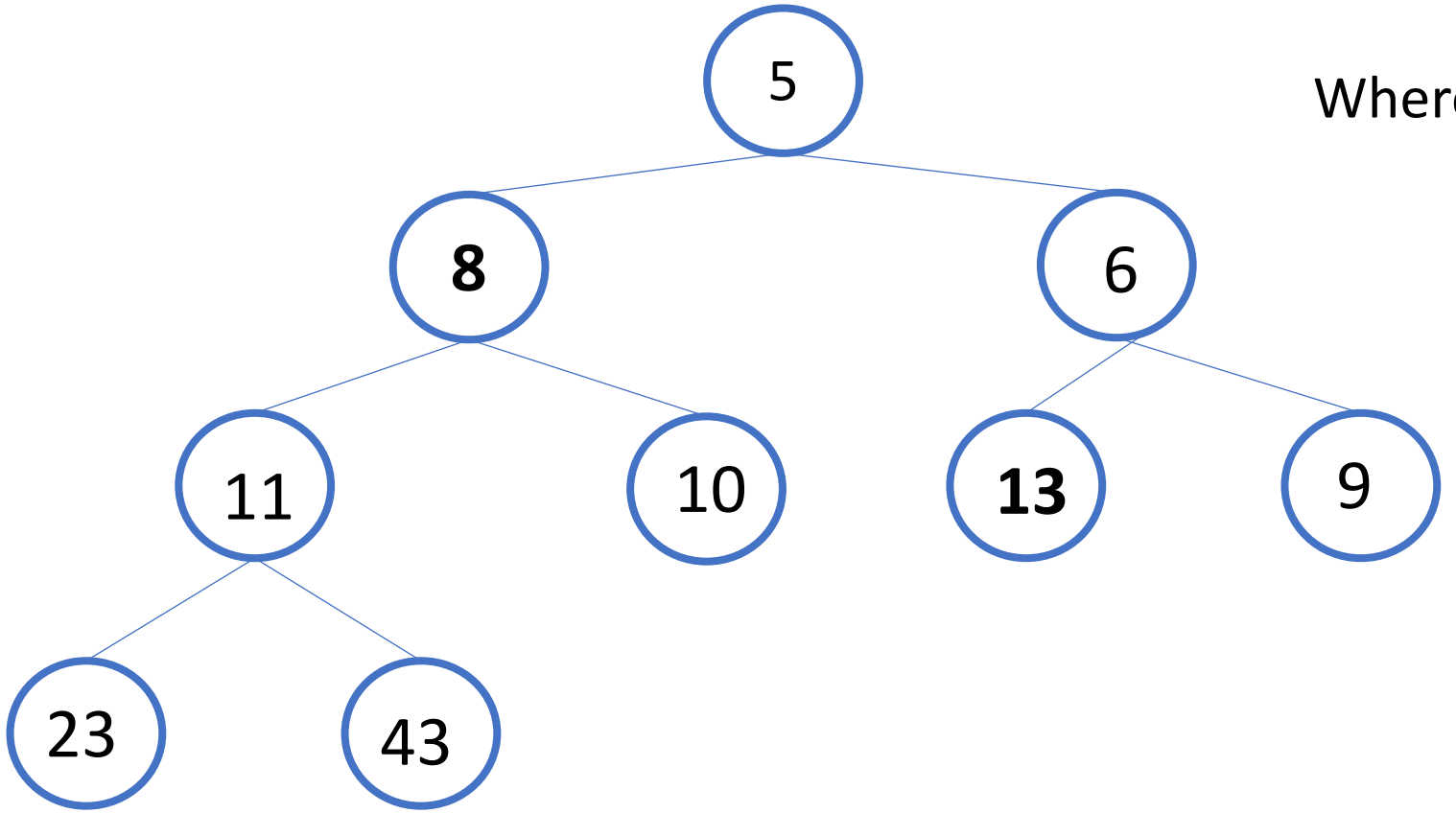
Binary heap

- **Extractmin:**

time:

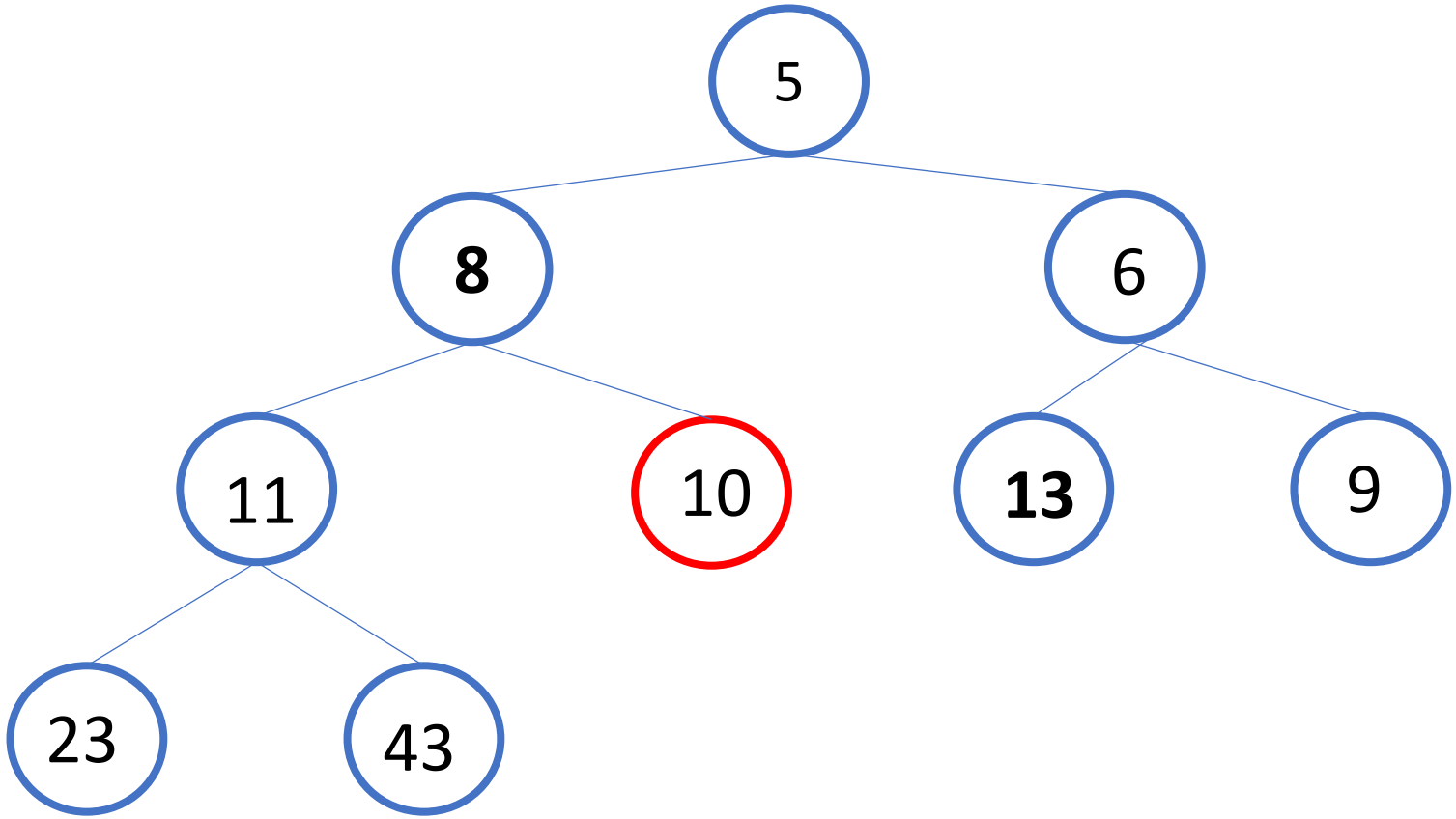
O(logn)

Where n is the size of the heap



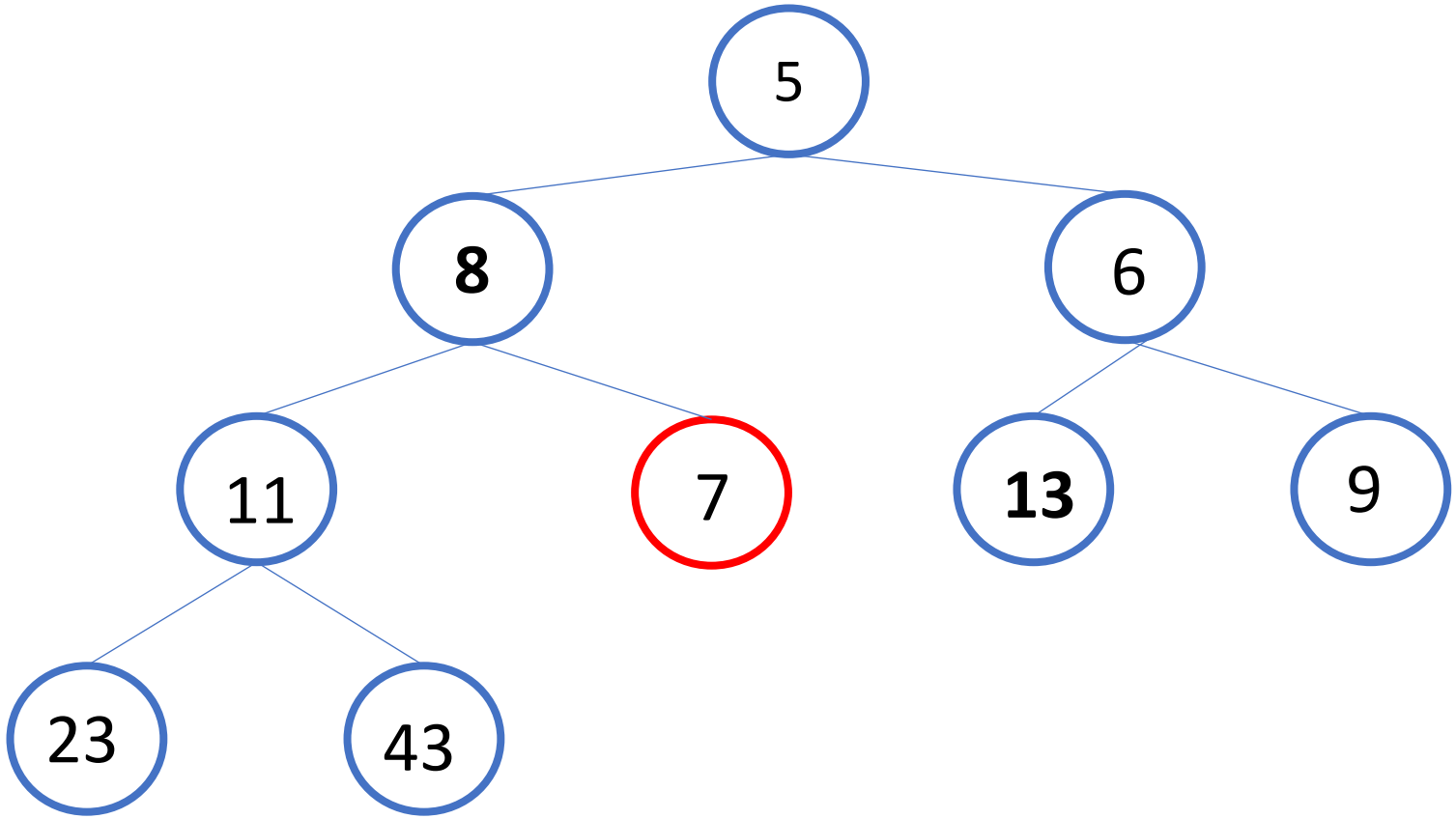
Binary heap

- **Decrease Key operation:**



Binary heap

- **Decrease Key operation:**



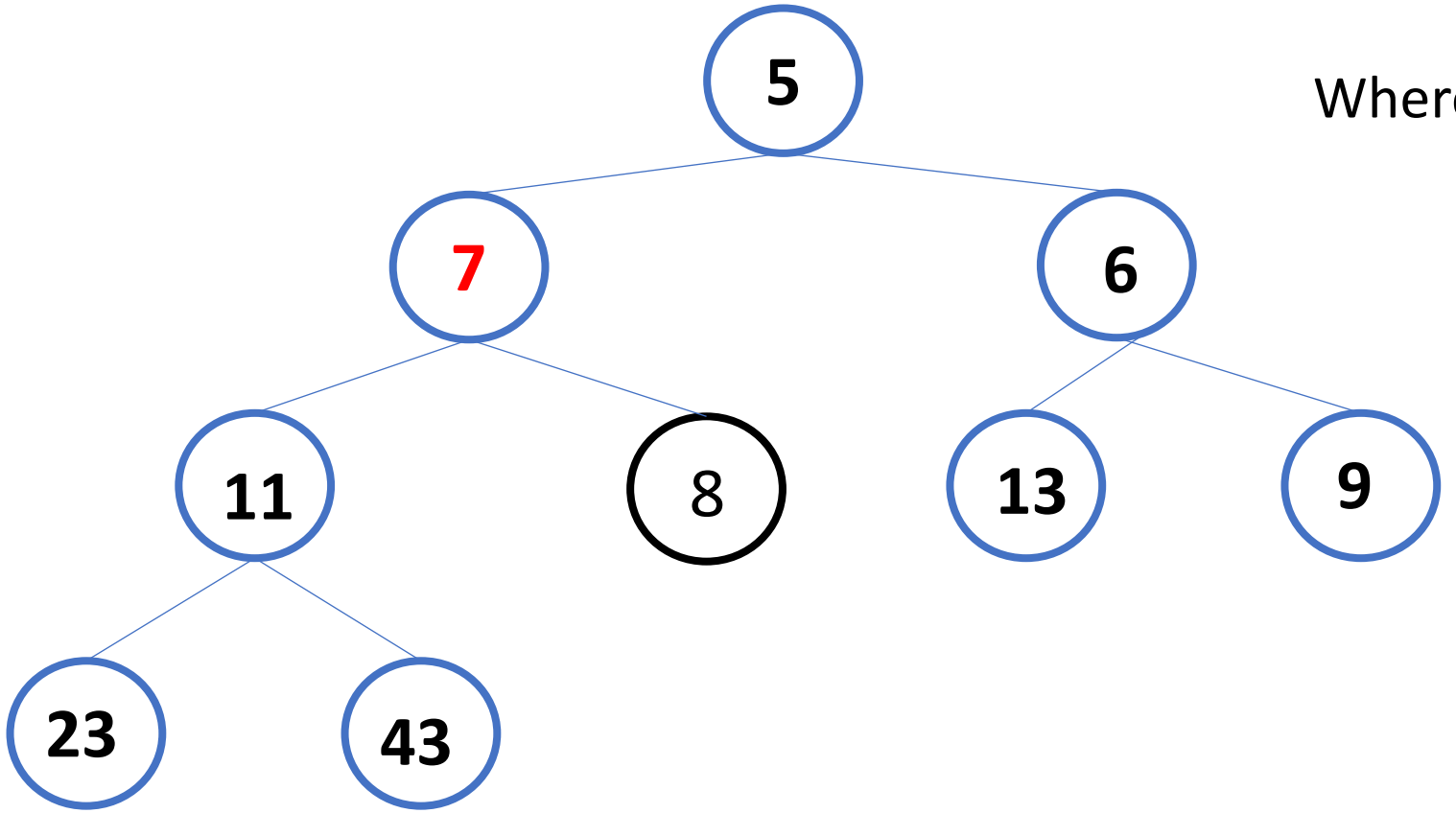
Binary heap

- **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 = ∅ // 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 ≠ ∅ :

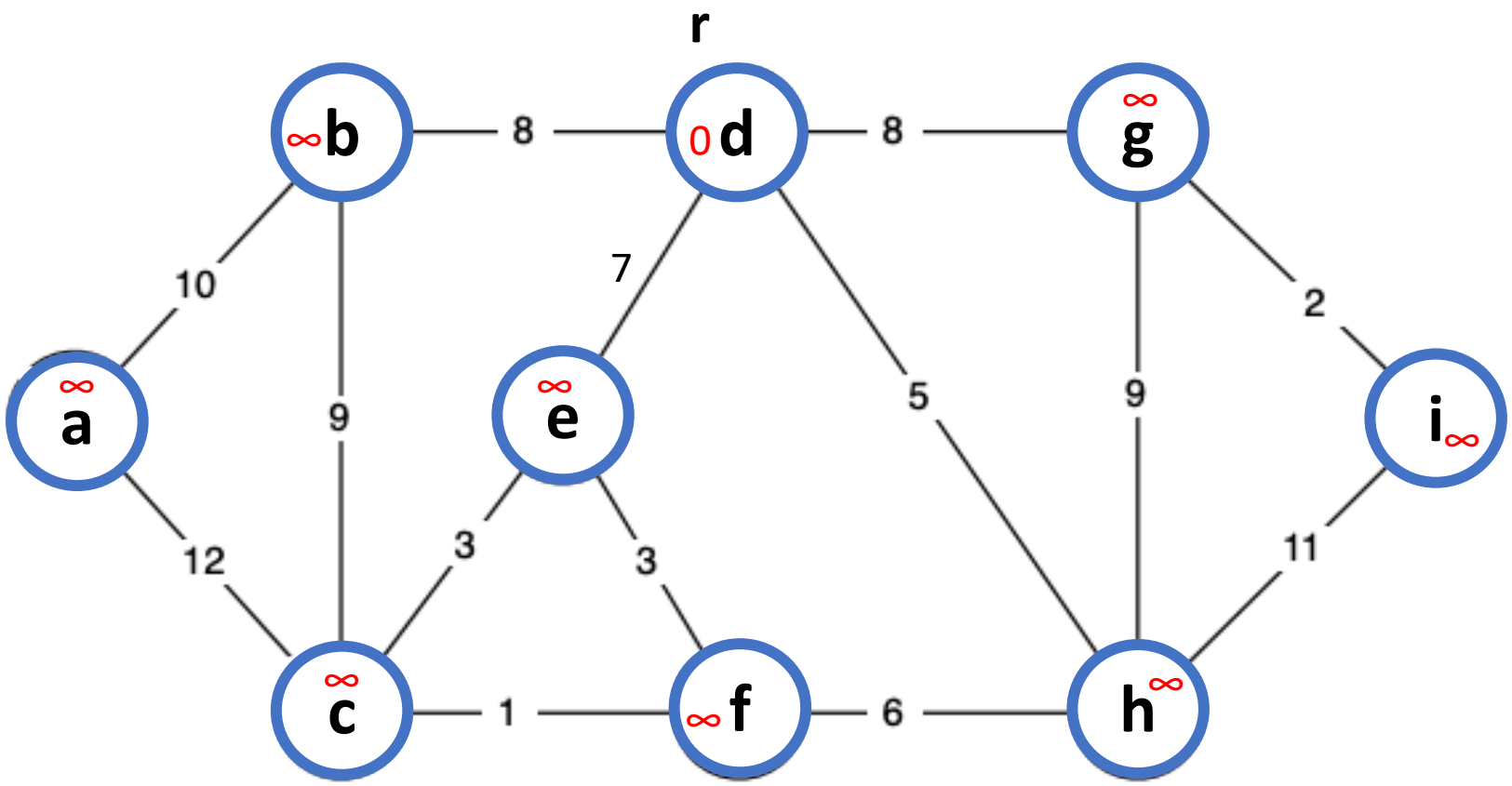
Do u ← Extract-MIN(Q)

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

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 ≠ ∅ :

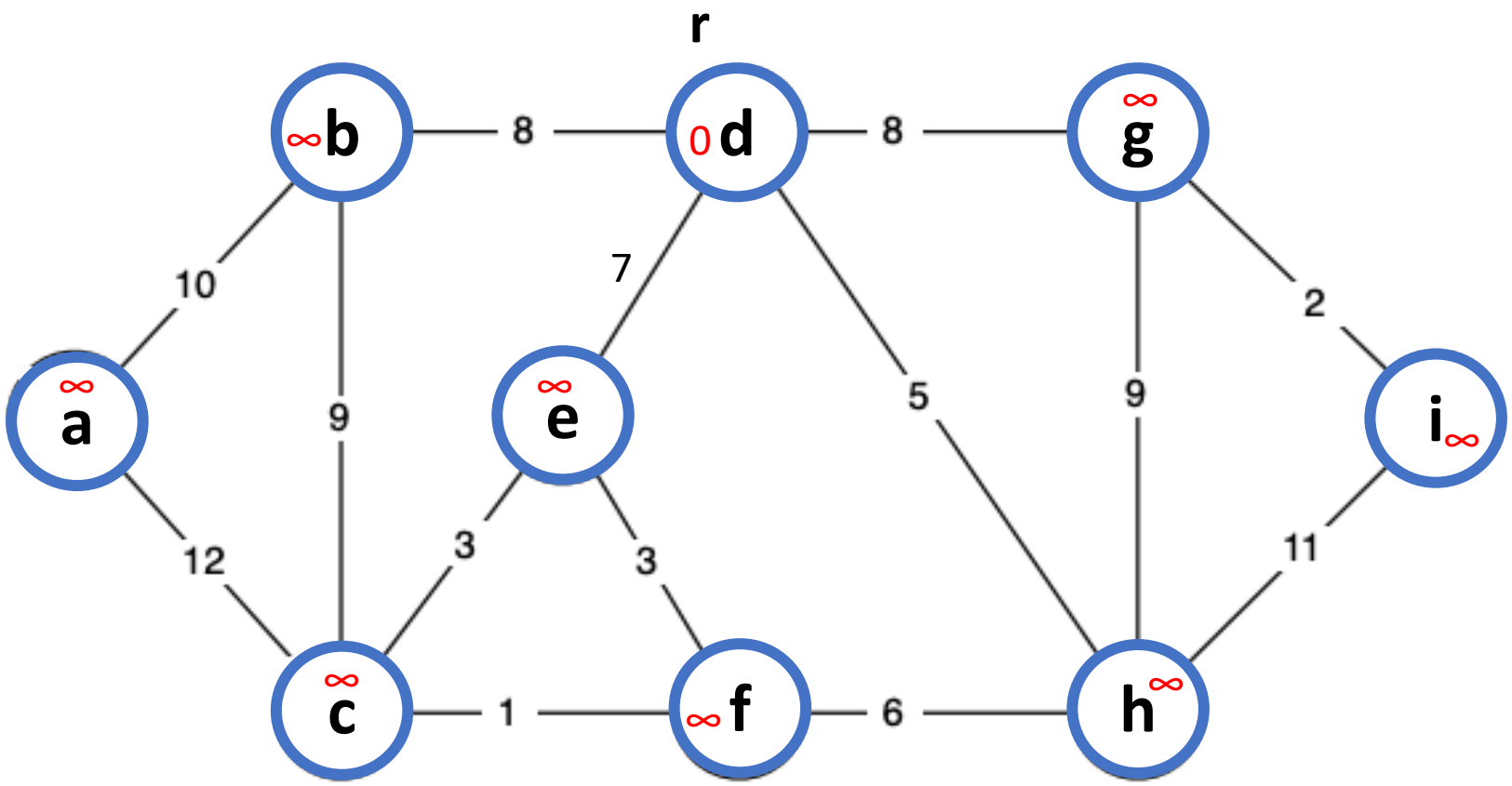
Do u ← Extract-MIN(Q)

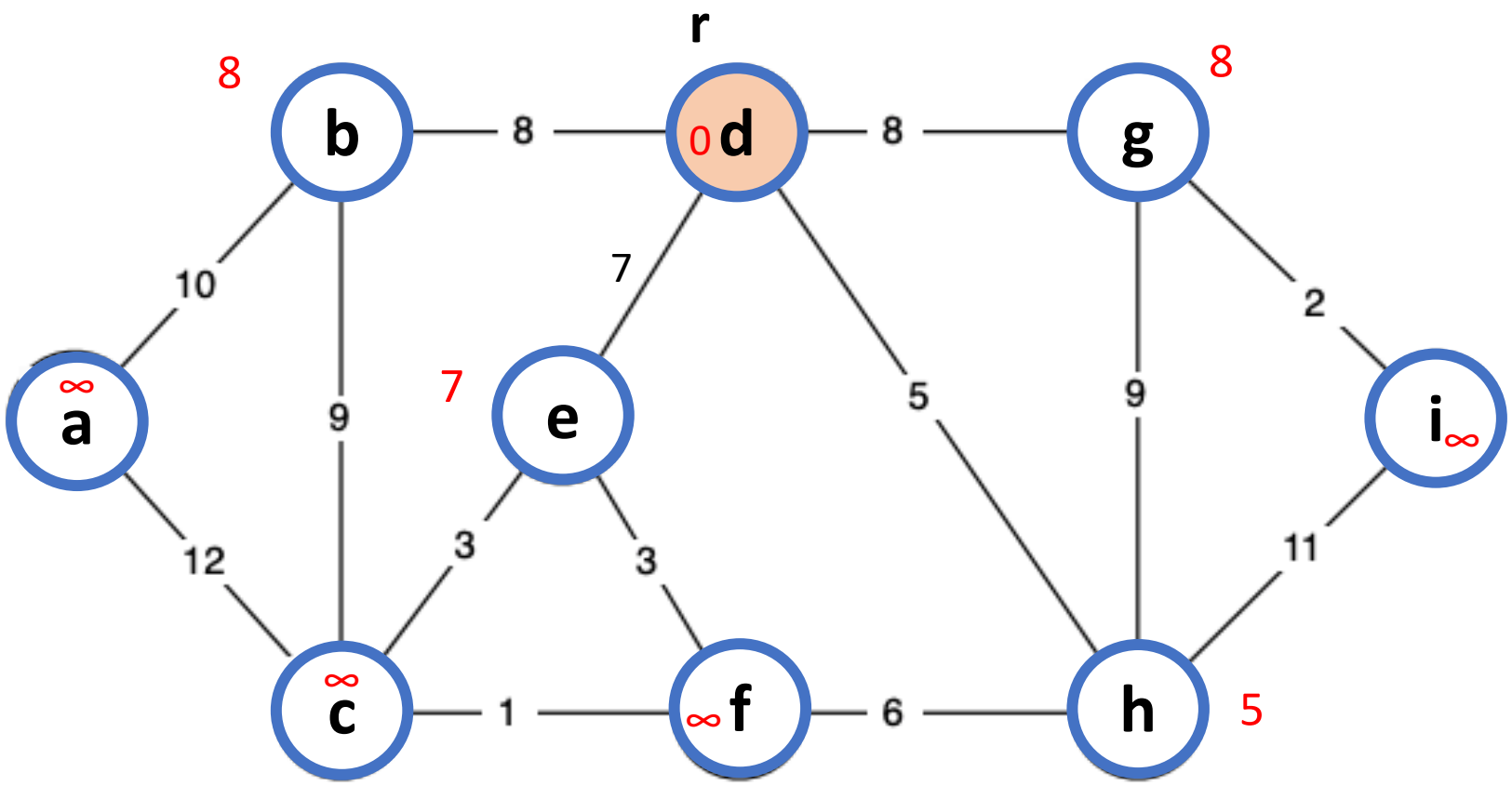
for each v ∈ Adj(u):

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


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 = ∅ // 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 ≠ ∅ :

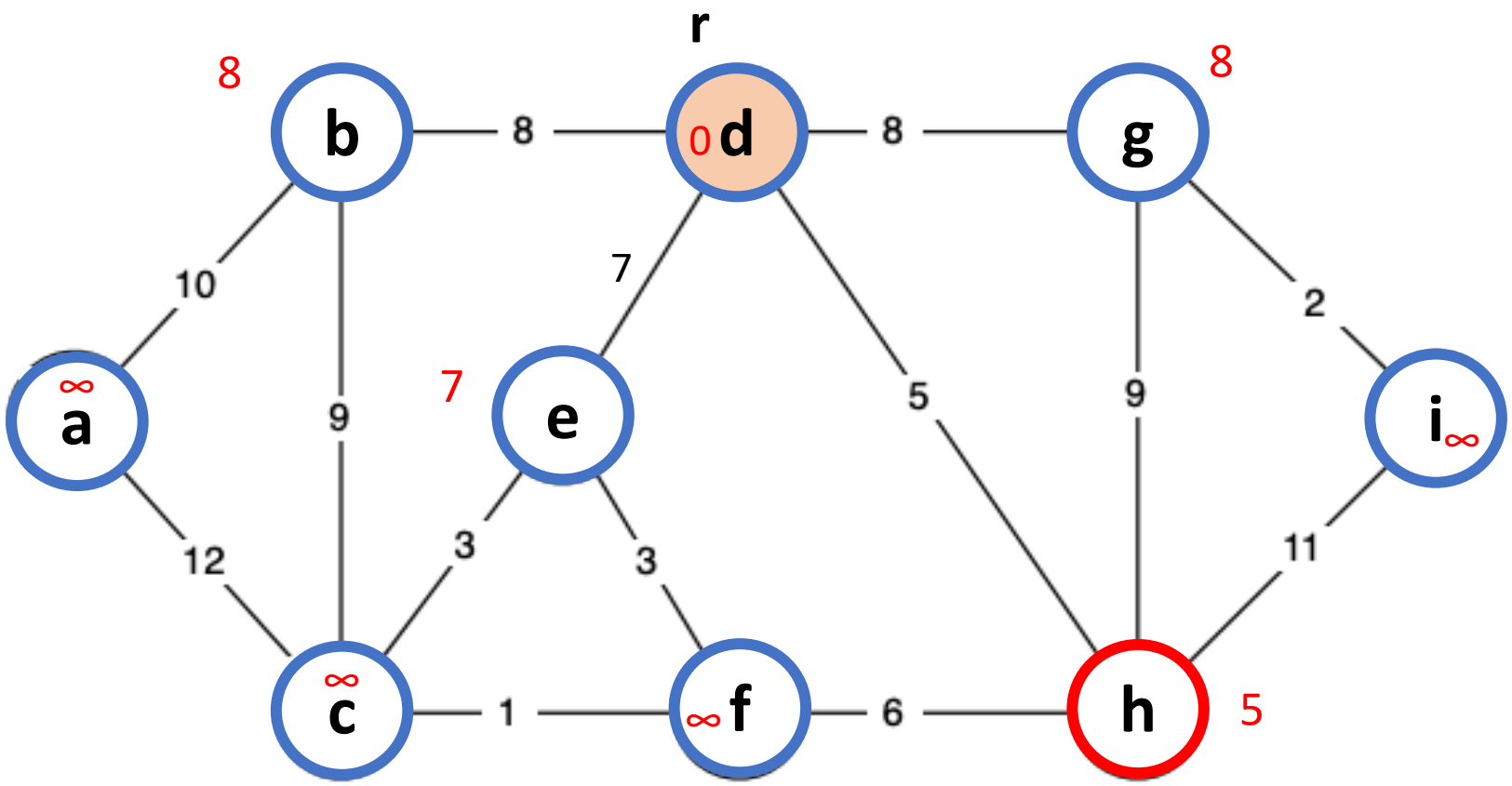
Do u ← Extract-MIN(Q)

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
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 = ∅ // 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 ≠ ∅ :

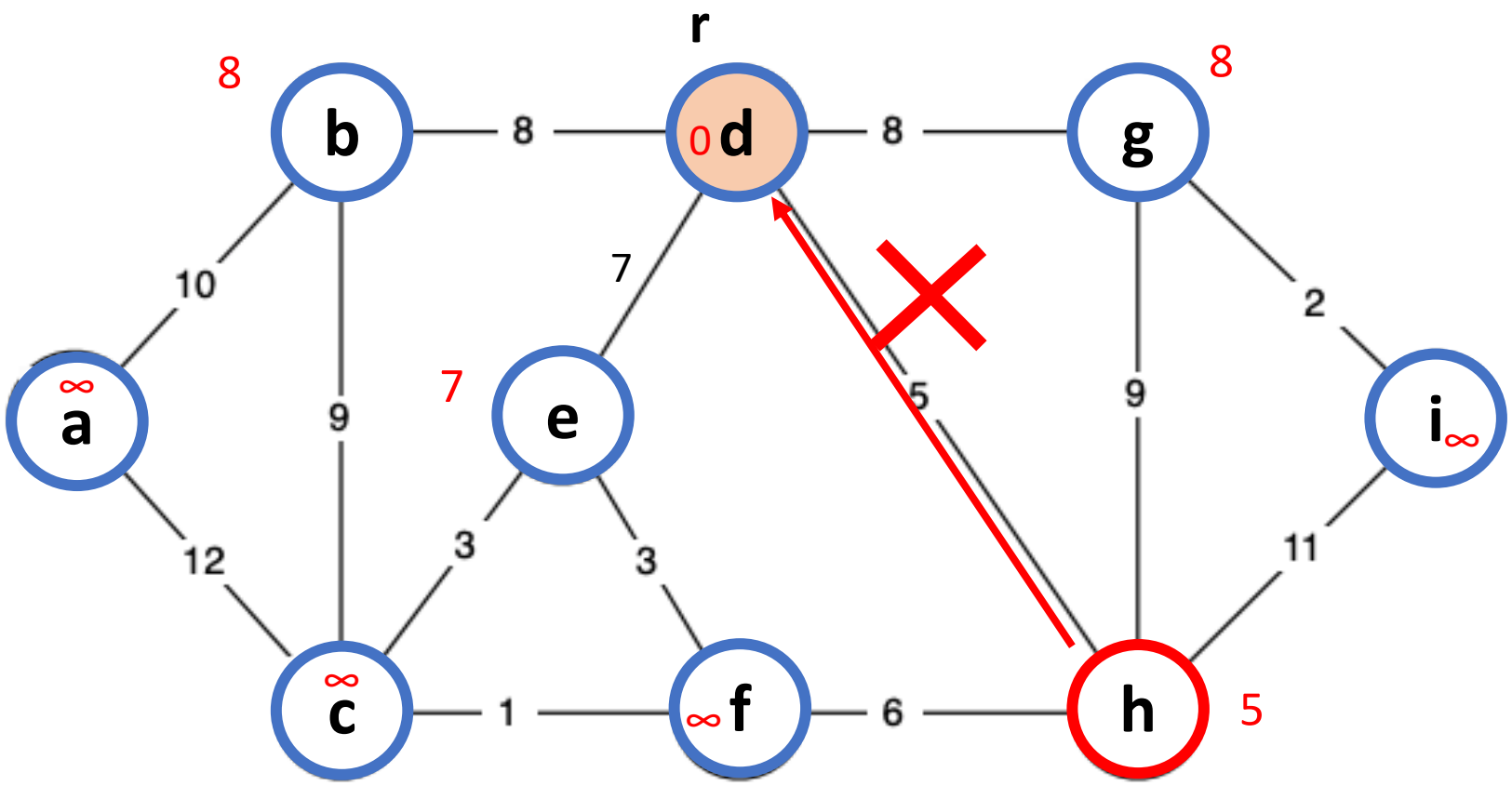
    Do u ← Extract-MIN(Q)

        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
8
8
8
b
8
0 d
8
g
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 = ∅ // 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 ≠ ∅ :

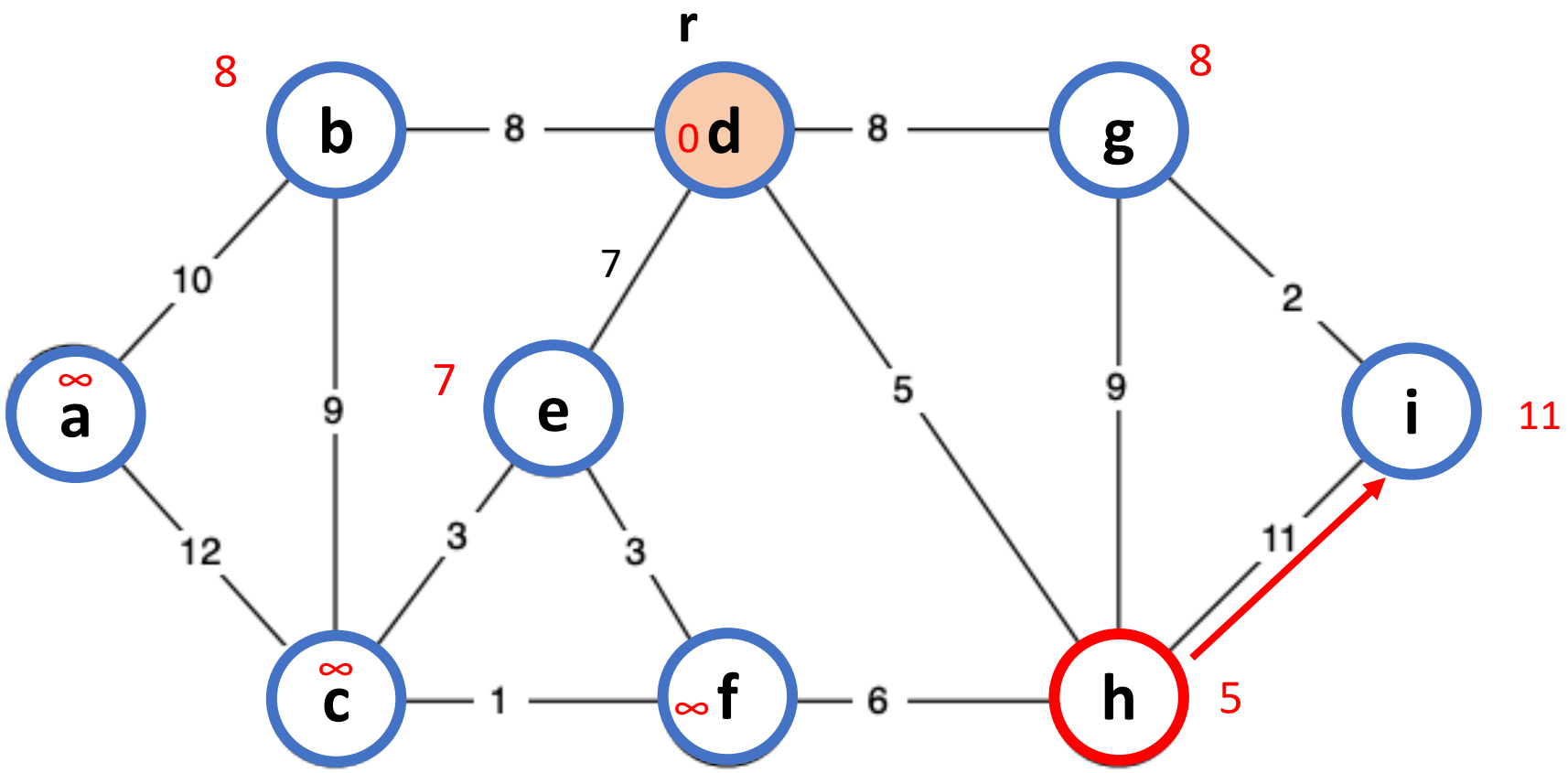
Do u ← Extract-MIN(Q)

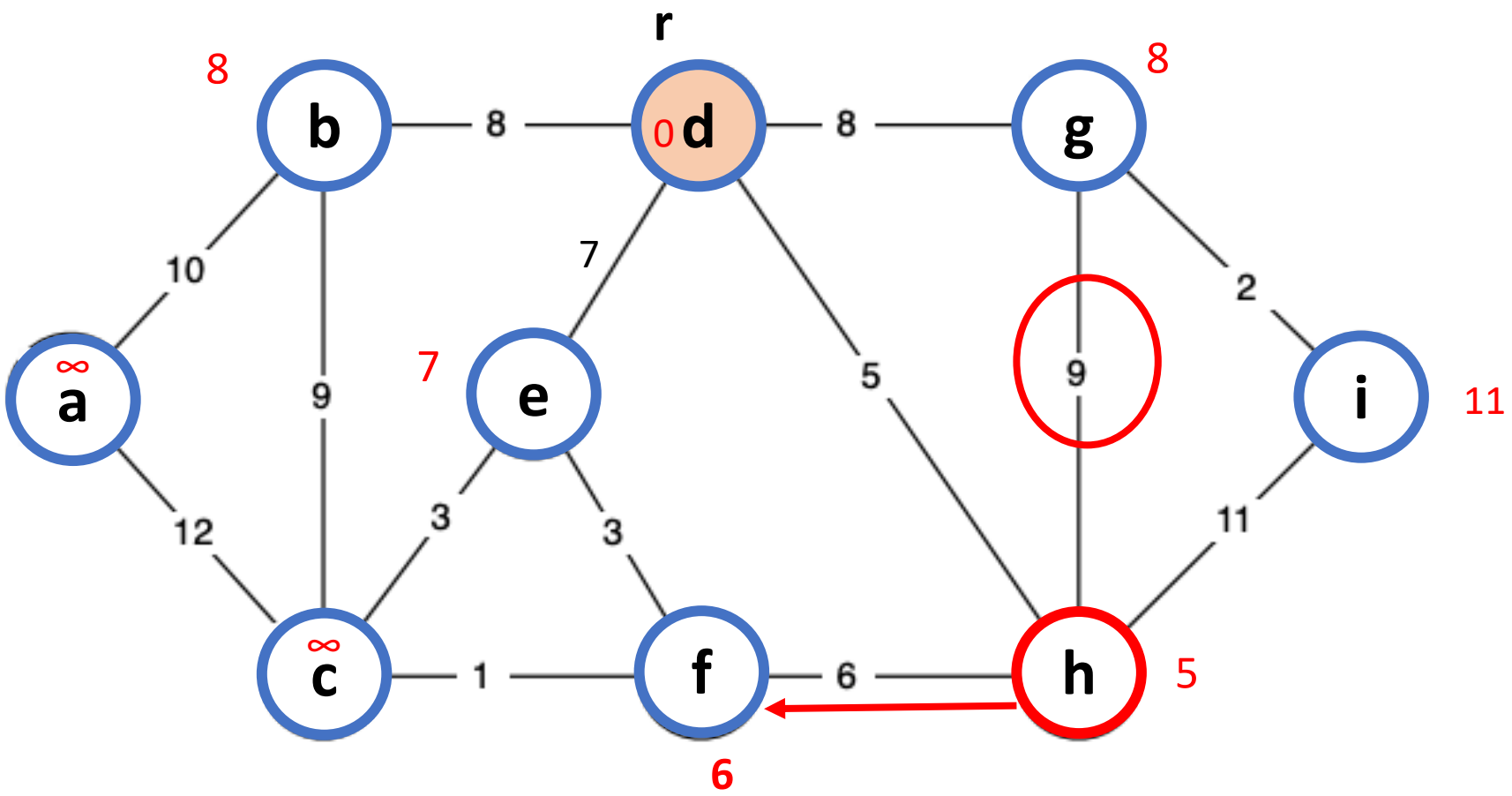
for each v ∈ Adj(u):

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


r
8
b
8
0 d
8
g
8
10
7
2
∞
a
9
7
e
5
9
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 ≠ ∅ :

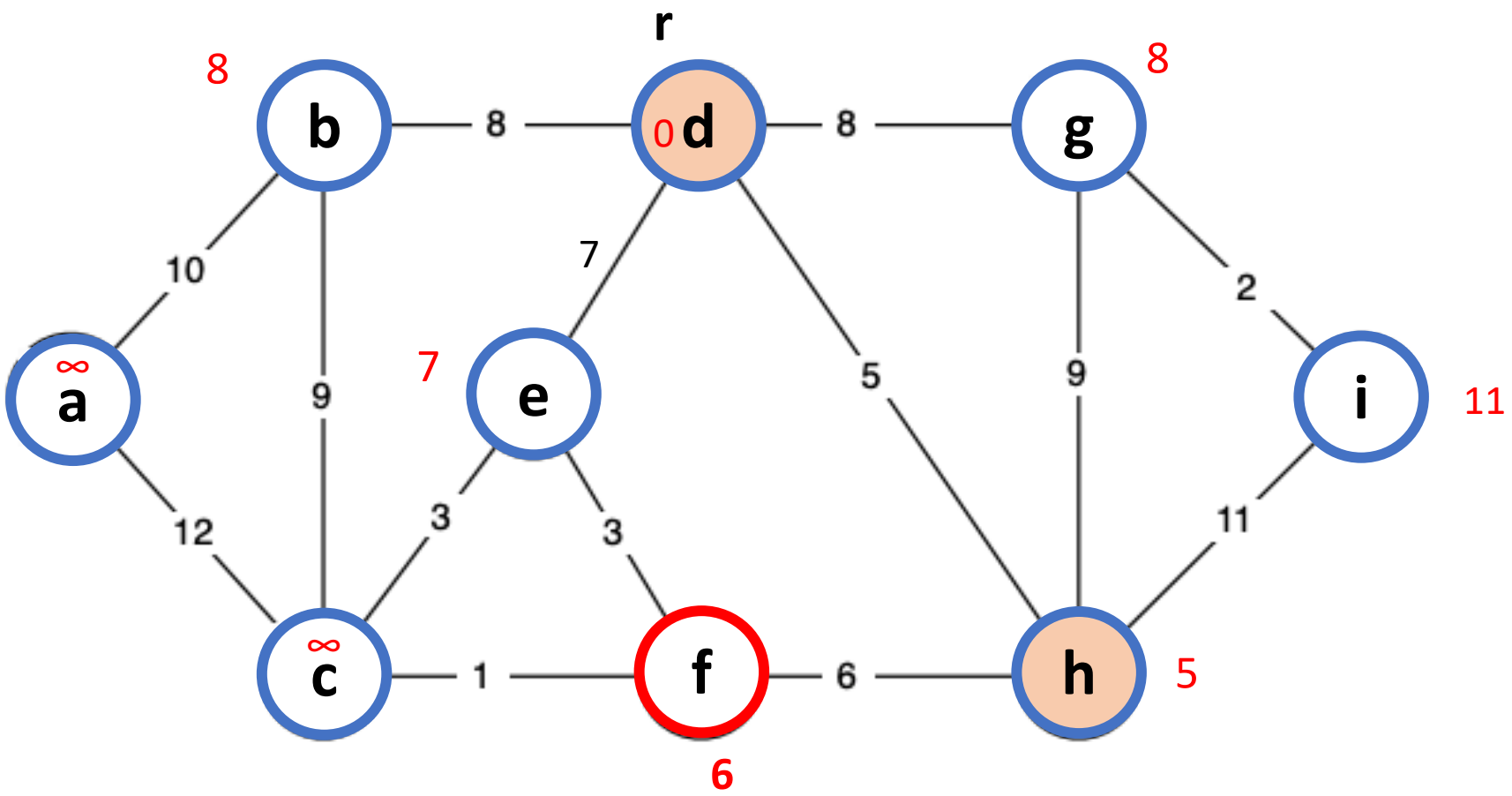
Do u ← Extract-MIN(Q)

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
8
b
8
0 d
8
g
8
10
7
2
∞
a
9
7
e
5
9
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 ≠ ∅ :

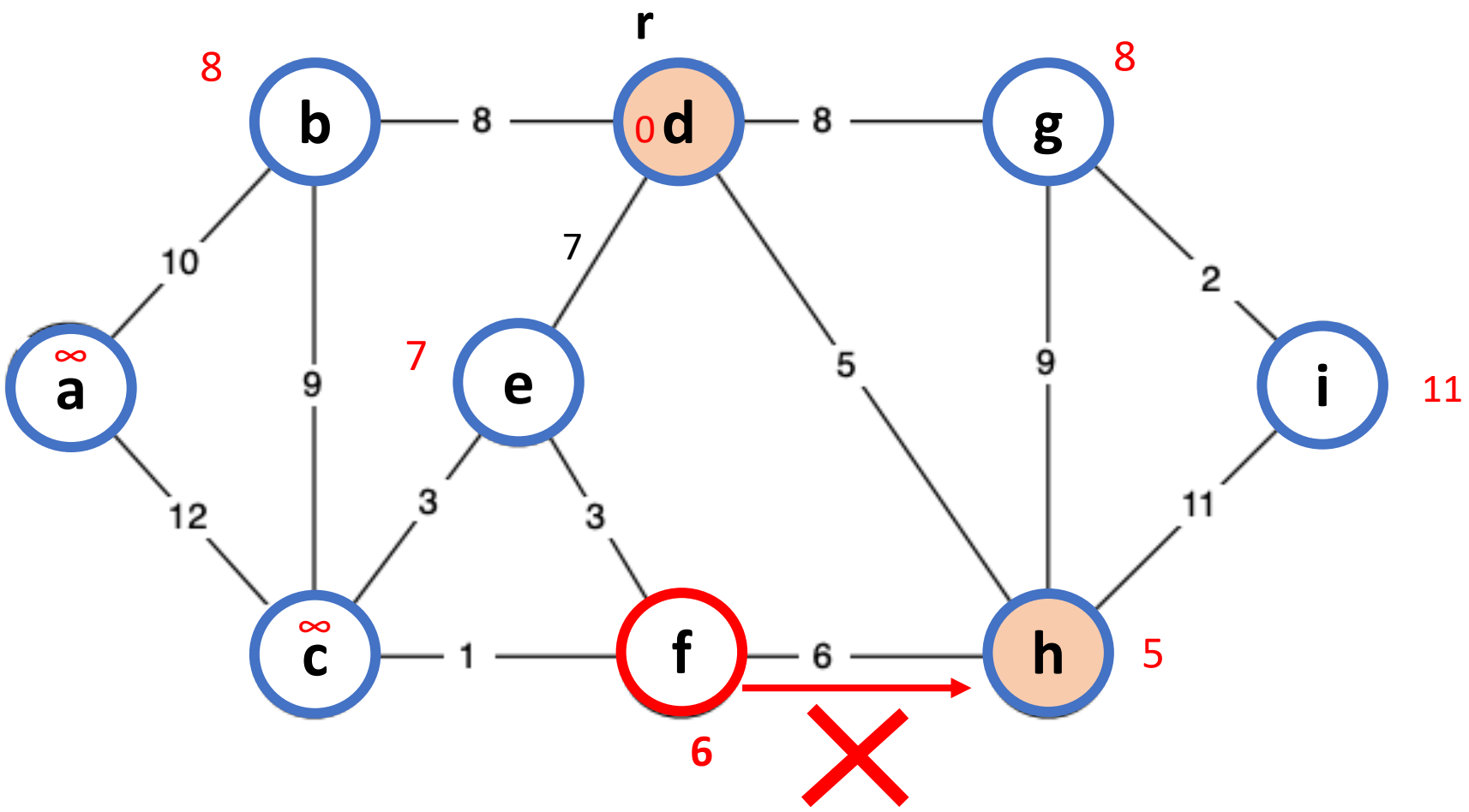
Do u ← Extract-MIN(Q)

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
8
b
8
0 d
8
g
8
10
7
2
∞
a
9
7
e
5
9
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 ≠ ∅ :

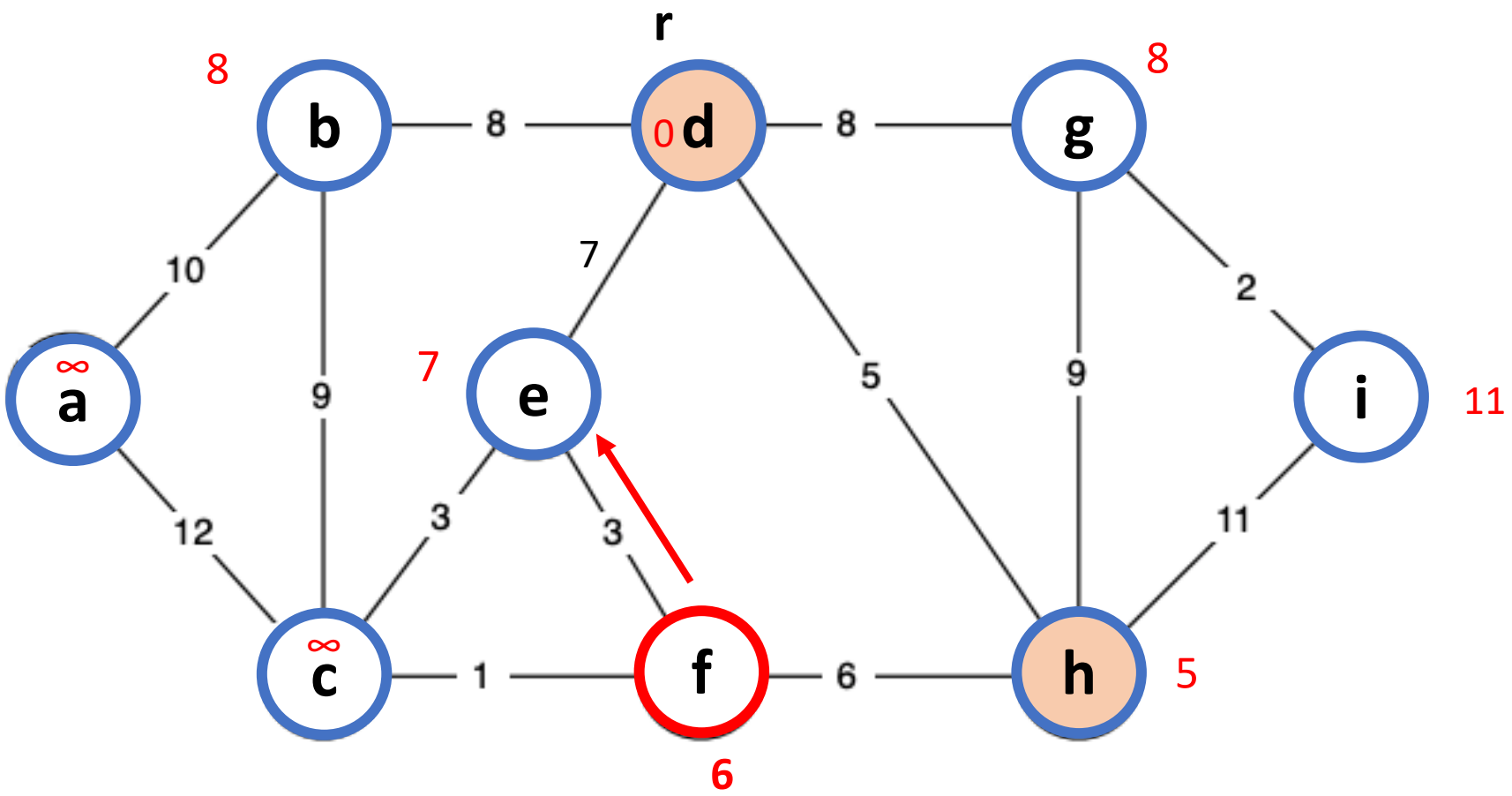
Do u ← Extract-MIN(Q)

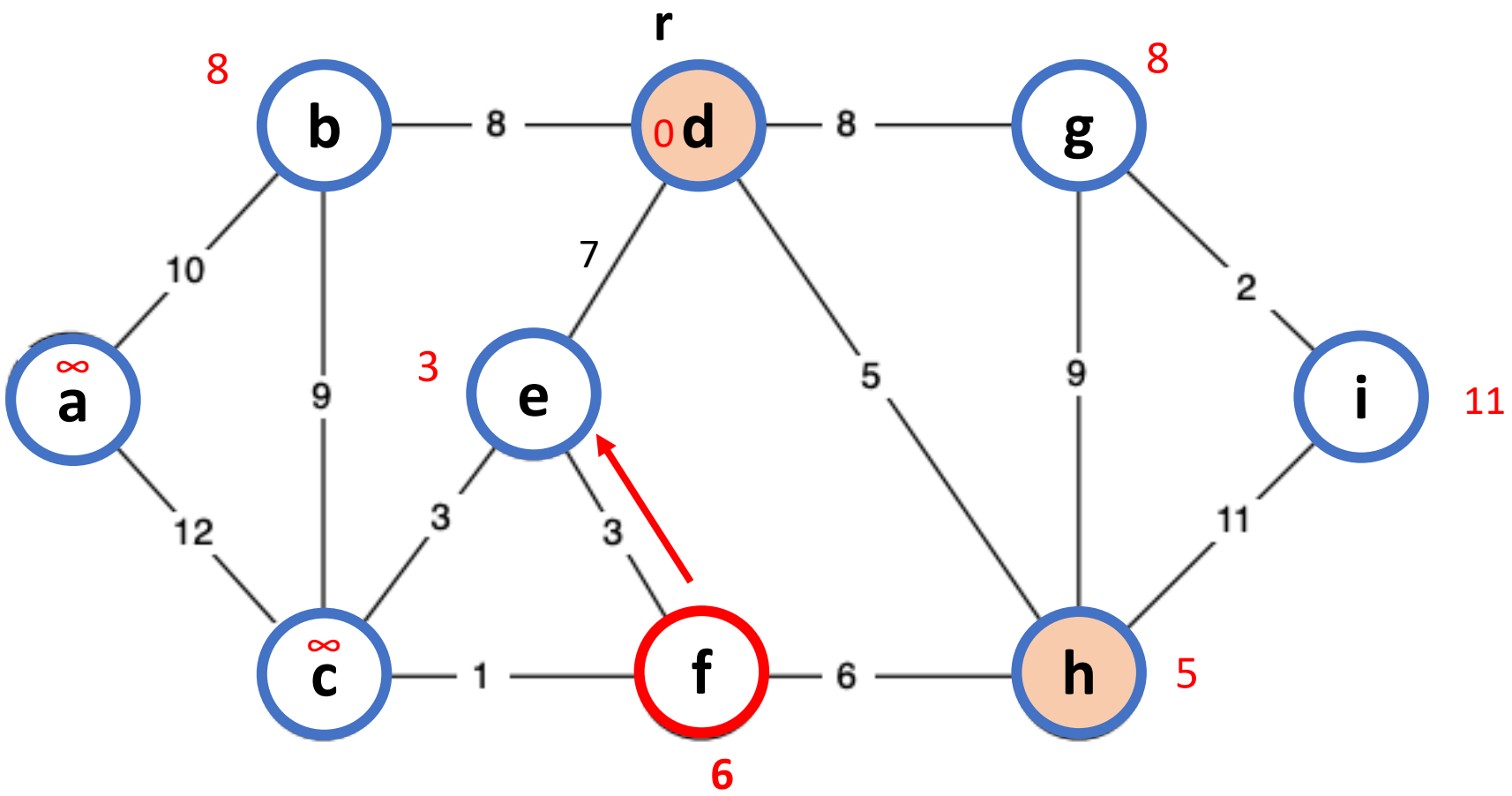
for each v ∈ Adj(u):

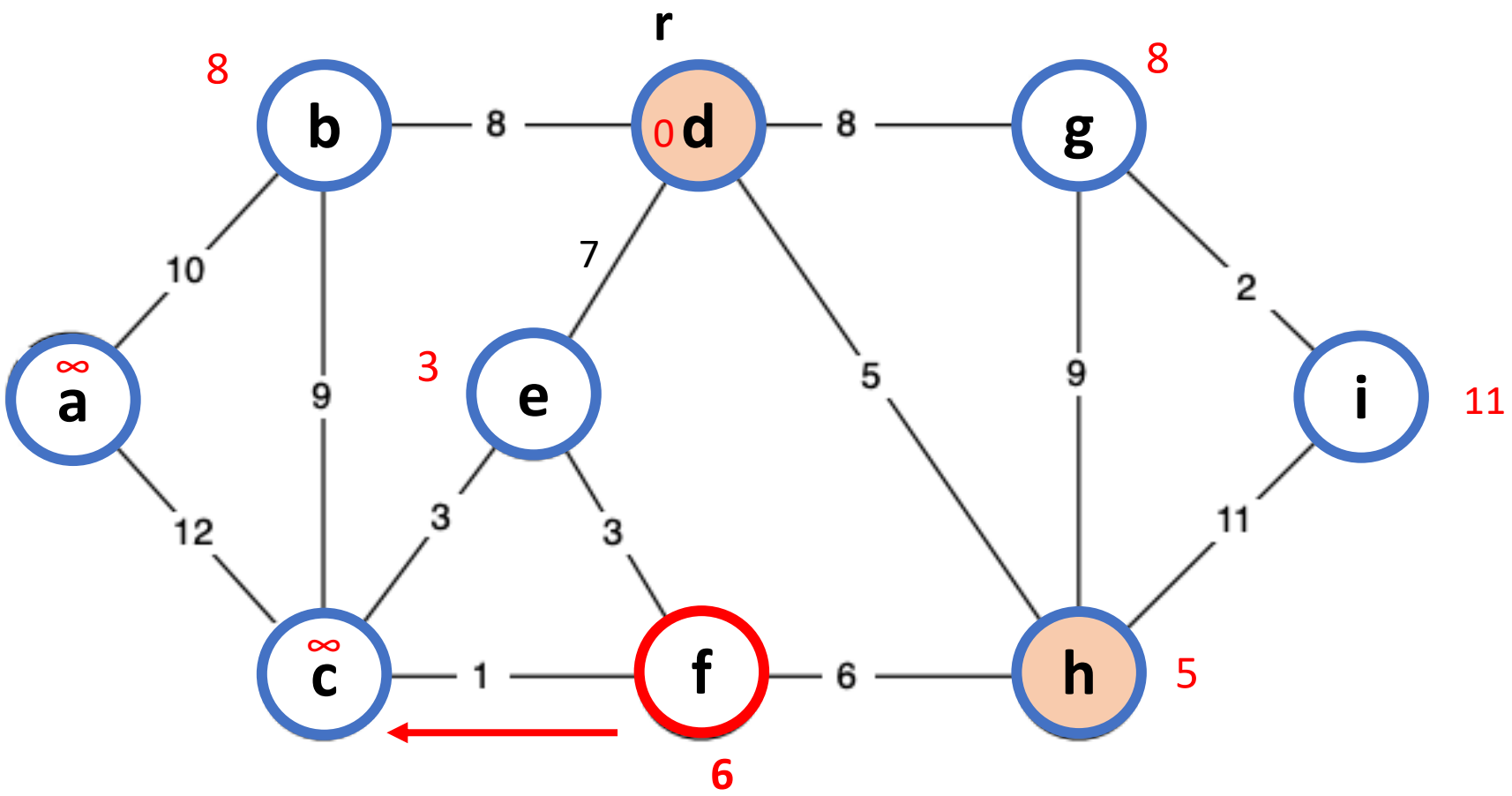
do if v ∈ Q and w(u,v) < k_v:

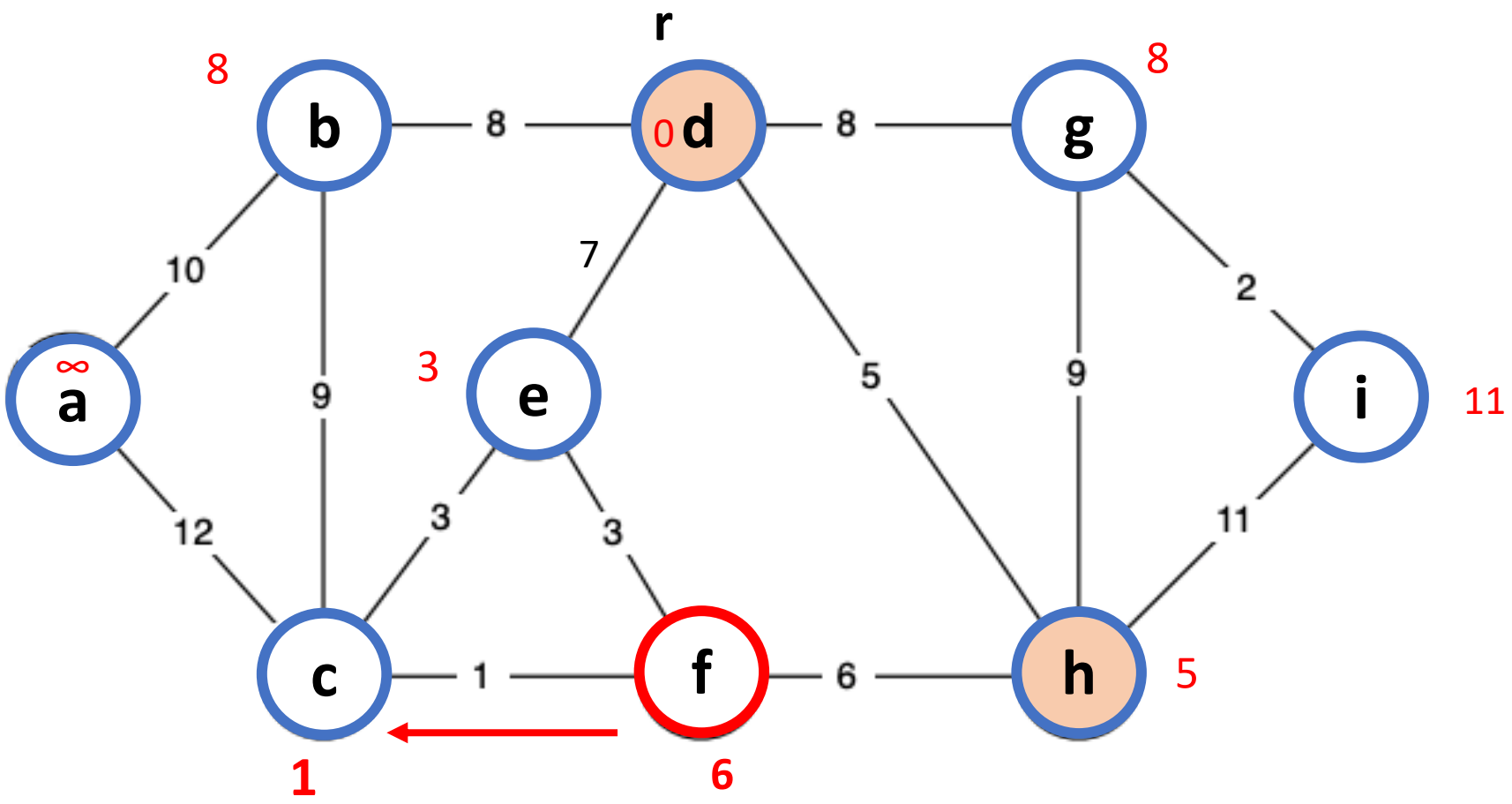
then $\prod_v \leftarrow$ u

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

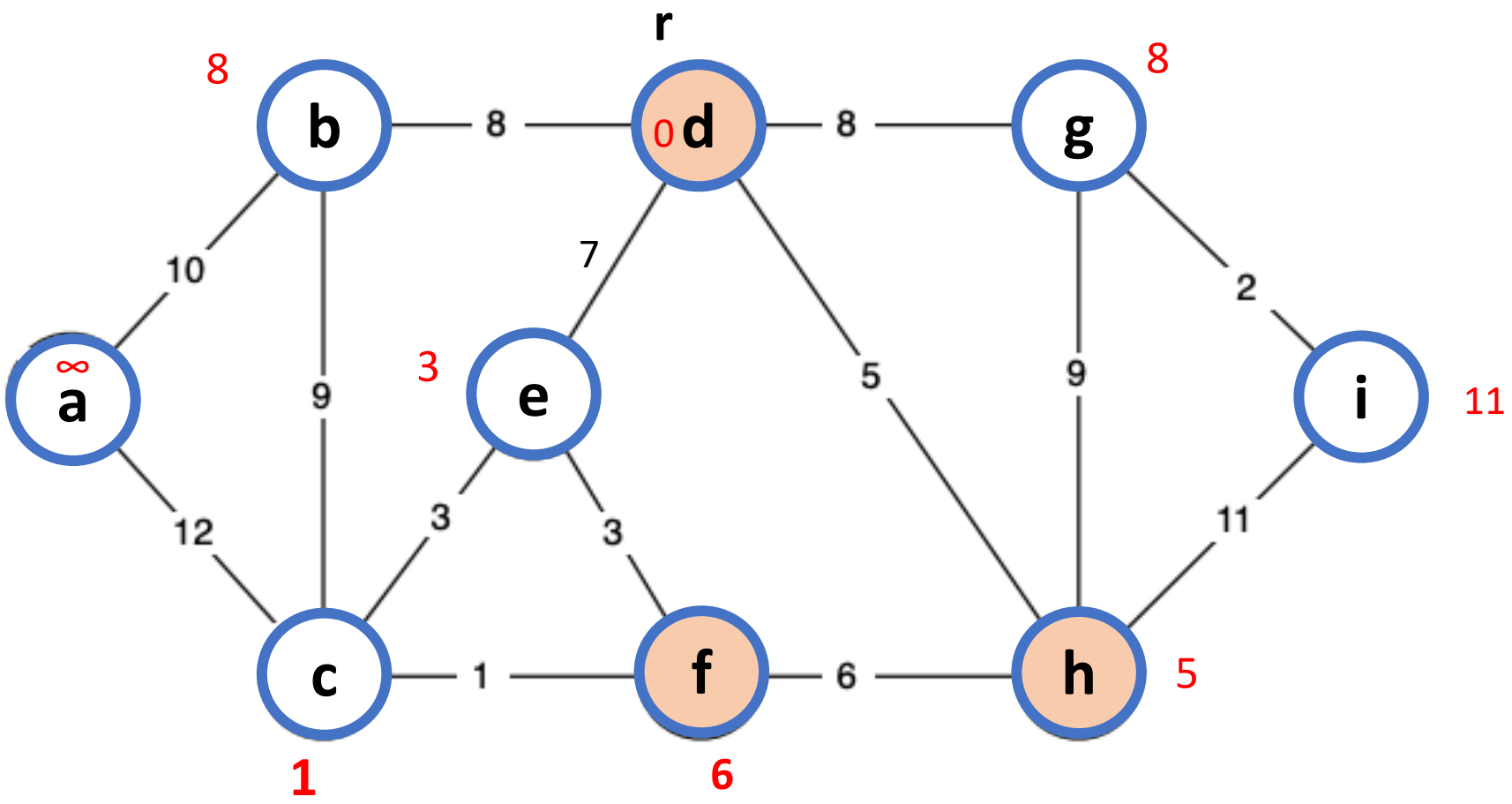

r
8
b
8
0 d
8
g
8
10
7
2
∞
a
9
7
e
5
9
i
11
12
3
3
11
∞
c
1
f
6
h
5
6


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

Number of call of : DECREASE-KEY ?



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)

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

O(vlogv)

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 $k_v \leftarrow w(u,v)$

Implementation

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

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

O(vlogv)

While Q ≠ ∅ :

Do u ← Extract-MIN(Q) → O(vlogv)

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 = ∅ // 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

O(vlogv)

While Q ≠ ∅ :

Do u ← Extract-MIN(Q) → O(vlogv)

E
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 = ∅ // 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

O(vlogv)

While Q ≠ ∅ :

Do u ← Extract-MIN(Q) → O(vlogv)

E

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)

O(Elogv)

Greedy

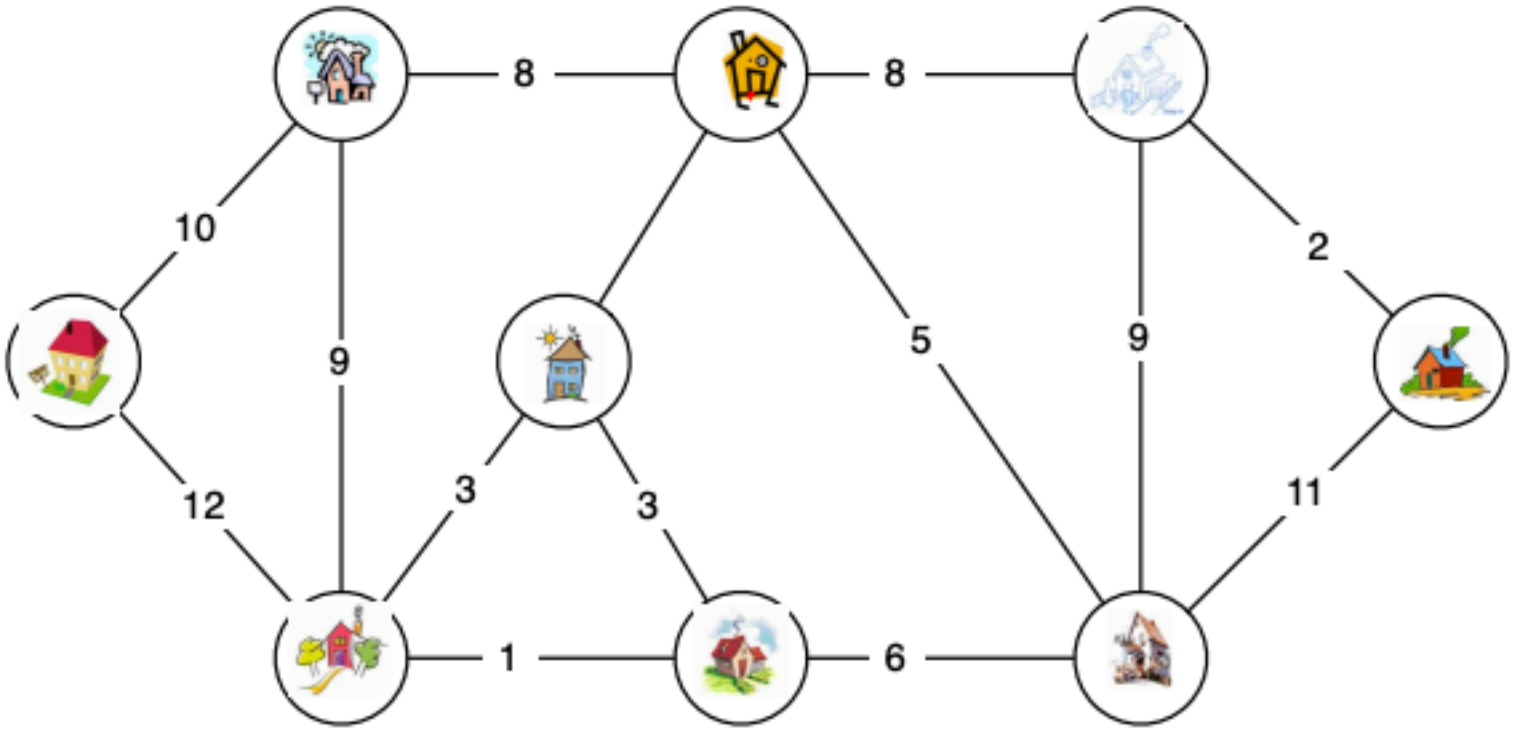
Lecture 5

Shortest Path Property

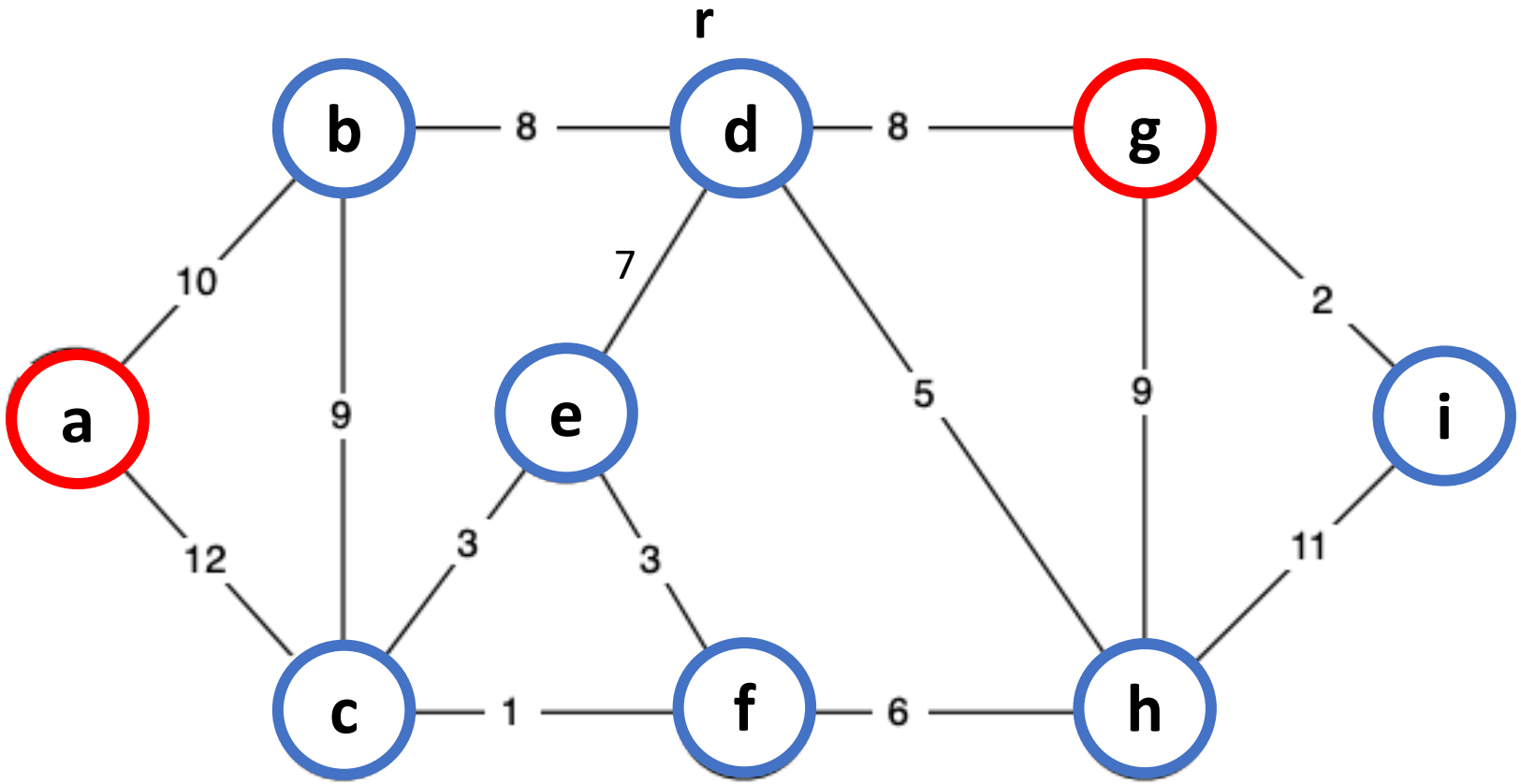
- Definition:
  - $\delta(s,v)$ = length of the shortest path from s to v in G

Shortest Path Property

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

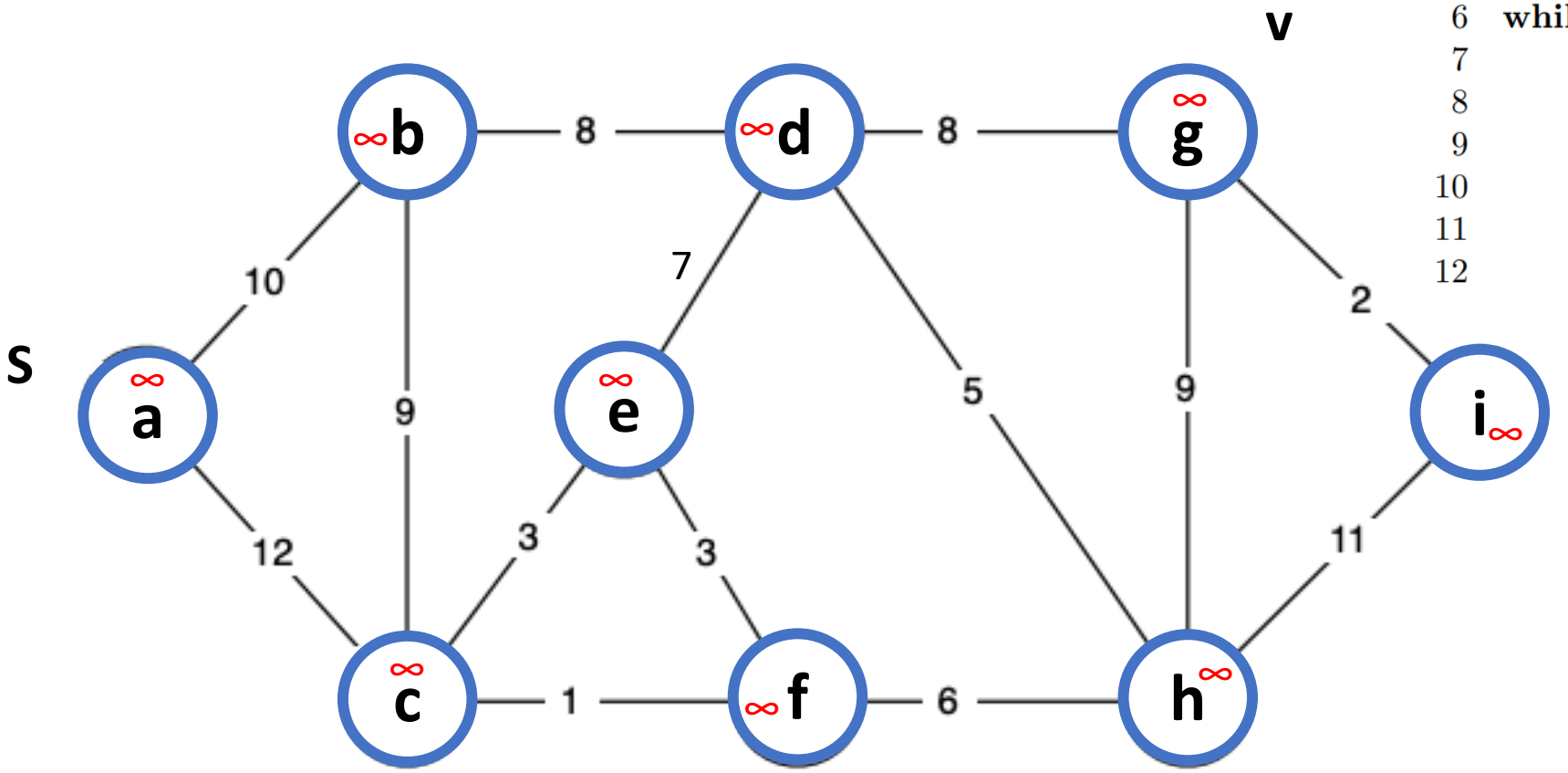


Shortest Path (a,g)



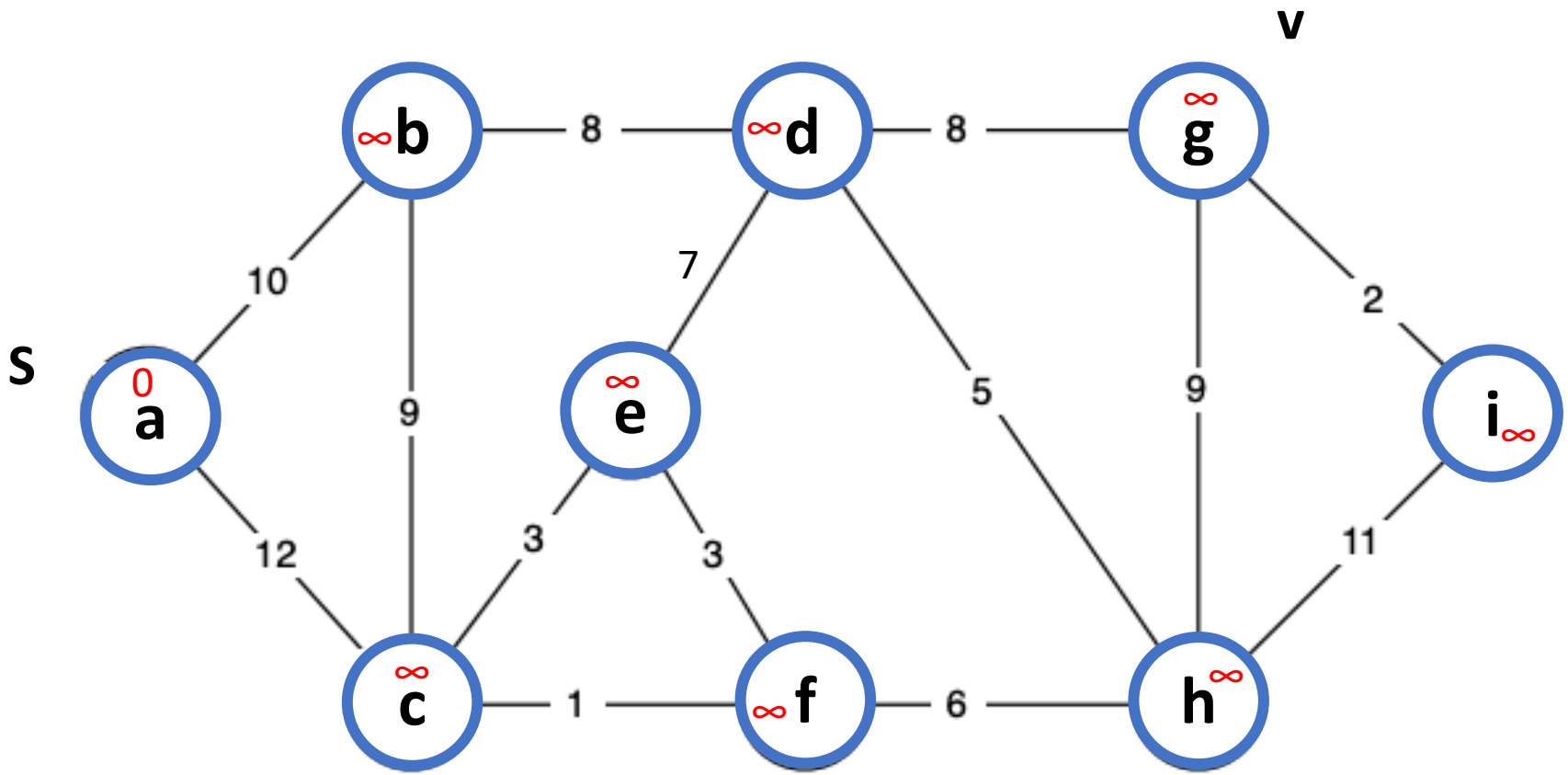
```
DIJKSTRA(G = (V, E), s)
 1  for all v ∈ V
 2        do d_u ← ∞
 3           π_u ← NIL
 4  d_s ← 0
 5  Q ← MAKEQUEUE(V)      ▷ use d_u as key
 6  while Q ≠ ∅
 7        do u ← EXTRACTMIN(Q)
 8           for each v ∈ Adj(u)
 9                 do if d_v > d_u + w(u, v)
10                       then d_v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

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

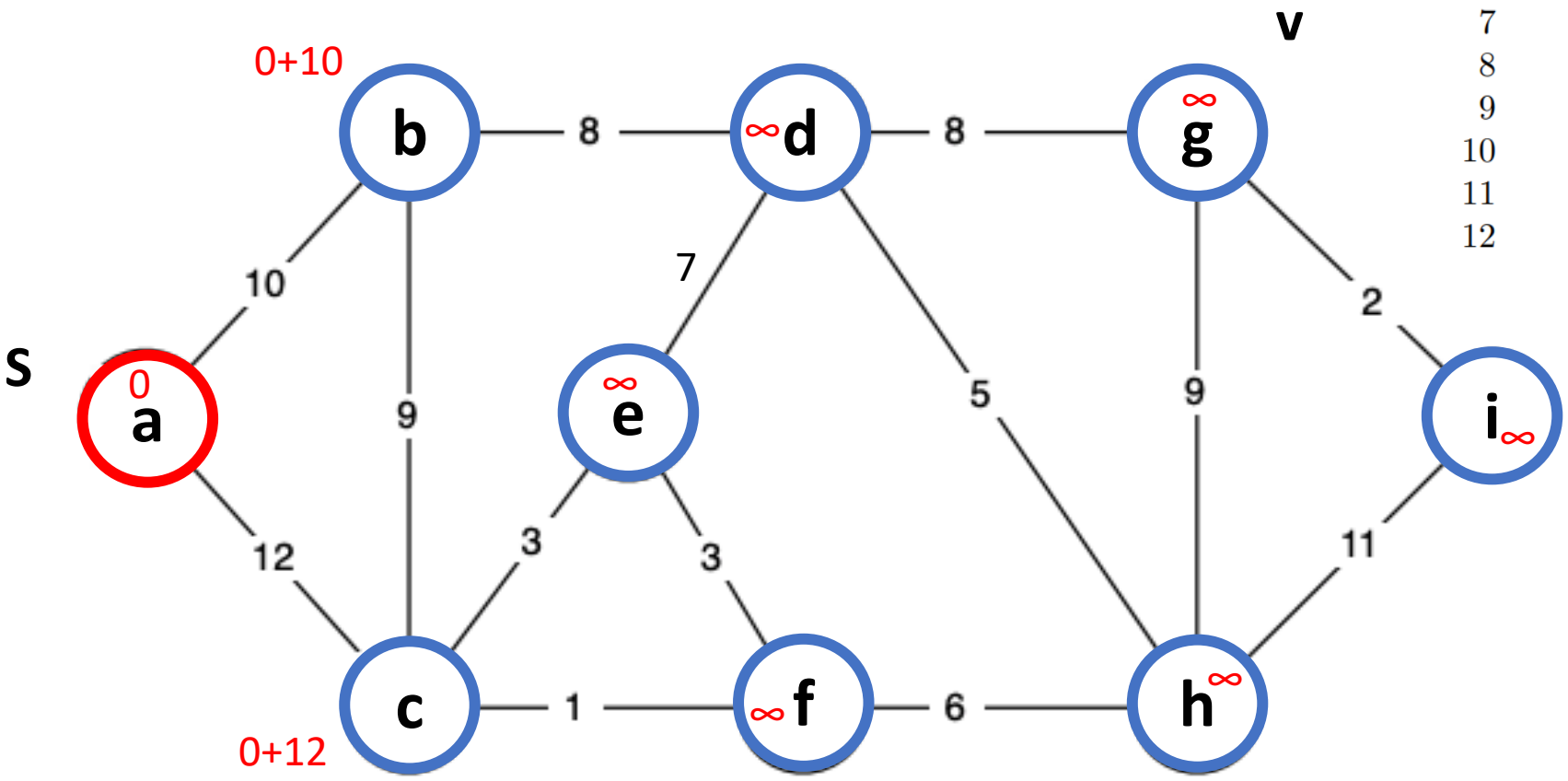



```
DIJKSTRA(G = (V, E), s)
 1  for all v ∈ V
 2        do d_u ← ∞
 3           π_u ← NIL
 4  d_s ← 0
 5  Q ← MAKEQUEUE(V)        ▷ use d_u as key
 6  while Q ≠ ∅
 7        do u ← EXTRACTMIN(Q)
 8           for each v ∈ Adj(u)
 9                 do if d_v > d_u + w(u, v)
10                       then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

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

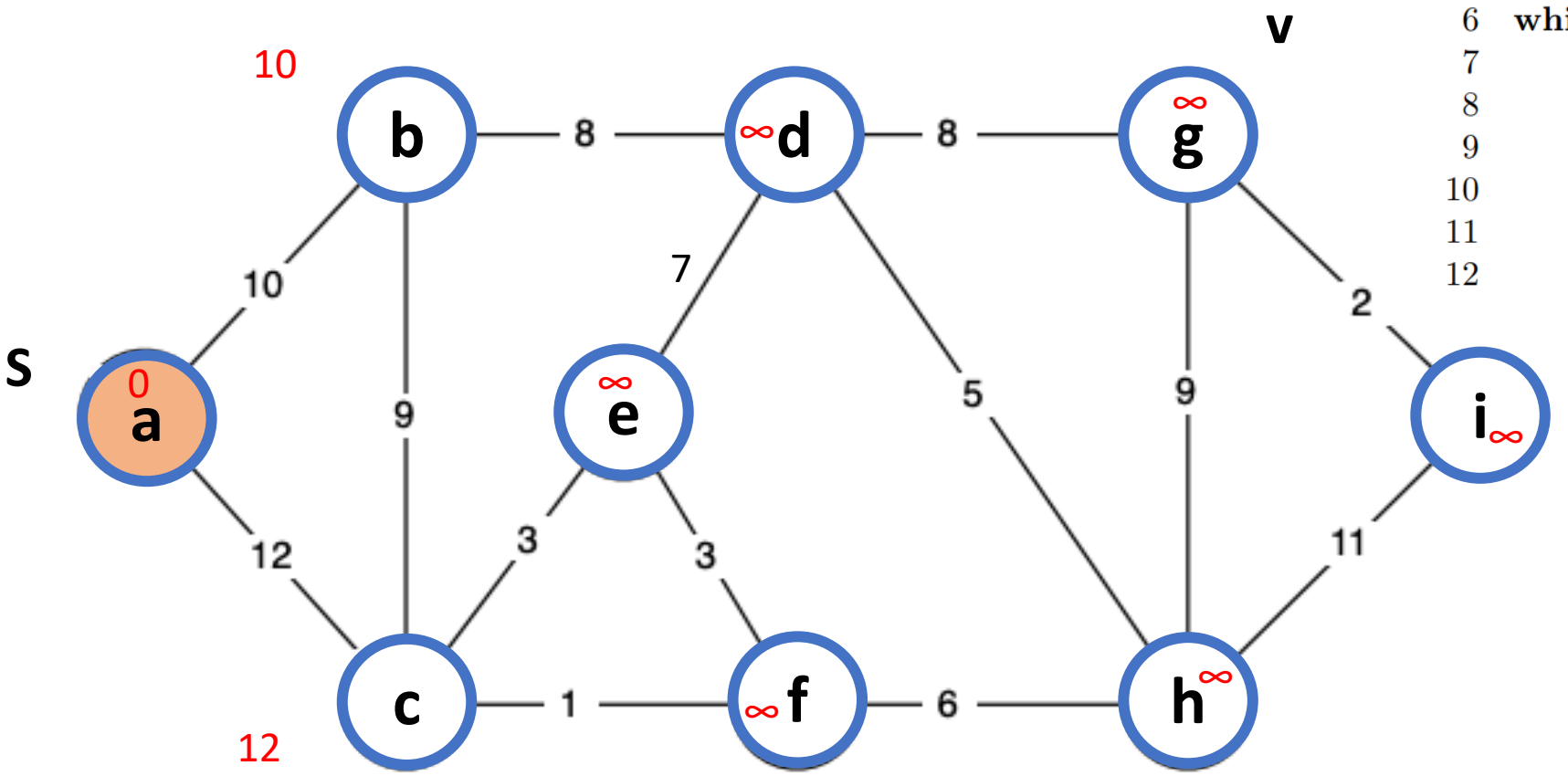


- Definition: δ (s,v) = length of the shortest path from s to v in G



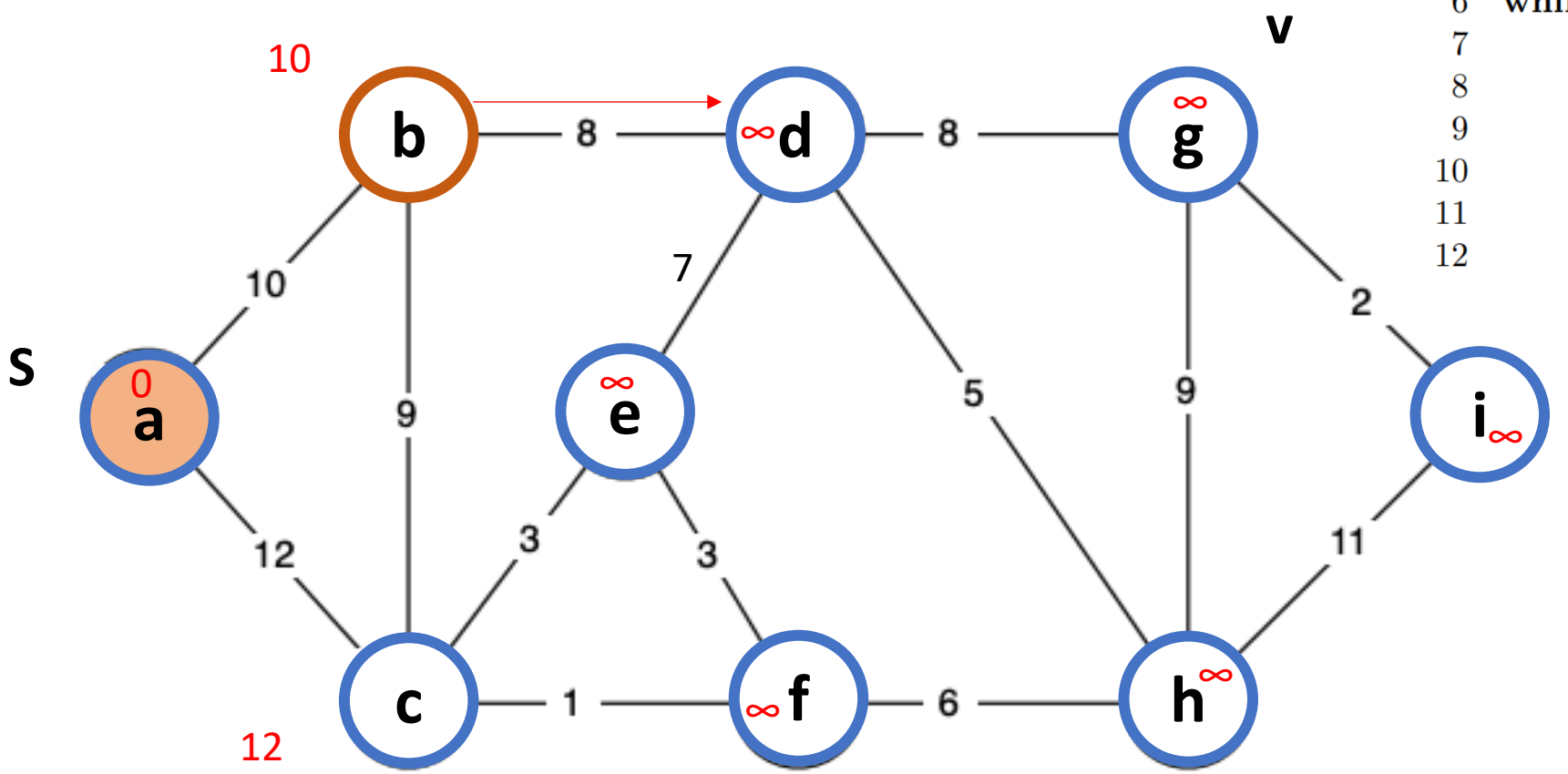

```
DIJKSTRA(G = (V, E), s)
 1  for all v ∈ V
 2        do d_u ← ∞
 3           π_u ← NIL
 4  d_s ← 0
 5  Q ← MAKEQUEUE(V)        ▷ use d_u as key
 6  while Q ≠ ∅
 7        do u ← EXTRACTMIN(Q)
 8           for each v ∈ Adj(u)
 9                 do if d_v > d_u + w(u, v)
10                       then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

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



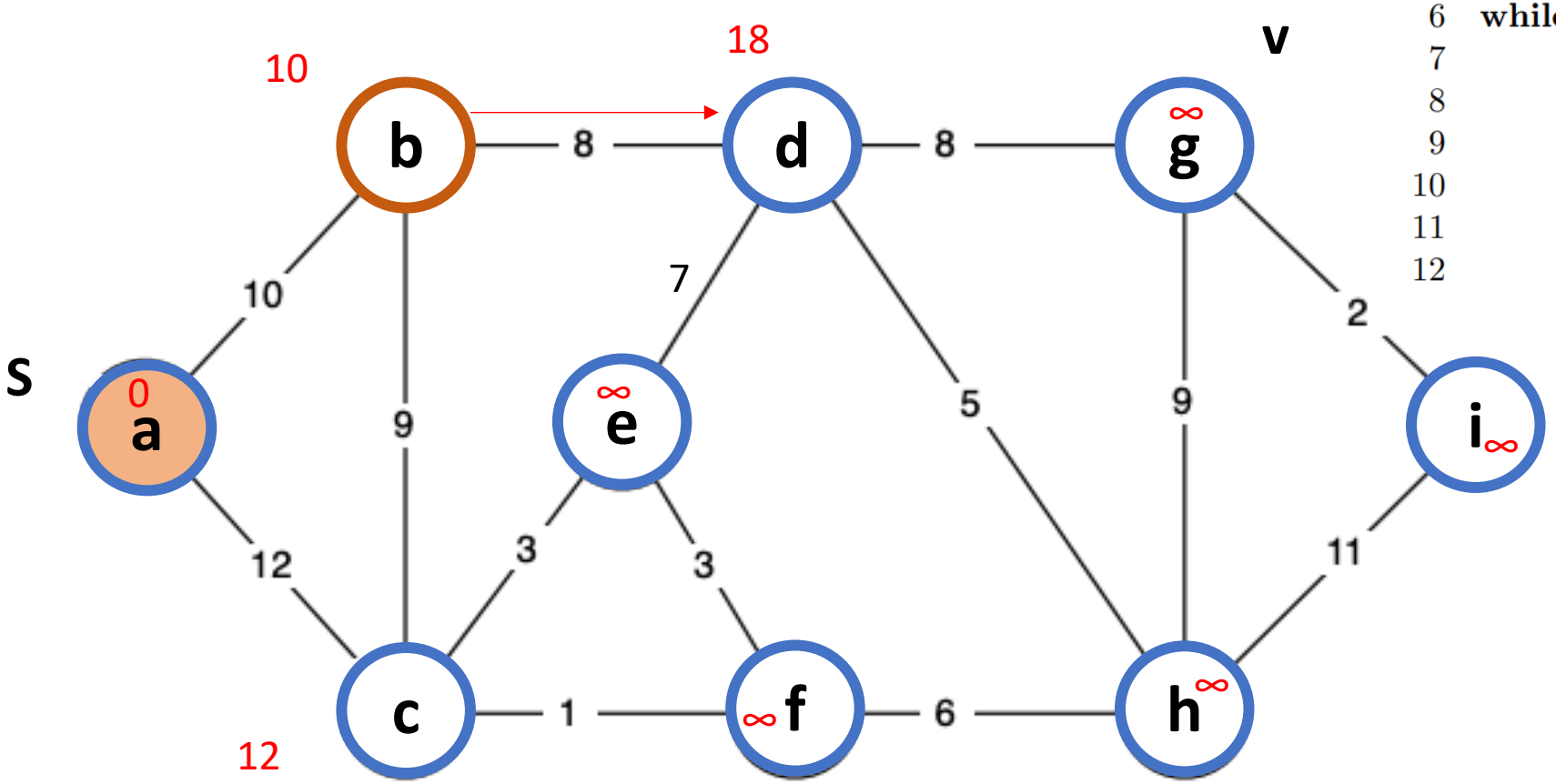

```
DIJKSTRA(G = (V, E), s)
 1  for all v ∈ V
 2        do d_u ← ∞
 3           π_u ← NIL
 4  d_s ← 0
 5  Q ← MAKEQUEUE(V)        ▷ use d_u as key
 6  while Q ≠ ∅
 7        do u ← EXTRACTMIN(Q)
 8           for each v ∈ Adj(u)
 9                 do if d_v > d_u + w(u, v)
10                       then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

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



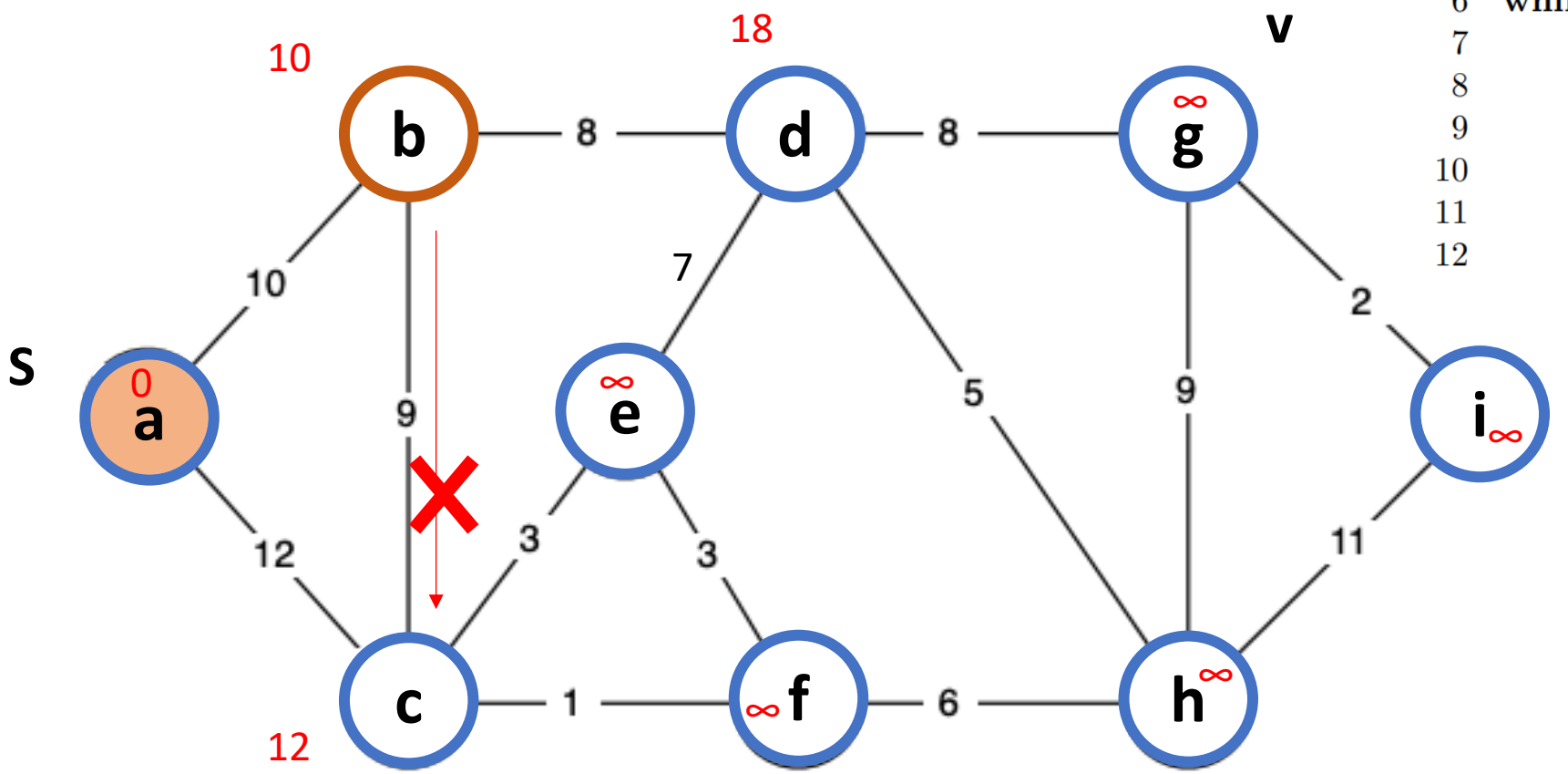

```
DIJKSTRA(G = (V, E), s)
 1  for all v ∈ V
 2        do d_u ← ∞
 3           π_u ← NIL
 4  d_s ← 0
 5  Q ← MAKEQUEUE(V)        ▷ use d_u as key
 6  while Q ≠ ∅
 7        do u ← EXTRACTMIN(Q)
 8           for each v ∈ Adj(u)
 9                 do if d_v > d_u + w(u, v)
10                       then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

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



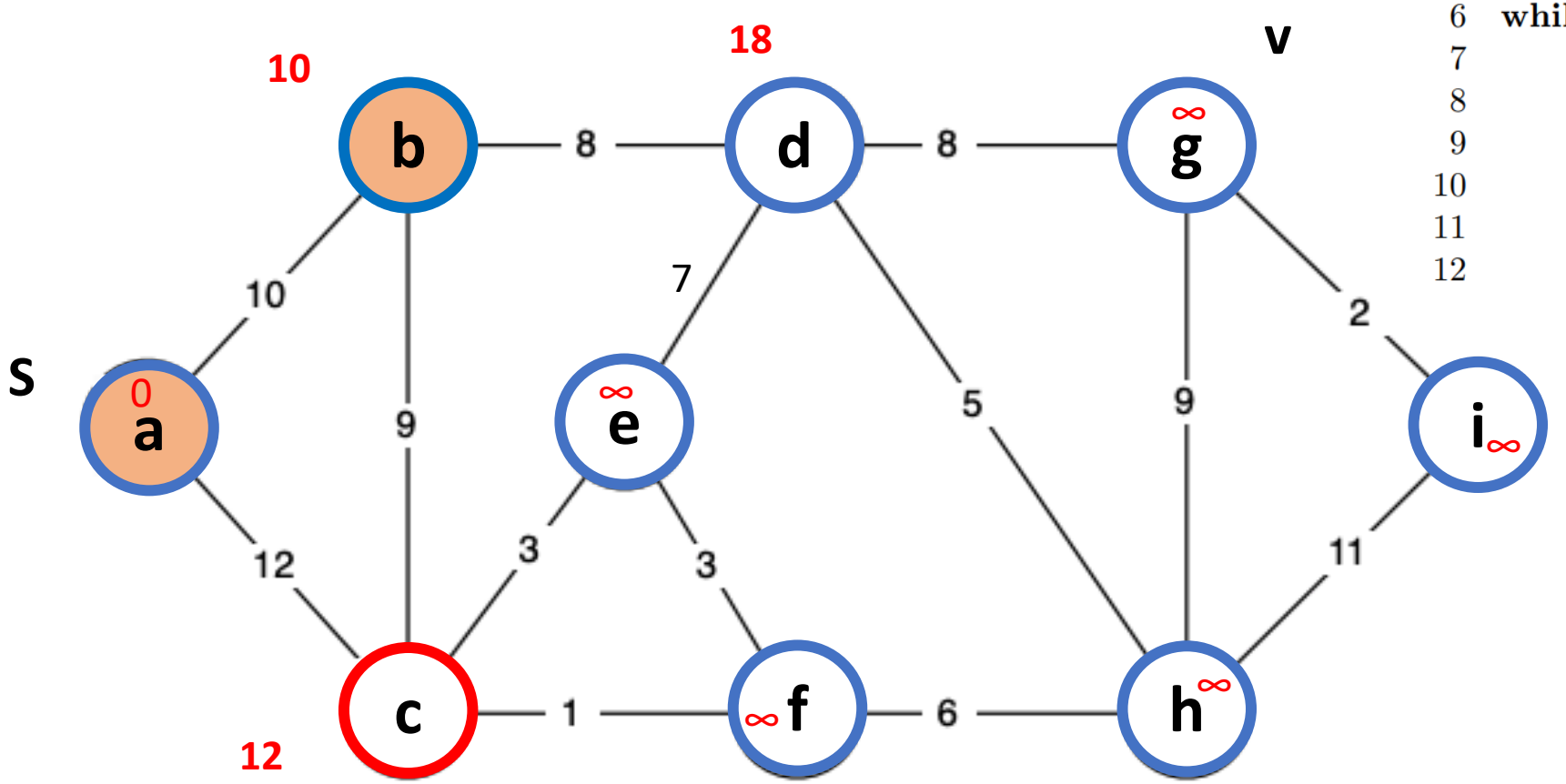

```
DIJKSTRA(G = (V, E), s)
 1  for all v ∈ V
 2        do d_u ← ∞
 3           π_u ← NIL
 4  d_s ← 0
 5  Q ← MAKEQUEUE(V)        ▷ use d_u as key
 6  while Q ≠ ∅
 7        do u ← EXTRACTMIN(Q)
 8           for each v ∈ Adj(u)
 9                 do if d_v > d_u + w(u, v)
10                       then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

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



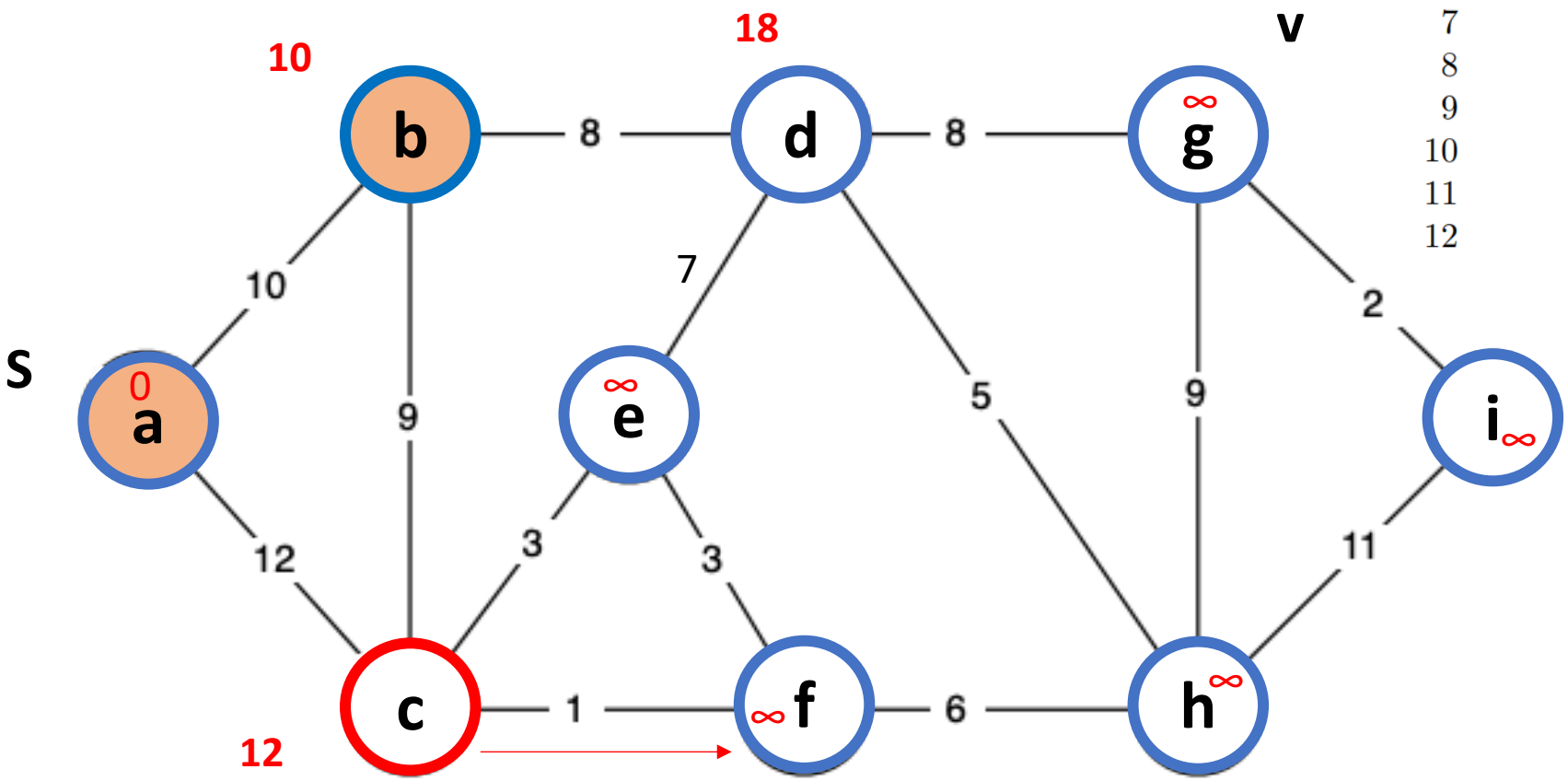

```
DIJKSTRA(G = (V, E), s)
 1  for all v ∈ V
 2        do d_u ← ∞
 3           π_u ← NIL
 4  d_s ← 0
 5  Q ← MAKEQUEUE(V)        ▷ use d_u as key
 6  while Q ≠ ∅
 7        do u ← EXTRACTMIN(Q)
 8           for each v ∈ Adj(u)
 9                 do if d_v > d_u + w(u, v)
10                       then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

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



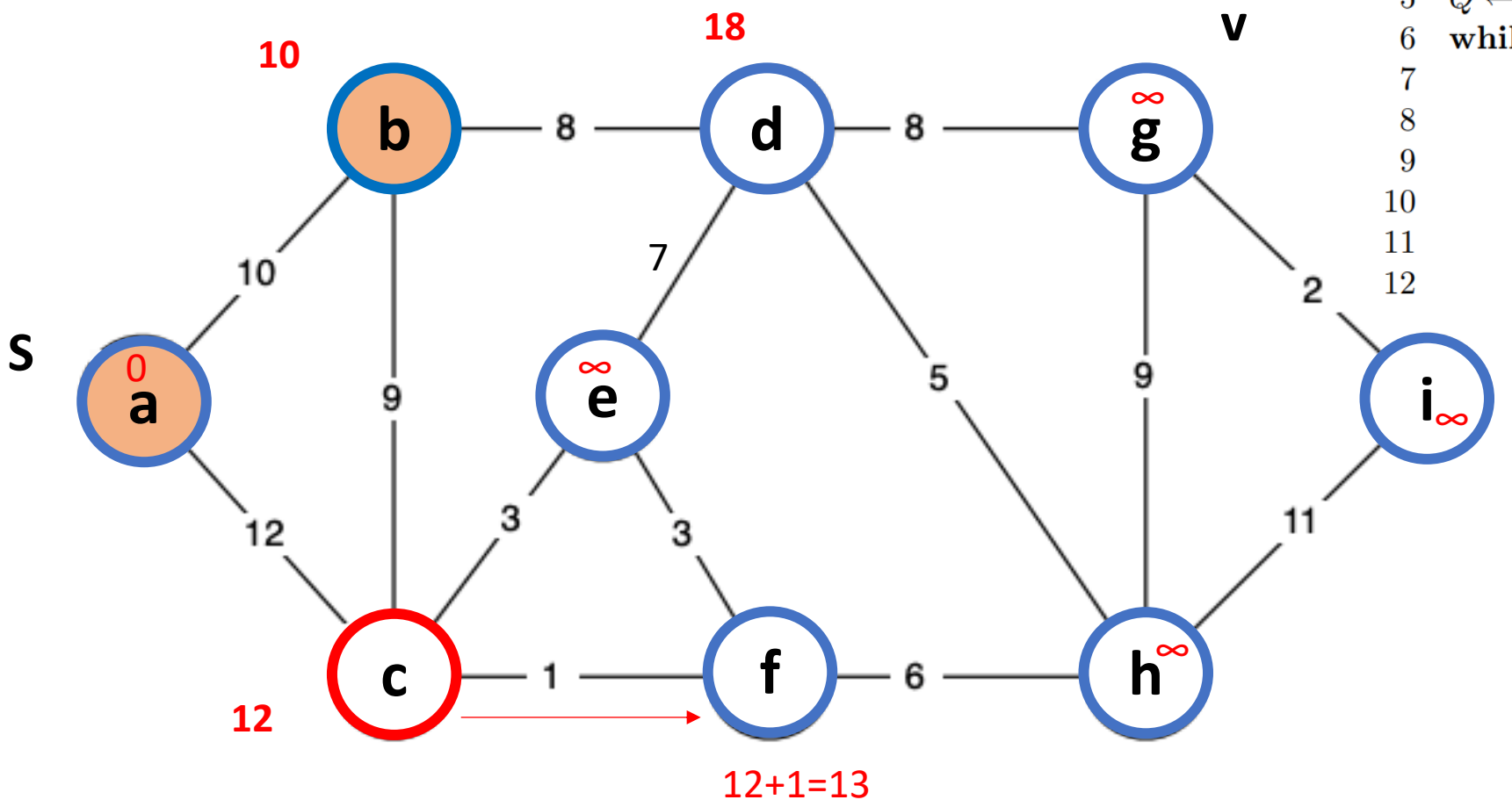

```
DIJKSTRA(G = (V, E), s)
1   for all v ∈ V
2         do d_u ← ∞
3            π_u ← NIL
4   d_s ← 0
5   Q ← MAKEQUEUE(V)        ▷ use d_u as key
6   while Q ≠ ∅
7         do u ← EXTRACTMIN(Q)
8            for each v ∈ Adj(u)
9                  do if d_v > d_u + w(u, v)
10                       then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

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



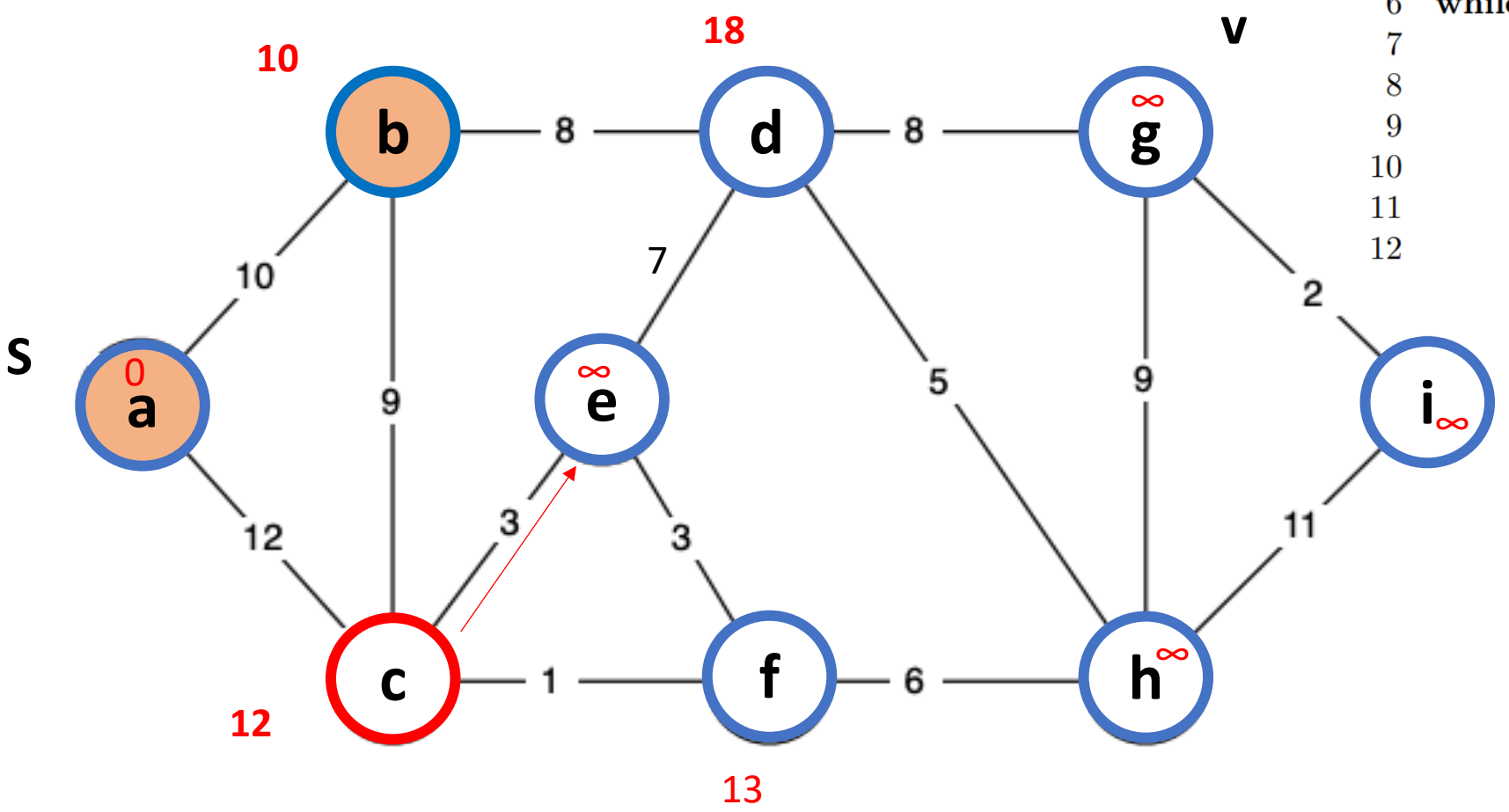

```
DIJKSTRA(G = (V, E), s)
1   for all v ∈ V
2         do d_u ← ∞
3            π_u ← NIL
4   d_s ← 0
5   Q ← MAKEQUEUE(V)        ▷ use d_u as key
6   while Q ≠ ∅
7         do u ← EXTRACTMIN(Q)
8            for each v ∈ Adj(u)
9                  do if d_v > d_u + w(u, v)
10                       then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

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



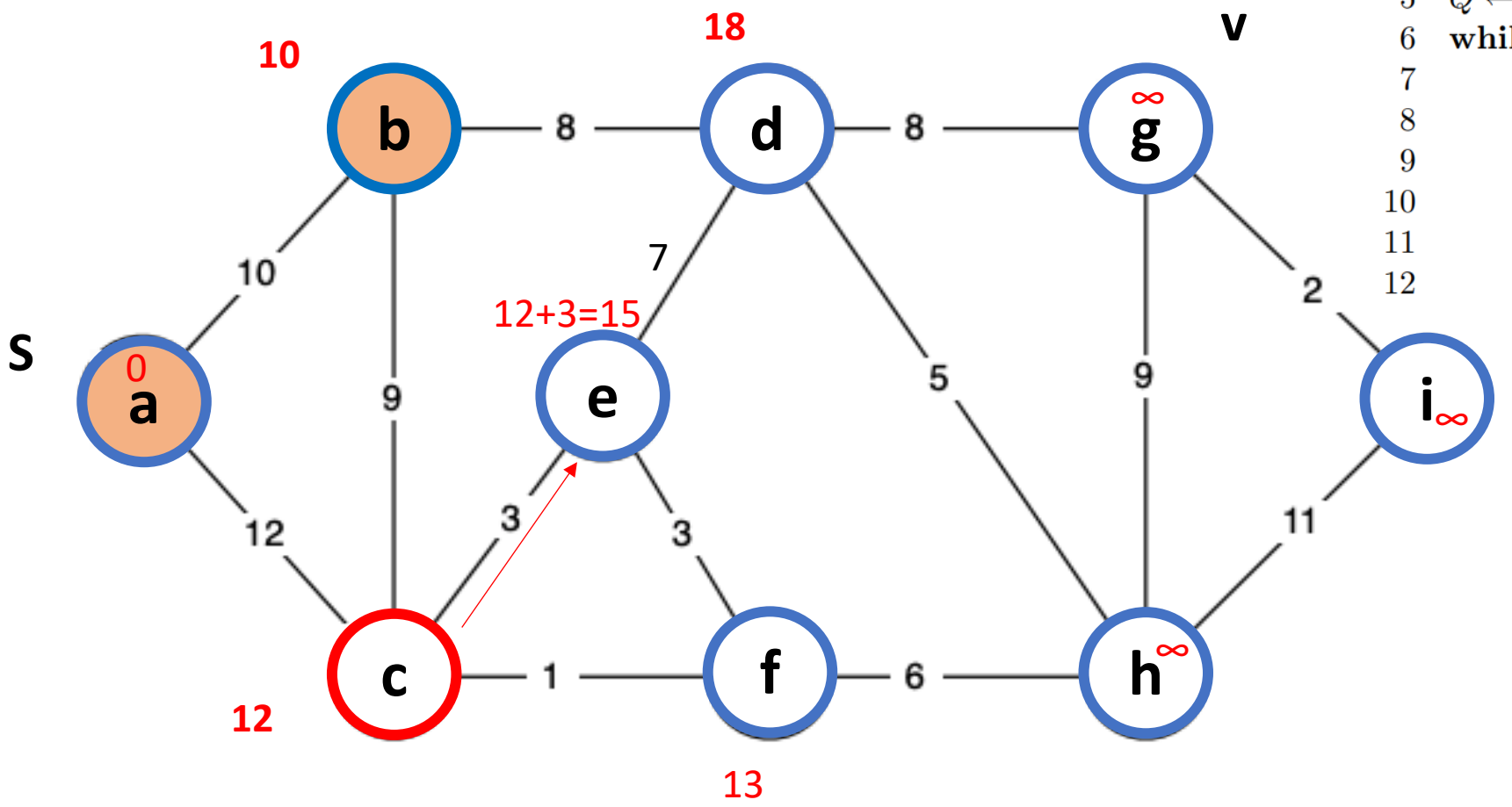

```
DIJKSTRA(G = (V, E), s)
1   for all v ∈ V
2         do d_u ← ∞
3            π_u ← NIL
4   d_s ← 0
5   Q ← MAKEQUEUE(V)        ▷ use d_u as key
6   while Q ≠ ∅
7         do u ← EXTRACTMIN(Q)
8            for each v ∈ Adj(u)
9                  do if d_v > d_u + w(u, v)
10                       then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

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



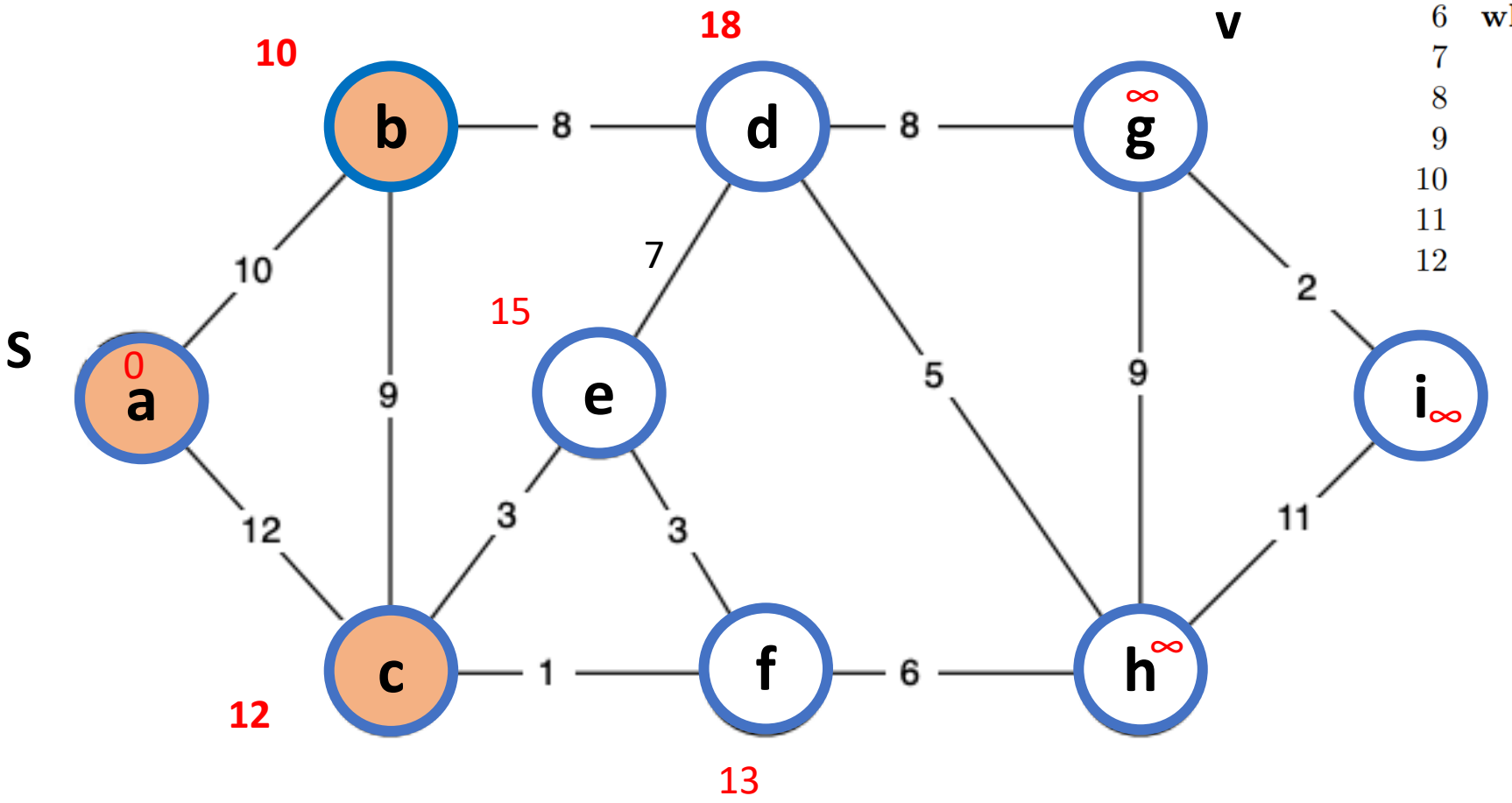

```
DIJKSTRA(G = (V, E), s)
1   for all v ∈ V
2         do d_u ← ∞
3            π_u ← NIL
4   d_s ← 0
5   Q ← MAKEQUEUE(V)        ▷ use d_u as key
6   while Q ≠ ∅
7         do u ← EXTRACTMIN(Q)
8            for each v ∈ Adj(u)
9                  do if d_v > d_u + w(u, v)
10                       then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

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



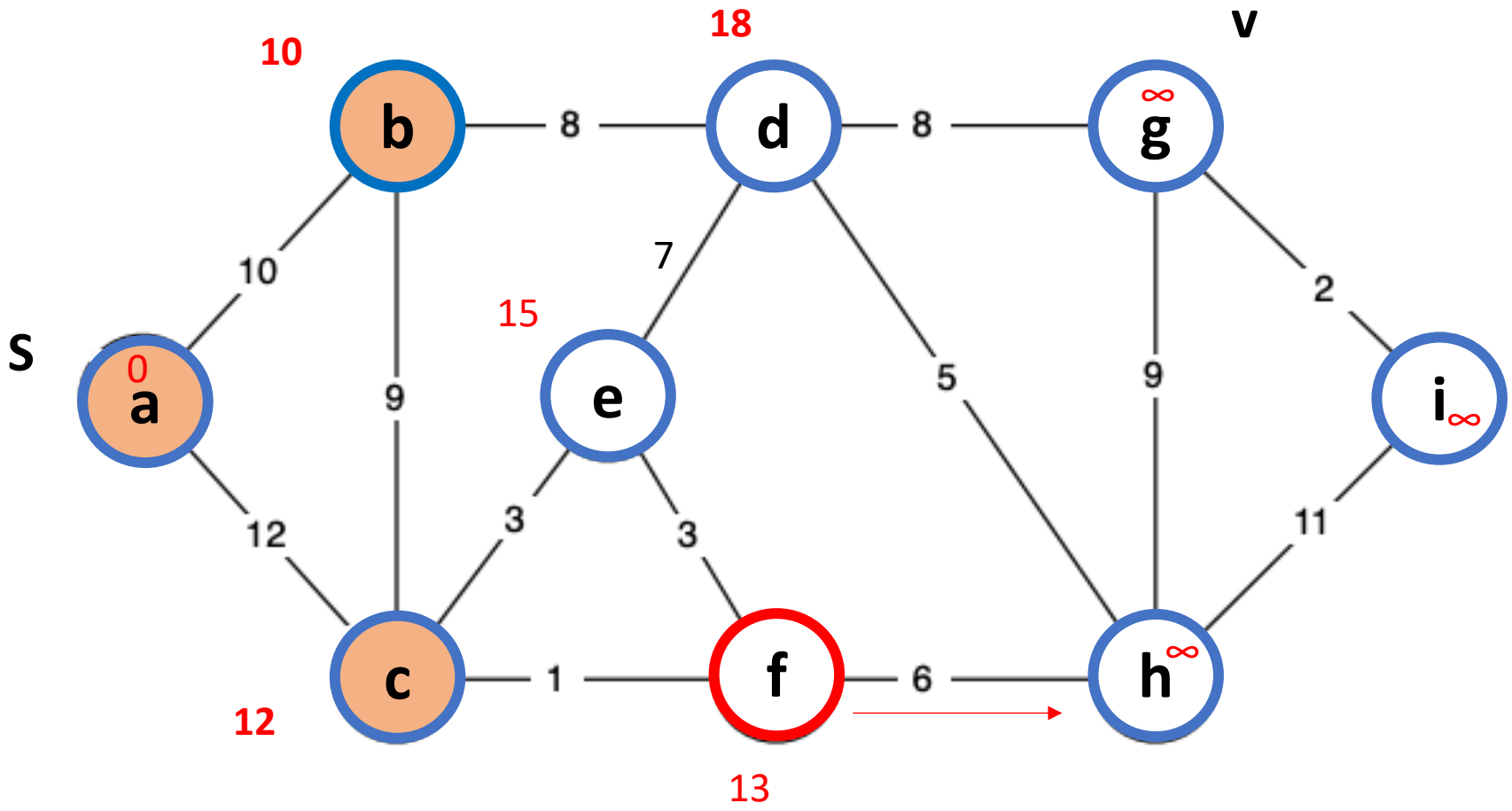

```
DIJKSTRA(G = (V, E), s)
1   for all v ∈ V
2         do d_u ← ∞
3            π_u ← NIL
4   d_s ← 0
5   Q ← MAKEQUEUE(V)        ▷ use d_u as key
6   while Q ≠ ∅
7         do u ← EXTRACTMIN(Q)
8            for each v ∈ Adj(u)
9                  do if d_v > d_u + w(u, v)
10                       then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

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

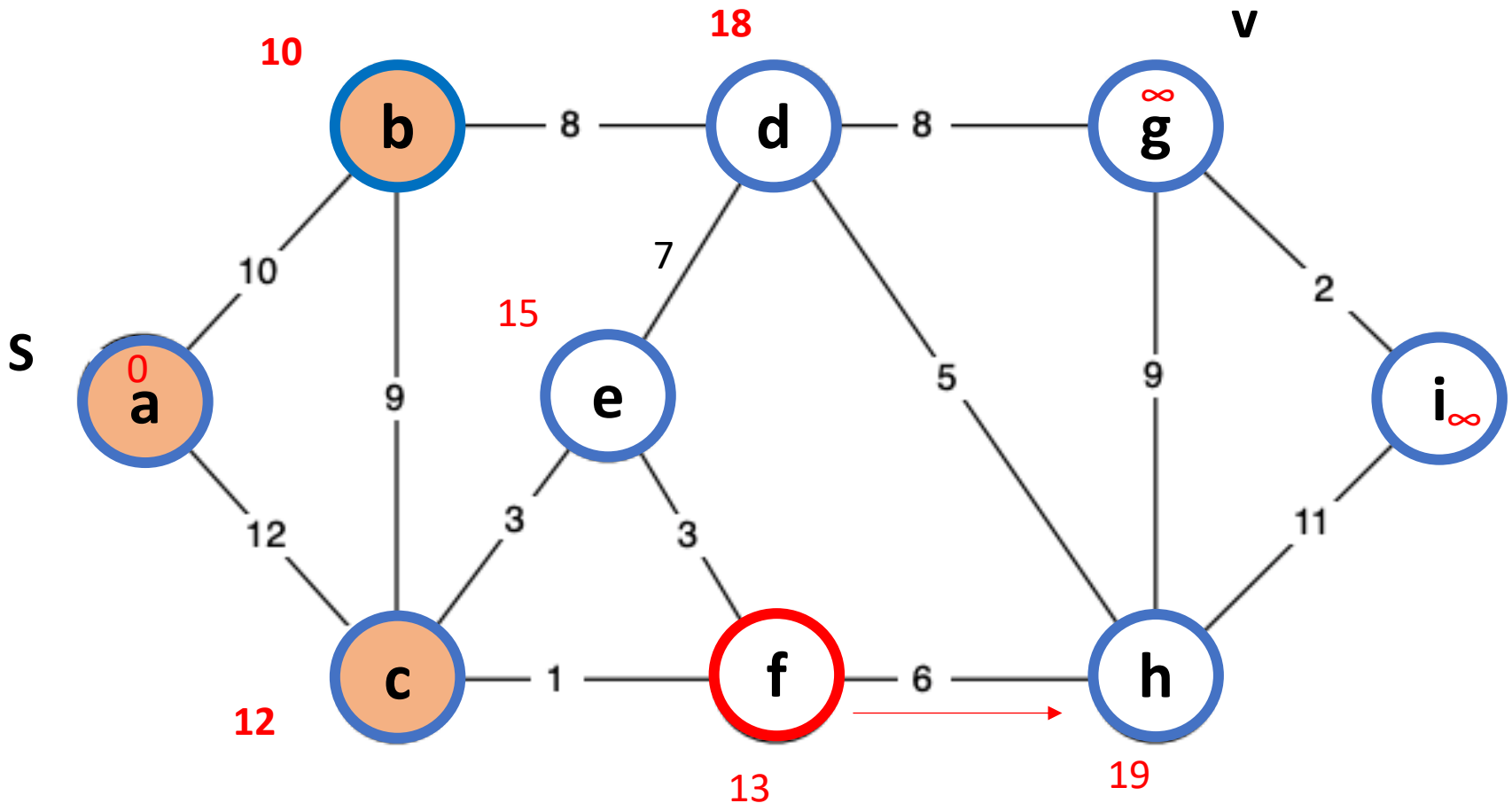



```
DIJKSTRA(G = (V, E), s)
1  for all v ∈ V
2        do d_u ← ∞
3           π_u ← NIL
4  d_s ← 0
5  Q ← MAKEQUEUE(V)        ▷ use d_u as key
6  while Q ≠ ∅
7        do u ← EXTRACTMIN(Q)
8           for each v ∈ Adj(u)
9                 do if d_v > d_u + w(u, v)
10                      then d_v ← d_u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

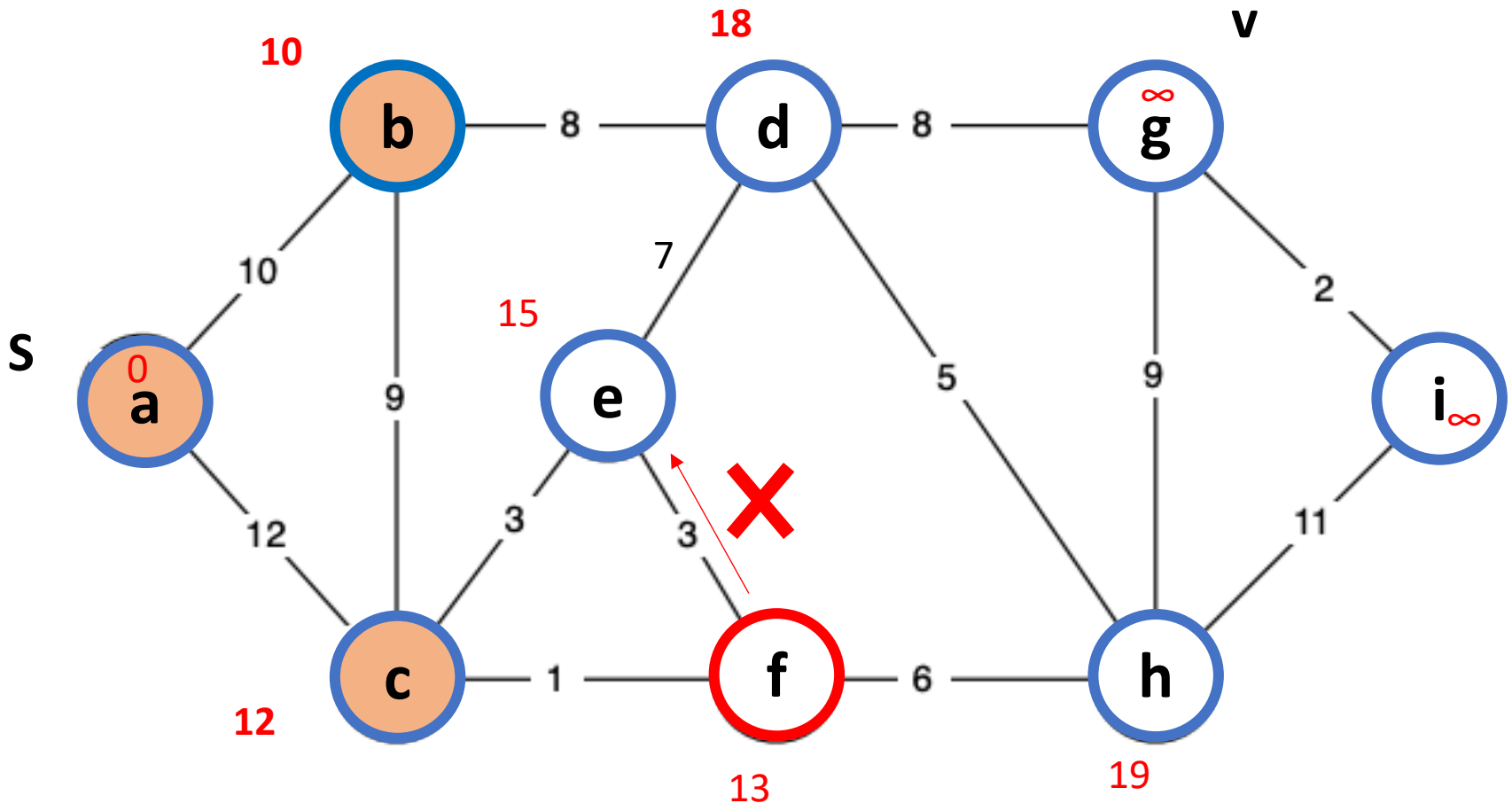
- Definition: δ (s,v) = length of the shortest path from s to v in G



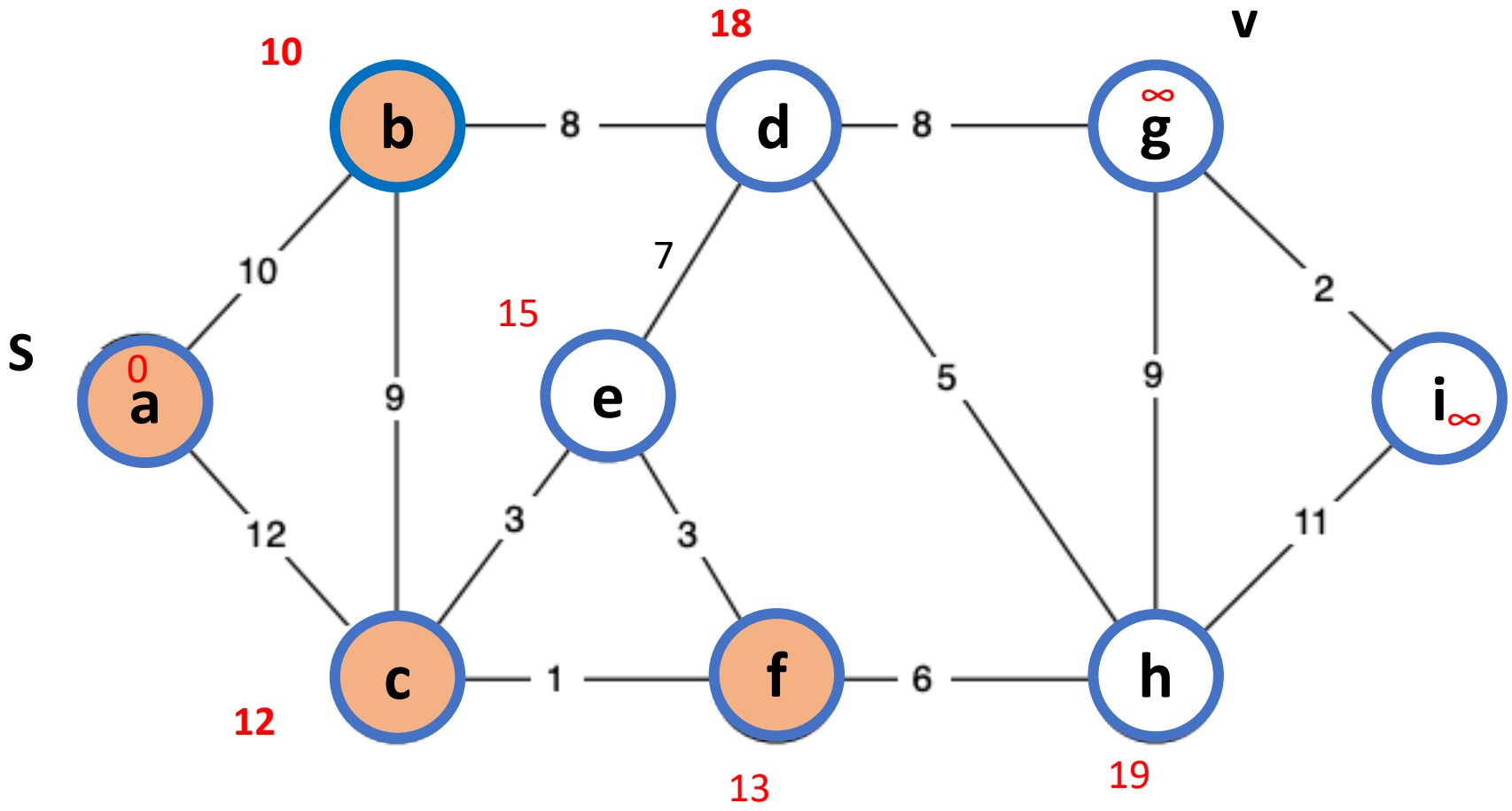
- Definition: δ (s,v) = length of the shortest path from s to v in G



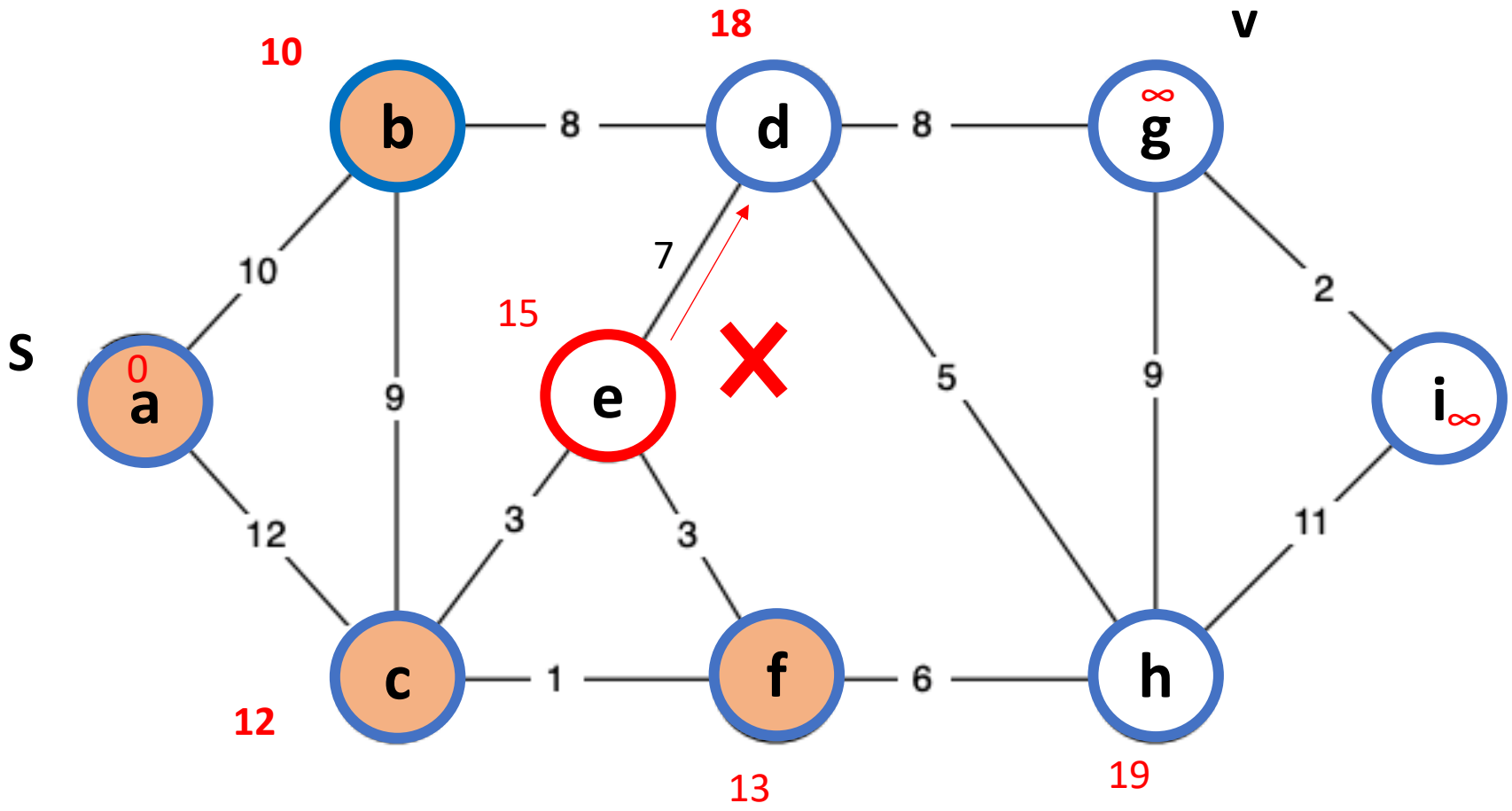
- Definition: δ (s,v) = length of the shortest path from s to v in G



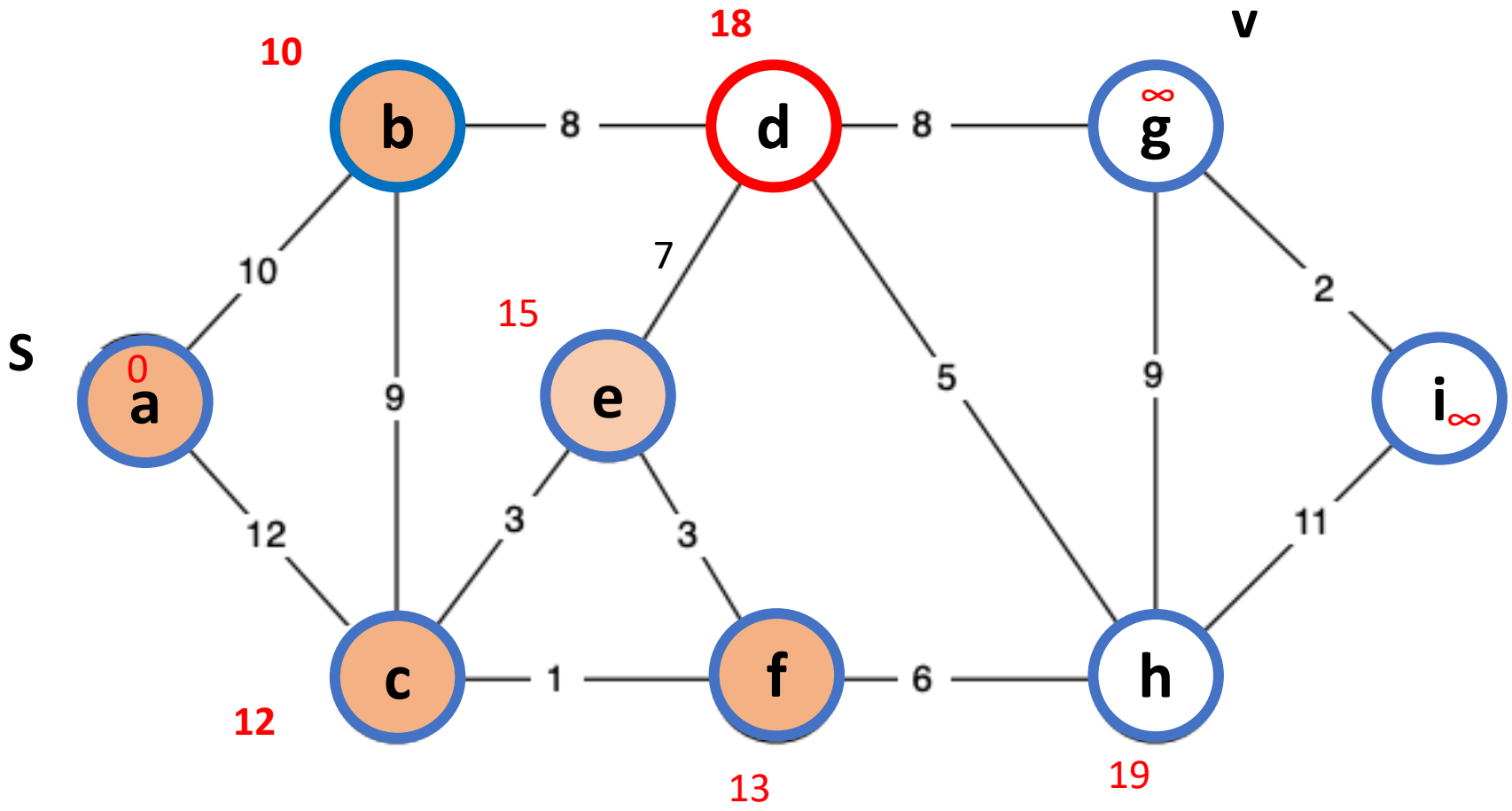
- Definition: δ (s,v) = length of the shortest path from s to v in G



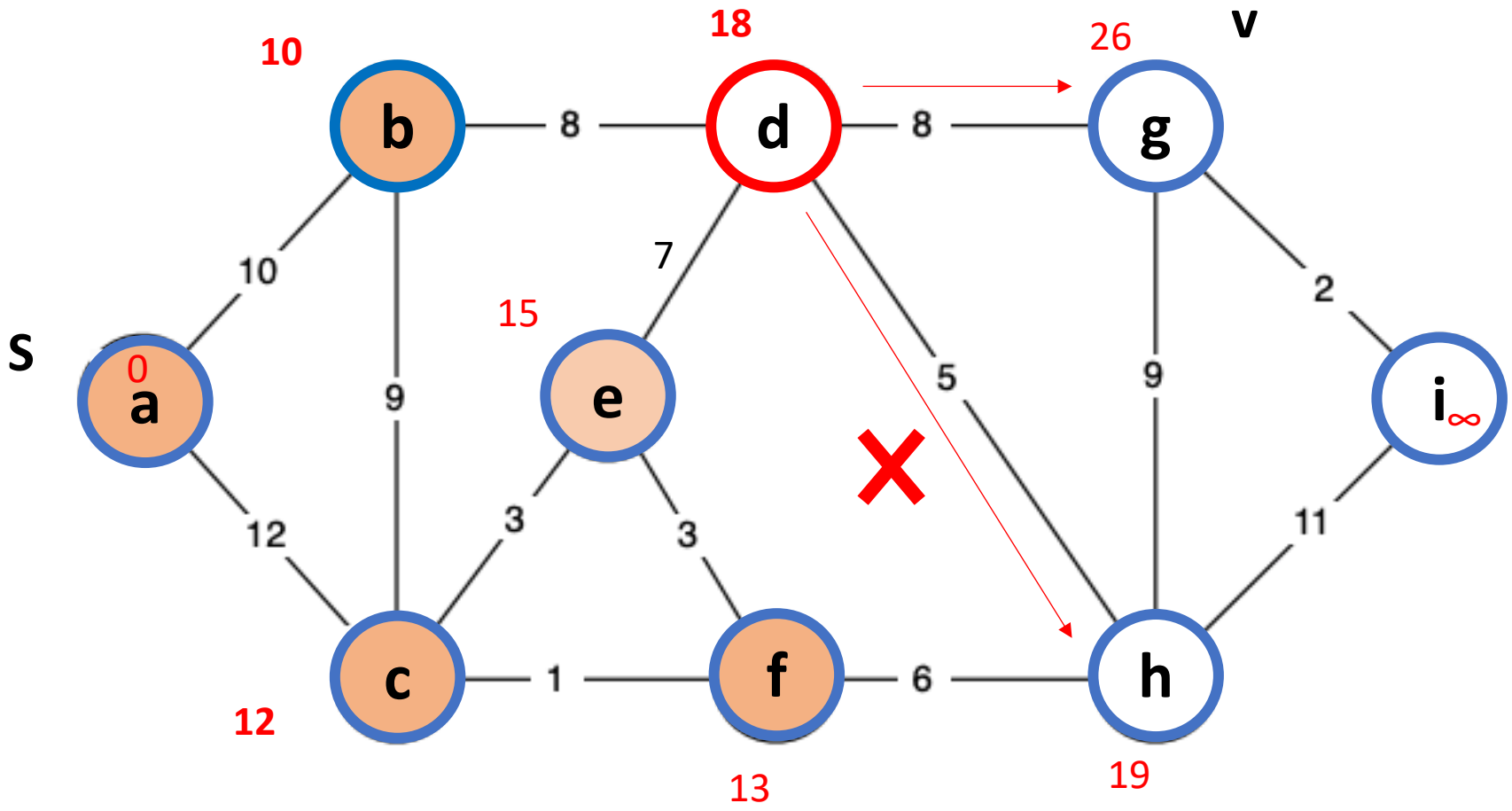
- Definition: δ (s,v) = length of the shortest path from s to v in G



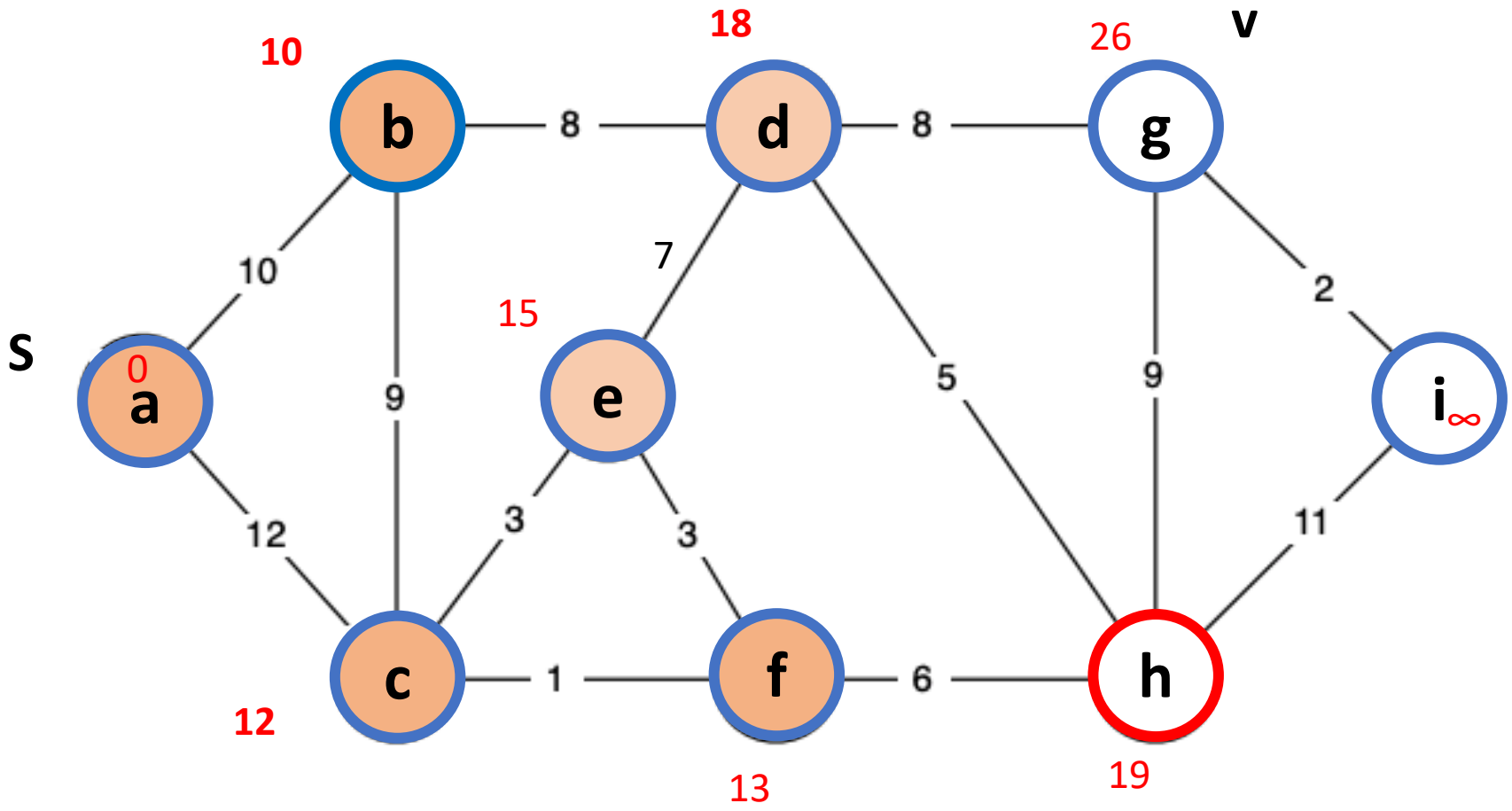
- Definition: δ (s,v) = length of the shortest path from s to v in G



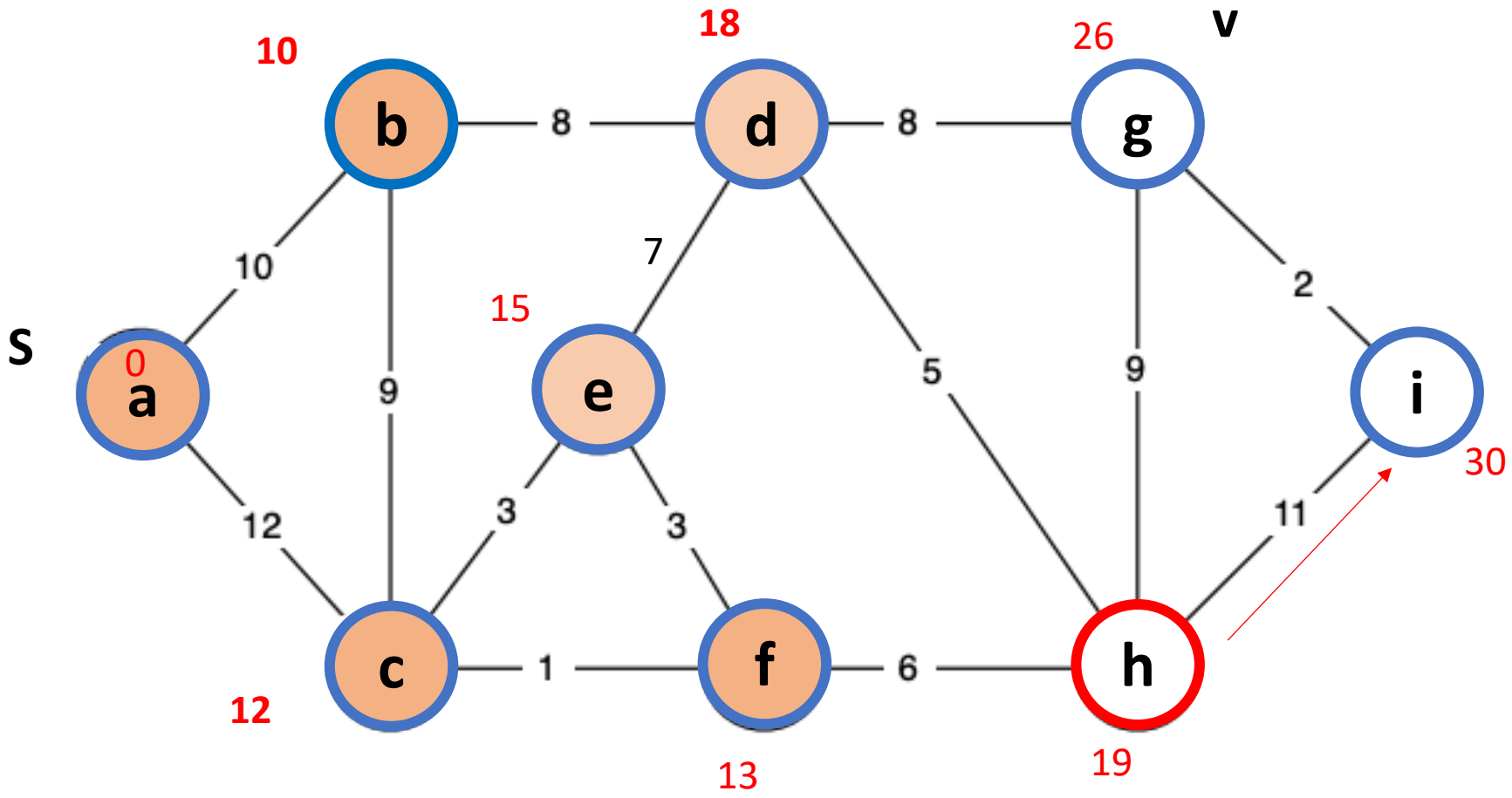
- Definition: δ (s,v) = length of the shortest path from s to v in G



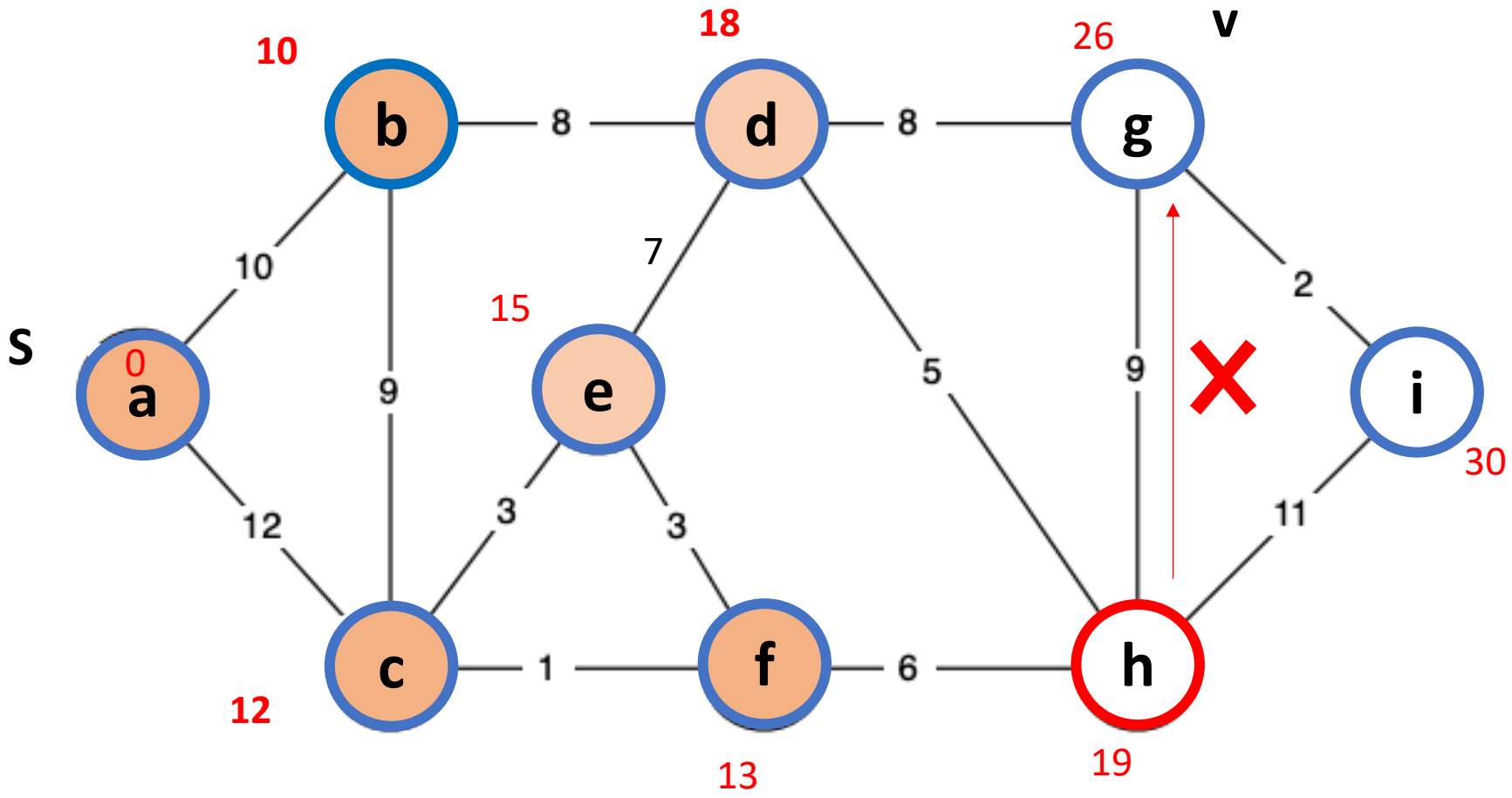
- Definition: δ (s,v) = length of the shortest path from s to v in G



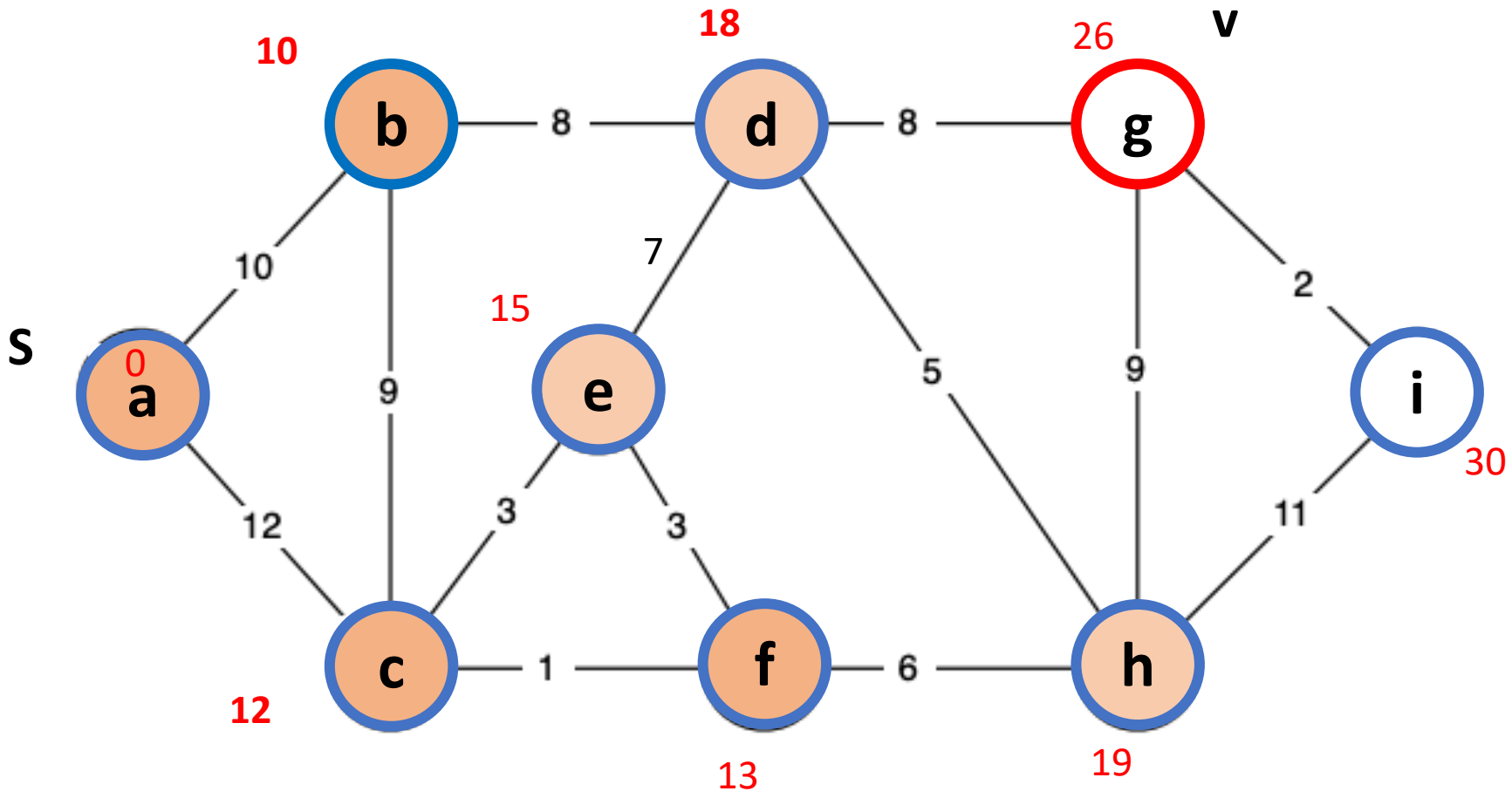
- Definition: δ (s,v) = length of the shortest path from s to v in G



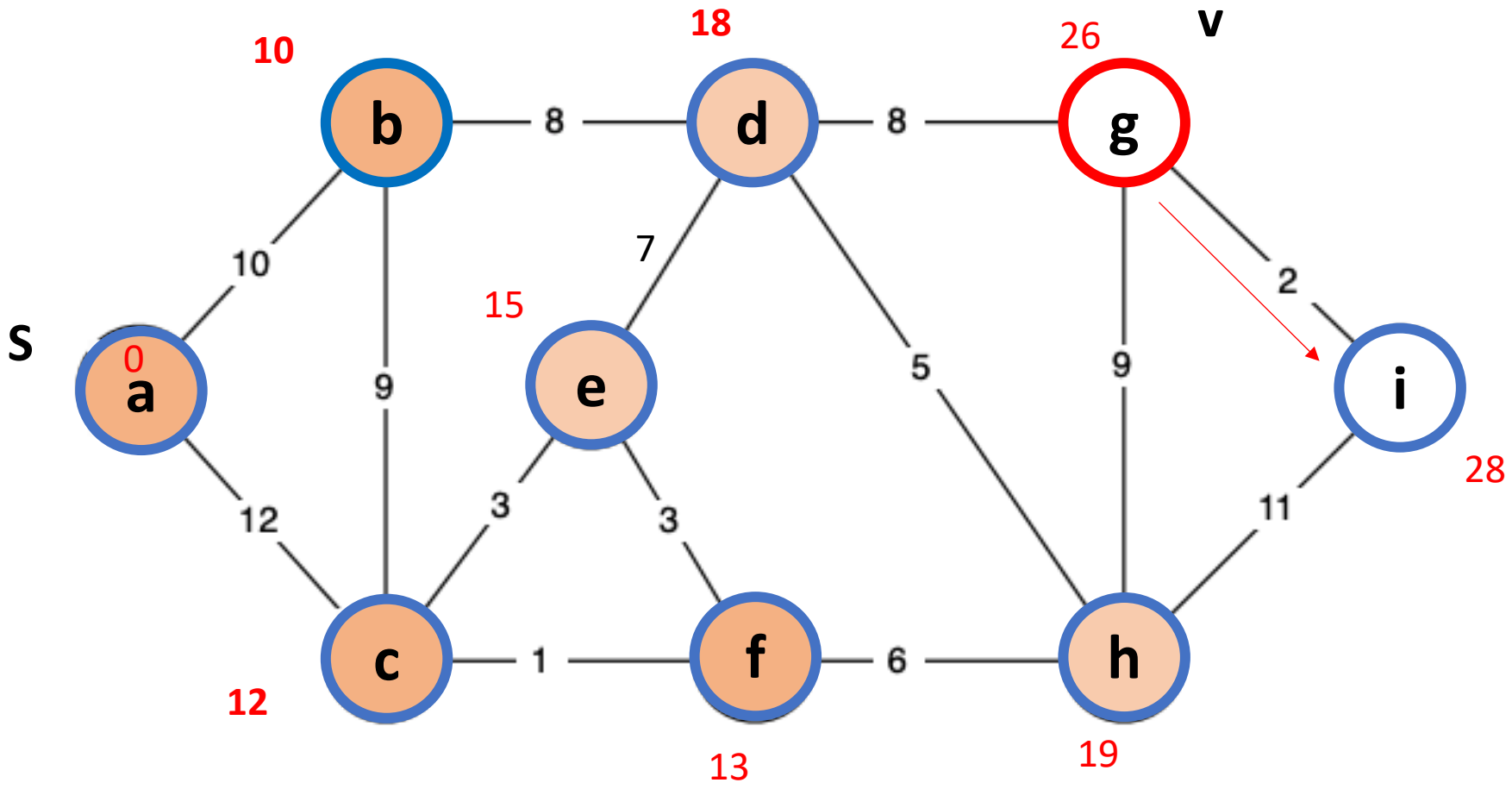
- Definition: δ (s,v) = length of the shortest path from s to v in G



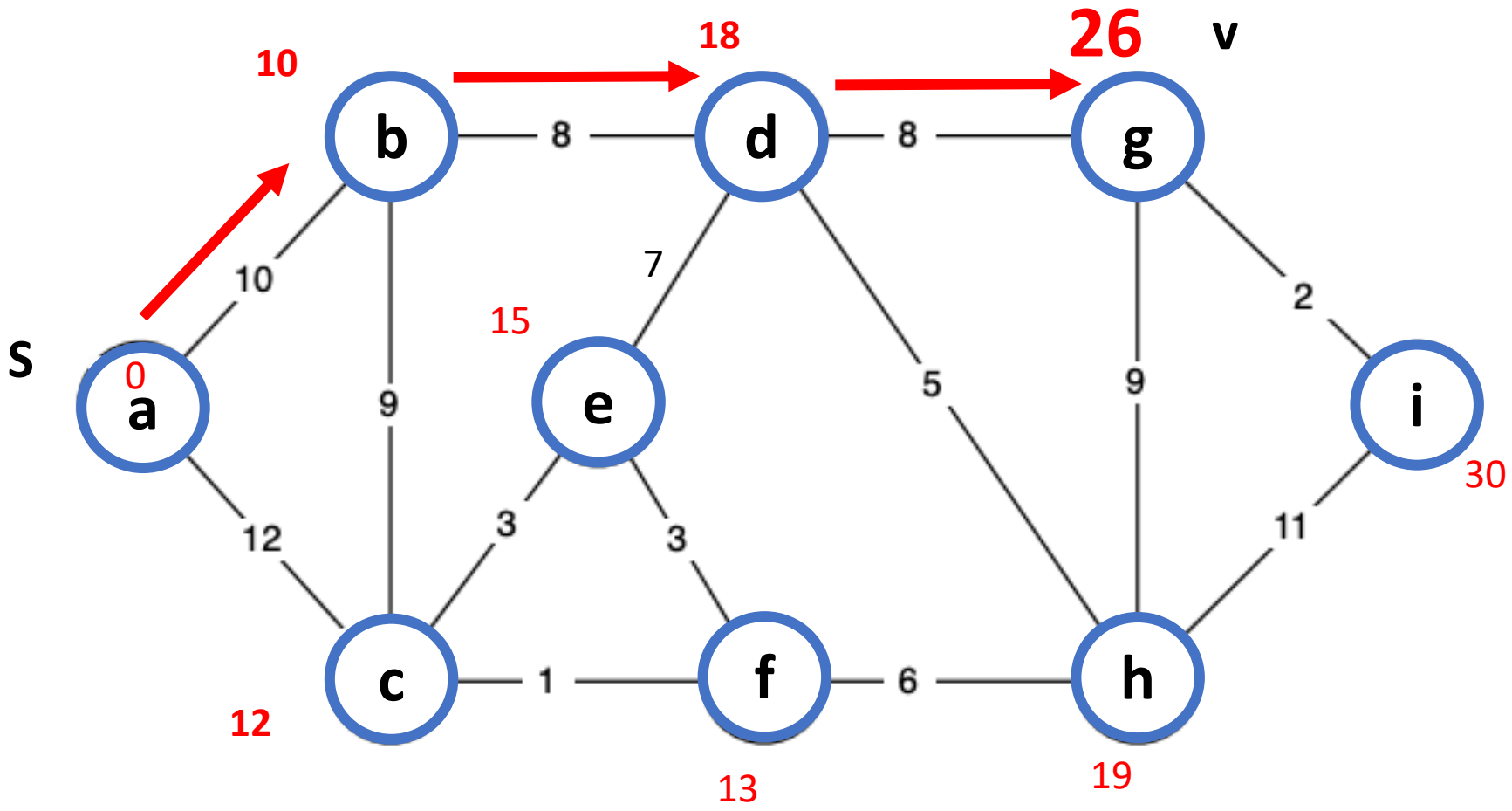
- Definition: δ (s,v) = length of the shortest path from s to v in G



- 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



```
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)
```

```
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)
```

```
PRIM(G = (V, E))
Q ← ∅      ▷ Q is a Priority Queue
Initialize each v ∈ V with key k_v ← ∞, π_v ← NIL
Pick a starting node r and set k_r ← 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 π_v ← u
                        DECREASE-KEY(Q, v, w(u, v))    ▷ Sets k_v ← w(u, v)
```

```
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)
```

(V-1) times

E times

0(ElogV + VlogV)

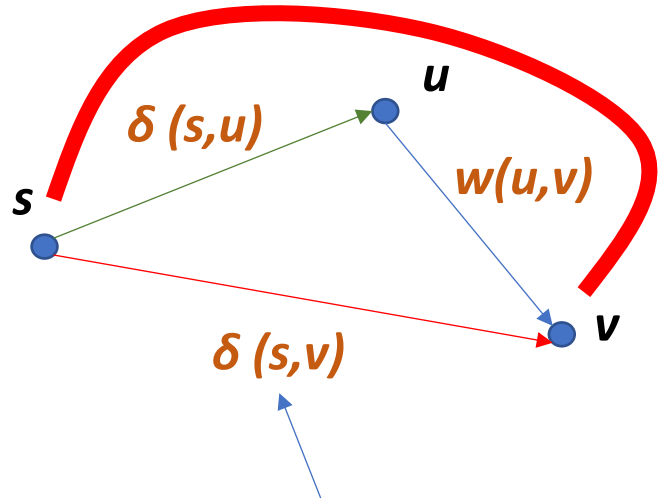
```
DIJKSTRA(G = (V, E), s)
 1  for all v ∈ V
 2        do d_u ← ∞
 3           π_u ← NIL
 4  d_s ← 0
 5  Q ← MAKEQUEUE(V)      ▷ use d_u as key
 6  while Q ≠ ∅
 7        do u ← EXTRACTMIN(Q)
 8           for each v ∈ Adj(u)
 9                do if d_v > d_u + w(u, v)
10                     then d_v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

```
PRIM(G = (V, E))
    Q ← ∅      ▷ Q is a Priority Queue
    Initialize each v ∈ V with key k_v ← ∞, π_v ← NIL
    Pick a starting node r and set k_r ← 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 π_v ← u
                          DECREASE-KEY(Q, v, w(u, v))    ▷ Sets k_v ← 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?



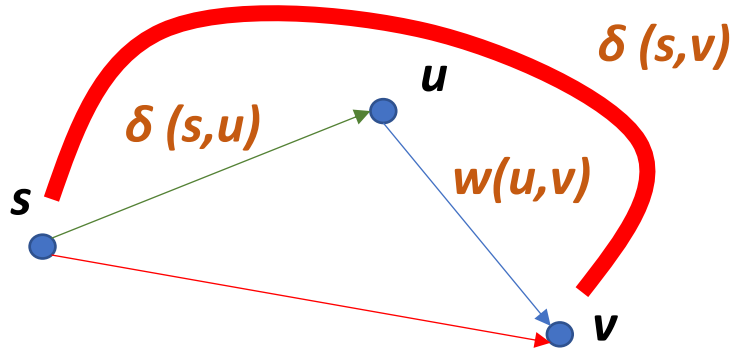
What if this is not the shortest path?

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

Why does Dijkstra work?

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

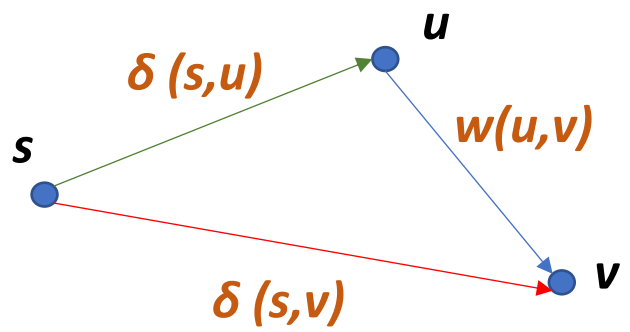
What if this is the shortest path from s-> v?



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

Why does Dijkstra work?

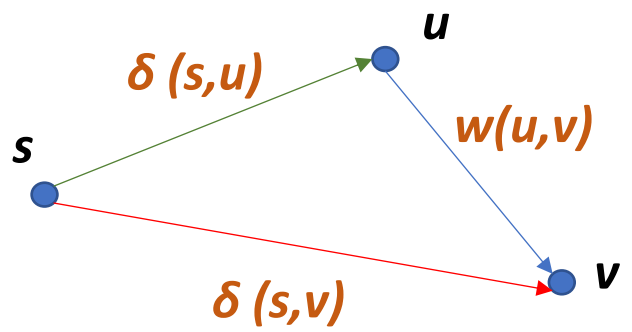
Triangle inequality: $\forall(u,v) \in E, \delta(s,v) <= \delta(s,u)+w(u,v)$



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

Why does Dijkstra work?

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



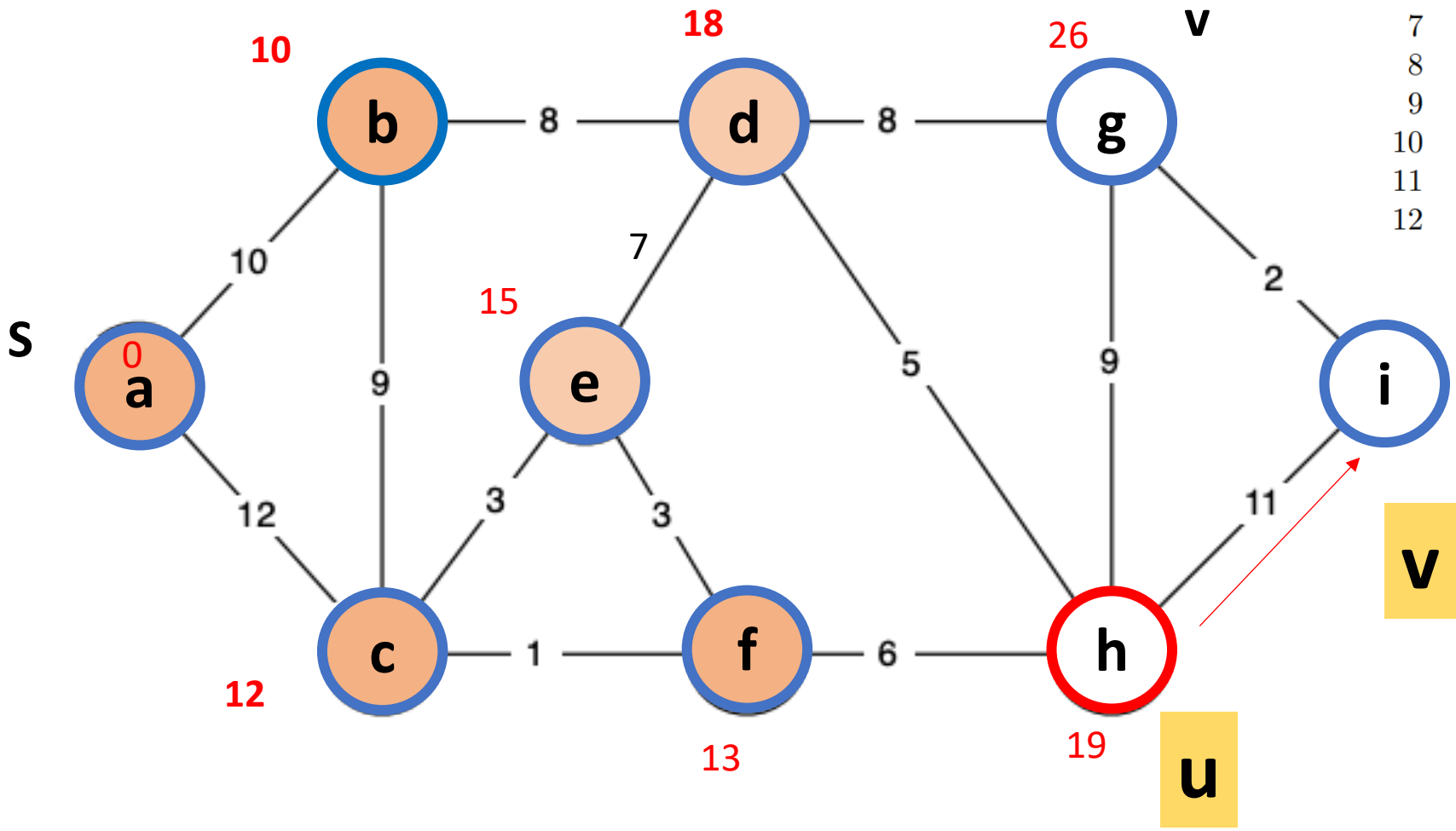
Upper bound: $d_v >= \delta(s,v)$

Follows because, we initiate all d_v with ∞

And we only update d_v by using

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

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

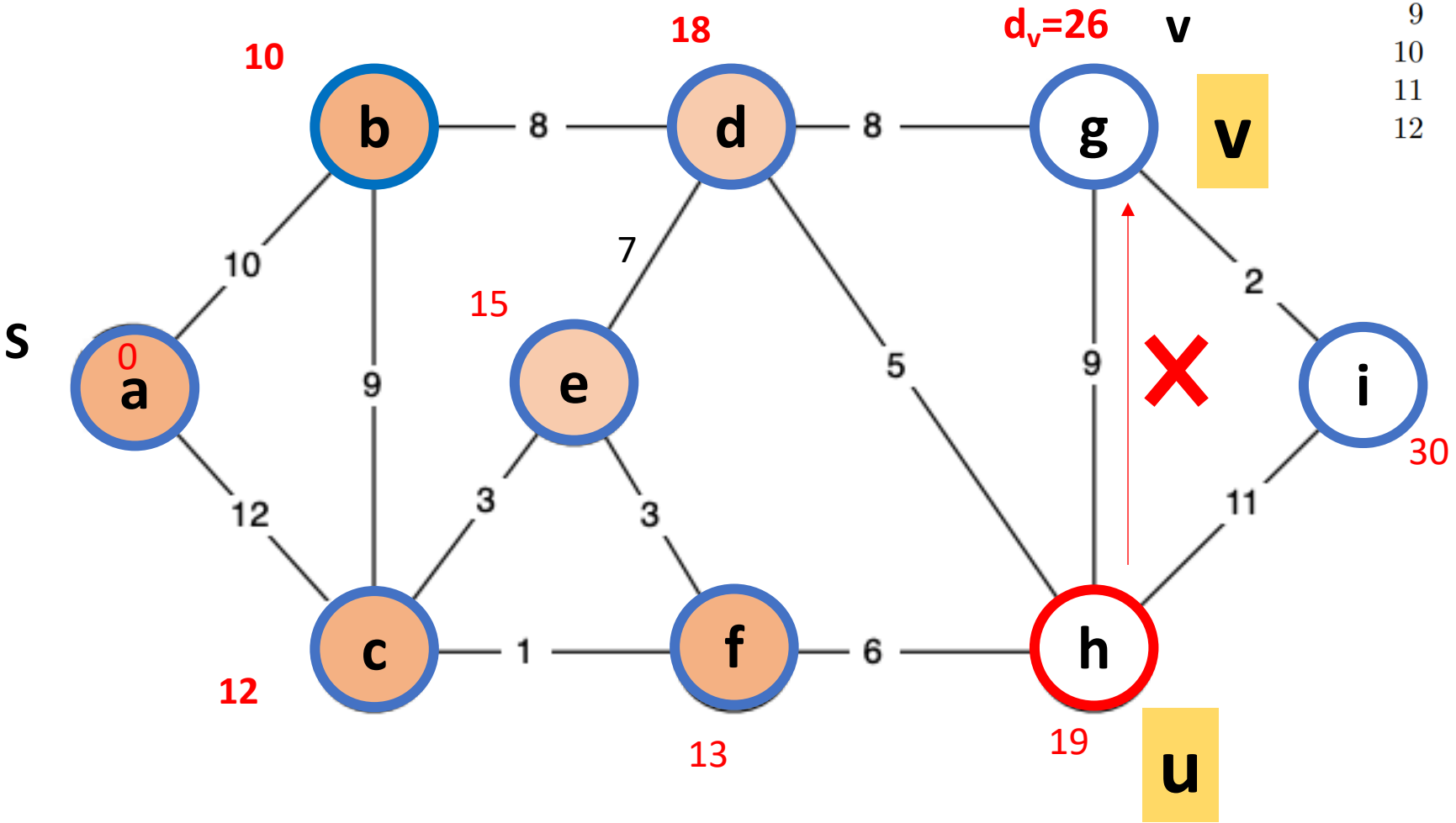



```
DIJKSTRA(G = (V, E), s)
1  for all v ∈ V
2        do d_u ← ∞
3           π_u ← NIL
4  d_s ← 0
5  Q ← MAKEQUEUE(V)        ▷ use d_u as key
6  while Q ≠ ∅
7        do u ← EXTRACTMIN(Q)
8           for each v ∈ Adj(u)
9                 do if d_v > d_u + w(u, v)
10                      then d_v ← d_u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

$d_v > \delta(s,v)$

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

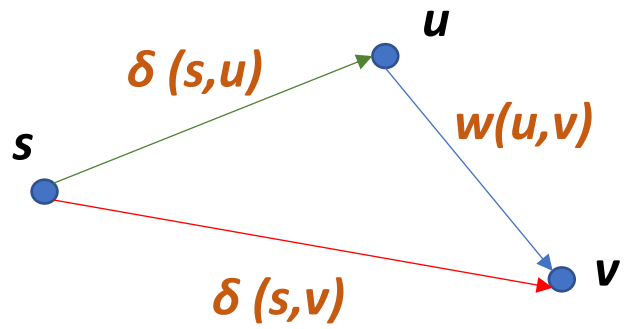
$d_v = \delta(s,v)$




```
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)
```

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)
 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)
```

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 ∈ S, $\mathbf{d_v = \delta(s,v)}$

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, $\mathbf{d_u = \delta(s,u)}$

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 ∈ S, $\mathbf{d_v = \delta(s,v)}$

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, $\mathbf{d_u = \delta(s,u)}$

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

```
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)
```

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, $\mathbf{d_v = \delta(s,v)}$

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

Claim 1: $\mathbf{d_u >= \delta(s,u)}$ *(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**

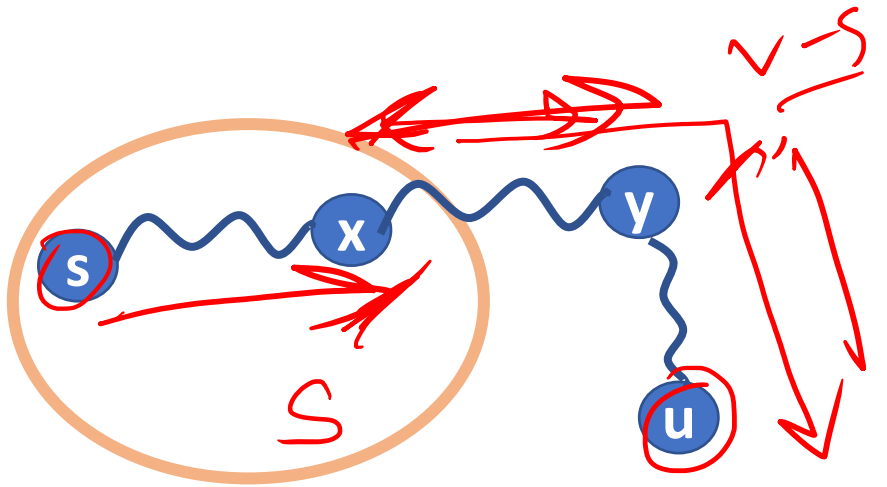
(x,y) is the first edge that cross the cut (S, V-S) where **S** is the Set right before **u** is extracted. **(until iteration i)**

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

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

Any path from y to u

```
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)
```



$d_x = \delta(s,x)$

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)**

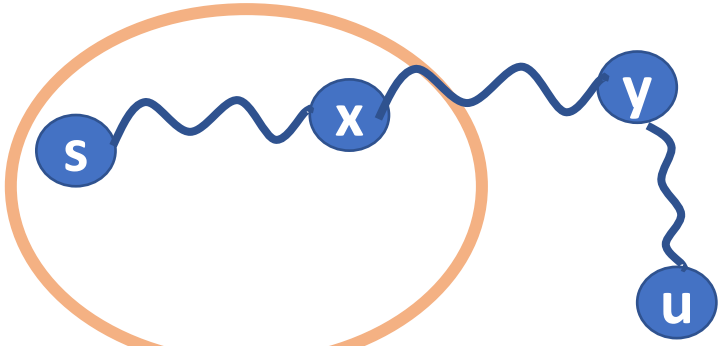
Let **x** be the very last node is **S** along the path from s to u: **path P**

(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)
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)
```

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

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



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 ← $\delta(s,x) + w(x,y)$ (10) >= dy

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**

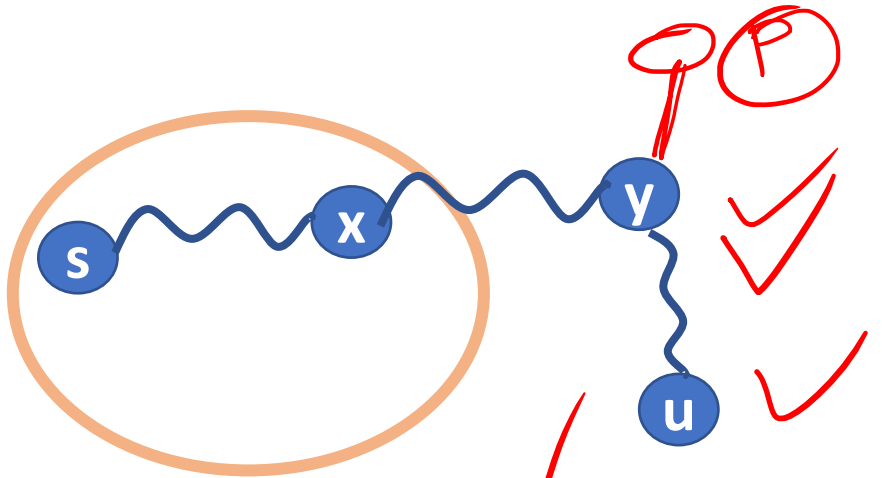
(x,y) is the first edge that cross the cut (S, V-S) where **S** is the Set right before **u** is extracted. **(until iteration i)**

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

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*

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)
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)
```

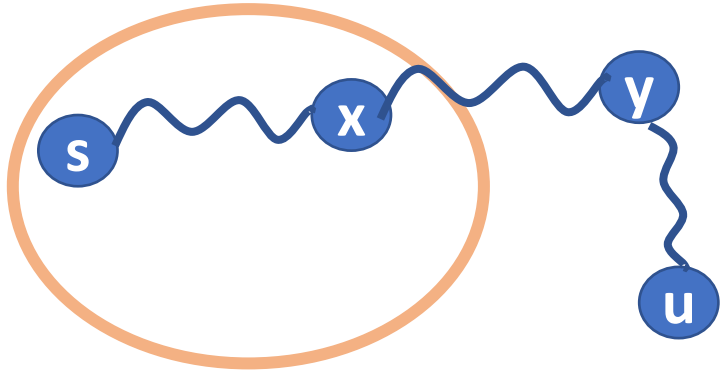
L(p)= *δ (s,x) + w(x,y) + l(y->u)*

>= dy + l(y->u)

>= du + l(y->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)**

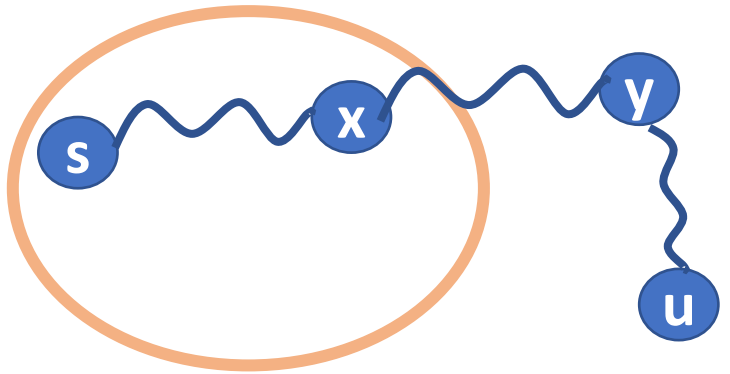
```
DIJKSTRA(G = (V, E), s)
1  for all v ∈ V
2       do d_u ← ∞
3          π_u ← NIL
4  d_s ← 0
5  Q ← MAKEQUEUE(V)        ▷ use d_u as key
6  while Q ≠ ∅
7       do u ← EXTRACTMIN(Q)
8          for each v ∈ Adj(u)
9               do if d_v > d_u + w(u, v)
10                 then d_v ← d_u + w(u, v)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

L(p)= *δ (s,x) + w(x,y) + l(y->u)*

>= dy + l(y->u)

>= du + l(y->u)

>= 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)**

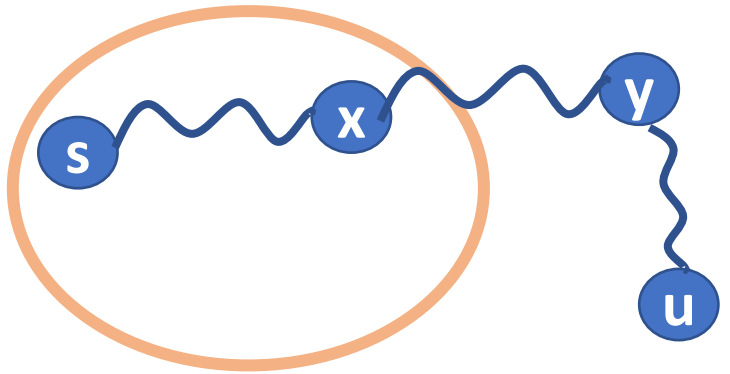
```
DIJKSTRA(G = (V, E), s)
1  for all v ∈ V
2        do d_u ← ∞
3           π_u ← NIL
4  d_s ← 0
5  Q ← MAKEQUEUE(V)        ▷ use d_u as key
6  while Q ≠ ∅
7        do u ← EXTRACTMIN(Q)
8           for each v ∈ Adj(u)
9                 do if d_v > d_u + w(u, v)
10                      then d_v ← d_u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

L(p)= *δ (s,x) + w(x,y) + l(y->u)*

>= dy + l(y->u)

>= du + l(y->u)

>= du



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 >= δ (s,u)***

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

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

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

```
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)
```

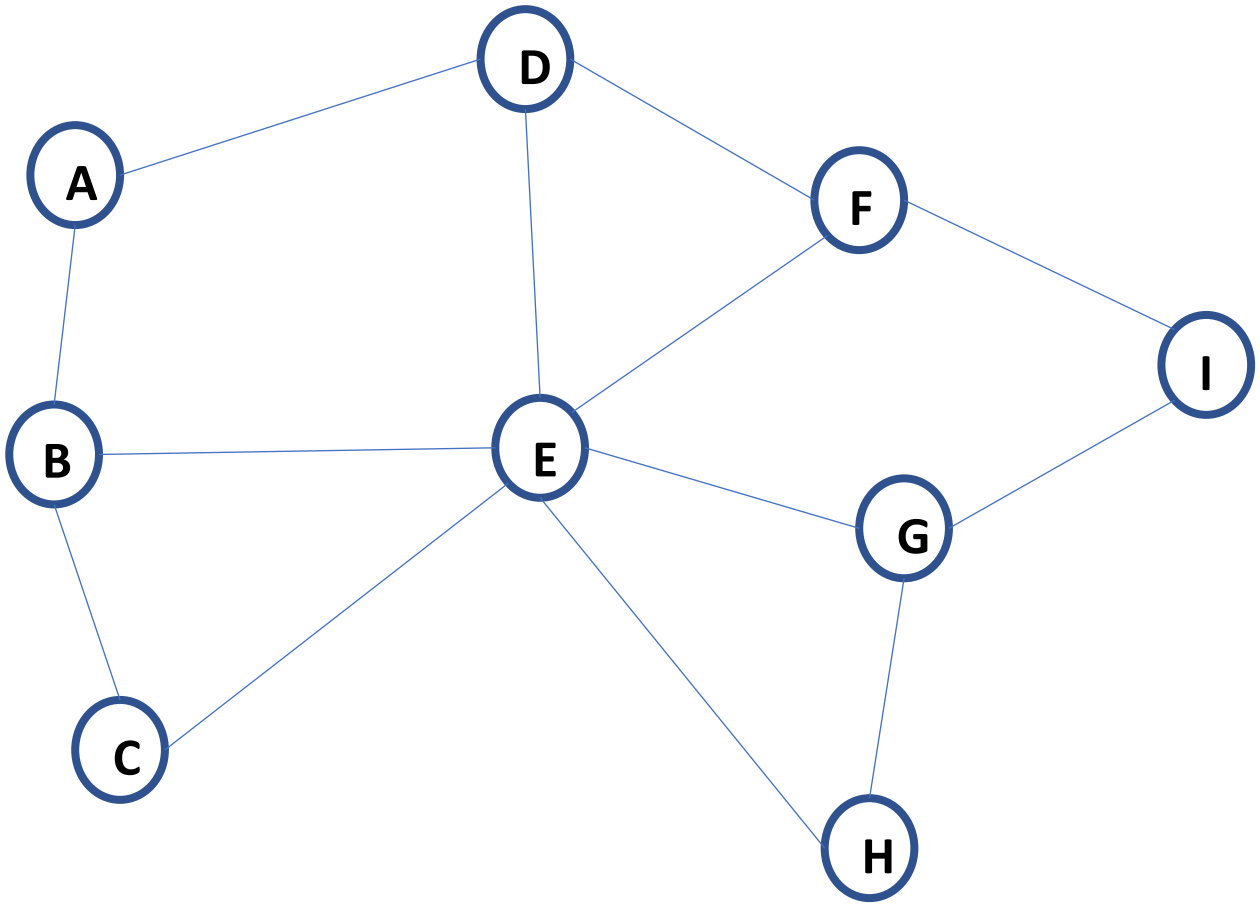
DIJKSTRA

```
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)
```

Breadth first search

- Input : G=(V,E),s
- Output: *∀v ∈ V* $d_v = \delta(s,v)$ *and weights are all 1, w(e) =1*
- *Goal: Smallest number of edges from s to v*

BFS

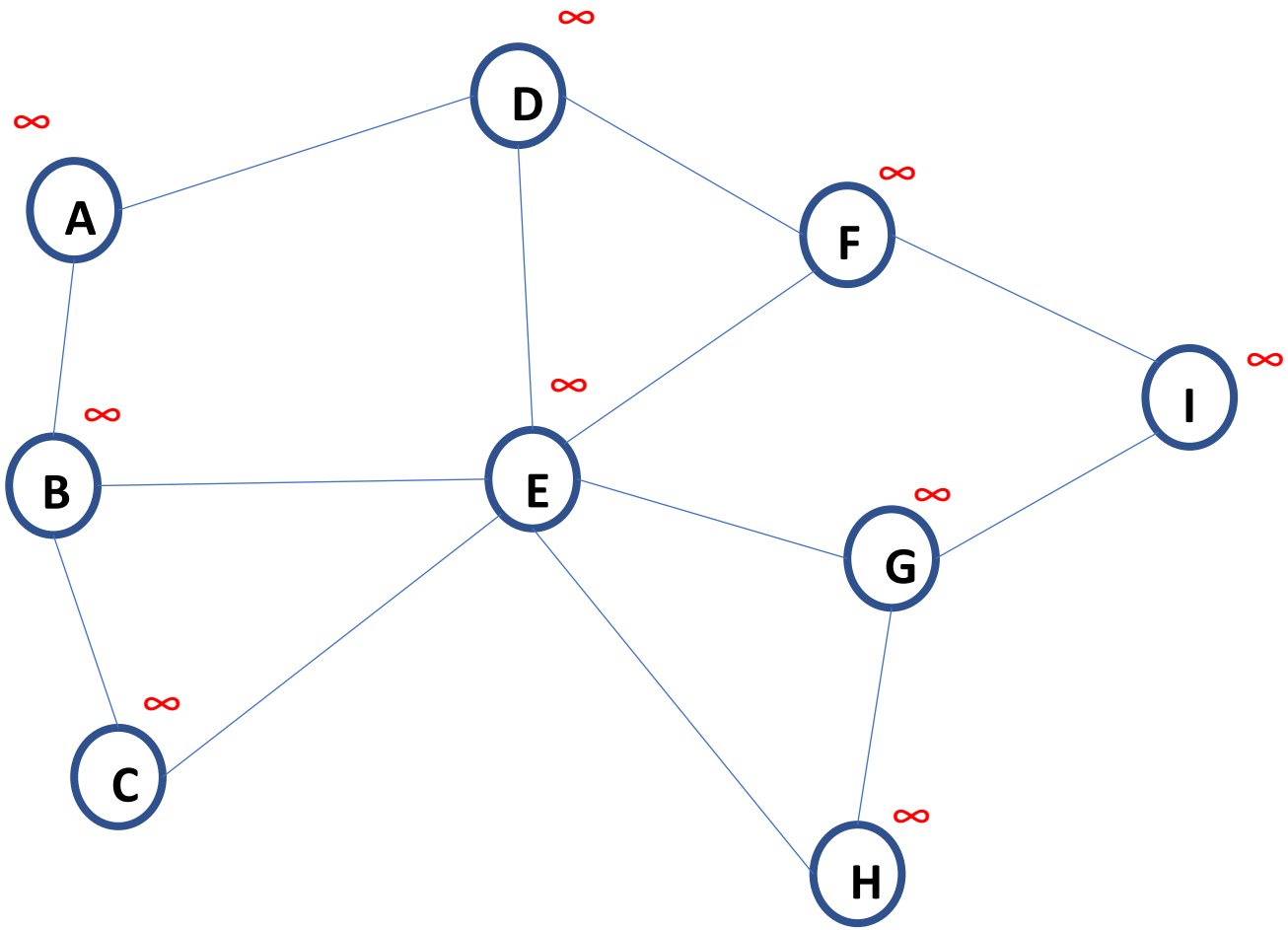


Queue

```
BFS(V, E, s)
for each u ∈ V − {s}
    do d[u] ← ∞
d[s] ← 0
Q ← ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    do u ← DEQUEUE(Q)
        for each v ∈ Adj[u]
            do if d[v] = ∞
                then d[v] ← d[u] + 1
                     ENQUEUE(Q, v)
```

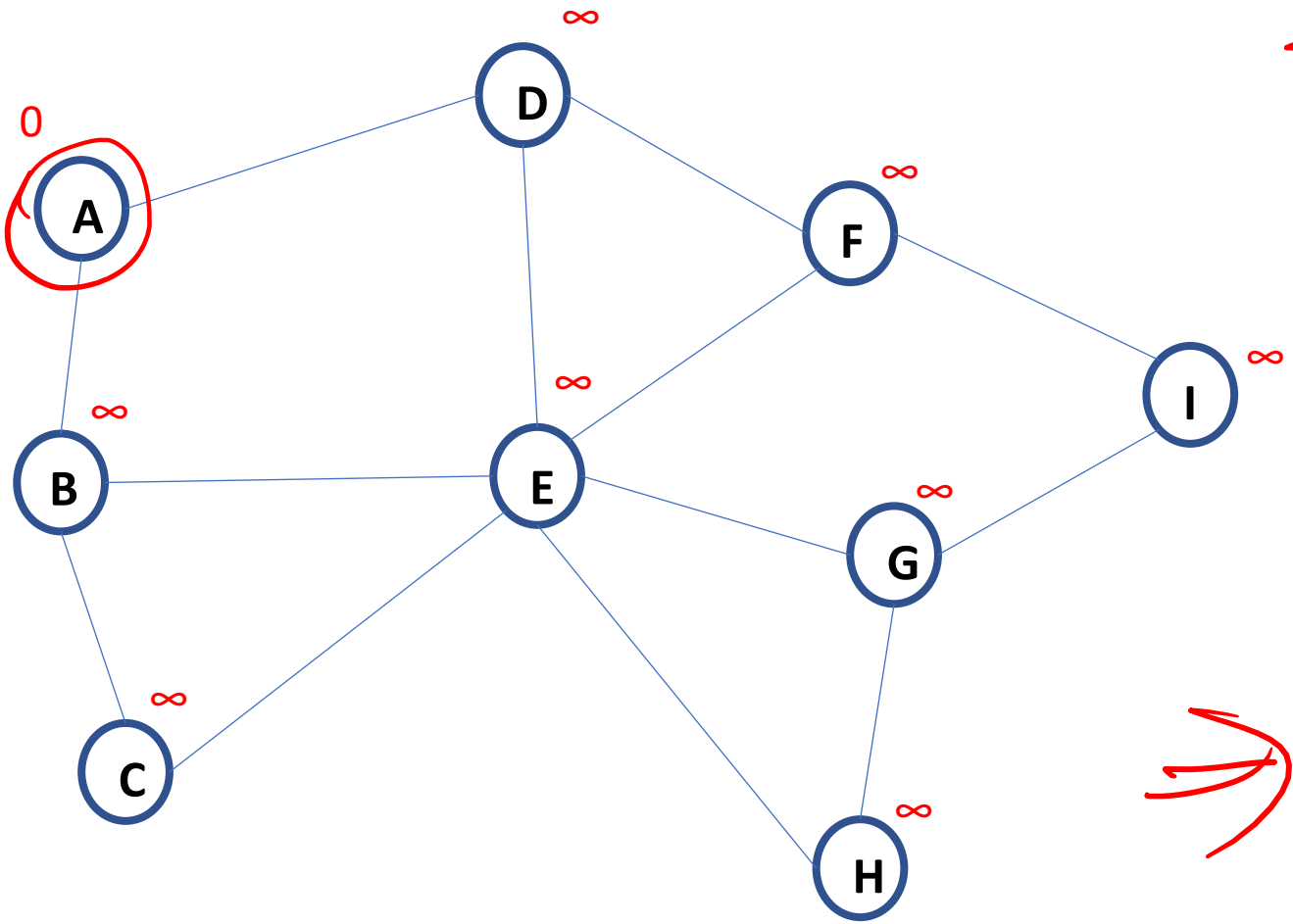
```
DIJKSTRA(G = (V, E), s)
1   for all v ∈ V
2         do d_u ← ∞
3            π_u ← NIL
4   d_s ← 0
5   Q ← MAKEQUEUE(V)        ▷ use d_u as key
6   while Q ≠ ∅
7         do u ← EXTRACTMIN(Q)
8            for each v ∈ Adj(u)
9                 do if d_v > d_u + w(u, v)
10                   then d_v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

BFS




```
BFS(V, E, s)
for each u ∈ V − {s}
    do d[u] ← ∞
d[s] ← 0
Q ← ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    do u ← DEQUEUE(Q)
        for each v ∈ Adj[u]
            do if d[v] = ∞
                then d[v] ← d[u] + 1
                     ENQUEUE(Q, v)
```

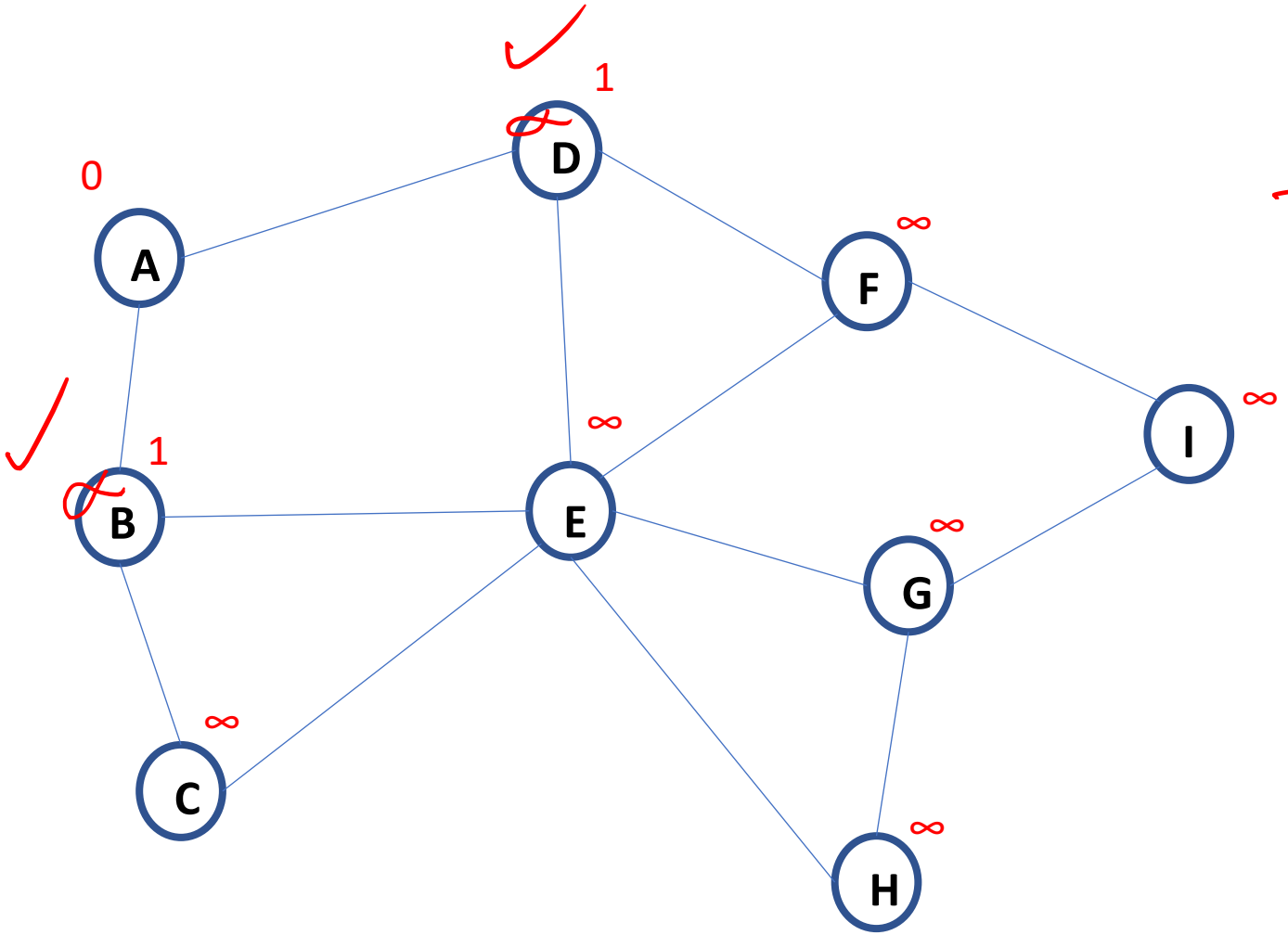
BFS




```
BFS(V, E, s)
for each u ∈ V − {s}
    do d[u] ← ∞
d[s] ← 0
Q ← ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    do u ← DEQUEUE(Q)
        for each v ∈ Adj[u]
            do if d[v] = ∞
                then d[v] ← d[u] + 1
                     ENQUEUE(Q, v)
```

Q: A

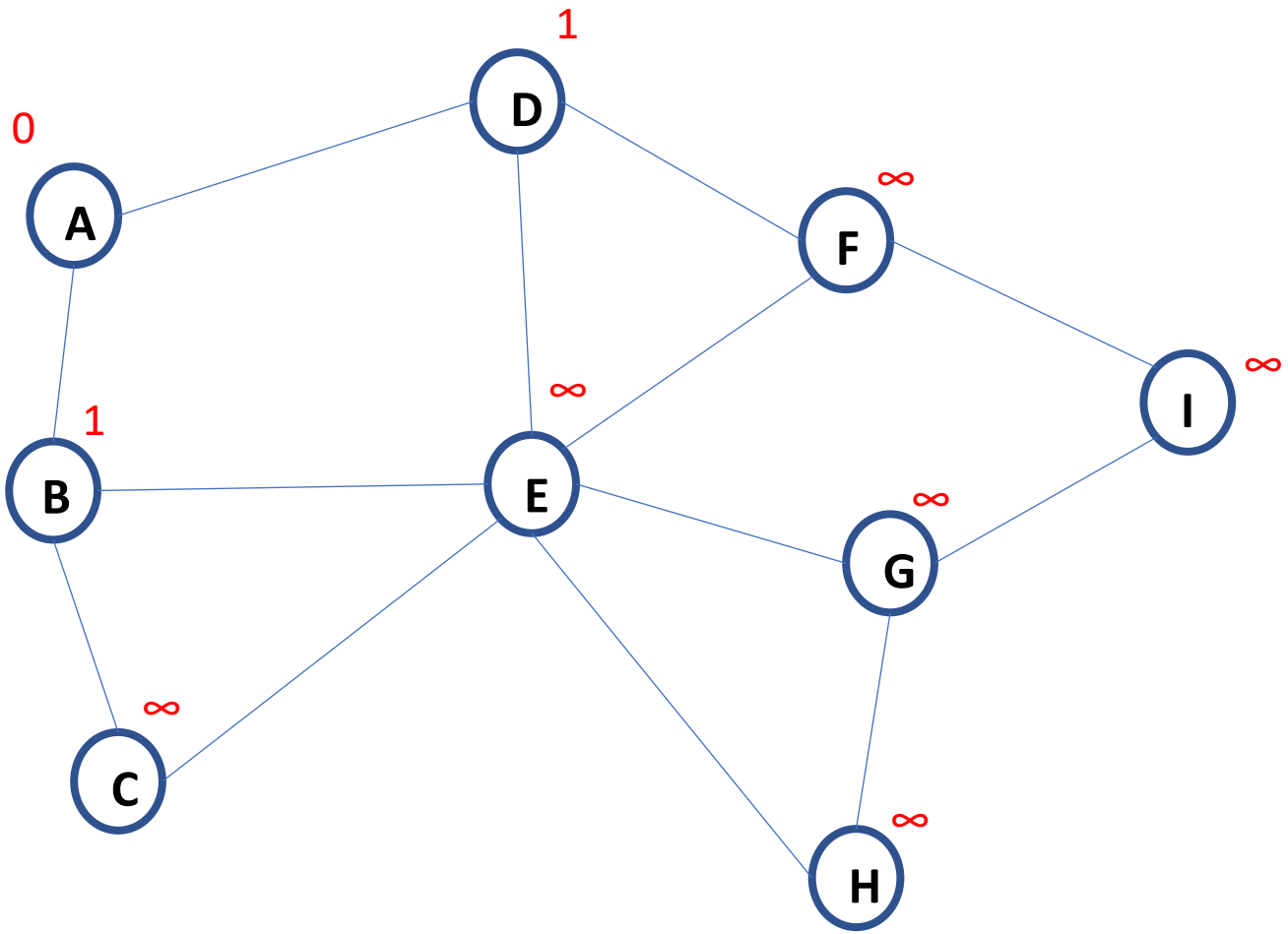
BFS




```
BFS(V, E, s)
for each u ∈ V − {s}
    do d[u] ← ∞
d[s] ← 0
Q ← ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    do u ← DEQUEUE(Q)
       for each v ∈ Adj[u]
           do if d[v] = ∞
              then d[v] ← d[u] + 1
                   ENQUEUE(Q, v)
```

Q:

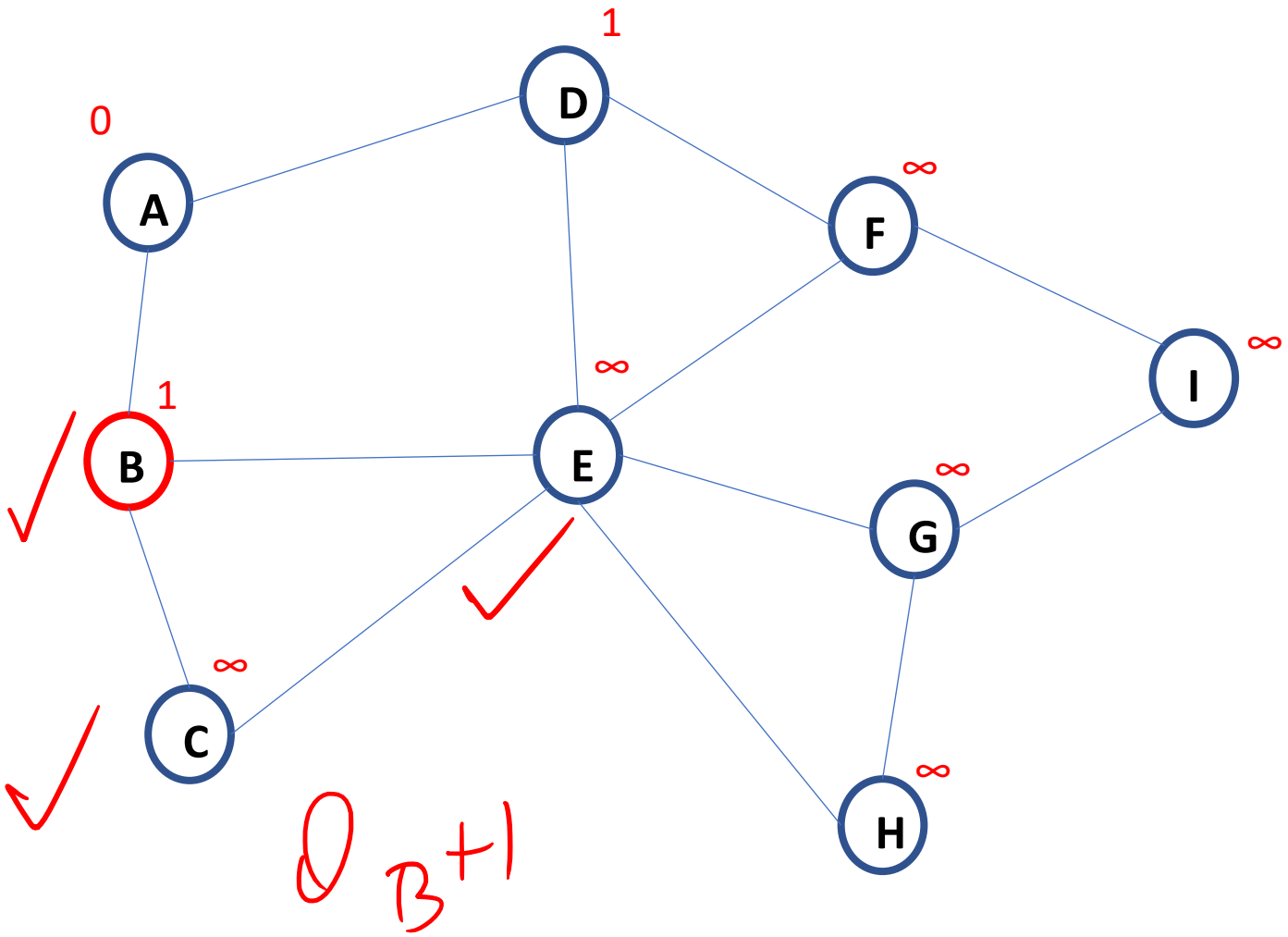
BFS




```
BFS(V, E, s)
for each u ∈ V − {s}
    do d[u] ← ∞
d[s] ← 0
Q ← ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    do u ← DEQUEUE(Q)
        for each v ∈ Adj[u]
            do if d[v] = ∞
                then d[v] ← d[u] + 1
                     ENQUEUE(Q, v)
```

Q: ~~A~~ B D

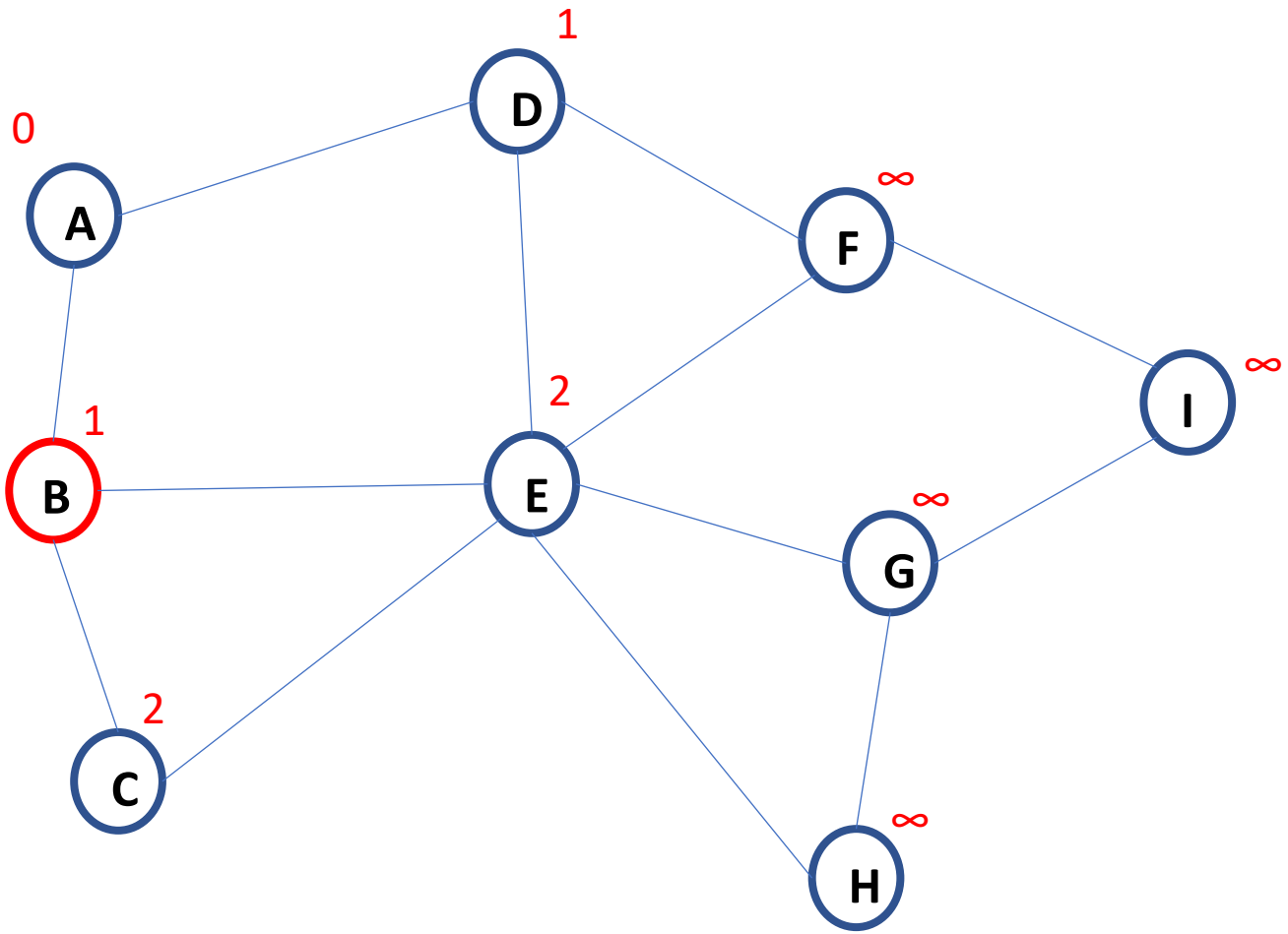
BFS




```
BFS(V, E, s)
for each u ∈ V − {s}
    do d[u] ← ∞
d[s] ← 0
Q ← ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    do u ← DEQUEUE(Q)
        for each v ∈ Adj[u]
            do if d[v] = ∞
                then d[v] ← d[u] + 1
                     ENQUEUE(Q, v)
```

Q: ~~A C E~~ D

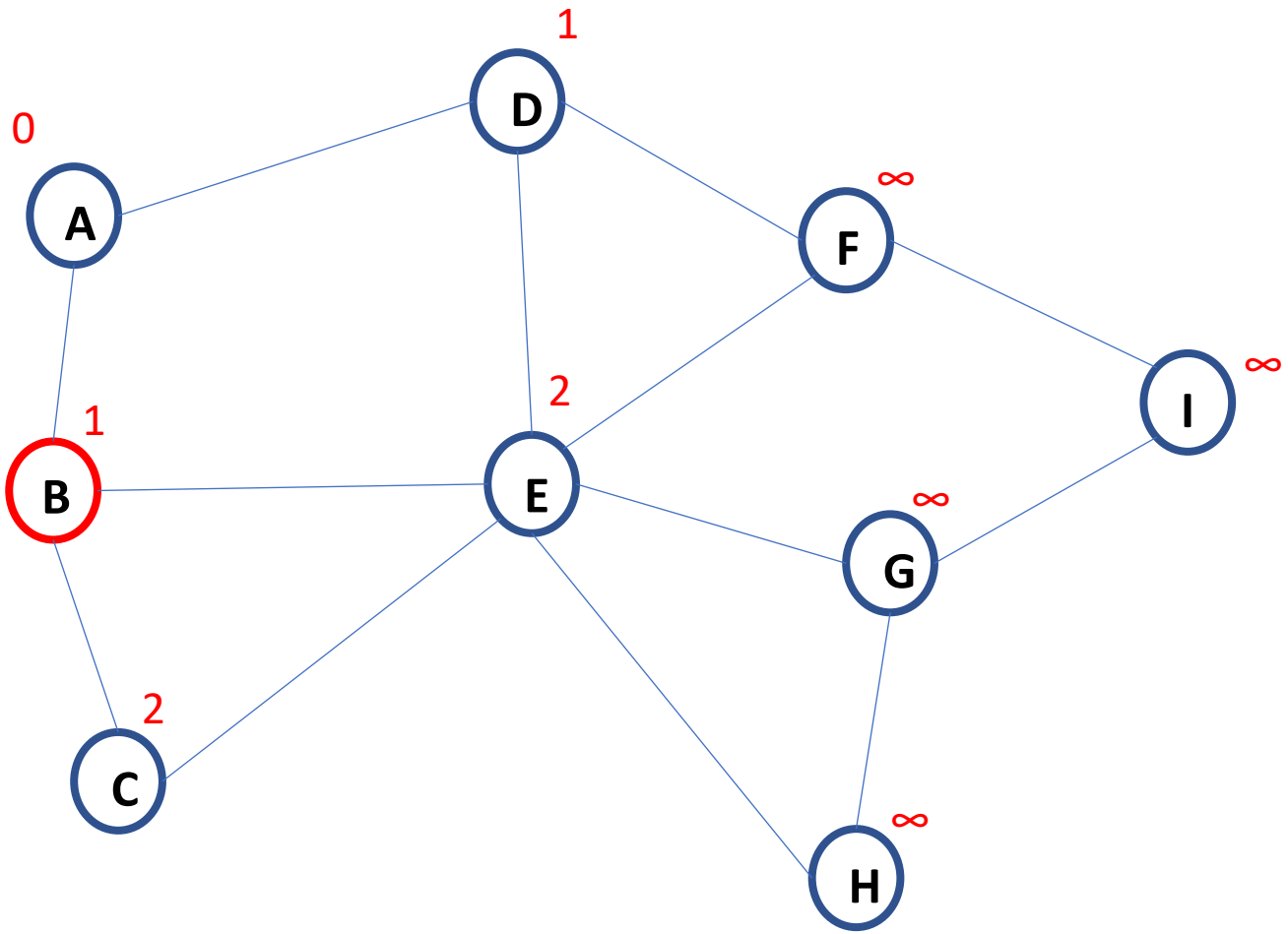
BFS




```
BFS(V, E, s)
for each u ∈ V − {s}
      do d[u] ← ∞
d[s] ← 0
Q ← ∅
ENQUEUE(Q, s)
while Q ≠ ∅
      do u ← DEQUEUE(Q)
         for each v ∈ Adj[u]
               do if d[v] = ∞
                     then d[v] ← d[u] + 1
                          ENQUEUE(Q, v)
```

Q: ~~A C E~~ D

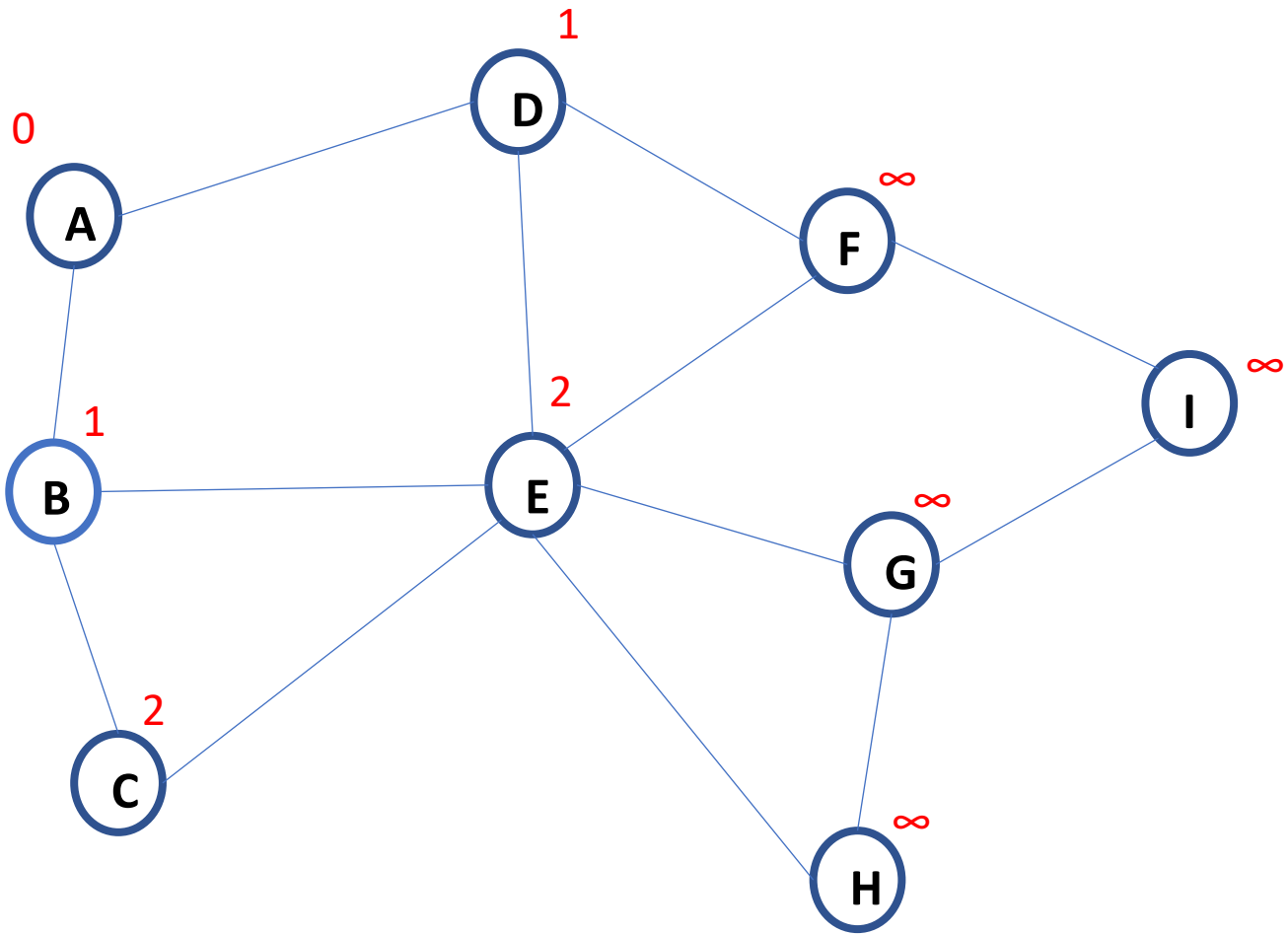
BFS




```
BFS(V, E, s)
for each u ∈ V − {s}
    do d[u] ← ∞
d[s] ← 0
Q ← ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    do u ← DEQUEUE(Q)
        for each v ∈ Adj[u]
            do if d[v] = ∞
                then d[v] ← d[u] + 1
                     ENQUEUE(Q, v)
```

Q: ~~A B~~ D C E

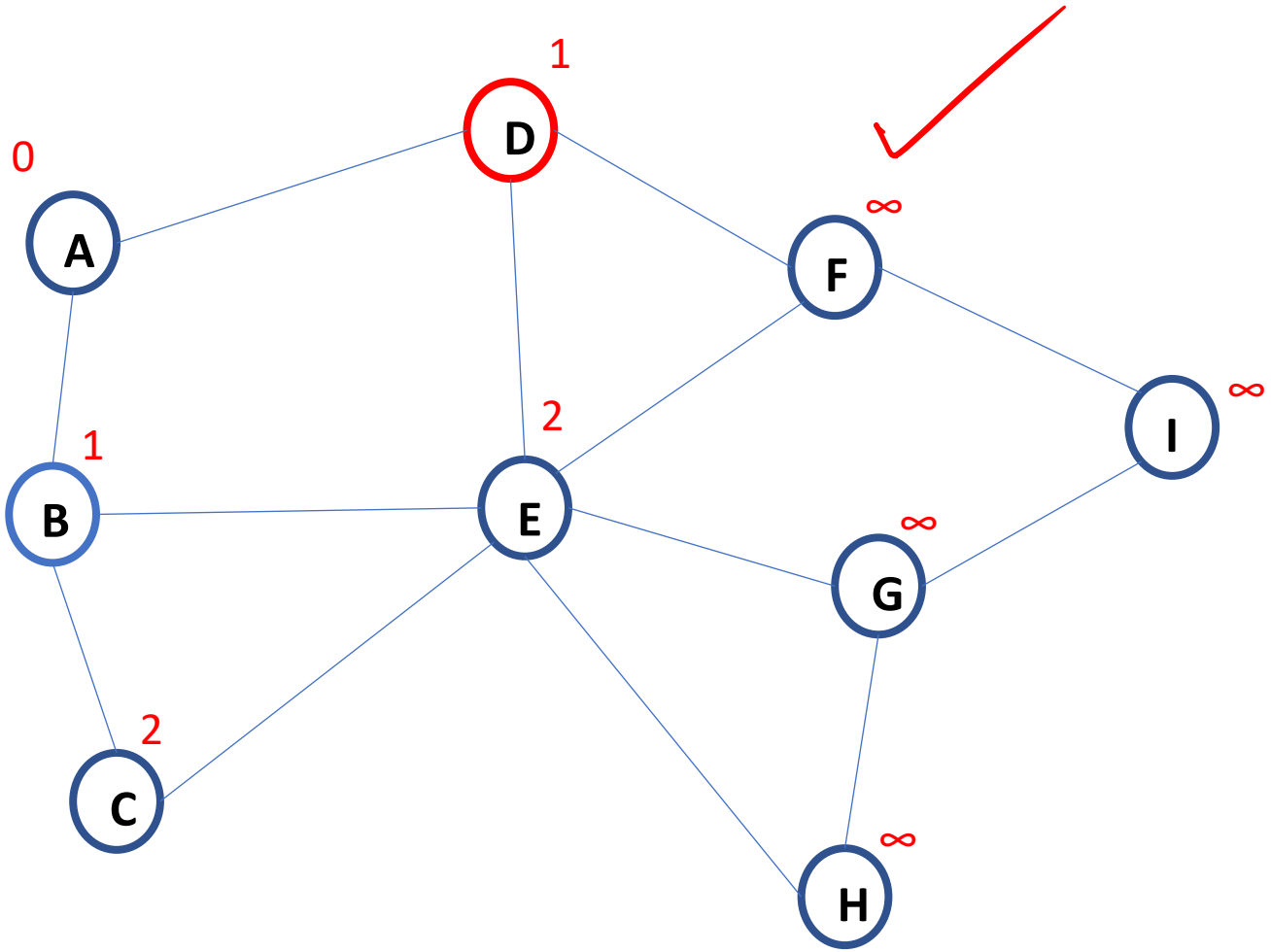
BFS




```
BFS(V, E, s)
for each u ∈ V − {s}
      do d[u] ← ∞
d[s] ← 0
Q ← ∅
ENQUEUE(Q, s)
while Q ≠ ∅
      do u ← DEQUEUE(Q)
         for each v ∈ Adj[u]
               do if d[v] = ∞
                     then d[v] ← d[u] + 1
                          ENQUEUE(Q, v)
```

Q: ~~A B~~ D C E

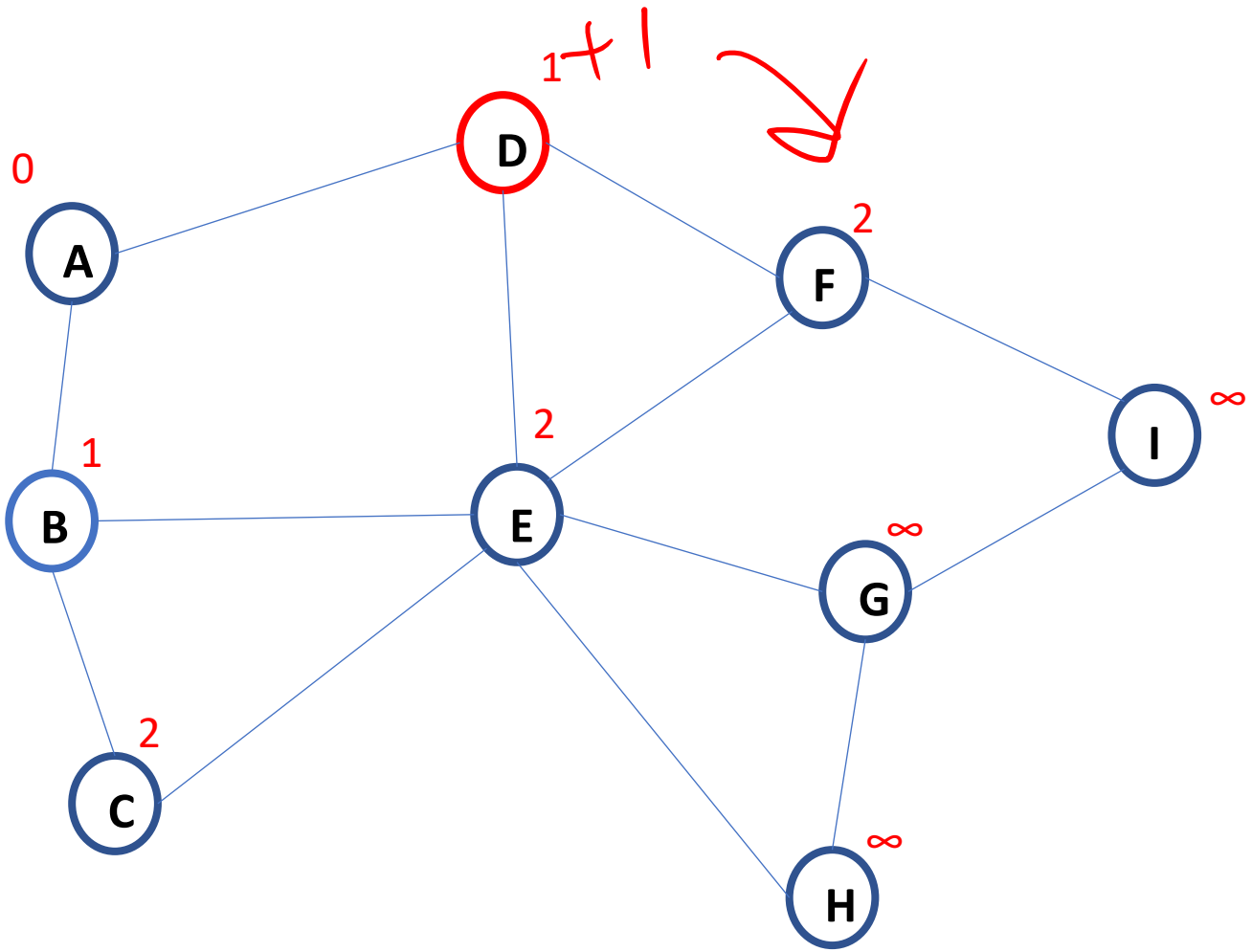
BFS




```
BFS(V, E, s)
for each u ∈ V − {s}
      do d[u] ← ∞
d[s] ← 0
Q ← ∅
ENQUEUE(Q, s)
while Q ≠ ∅
      do u ← DEQUEUE(Q)
         for each v ∈ Adj[u]
               do if d[v] = ∞
                     then d[v] ← d[u] + 1
                          ENQUEUE(Q, v)
```

Q: ~~A D B~~ C E

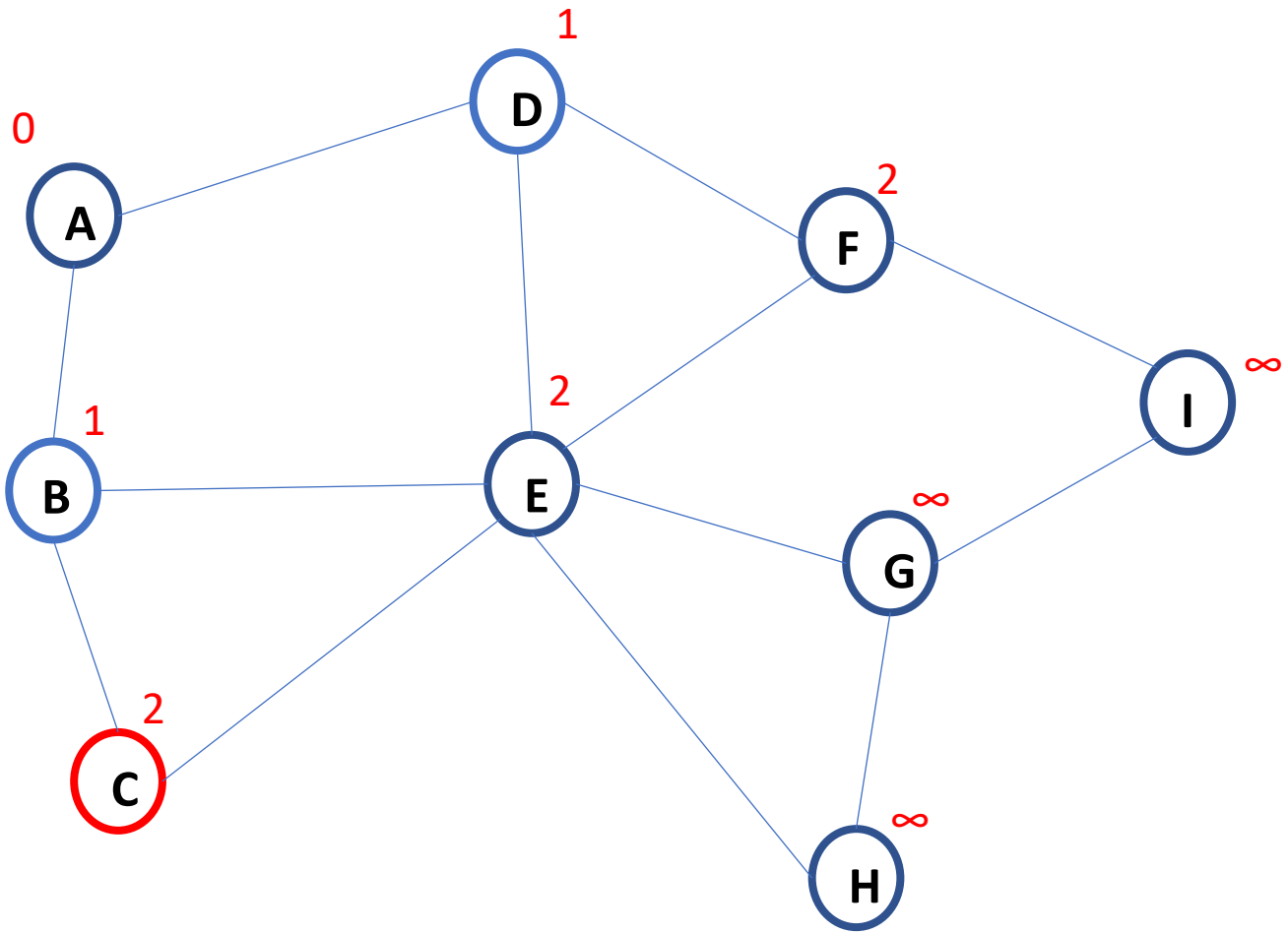
BFS




```
BFS(V, E, s)
for each u ∈ V − {s}
    do d[u] ← ∞
d[s] ← 0
Q ← ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    do u ← DEQUEUE(Q)
        for each v ∈ Adj[u]
            do if d[v] = ∞
                then d[v] ← d[u] + 1
                    ENQUEUE(Q, v)
```

Q: ~~A D B~~ C E F

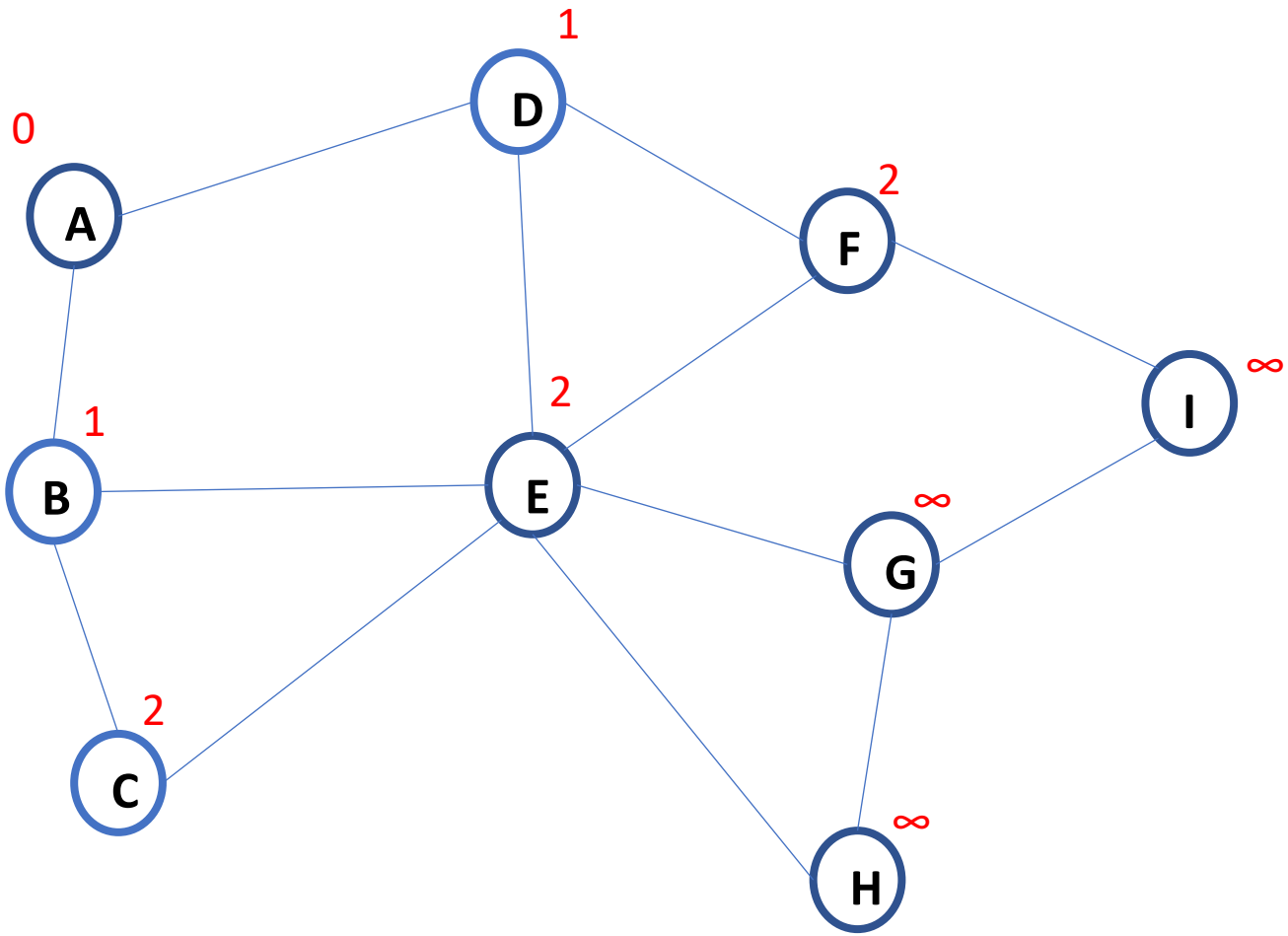
BFS




```
BFS(V, E, s)
for each u ∈ V − {s}
    do d[u] ← ∞
d[s] ← 0
Q ← ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    do u ← DEQUEUE(Q)
        for each v ∈ Adj[u]
            do if d[v] = ∞
                then d[v] ← d[u] + 1
                     ENQUEUE(Q, v)
```

Q: ~~A E B C~~ E F

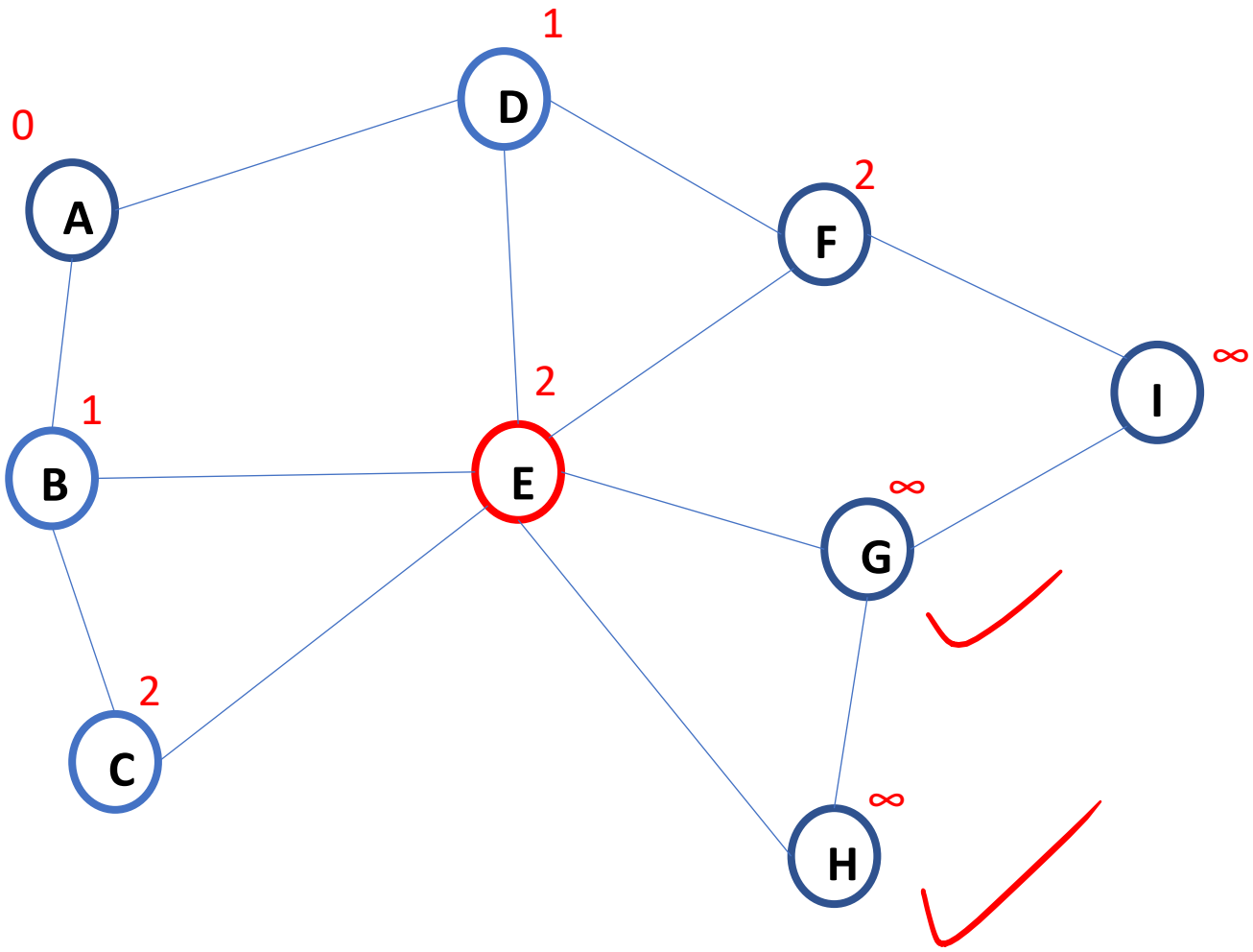
BFS




```
BFS(V, E, s)
for each u ∈ V − {s}
    do d[u] ← ∞
d[s] ← 0
Q ← ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    do u ← DEQUEUE(Q)
        for each v ∈ Adj[u]
            do if d[v] = ∞
                then d[v] ← d[u] + 1
                    ENQUEUE(Q, v)
```

Q: ~~A B D C~~ E F

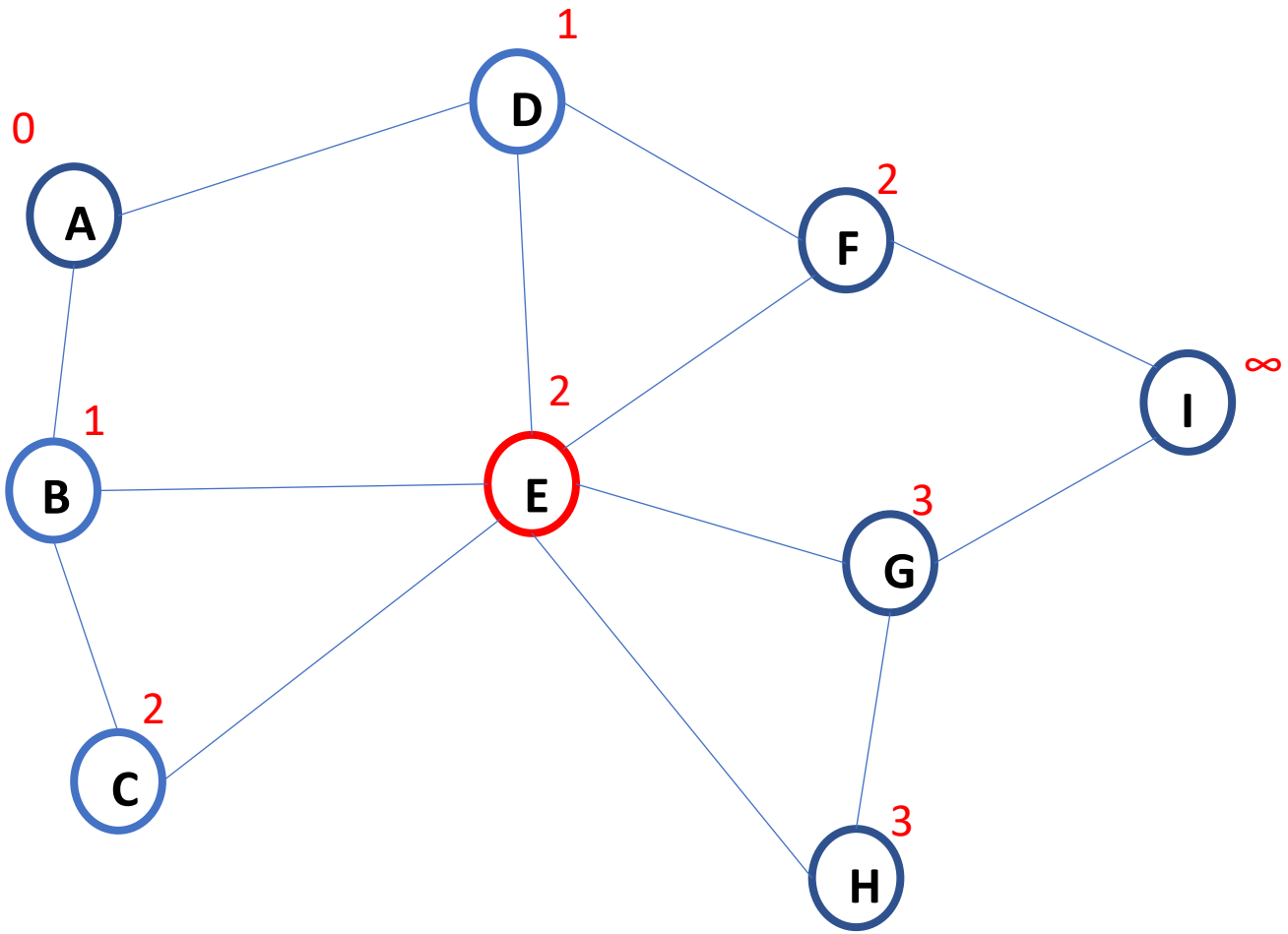
BFS




```
BFS(V, E, s)
for each u ∈ V − {s}
    do d[u] ← ∞
d[s] ← 0
Q ← ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    do u ← DEQUEUE(Q)
        for each v ∈ Adj[u]
            do if d[v] = ∞
                then d[v] ← d[u] + 1
                    ENQUEUE(Q, v)
```

Q: ~~A E B C E~~ F

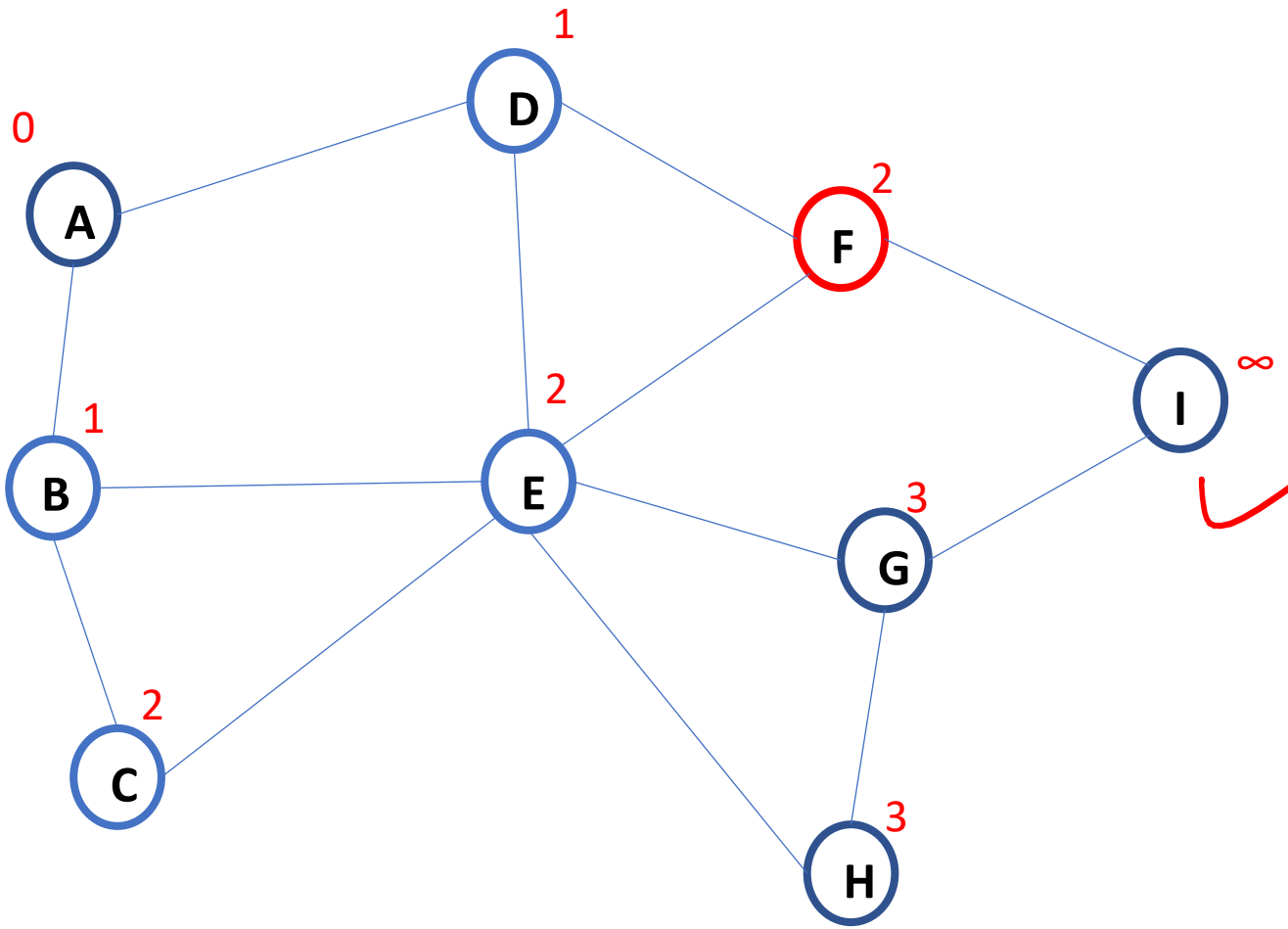
BFS




```
BFS(V, E, s)
for each u ∈ V − {s}
      do d[u] ← ∞
d[s] ← 0
Q ← ∅
ENQUEUE(Q, s)
while Q ≠ ∅
      do u ← DEQUEUE(Q)
         for each v ∈ Adj[u]
             do if d[v] = ∞
                   then d[v] ← d[u] + 1
                        ENQUEUE(Q, v)
```

Q: ~~A E B C E~~ F G H

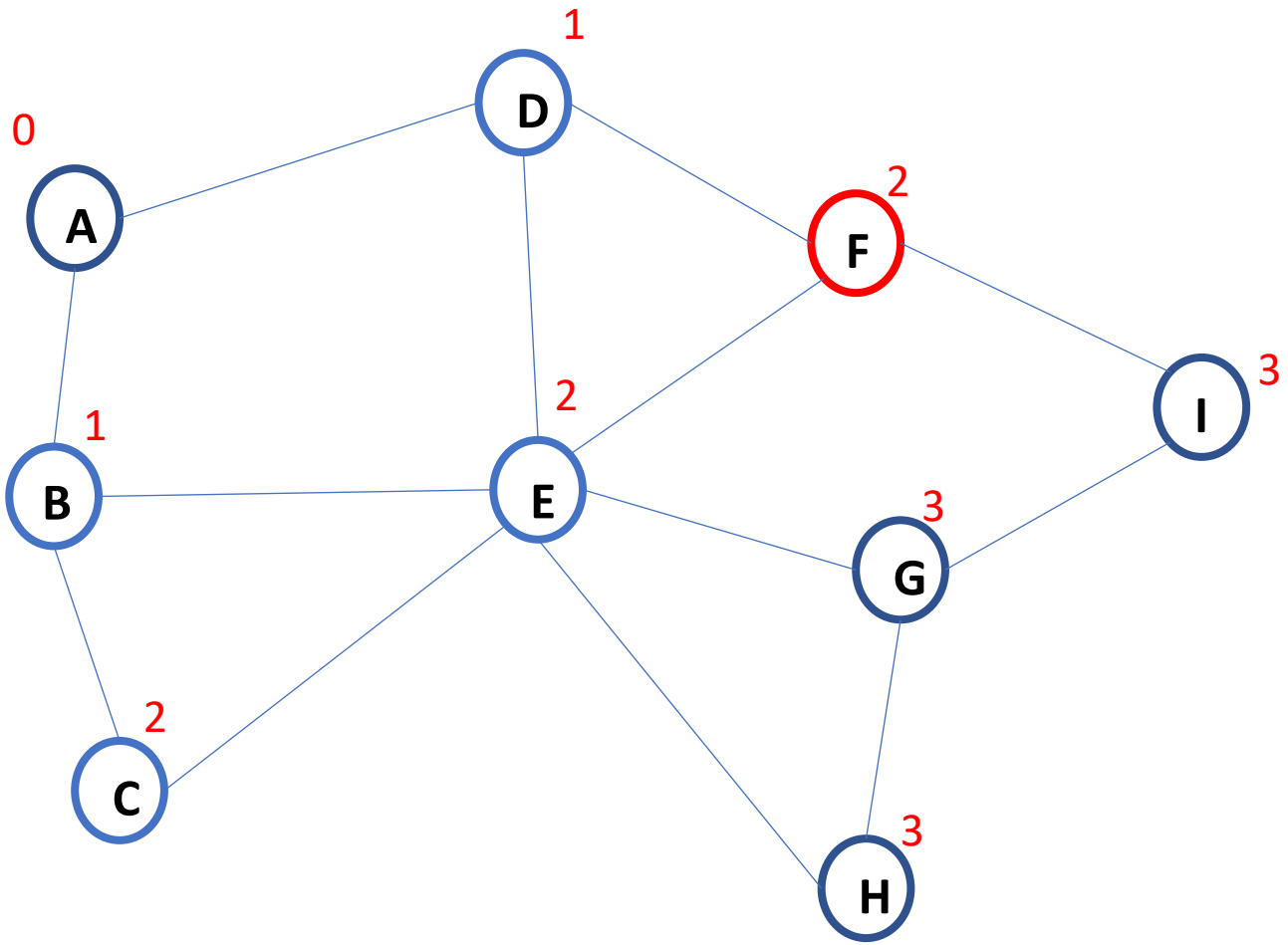
BFS




```
BFS(V, E, s)
for each u ∈ V − {s}
      do d[u] ← ∞
d[s] ← 0
Q ← ∅
ENQUEUE(Q, s)
while Q ≠ ∅
      do u ← DEQUEUE(Q)
         for each v ∈ Adj[u]
             do if d[v] = ∞
                   then d[v] ← d[u] + 1
                        ENQUEUE(Q, v)
```

Q: ~~A B D C E F~~ G H

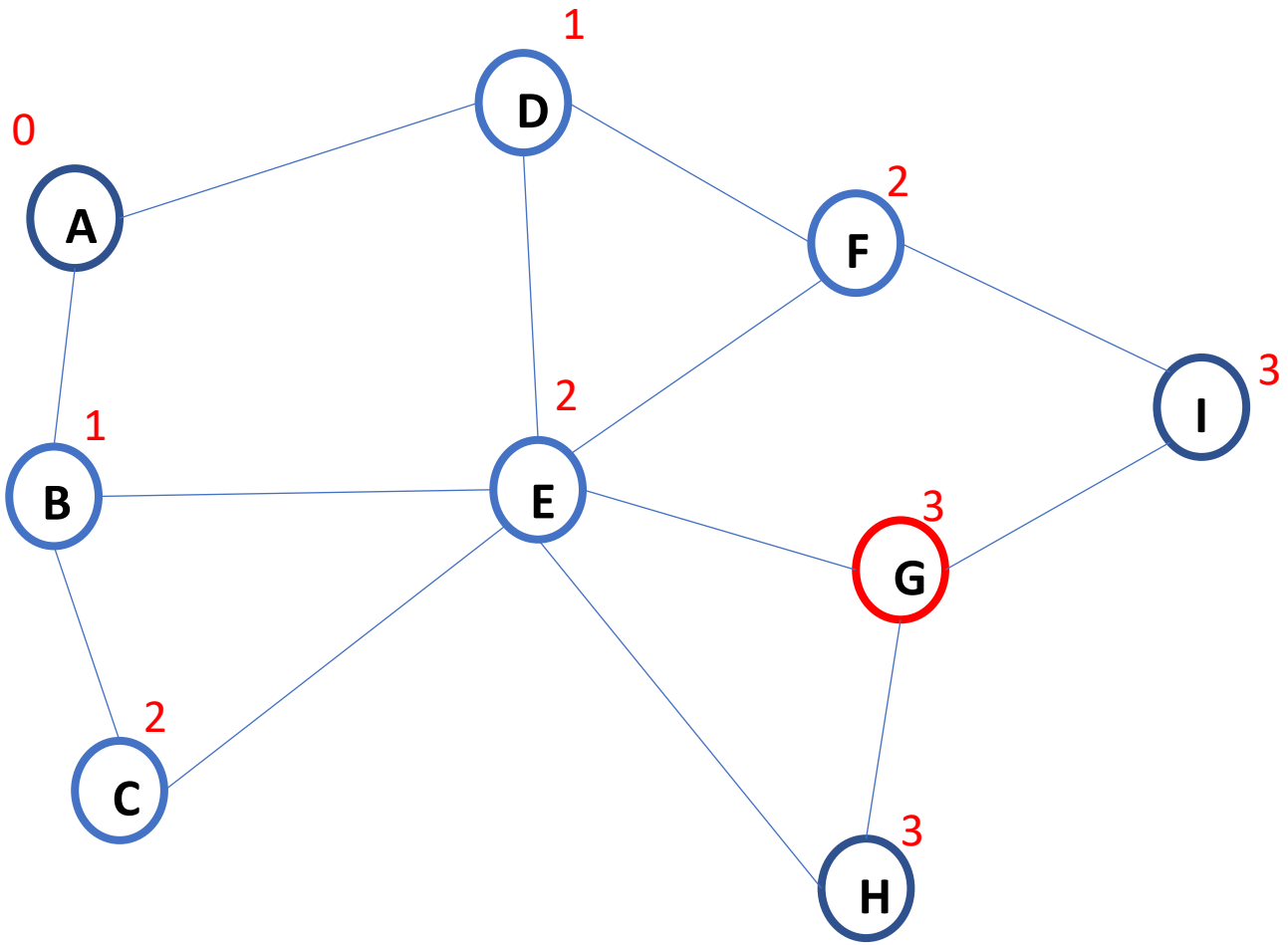
BFS




```
BFS(V, E, s)
for each u ∈ V − {s}
      do d[u] ← ∞
d[s] ← 0
Q ← ∅
ENQUEUE(Q, s)
while Q ≠ ∅
      do u ← DEQUEUE(Q)
         for each v ∈ Adj[u]
             do if d[v] = ∞
                   then d[v] ← d[u] + 1
                        ENQUEUE(Q, v)
```

Q: ~~A B D C E F~~ G H I

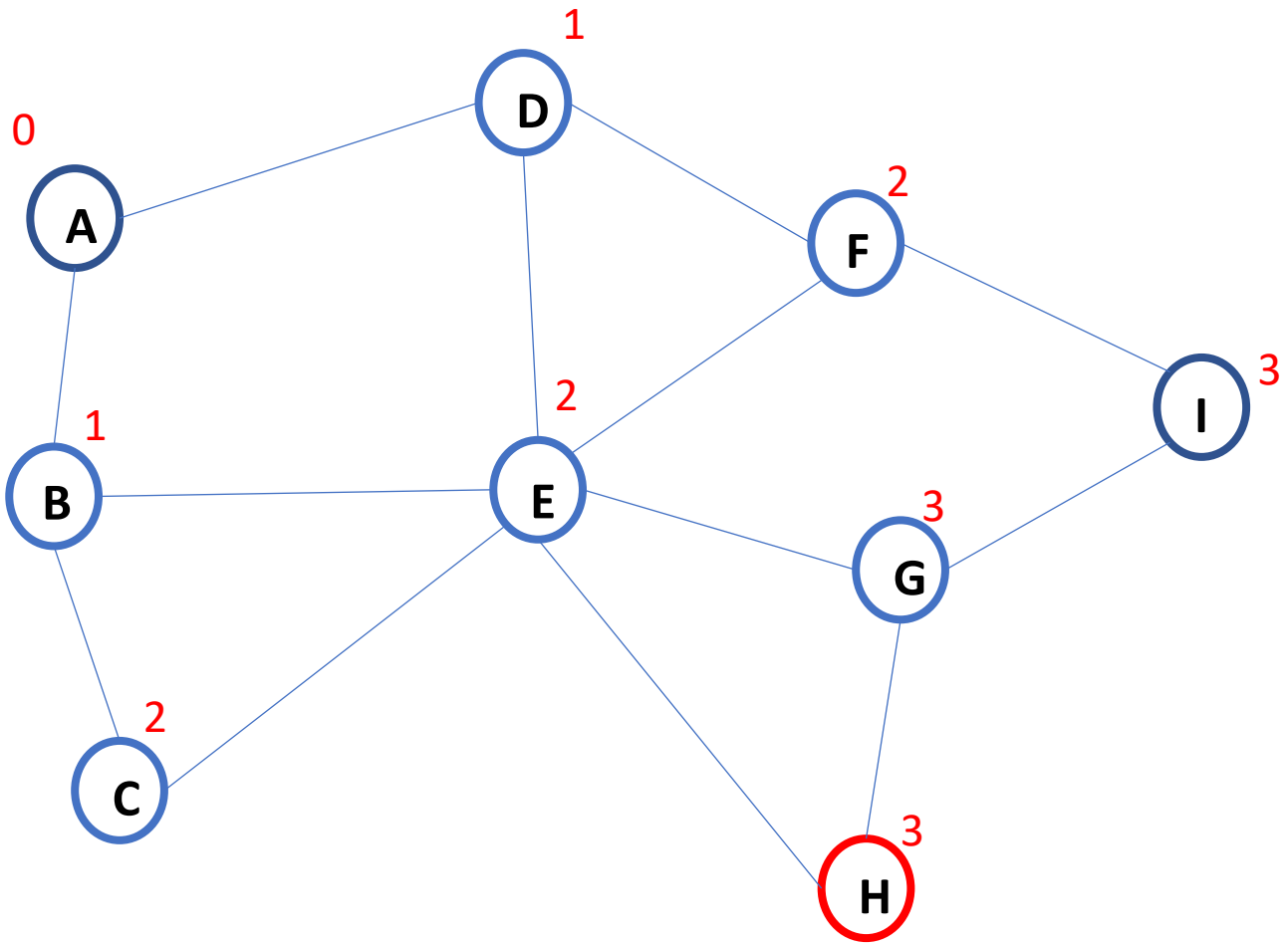
BFS




```
BFS(V, E, s)
for each u ∈ V − {s}
      do d[u] ← ∞
d[s] ← 0
Q ← ∅
ENQUEUE(Q, s)
while Q ≠ ∅
      do u ← DEQUEUE(Q)
         for each v ∈ Adj[u]
               do if d[v] = ∞
                     then d[v] ← d[u] + 1
                          ENQUEUE(Q, v)
```

Q: ~~A B C D E F G~~ H I

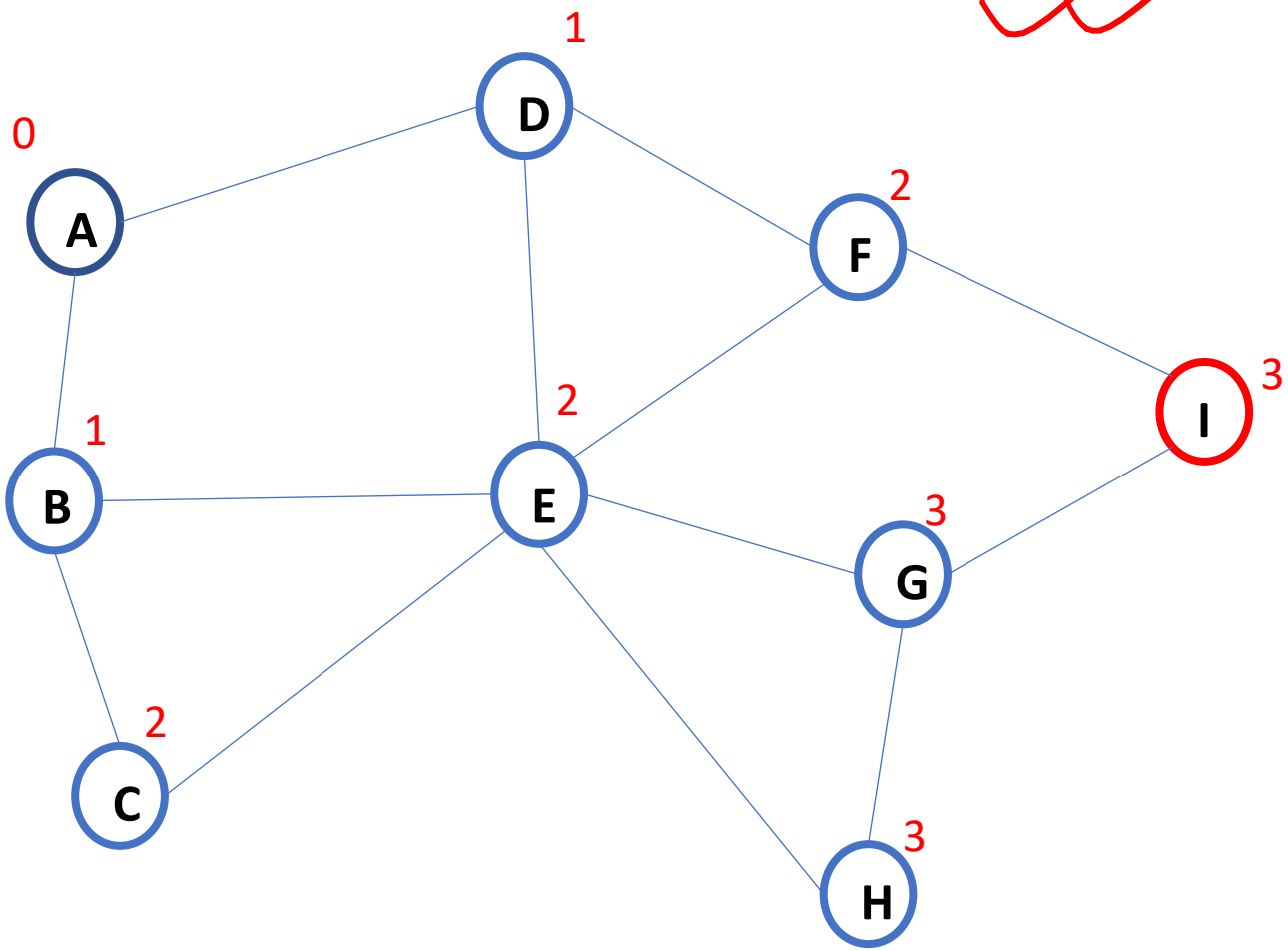
BFS




```
BFS(V, E, s)
for each u ∈ V − {s}
      do d[u] ← ∞
d[s] ← 0
Q ← ∅
ENQUEUE(Q, s)
while Q ≠ ∅
      do u ← DEQUEUE(Q)
         for each v ∈ Adj[u]
               do if d[v] = ∞
                     then d[v] ← d[u] + 1
                          ENQUEUE(Q, v)
```

Q: ~~A E B C D F G H I~~

BFS




```
BFS(V, E, s)
for each u ∈ V − {s}
      do d[u] ← ∞
d[s] ← 0
Q ← ∅
ENQUEUE(Q, s)
while Q ≠ ∅
      do u ← DEQUEUE(Q)
         for each v ∈ Adj[u]
             do if d[v] = ∞
                then d[v] ← d[u] + 1
                     ENQUEUE(Q, v)
```

Q: ~~XXBCEFGXX~~

In queue: how much time it takes to push and pop?

```
BFS(V, E, s)
for each u ∈ V − {s}
    do d[u] ← ∞
d[s] ← 0
Q ← ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    do u ← DEQUEUE(Q)
       for each v ∈ Adj[u]
           do if d[v] = ∞
              then d[v] ← d[u] + 1
                   ENQUEUE(Q, v)
```

```
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)
```

BCZ, w(e) =1 similar

θ(V+E)

θ(E)

θ(1)

θ(1)

θ(1)

V

V

E

push

θ(1)

θ(E log V)

θ(log V)

E log V

In queue: how much time it takes to push and pop?

```
BFS(V, E, s)
for each u ∈ V − {s}
    do d[u] ← ∞
d[s] ← 0
Q ← ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    do u ← DEQUEUE(Q)
       for each v ∈ Adj[u]
           do if d[v] = ∞
              then d[v] ← d[u] + 1
                   ENQUEUE(Q, v)
```

BCZ, w(e) =1 similar

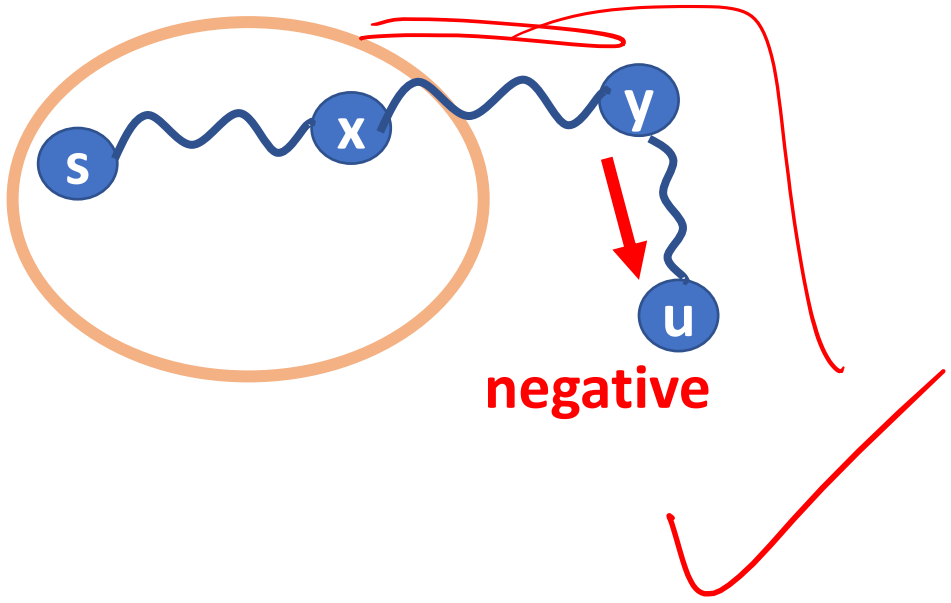
```
DIJKSTRA(G = (V, E), s)
1  for all v ∈ V
2        do d_u ← ∞
3           π_u ← NIL
4  d_s ← 0
5  Q ← MAKEQUEUE(V)        ▷ use d_u as key
6  while Q ≠ ∅
7        do u ← EXTRACTMIN(Q)
8           for each v ∈ Adj(u)
9                do if d_v > d_u + w(u, v)
10                  then d_v ← d_u + w(u,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

E

What if graph has negative weights? Dijkstra

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

```
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)
```



What if graph has negative weights? Dijkstra

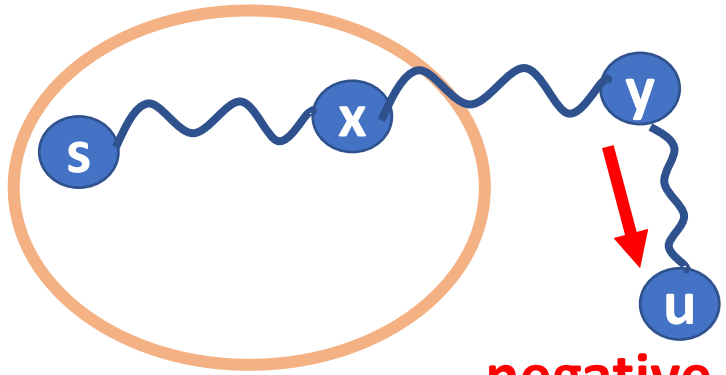
Consider any path from **s** to **u**. **(any include the shortest path too)**

weights neg!

$$L(p) = \delta(s,x) + w(x,y) + l(y \to u)$$
$$>= dy + l(y \to u)$$
$$>= du + l(y \to u) \quad \leftarrow -ve$$
$$>= du$$

If negative

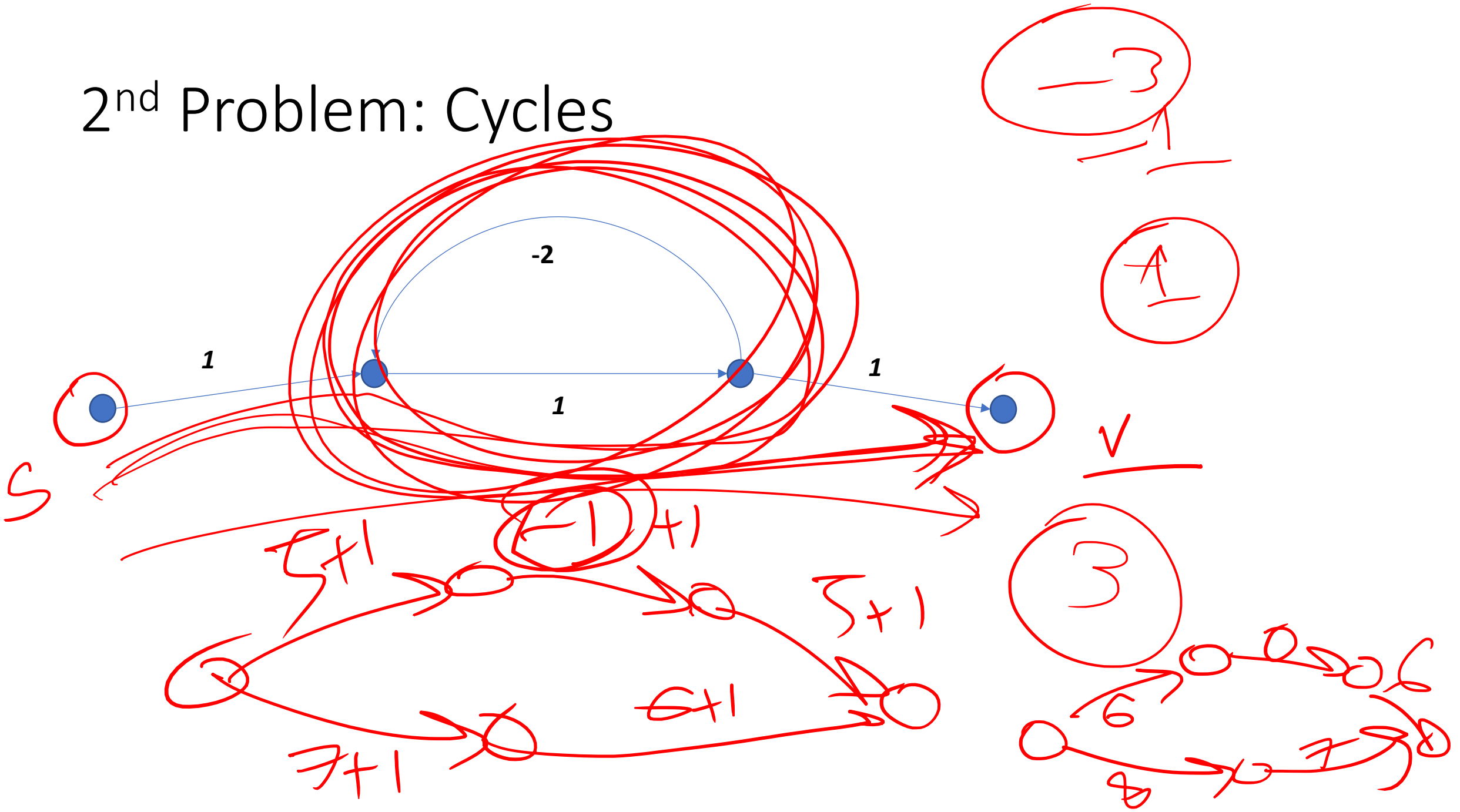
$x + (-ve) \geq x$



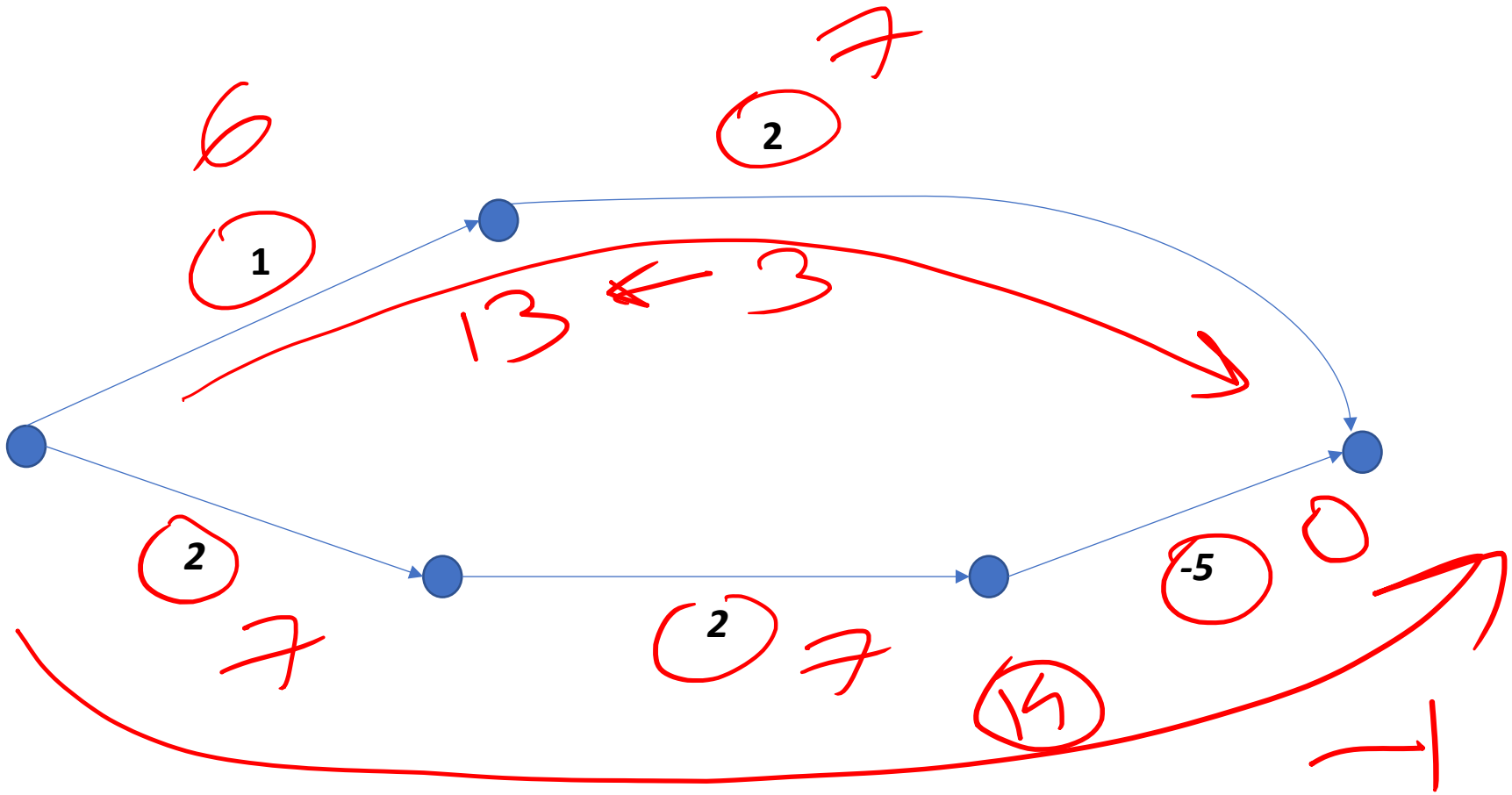
negative

```
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)
```

2nd Problem: Cycles



First Idea: add to each edge to make them positive



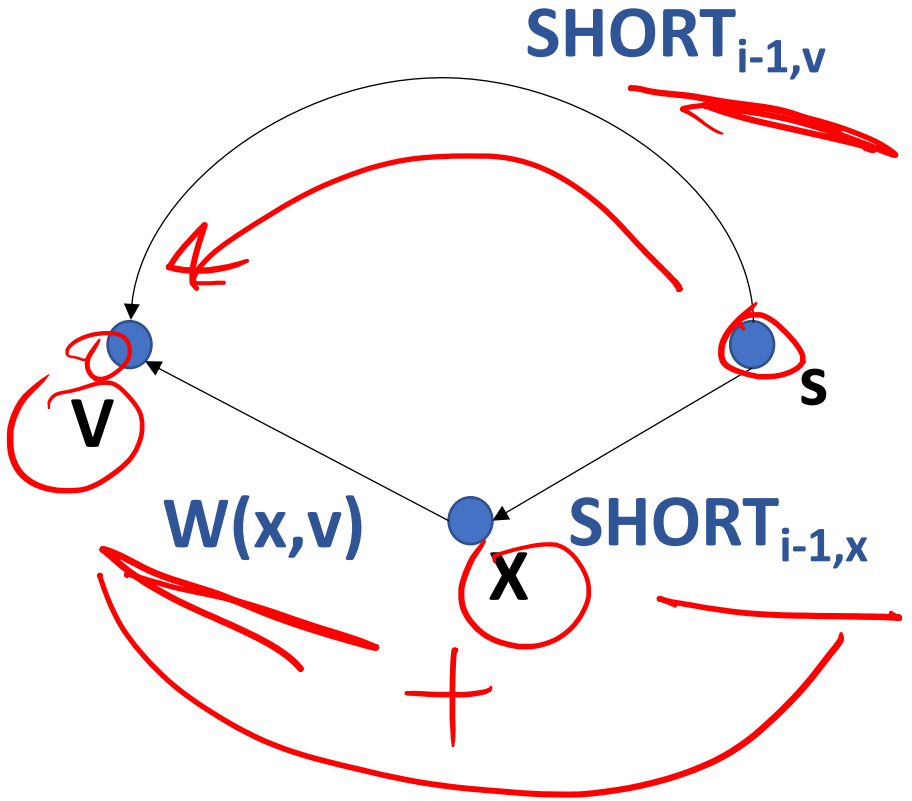
DP Solution

- **$SHORT_{i,v}$** = length of the shortest path from s-> v, that uses <= i edges

- **$SHORT_{i,v}$** = **Min**
  - **$SHORT_{i-1,v}$**
  - **$SHORT_{i-1,x} + w(x,v)$**

What are this x nodes?

The neighbors of v



DP Solution

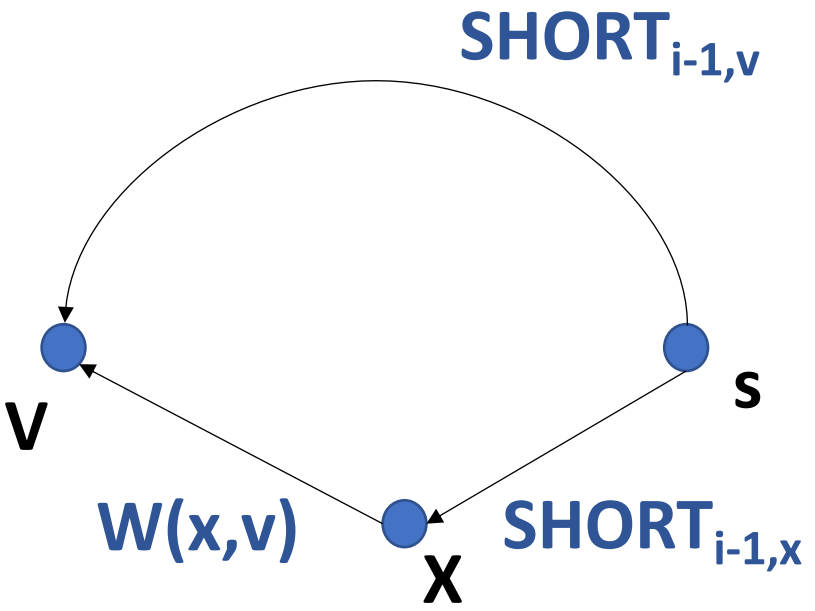
- **$SHORT_{i,v}$** = length of the shortest path from s-> v, that uses <= i edges

- **$SHORT_{i,v}$= Min**

$$\begin{cases} 0 & \text{if } v=s \\ \infty & \text{if } i=0 \\ SHORT_{i-1,v} & \\ SHORT_{i-1,x} + w(x,v) & \end{cases}$$

What are this x nodes?

The neighbors of v



Max length of a simple path

- Maximum length of a path can be V-1
- V: number of nodes

BELLMAN-FORD(G, s)

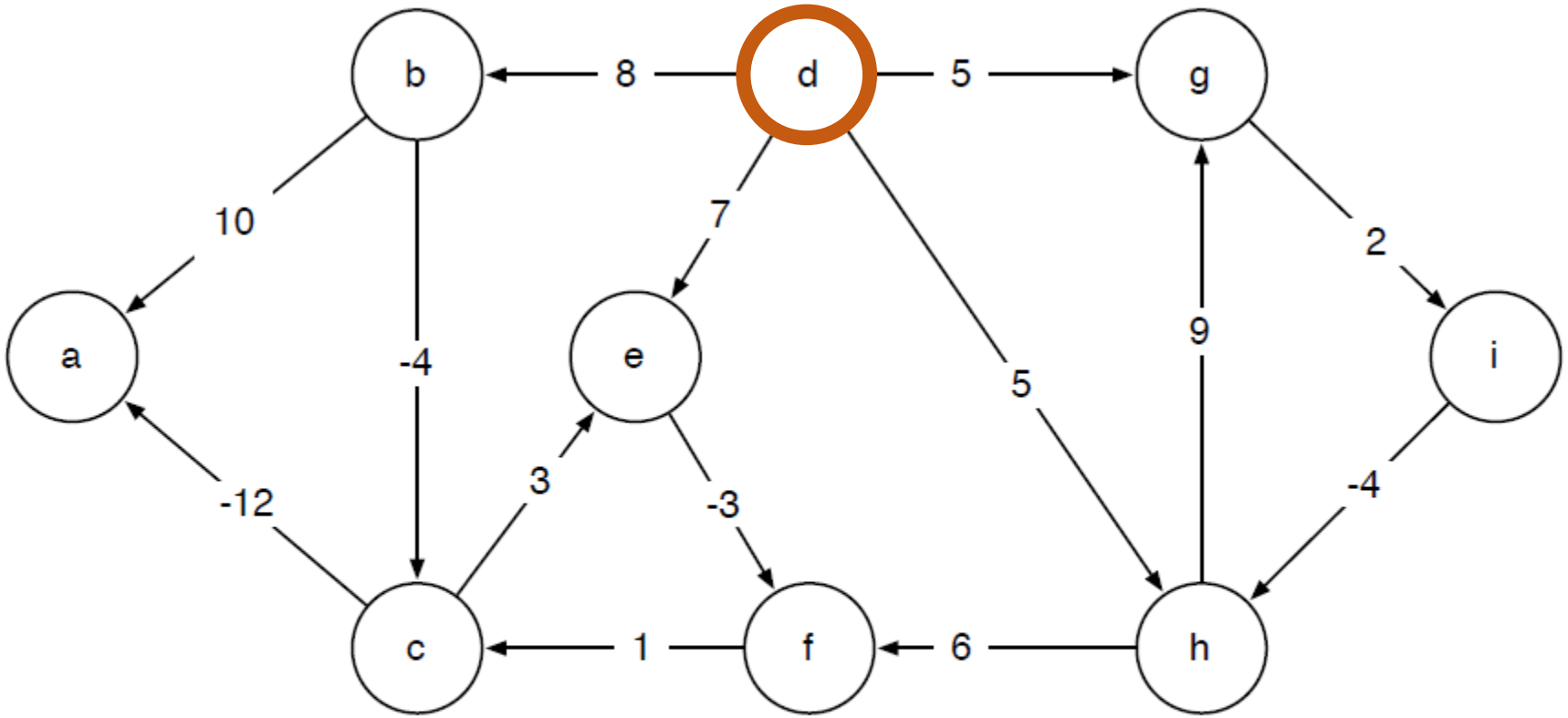
1 $\text{SHORT}_{0,s} \leftarrow 0$

2 $\forall v \in V - \{s\}$, $\text{SHORT}_{0,v} \leftarrow \infty$

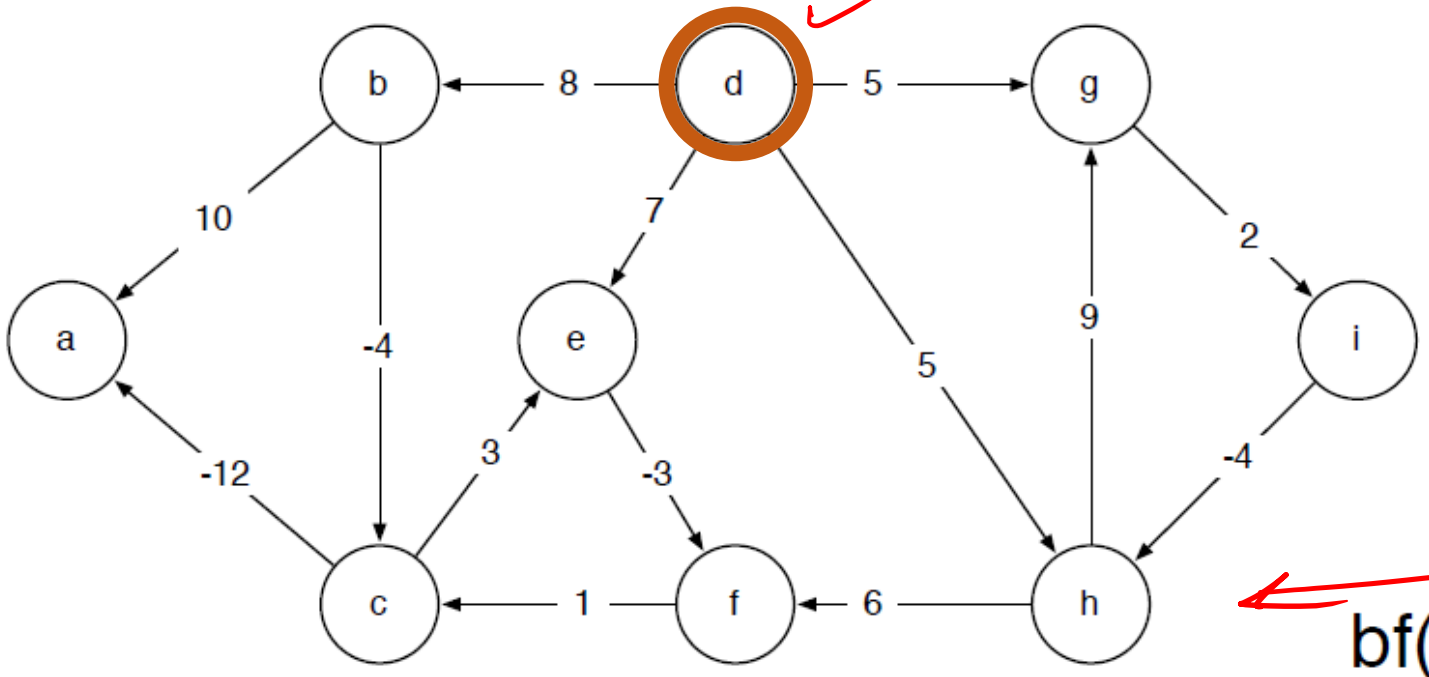
3 **for** $i = 1, \ldots, V - 1$

4 **do for** each $v \in V - \{s\}$

5 **do** $\text{SHORT}_{i,v} = \min_{x \in Adj(v)} \left\{ \begin{array}{l} \text{SHORT}_{i-1,v} \\ w(x, v) + \text{SHORT}_{i-1,x} \end{array} \right\}$



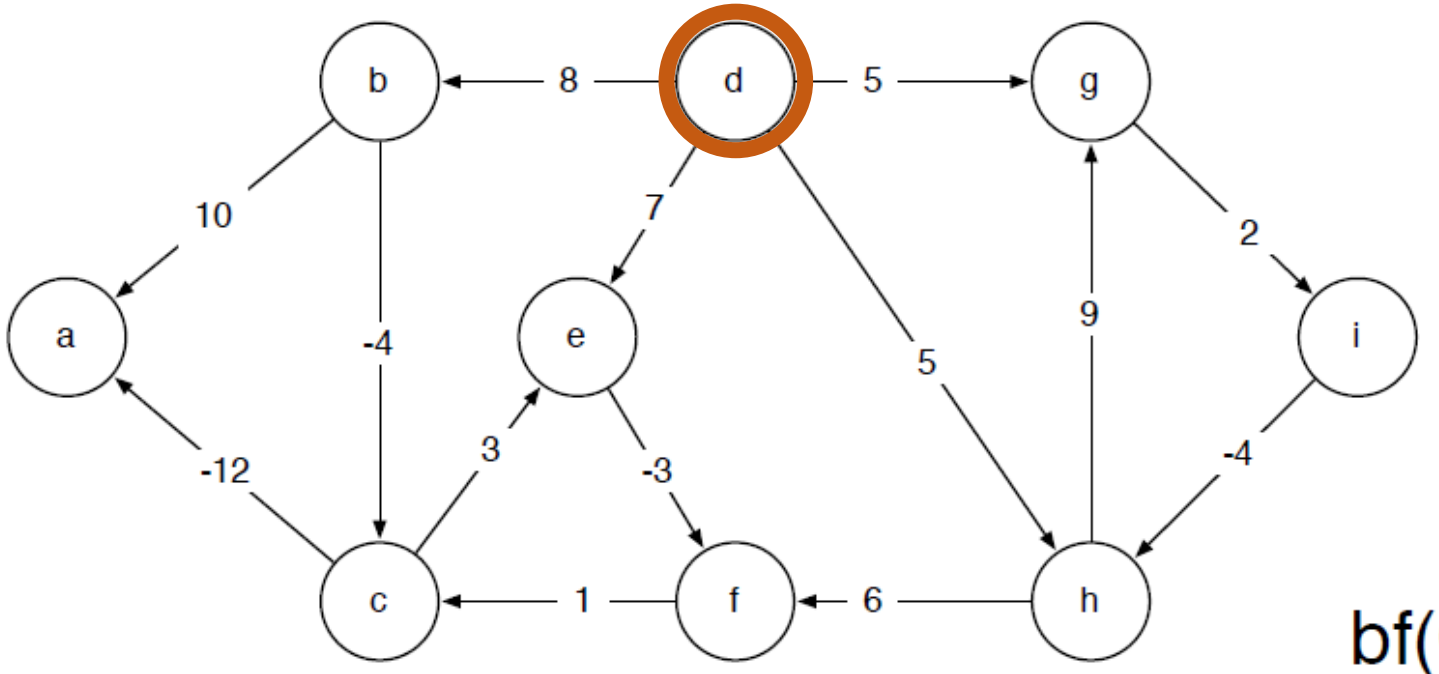
$$\text{SHORT}_{i,v} = \begin{cases} \infty & i = 0 \\ 0 & v = s \\ \min_{x \in V} \left\{ \begin{array}{l} \text{SHORT}_{i-1,v} \\ \text{SHORT}_{i-1,x} + w(x,v) \end{array} \right\} \end{cases}$$



$$\text{SHORT}_{i,v} = \begin{cases} \infty & i = 0 \\ 0 & v = s \\ \min_{x \in V} \left\{ \begin{array}{l} \text{SHORT}_{i-1,v} \\ \text{SHORT}_{i-1,x} + w(x,v) \end{array} \right\} \end{cases}$$

bf(G,d)

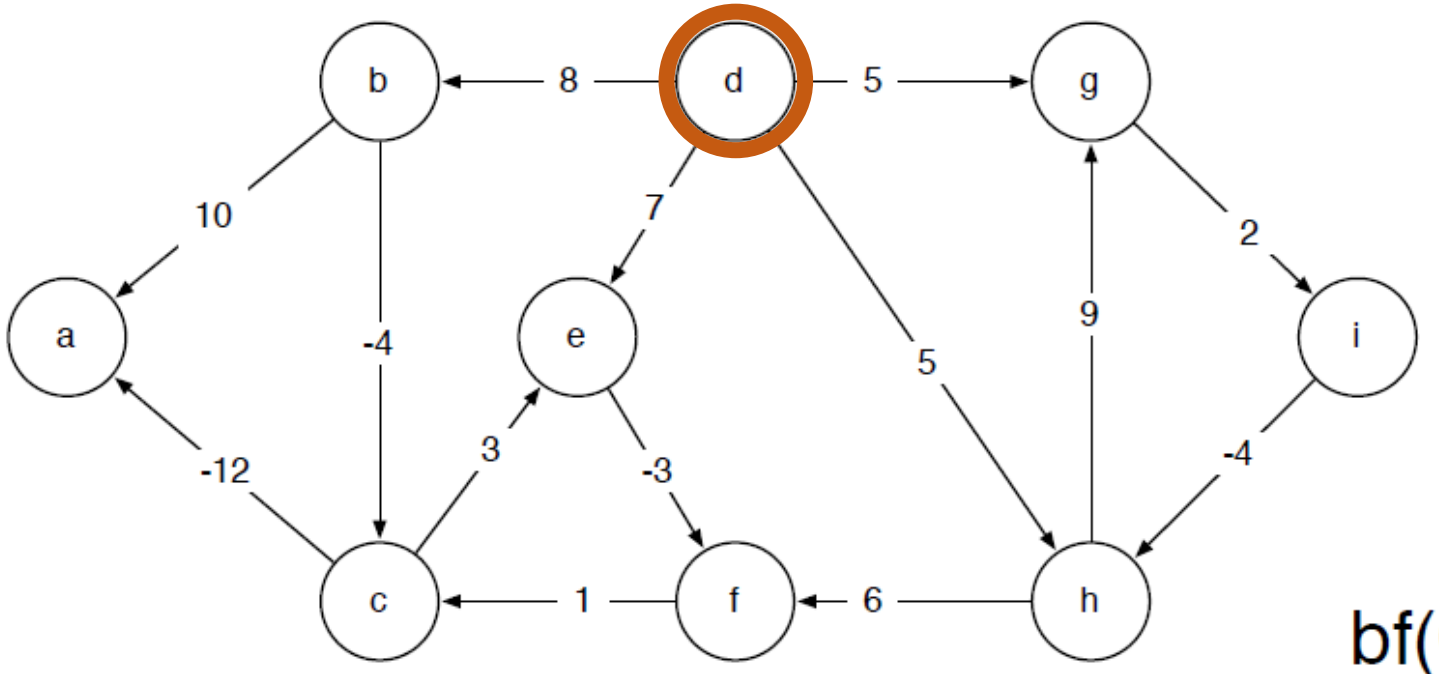
	0	1	2	3	4	5	6	7
A								
B								
C								
D	0							
E								
F								
G								
H								
I								



$$\text{SHORT}_{i,v} = \begin{cases} \infty & i = 0 \\ 0 & v = s \\ \min_{x \in V} \left\{ \begin{array}{l} \text{SHORT}_{i-1,v} \\ \text{SHORT}_{i-1,x} + w(x,v) \end{array} \right\} \end{cases}$$

bf(G,d)

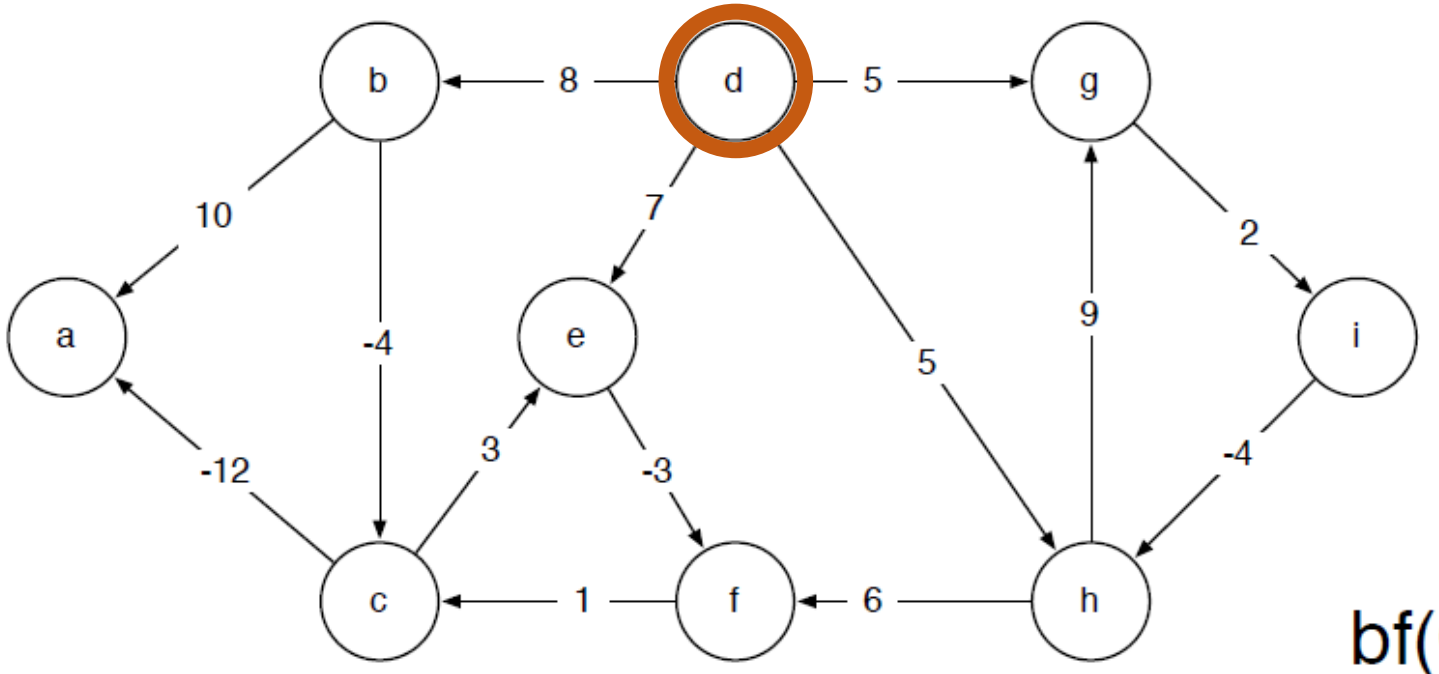
	0	1	2	3	4	5	6	7
A	∞							
B	∞							
C	∞							
D	0							
E	∞							
F	∞							
G	∞							
H	∞							
I	∞							



$$\text{SHORT}_{i,v} = \begin{cases} \infty & i = 0 \\ 0 & v = s \\ \min_{x \in V} \left\{ \begin{array}{l} \text{SHORT}_{i-1,v} \\ \text{SHORT}_{i-1,x} + w(x,v) \end{array} \right\} \end{cases}$$

bf(G,d)

	0	1	2	3	4	5	6	7
A	∞	∝						
B	∞	8						
C	∞	∠						
D	0	0						
E	∞	7						
F	∞	∠						
G	∞	5						
H	∞	5						
I	∞	∠						



$$\text{SHORT}_{i,v} = \begin{cases} \infty & i = 0 \\ 0 & v = s \\ \min_{x \in V} \left\{ \begin{array}{l} \text{SHORT}_{i-1,v} \\ \text{SHORT}_{i-1,x} + w(x,v) \end{array} \right\} \end{cases}$$

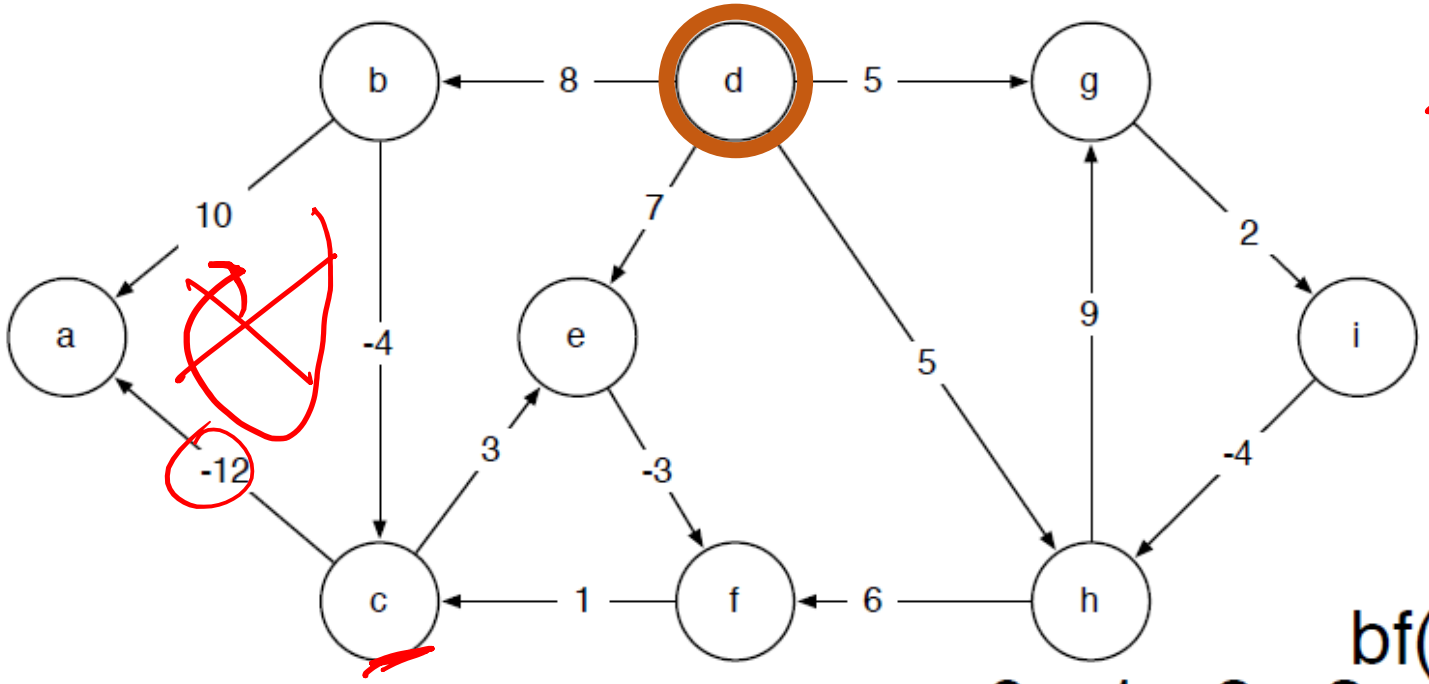
bf(G,d)

	0	1	2	3	4	5	6	7
A	∞		18					
B	∞	8	8					
C	∞		4					
D	0	0	0					
E	∞	7	7					
F	∞		4					
G	∞	5	5					
H	∞	5	5					
I	∞		7					

short 2,A

min short 1,b + 10

short 1,c (+17)



short 3, A

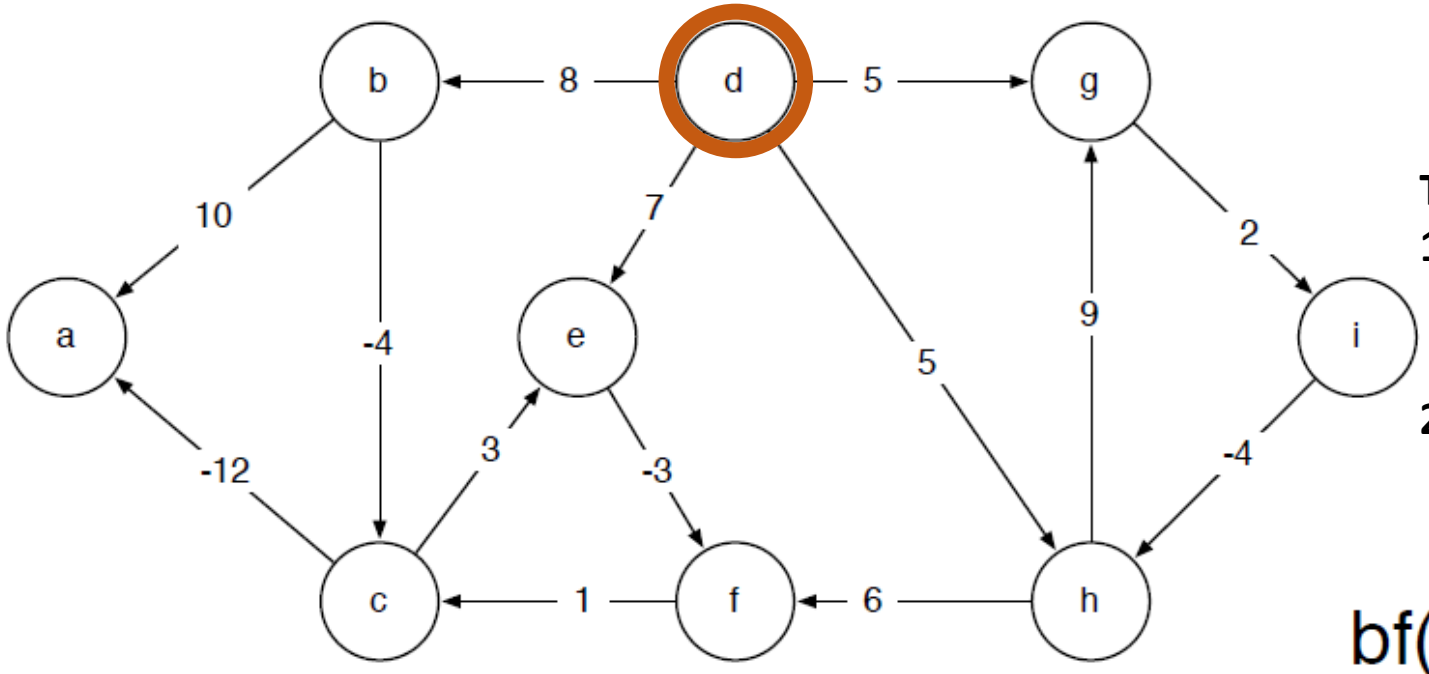
short 2, c + w(a,c)

4 + (-12)

$$\text{SHORT}_{i,v} = \begin{cases} \infty & i = 0 \\ 0 & v = s \\ \min_{x \in V} \left\{ \begin{array}{l} \text{SHORT}_{i-1,v} \\ \text{SHORT}_{i-1,x} + w(x,v) \end{array} \right\} \end{cases}$$

bf(G,d)

	0	1	2	3	4	5	6	7
A	∞		18	-8				
B	∞	8	8	8				
C	∞		4	4				
D	0	0	0	0				
E	∞	7	7	7				
F	∞		4	4				
G	∞	5	5	5				
H	∞	5	5	3				
I	∞		7	7				



Two things can happened:

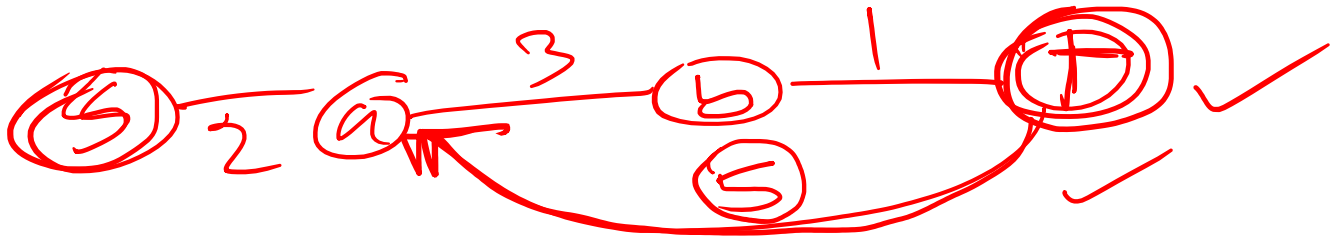
1. **Nothing changes**
   1. **Nothing will change in future**
   2. **You can stop**
2. **Continue to change and you go until V-1 steps.**

$$\text{SHORT}_{i,v} = \begin{cases} \infty & i = 0 \\ 0 & v = s \\ \min_{x \in V} \left\{ \begin{array}{l} \text{SHORT}_{i-1,v} \\ \text{SHORT}_{i-1,x} + w(x,v) \end{array} \right\} \end{cases}$$

bf(G,d)

	0	1	2	3	4	5	6	7
A	∞		18	-8				
B	∞	8	8	8				
C	∞		4	4				
D	0	0	0	0				
E	∞	7	7	7				
F	∞		4	4				
G	∞	5	5	5				
H	∞	5	5	3				
I	∞		7	7				

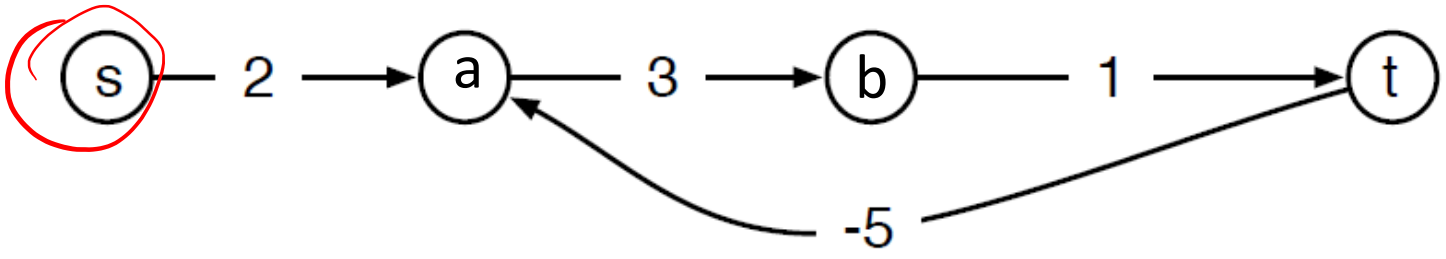
negative cycles?



	0	1	2	3	4
s		0			
a		2			
b					
t					

$$\text{SHORT}_{i,v} = \begin{cases} \infty & i = 0 \\ 0 & v = s \\ \min_{x \in V} & \left\{ \begin{array}{l} \text{SHORT}_{i-1,v} \\ \text{SHORT}_{i-1,x} + w(x,v) \end{array} \right\} \end{cases}$$

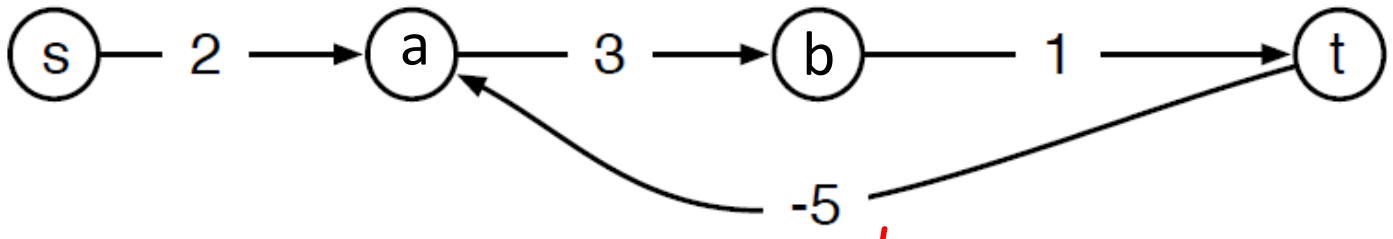
negative cycles?



	0	1	2	3	4
s	0	**0**			
a	**2**	**2**			
b		**5**			
t					

$$\text{SHORT}_{i,v} = \begin{cases} \infty & i = 0 \\ 0 & v = s \\ \min_{x \in V} & \left\{ \begin{array}{l} \text{SHORT}_{i-1,v} \\ \text{SHORT}_{i-1,x} + w(x,v) \end{array} \right\} \end{cases}$$

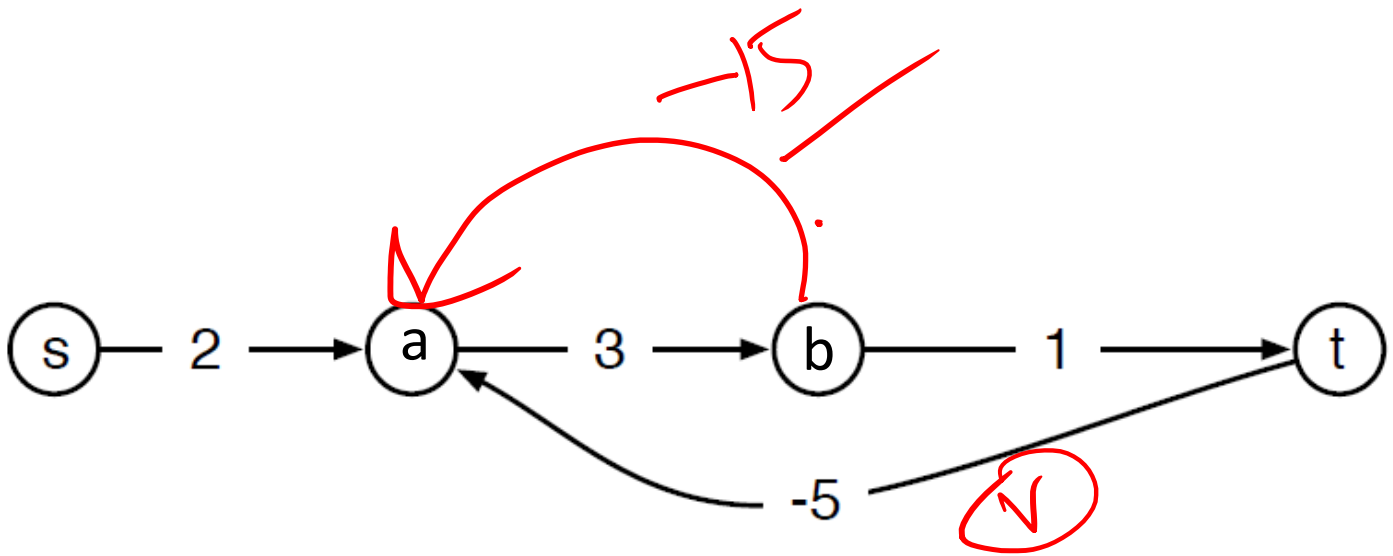
negative cycles?



	0	1	2	3	4
s	0	0	0		
a	2	2	2		
b		5	5		
t			6		

$$\text{SHORT}_{i,v} = \begin{cases} \infty & i = 0 \\ 0 & v = s \\ \min_{x \in V} & \left\{ \begin{array}{l} \text{SHORT}_{i-1,v} \\ \text{SHORT}_{i-1,x} + w(x,v) \end{array} \right\} \end{cases}$$

negative cycles?



0	1	2	3	4
s	0	0	0	
a	2	2	2	1
b		5	5	
t			6	

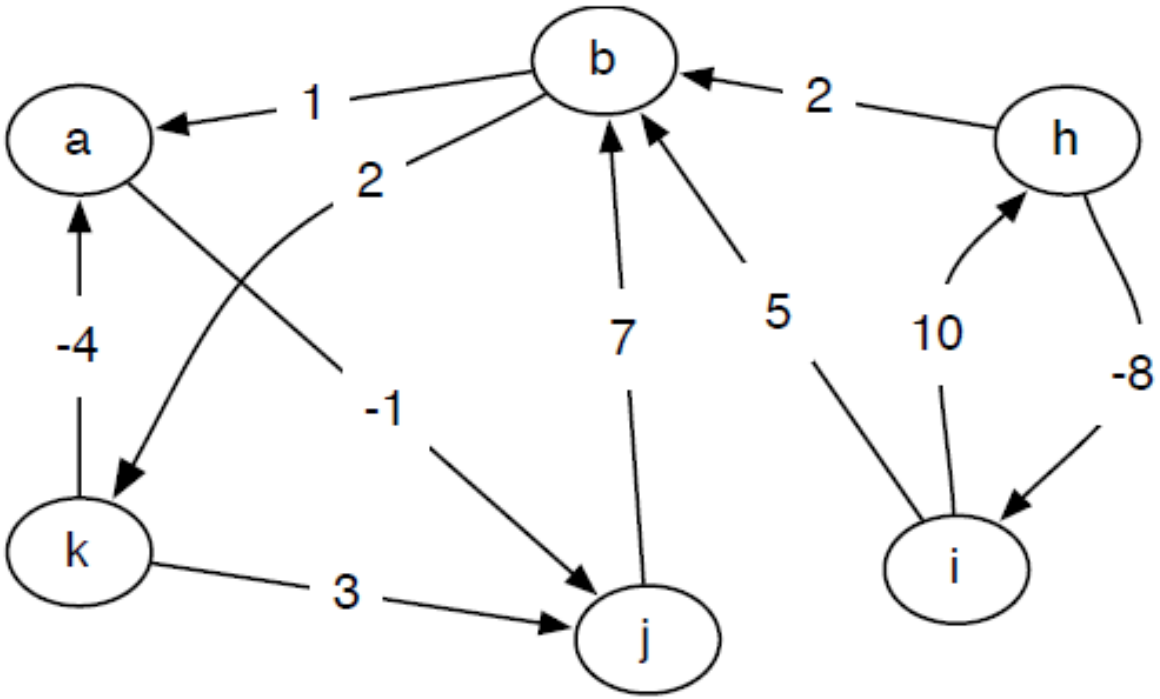
BF:

Run-time:θ(VE)

Space: V*E

$$\text{SHORT}_{i,v} = \begin{cases} \infty & i = 0 \\ 0 & v = s \\ \min_{x \in V} & \left\{ \begin{array}{l} \text{SHORT}_{i-1,v} \\ \text{SHORT}_{i-1,x} + w(x,v) \end{array} \right\} \end{cases}$$

All-pairs shortest path



BF:

Run-time:θ(VE)

Space: V*E

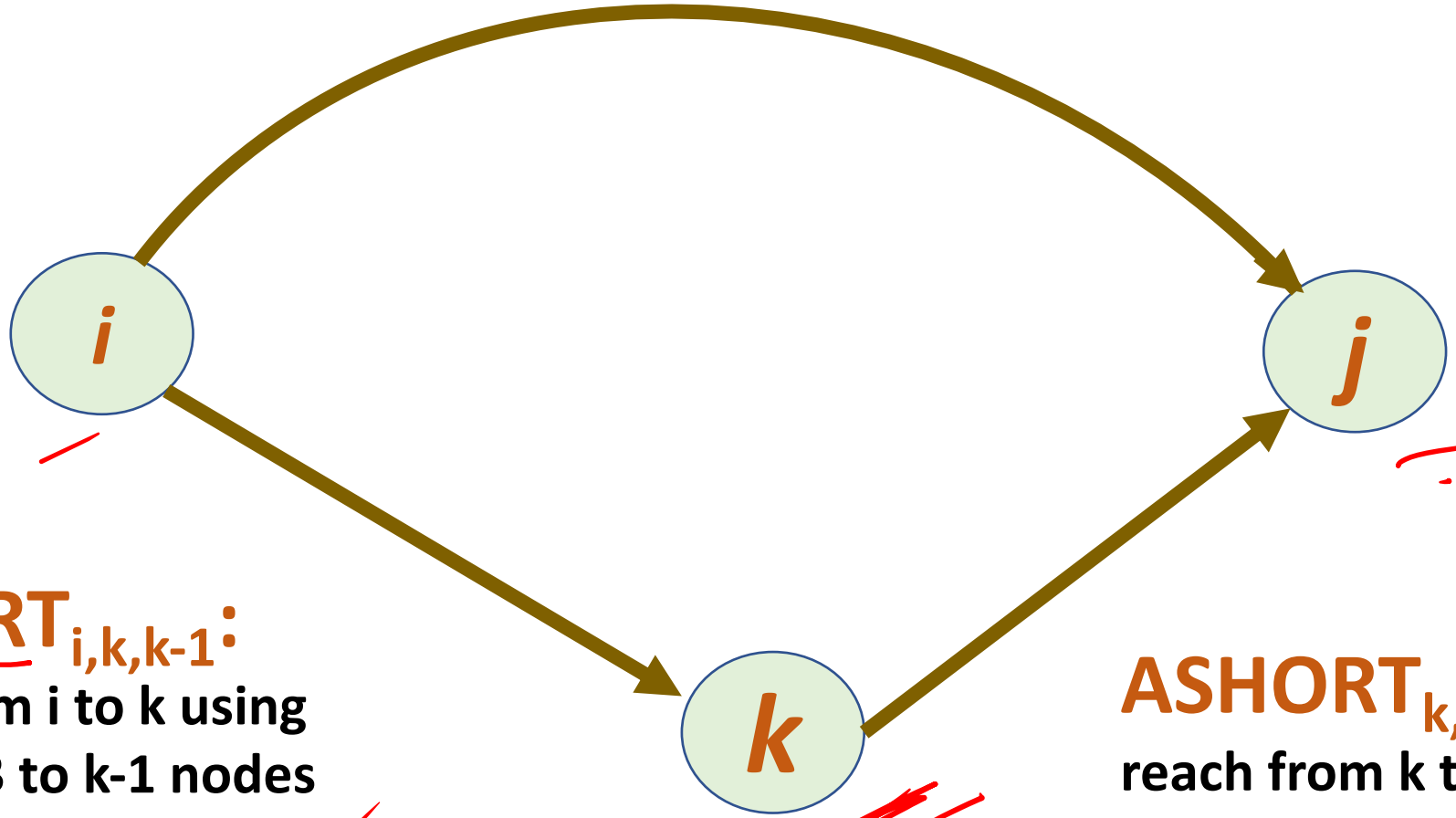
Run BF for all nodes:

θ(V²E)

- **$ASHORT_{i,j,k}$** = length of the shortest path from i to j that uses only nodes 1,2,3,…, to k nodes

$ASHORT_{i,j,k}$

$ASHORT_{i,j,k-1}$:
reach from i to j using only 1,2,3 to k-1 nodes

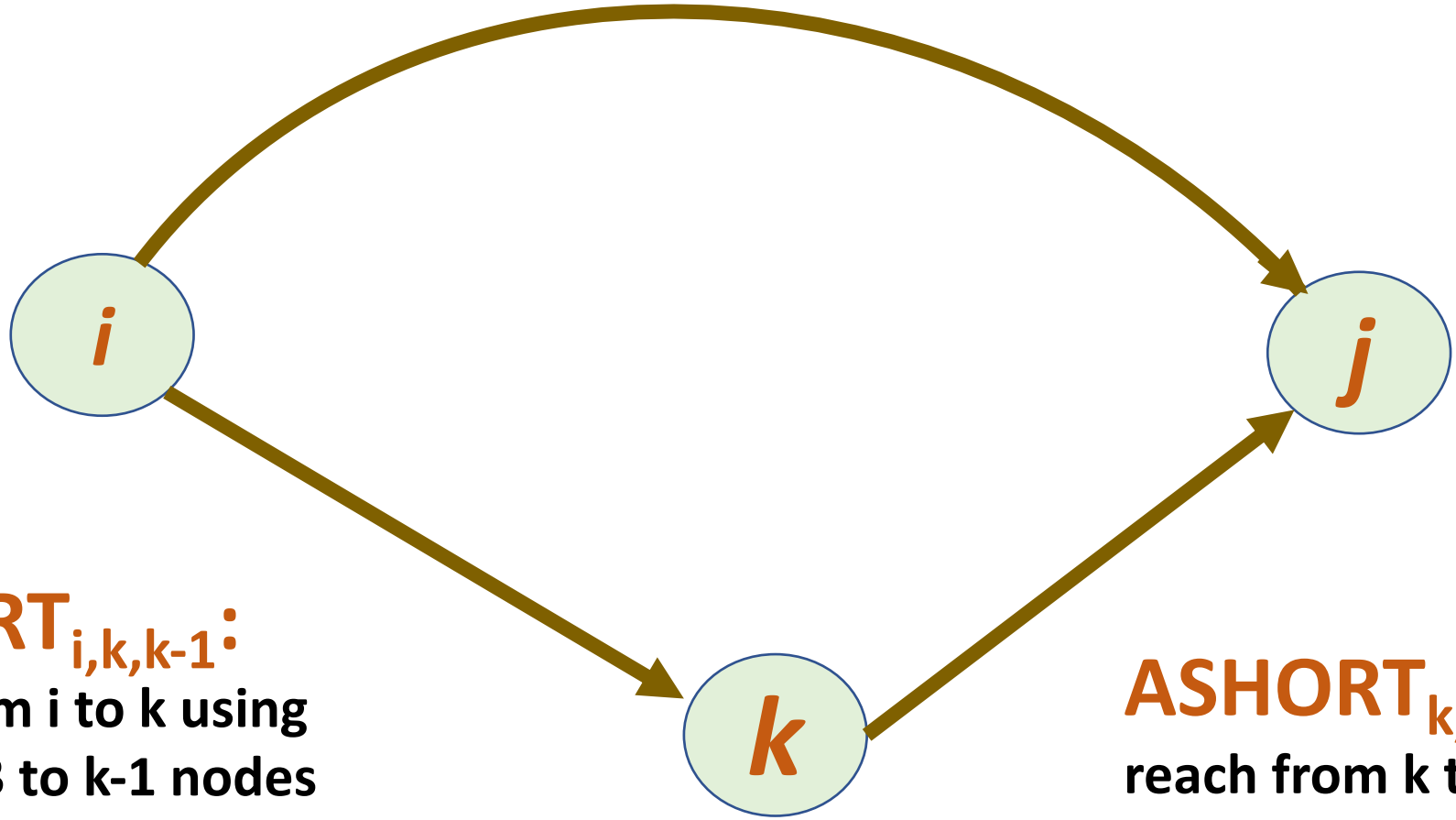


$ASHORT_{i,k,k-1}$:
reach from i to k using only 1,2,3 to k-1 nodes

$ASHORT_{k,j,k-1}$:
reach from k to j using only 1,2,3 to k-1 nodes

$ASHORT_{i,j,k}$

$ASHORT_{i,j,k-1}$:
reach from i to j using only 1,2,3 to k-1 nodes

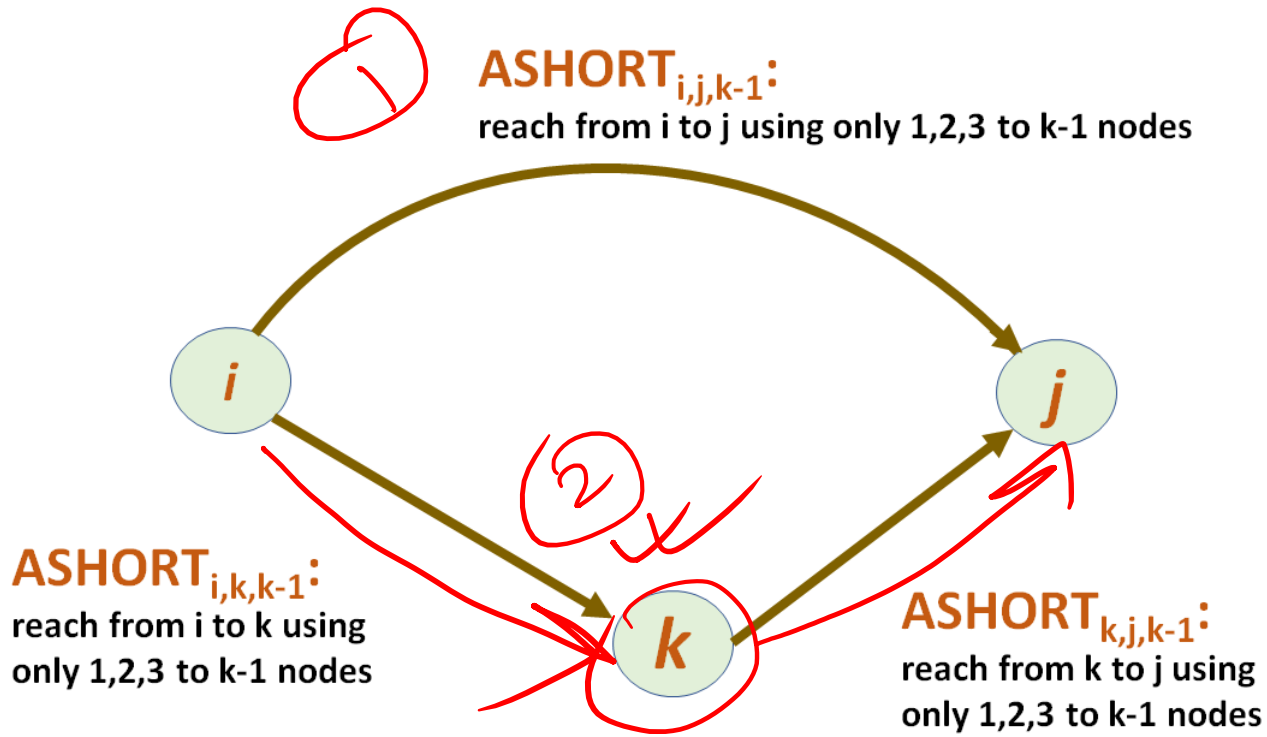


$ASHORT_{i,k,k-1}$:
reach from i to k using only 1,2,3 to k-1 nodes

$ASHORT_{k,j,k-1}$:
reach from k to j using only 1,2,3 to k-1 nodes

Floyd-Warshall(G,W) Algorithm

$$ASHORT_{i,j,k} = \begin{cases} W_{i,j} & \text{if } k=0 \\ \min \begin{cases} ASHORT_{i,j,k-1} \\ ASHORT_{i,k,k-1} + ASHORT_{k,j,k-1} \end{cases} & K>=0 \end{cases}$$



Floyd-Warshall(G,W) Algorithm

- for (k=1……n)
  {
    for (i=1……n)
    {
      for (j=1……n)
      {

$$\mathbf{ASHORT}_{i,j,k} = \min \begin{cases} ASHORT_{i,j,k-1} \\ ASHORT_{i,k,k-1} + ASHORT_{k,j,k-1} \end{cases}$$

Floyd-Warshall(G,W) Algorithm

Run-time:θ(V^3)

Space: V^2

int graph[128][128], n; // a weighted graph and its size

- initialize the graph[][] adjacency matrix and n
- graph[i][i] should be zero for all i.
- graph[i][j] should be "infinity" if edge (i, j) does not exist
- otherwise, graph[i][j] is the weight of the edge (i, j)

```
void floydWarshall() {
        for( int k = 0; k < n; k++ )
                for( int i = 0; i < n; i++ )
                        for( int j = 0; j < n; j++ )
                                graph[i][j] = min( graph[i][j], graph[i][k] + graph[k][j]);
}
```

Max flow
Min Cut

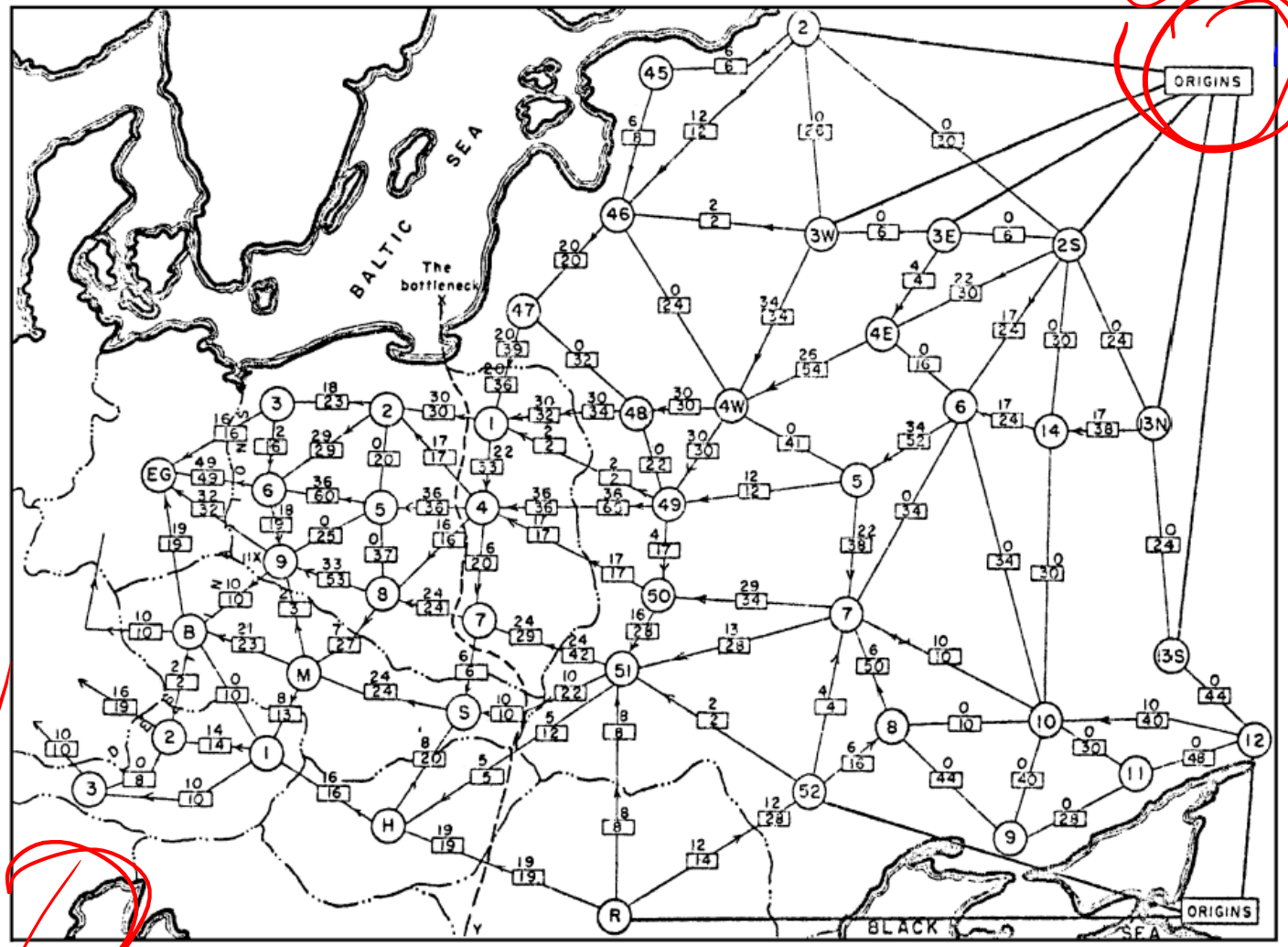


Figure 4 From Harris and Ross [3]: Schematic diagram of the railway network of the Western Soviet Union and East European countries, with a maximum flow of value 163,000 tons from Russia to Eastern Europe and a cut of capacity 163,000 tons indicated as 'The bottleneck'

courtesy Alexander Schrijver

"Consider a rail network connecting two cities by way of a number of intermediate cities, where each link of the network has a number assigned to it representing its capacity. Assuming a steady state condition, find a maximal flow from one given city to the other."

flow networks

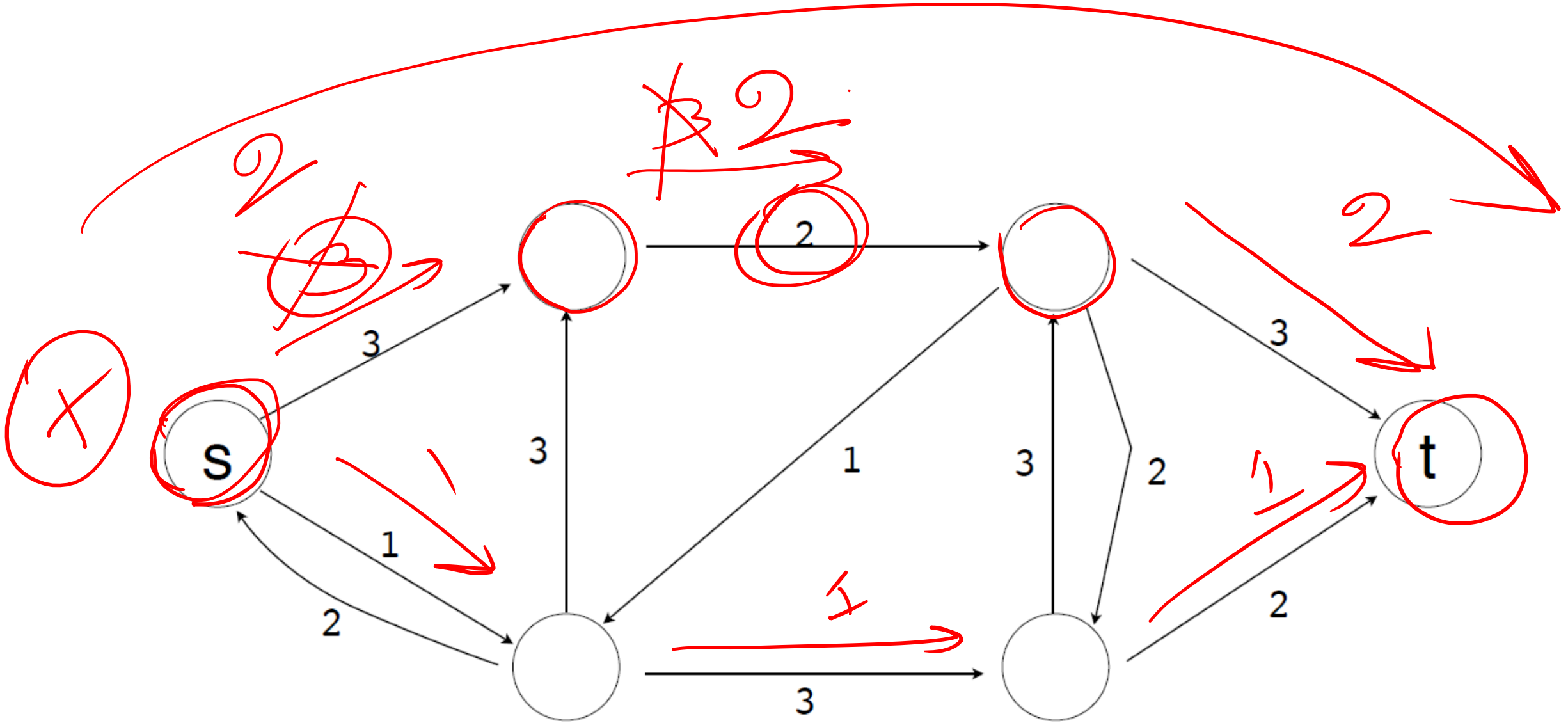
$$G = (V, E)$$

source + sink: node s, and t

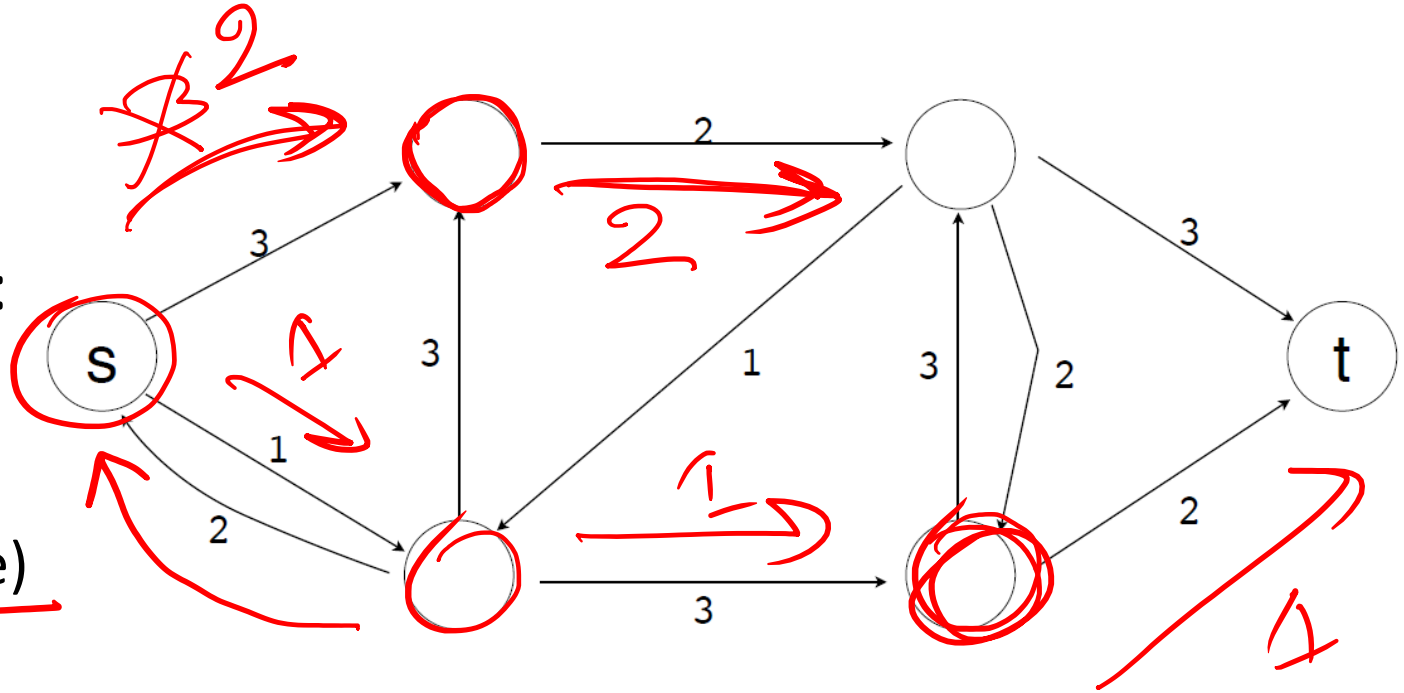
capacities: $c(u, v)$

assumed to be 0 if no (u,v) edge

example



FLOW



- Map from edges to numbers:
  - Each edge weights
- Capacity Constraint:
  - For every edge e∈E, f(e) <= C(e)
- Flow Constraint:

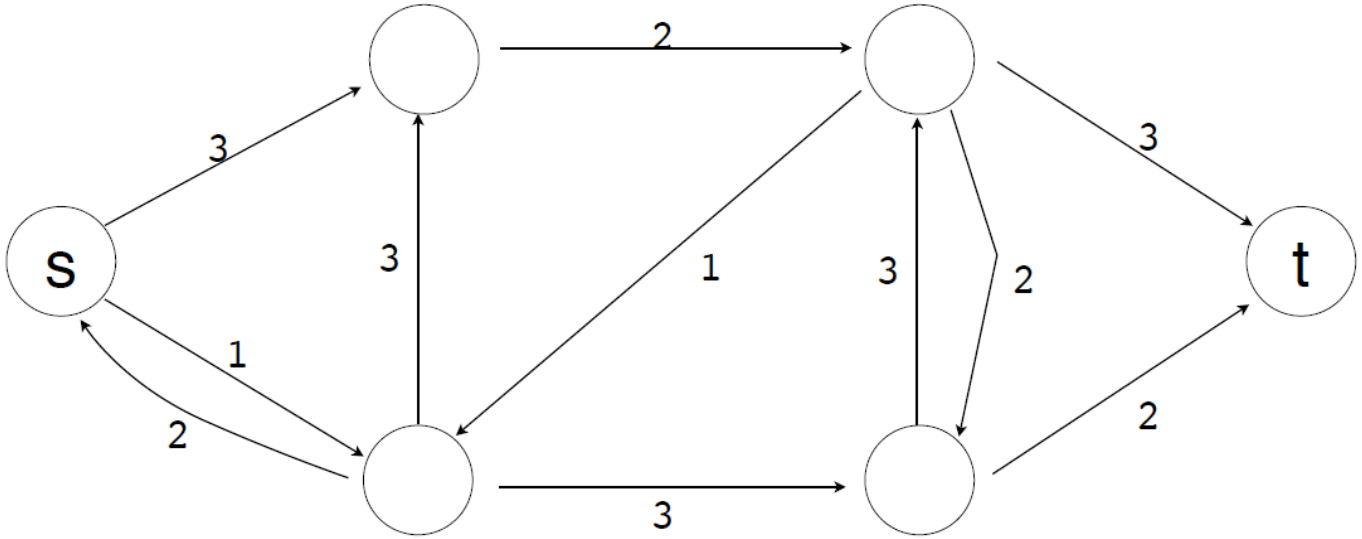
For every node v ∈ V-{s,t}, inflow(v)=outflow(v)

$$\sum_{x \in V} f(x, v) = \sum_{x \in V} f(v, x)$$

|f|= total flow of the graph = outflow from *S*

$$\sum_{x \in V} f(s, x) - \sum_{x \in V} f(x, s)$$

FLOW



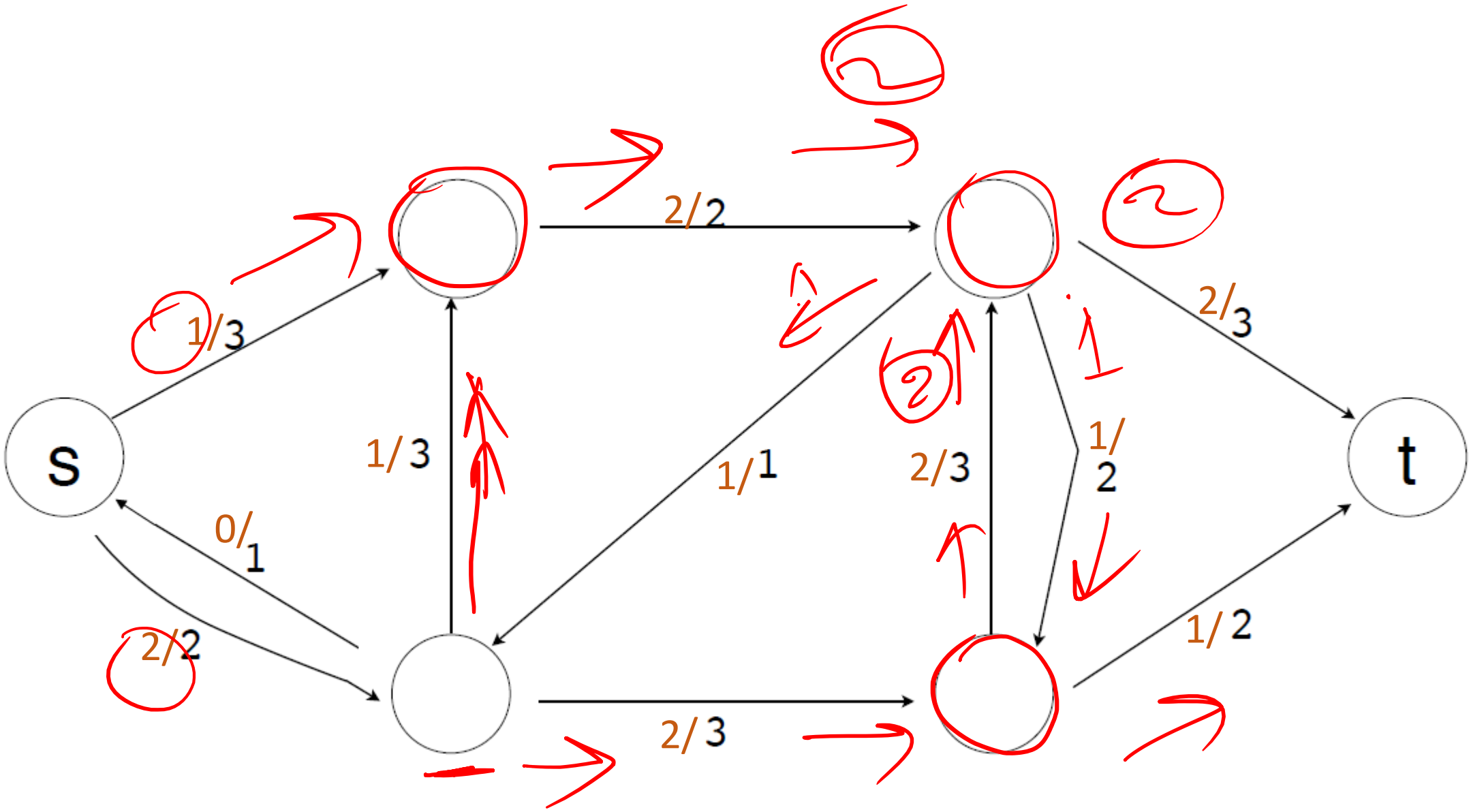
- Map from edges to numbers:
  - Each edge weights
- Capacity Constraint:
  - For every edge e∈E, f(e) <= C(e)
- Flow Constraint:

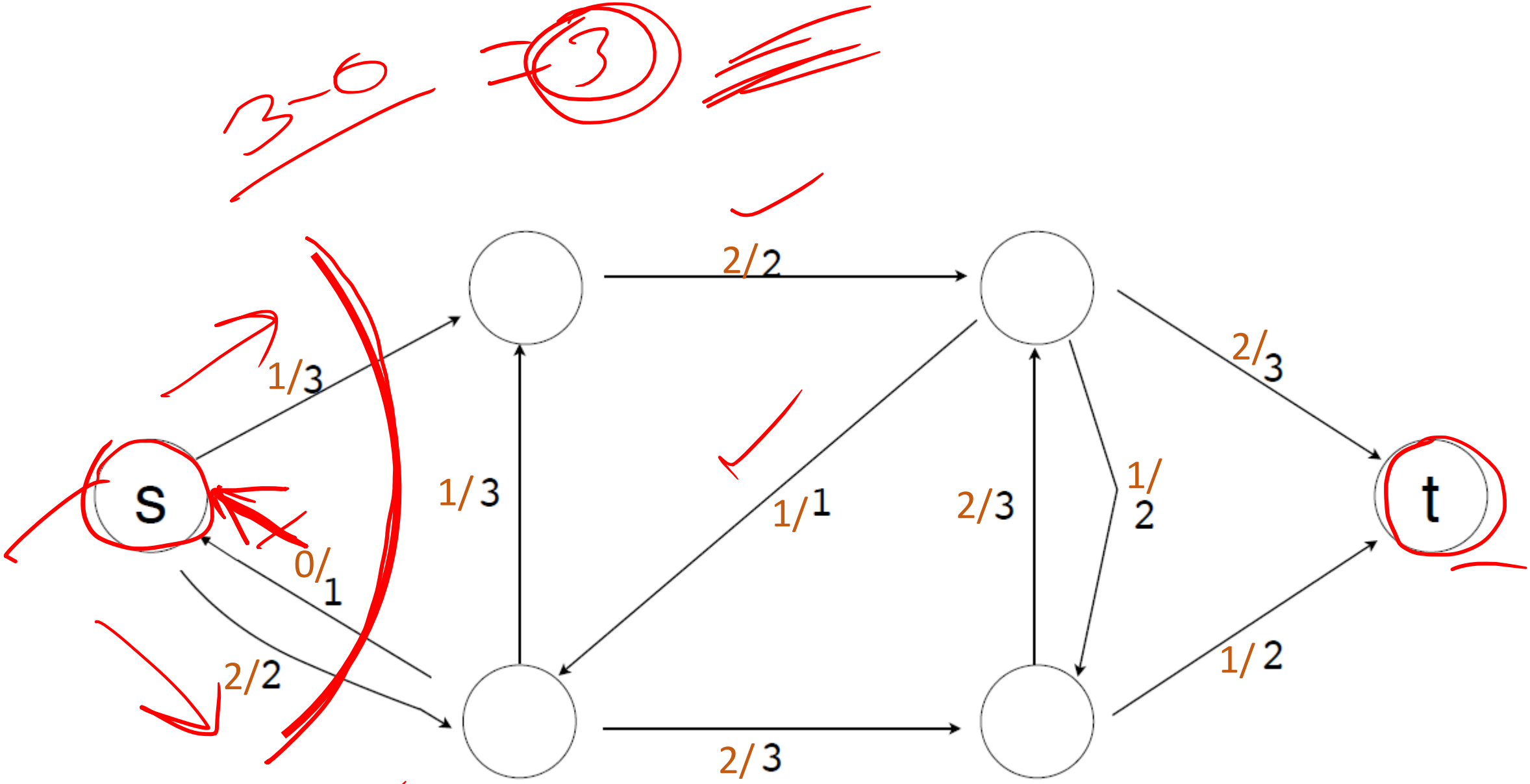
For every node v ∈ V-{s,t}, inflow(v)=outflow(v)

$$\sum_{x \in V} f(x, v) = \sum_{x \in V} f(v, x)$$

|f|= total flow of the graph = outflow from ***S***

$$\sum_{x \in V} f(s, x) - \sum_{x \in V} f(x, s)$$


s
t
1/3
2/2
1/3
1/1
2/3
1/2
2/3
0/1
2/2
2/3
1/2



$|f| = 3$

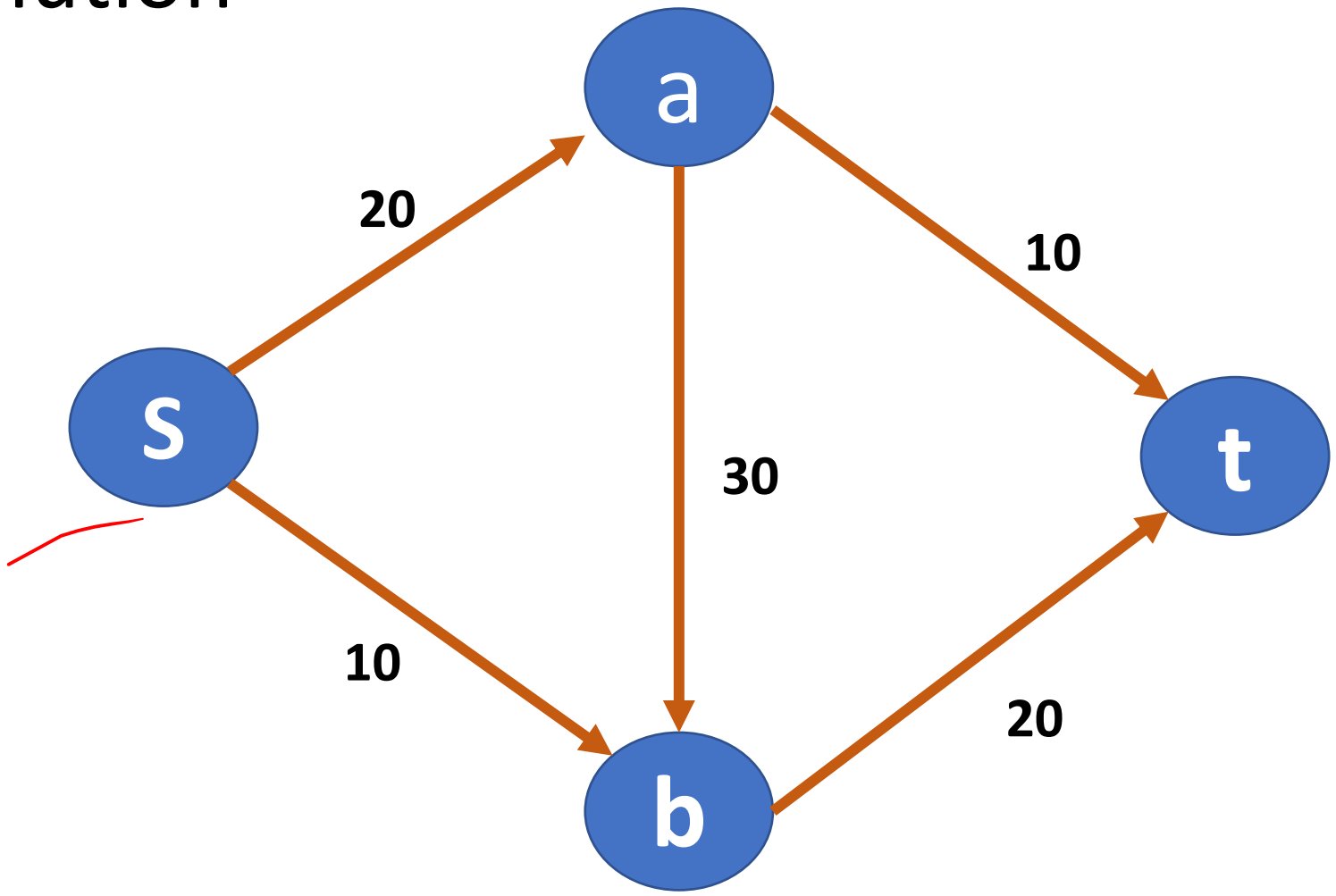
Max flow problem

- Given a graph G=(V,E), with each edge e has capacity c(e):

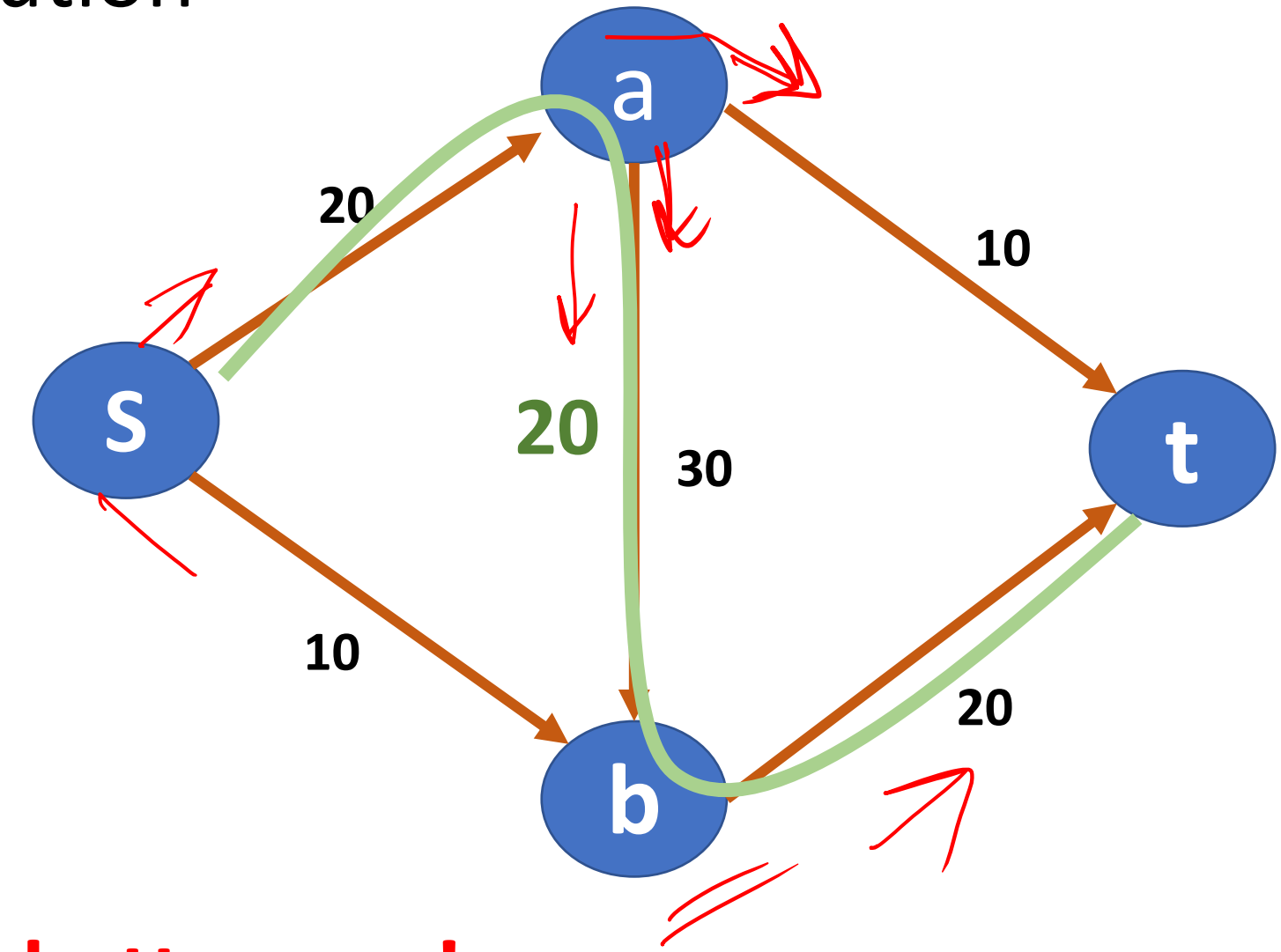
Compute: argmax |f|

- Where f is possible valid flow through the graph
- Find the maximum valid flow

Greedy Solution

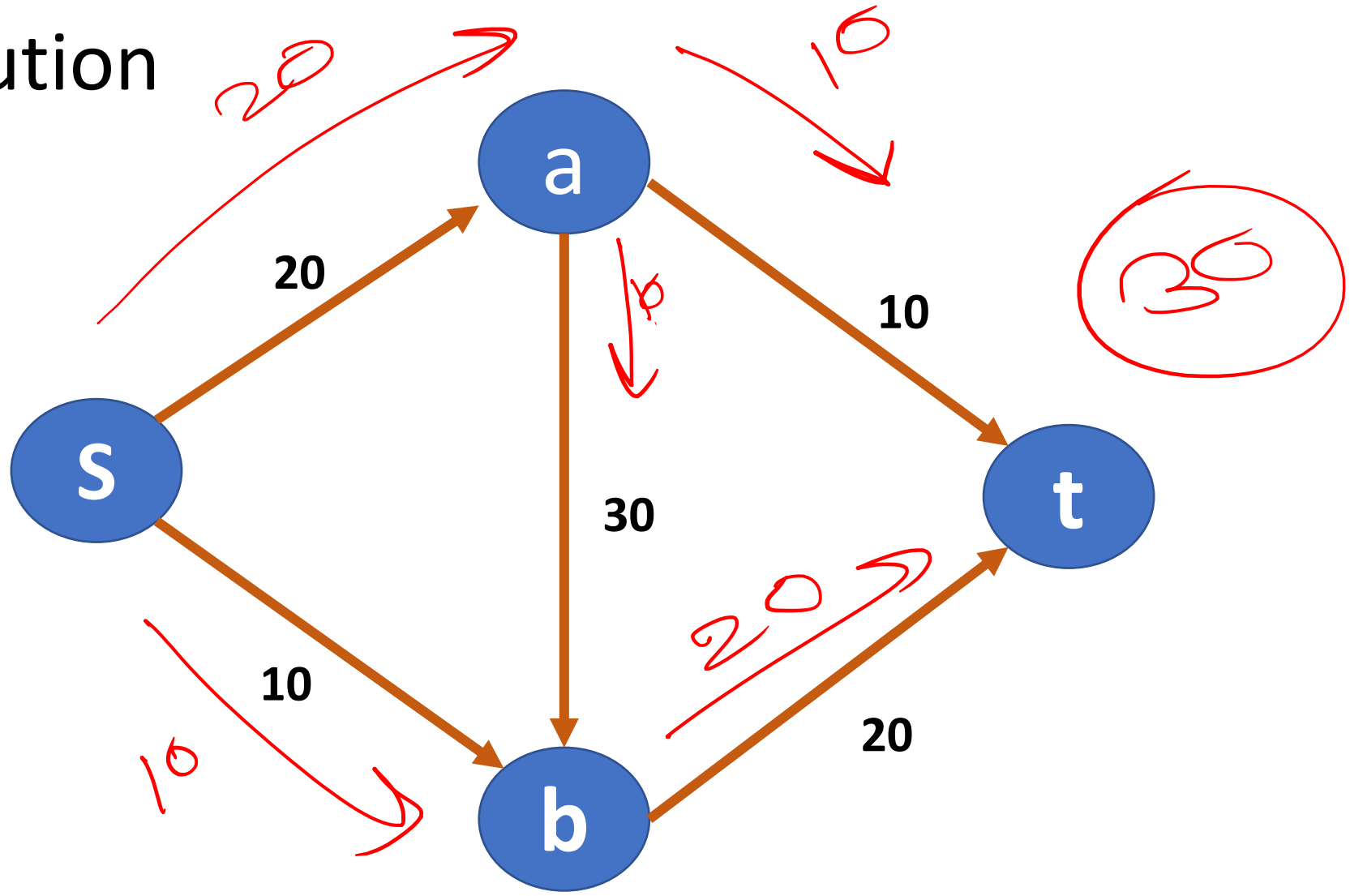


Greedy Solution



There is a better one!

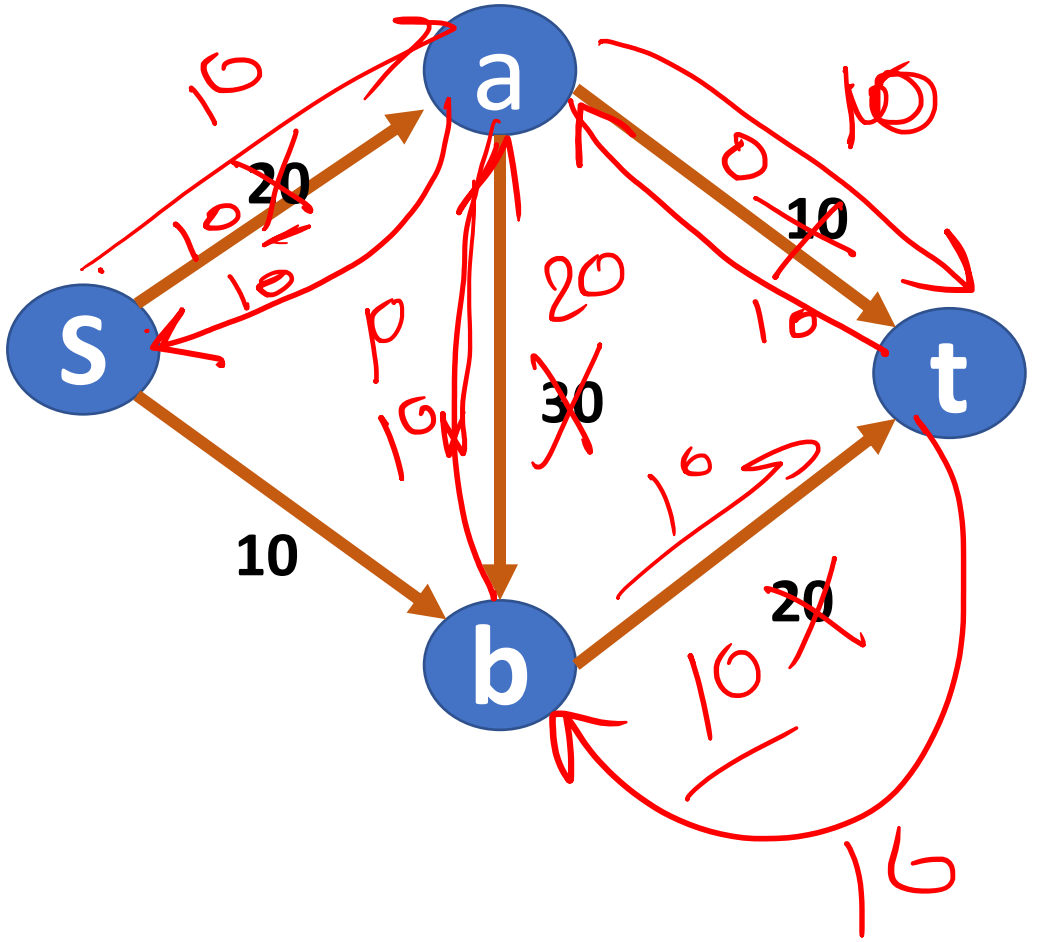
Greedy Solution



There is a better one!

Residual graphs

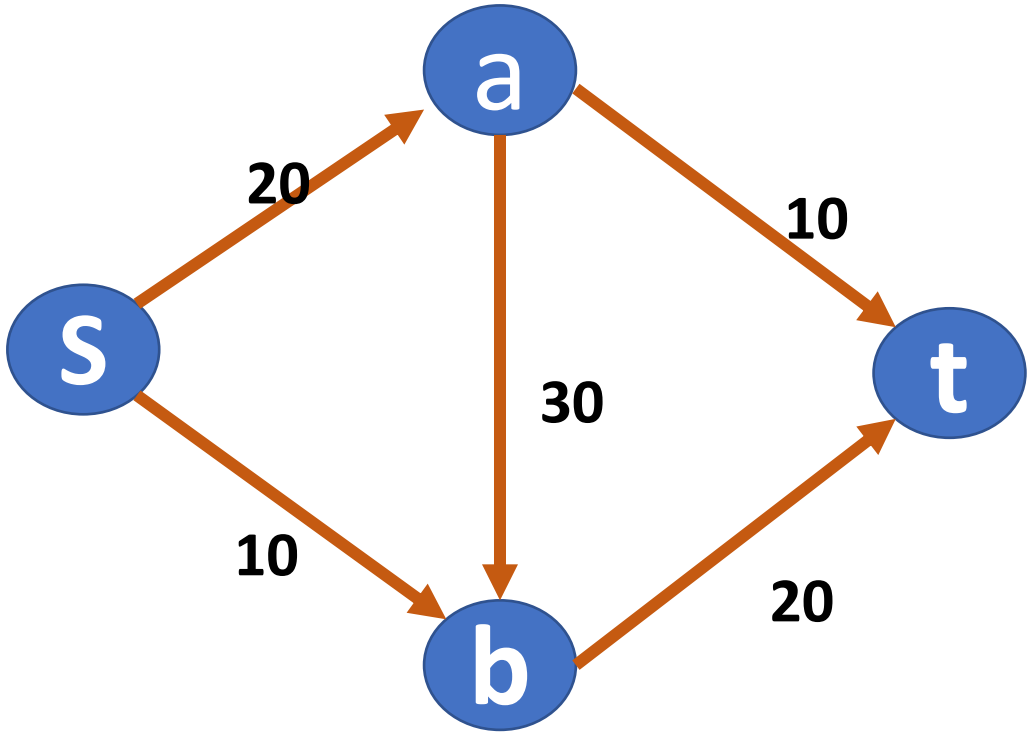
- $G_f=(V,E_f)$ based on flow f
- New set of edges **E_f:f**
  - **If f(e) > 0 in f , where e=(u,v),**
    then update the edge e=(u,v)
    with new capacity c(e) – f(e)
  - And add a new edge e'=(v,u)
    with capacity c(c')=f(e)

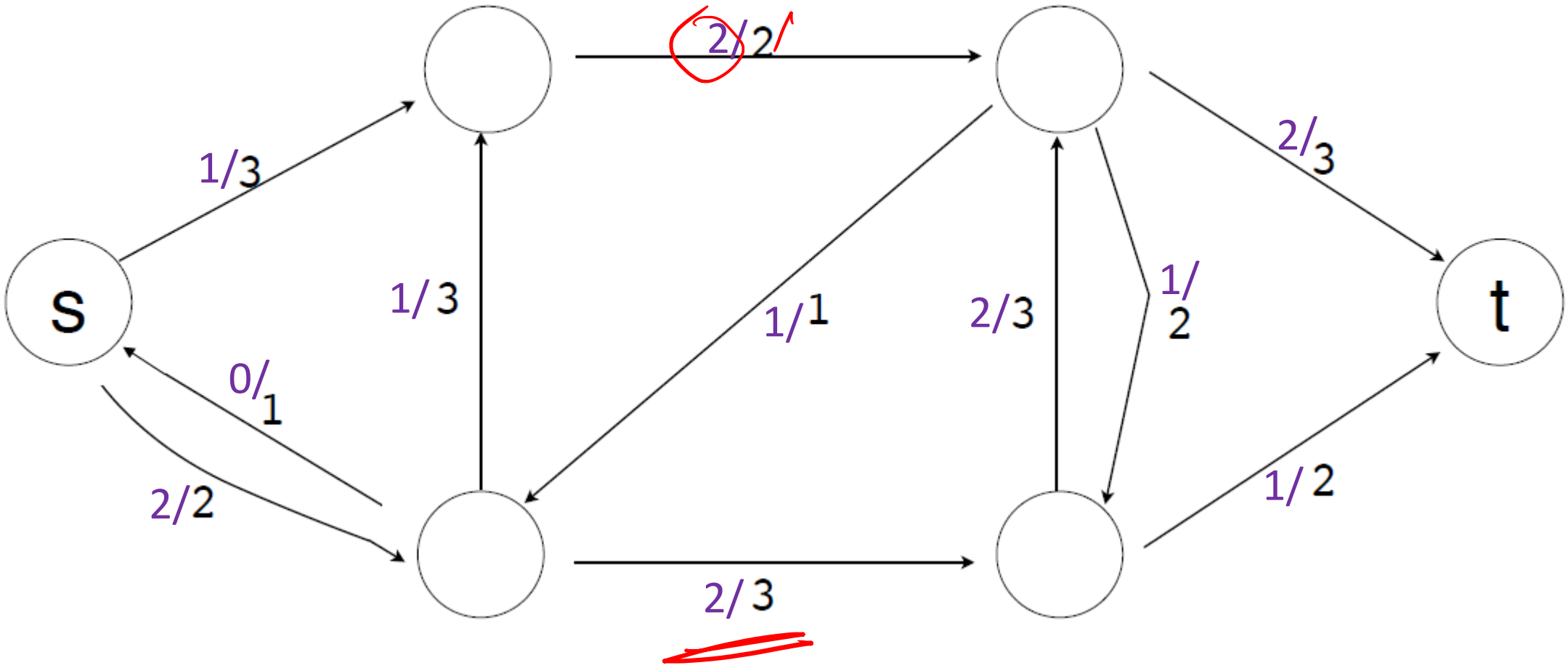


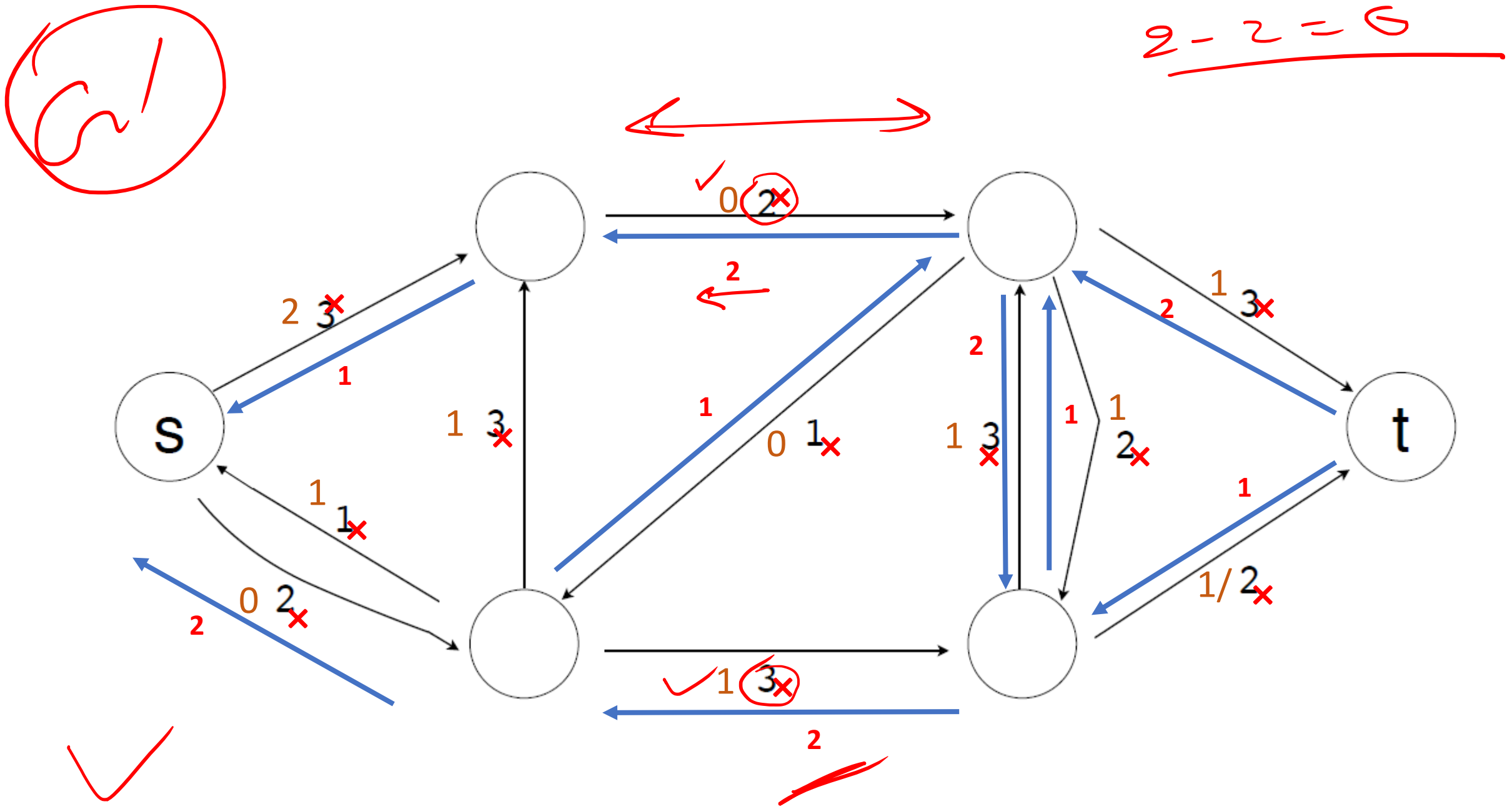
Residual graphs

- $G_f=(V,E_f)$ based on flow f
- New set of edges **E_f:f**
  - **If f(e) > 0 in f , where e=(u,v),**

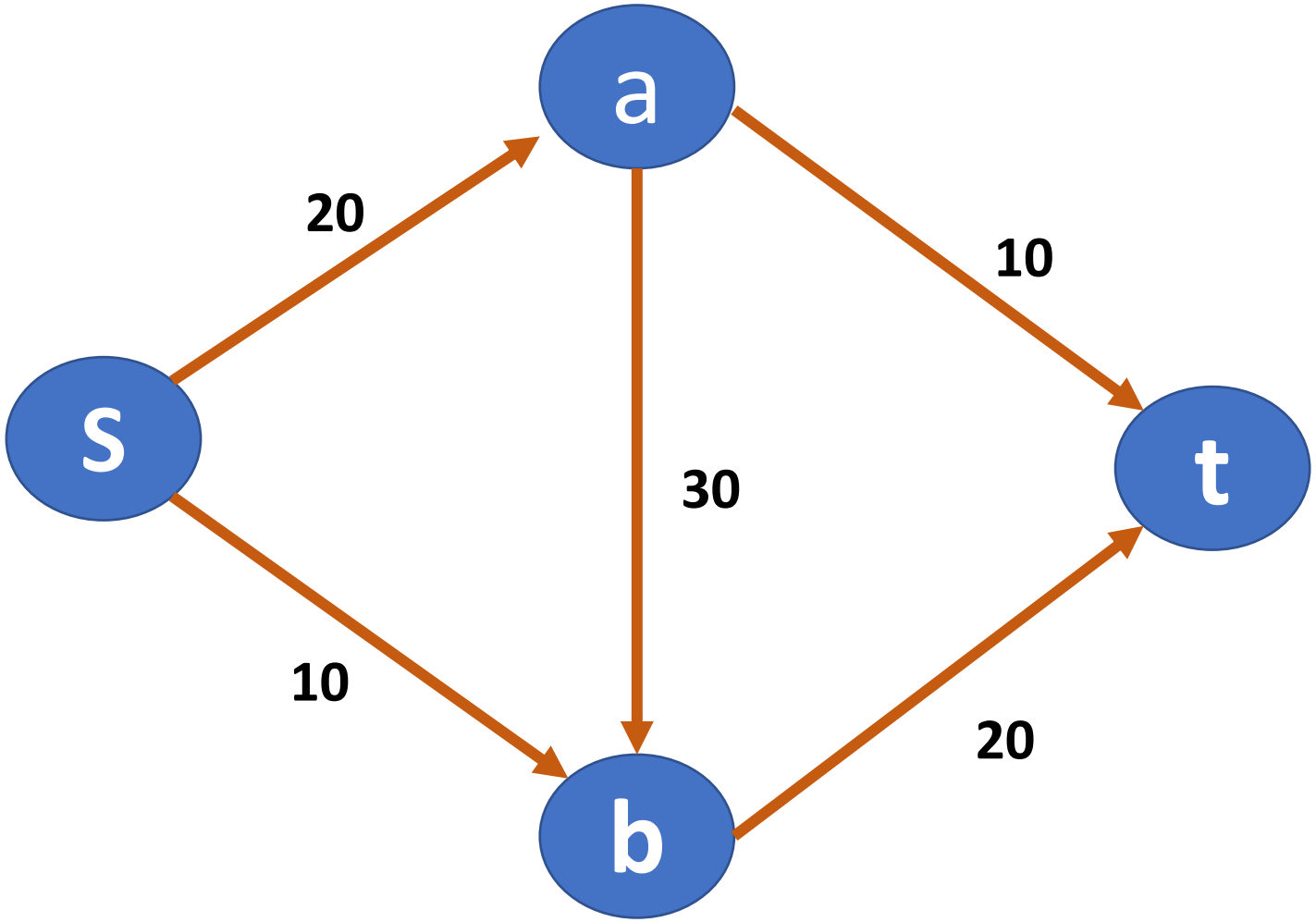
  then update the edge e=(u,v) with new capacity c(e) – f(e)
  - And add a new edge e'=(v,u) with capacity c(c')=f(e)



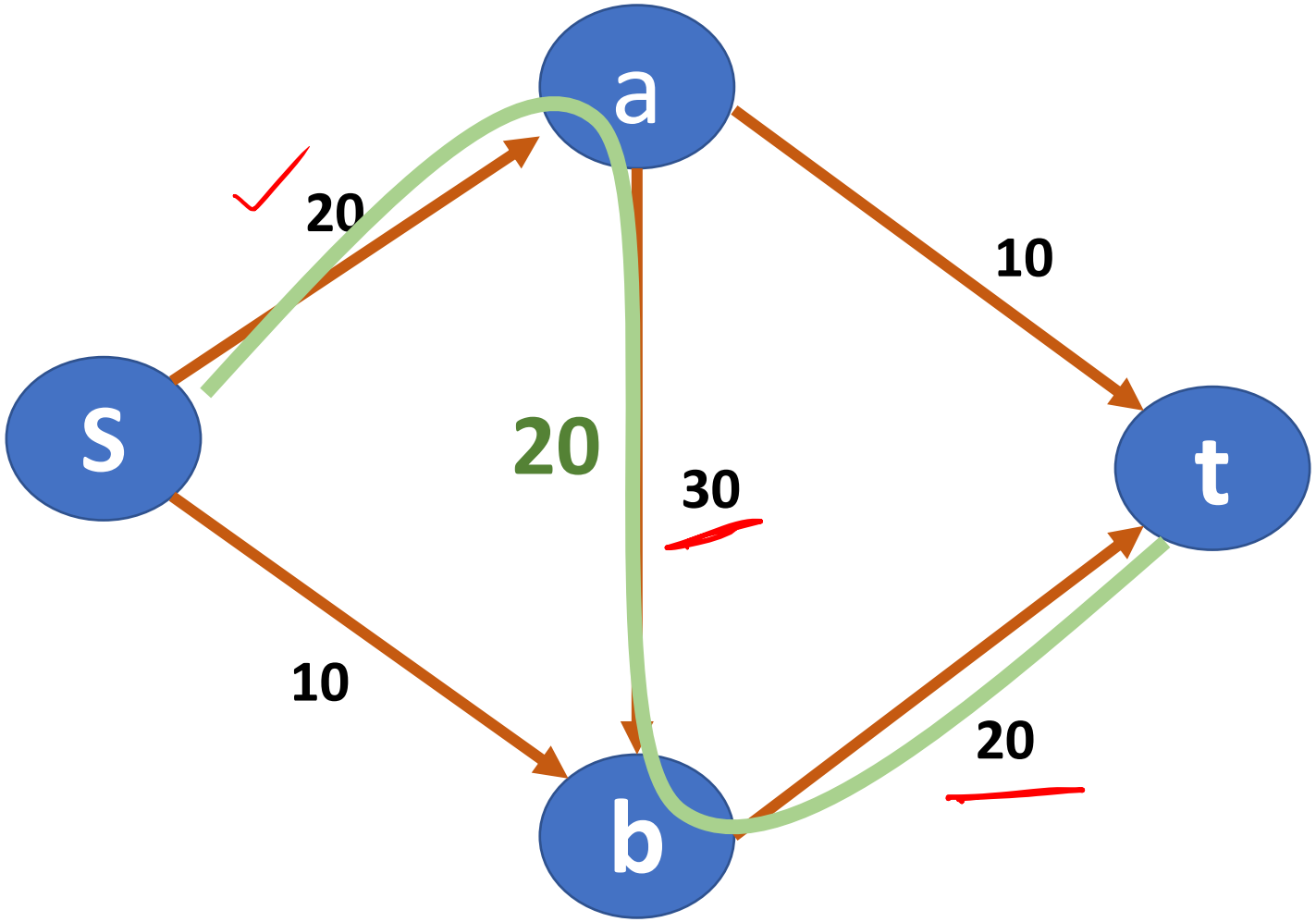

s
t
1/3
2/2
2/3
1/3
1/1
2/3
1/2
0/1
2/2
1/2
2/3


s
t
0 2
2
2 3
1
1 3
1
0 1
1 3
2
1
1
2
1 3
2
1 1
0 2
2
1 3
2
1
1/ 2

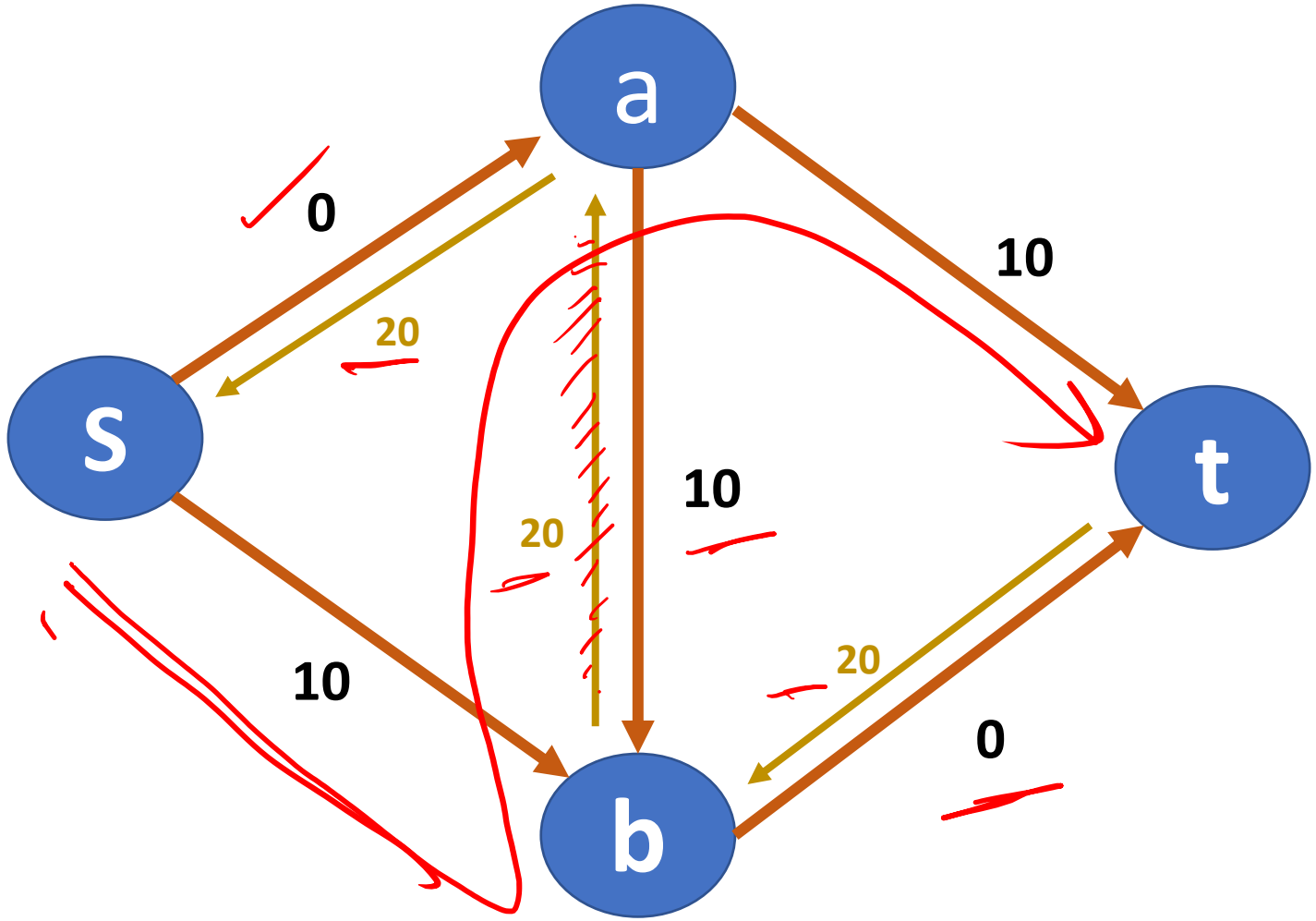
Greedy with Residual Graph Solution



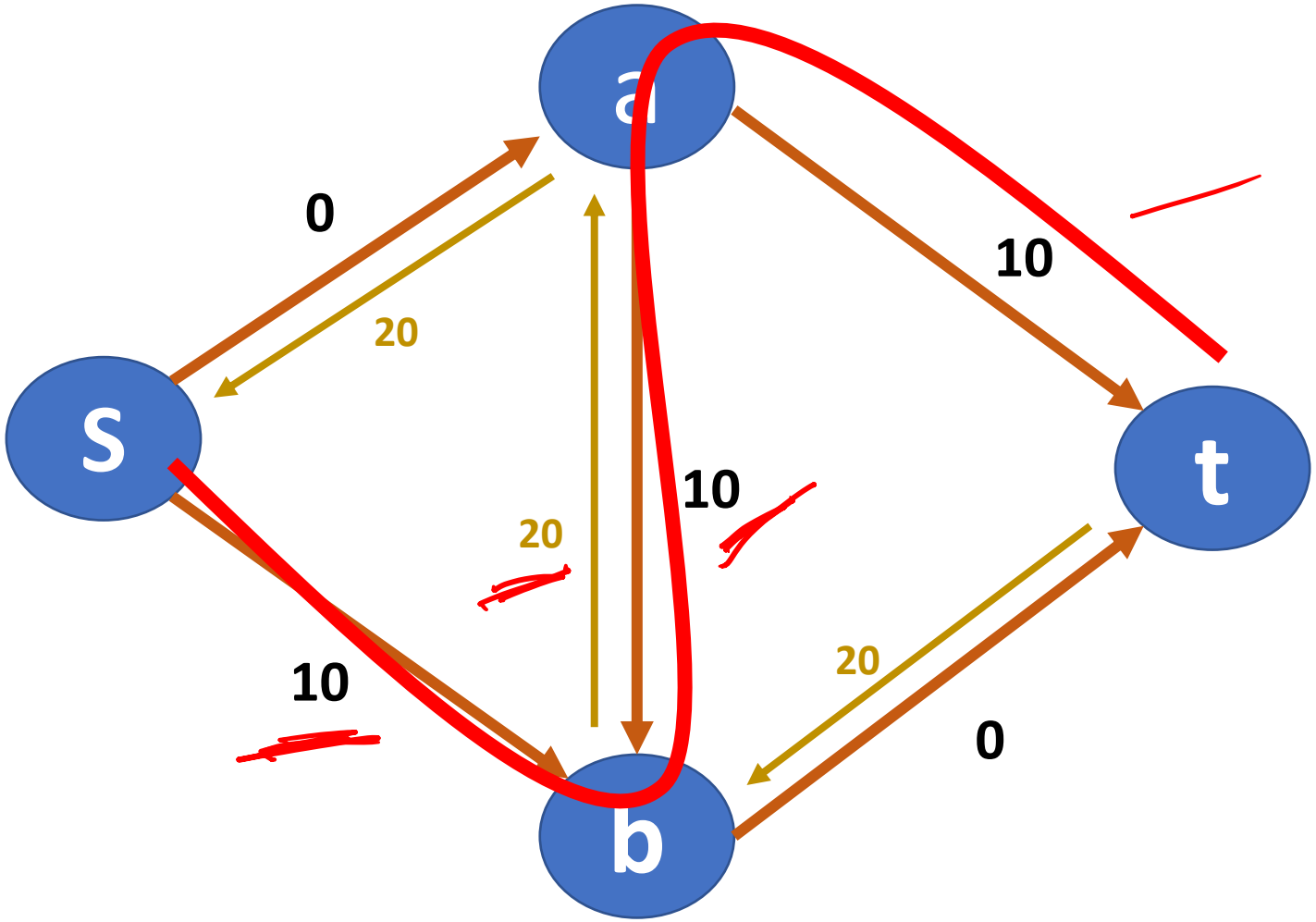
Greedy with Residual Graph Solution



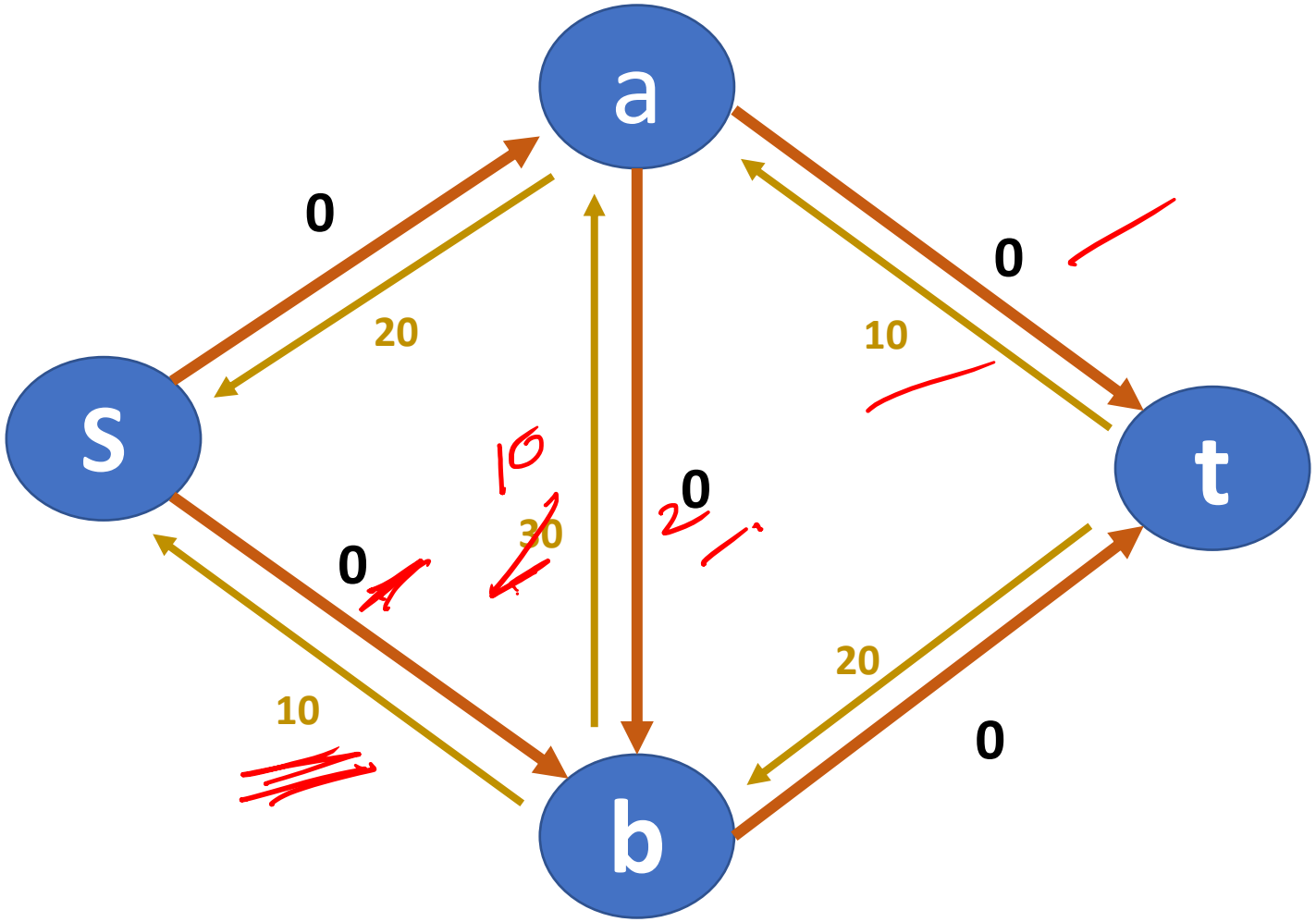
Greedy with Residual Graph Solution



Greedy with Residual Graph Solution



Greedy with Residual Graph Solution

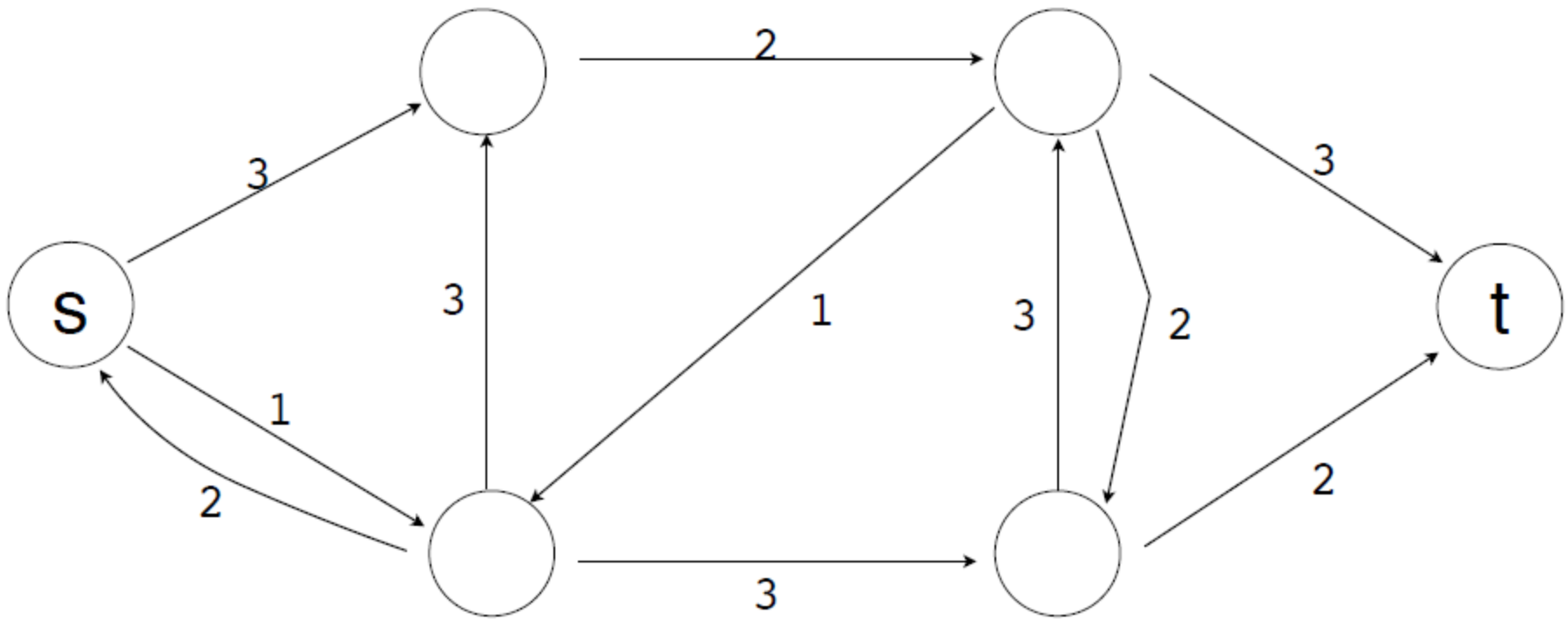


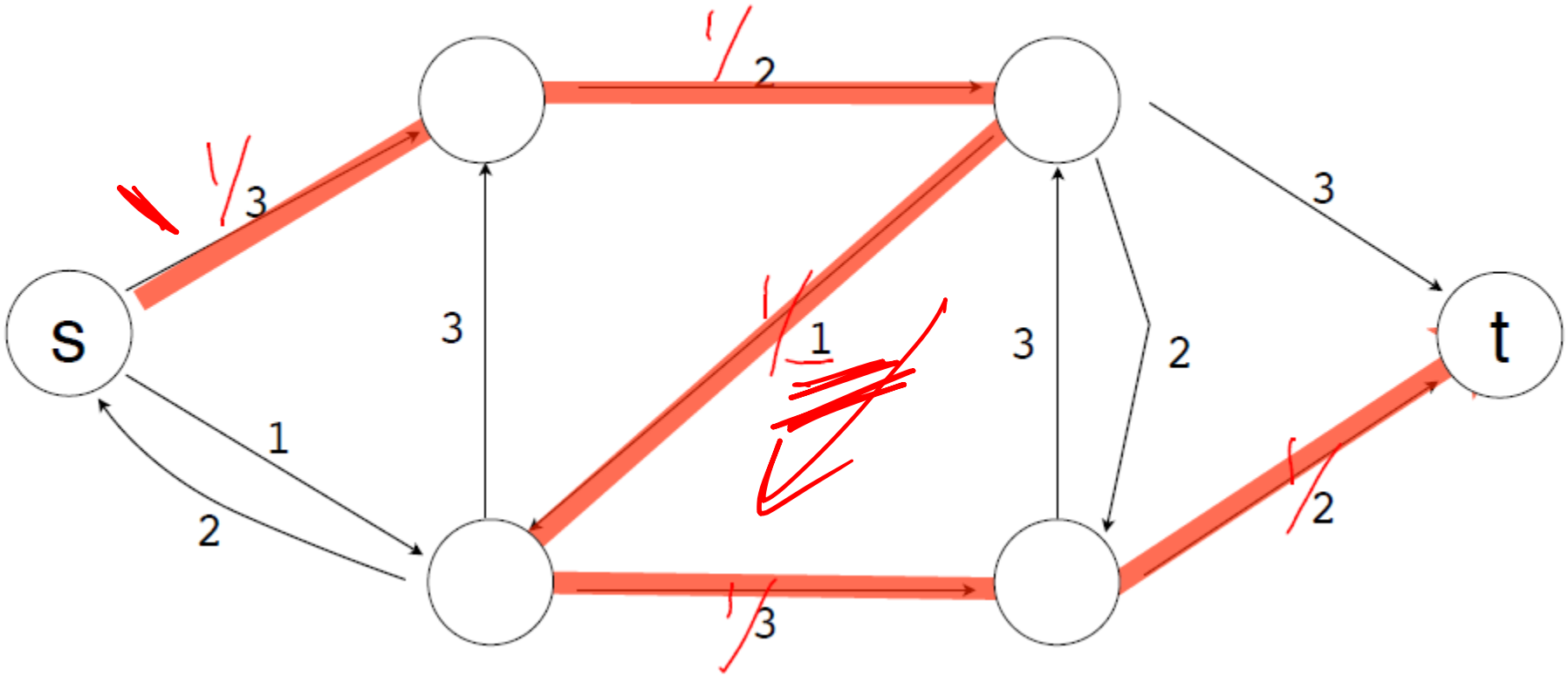
ford-fulkerson

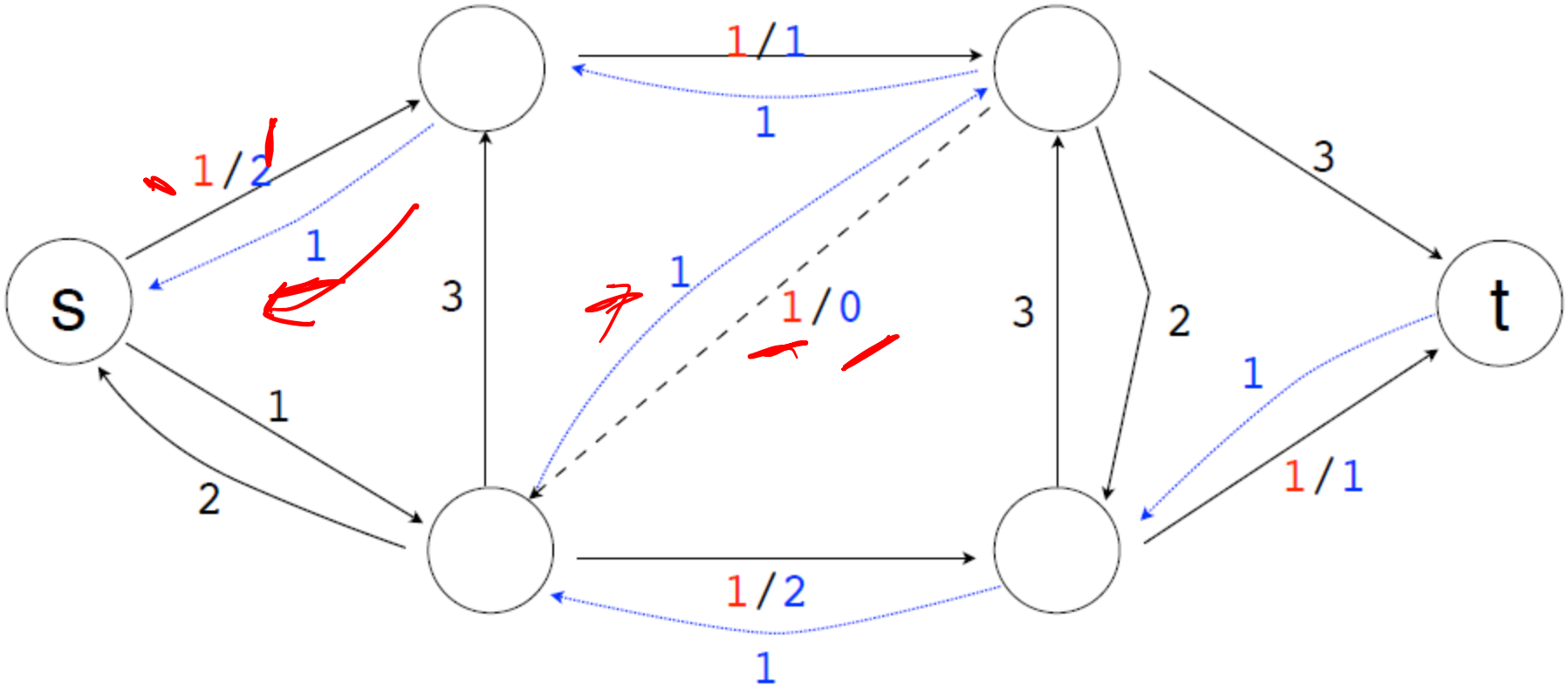
initialize $f(u,v) \leftarrow 0 \; \forall u,v$

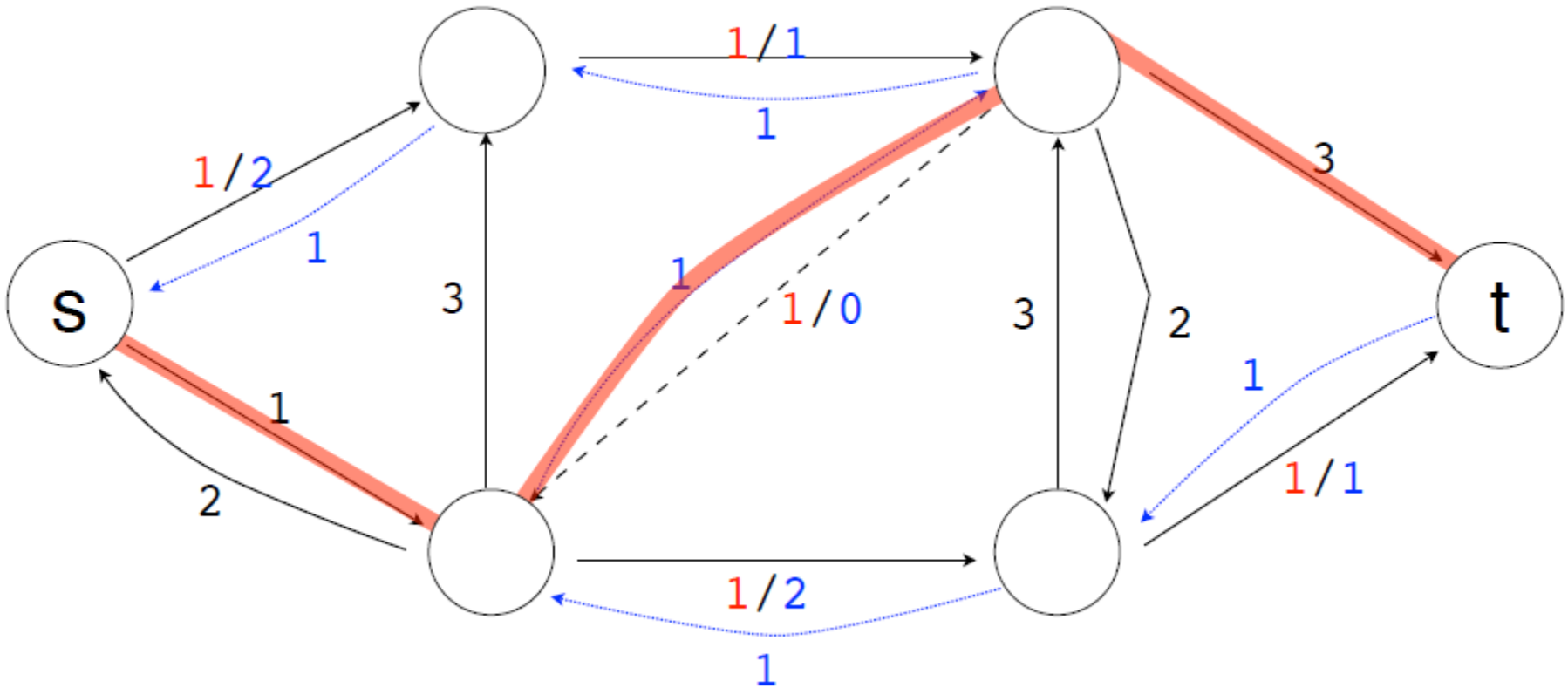
while exists an augmenting path p in G_f

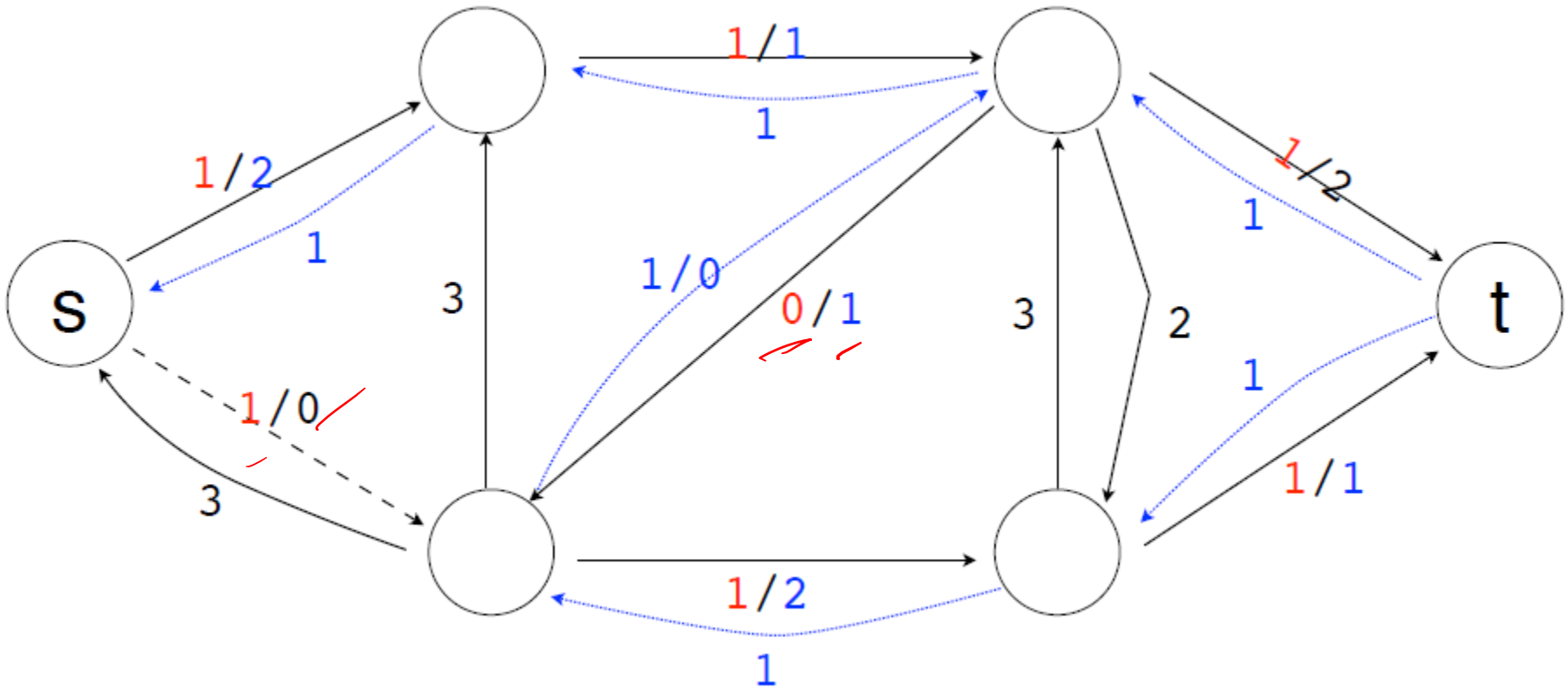
augment f with $c_f(p) = \min_{(u,v)\in p} c_f(u,v)$

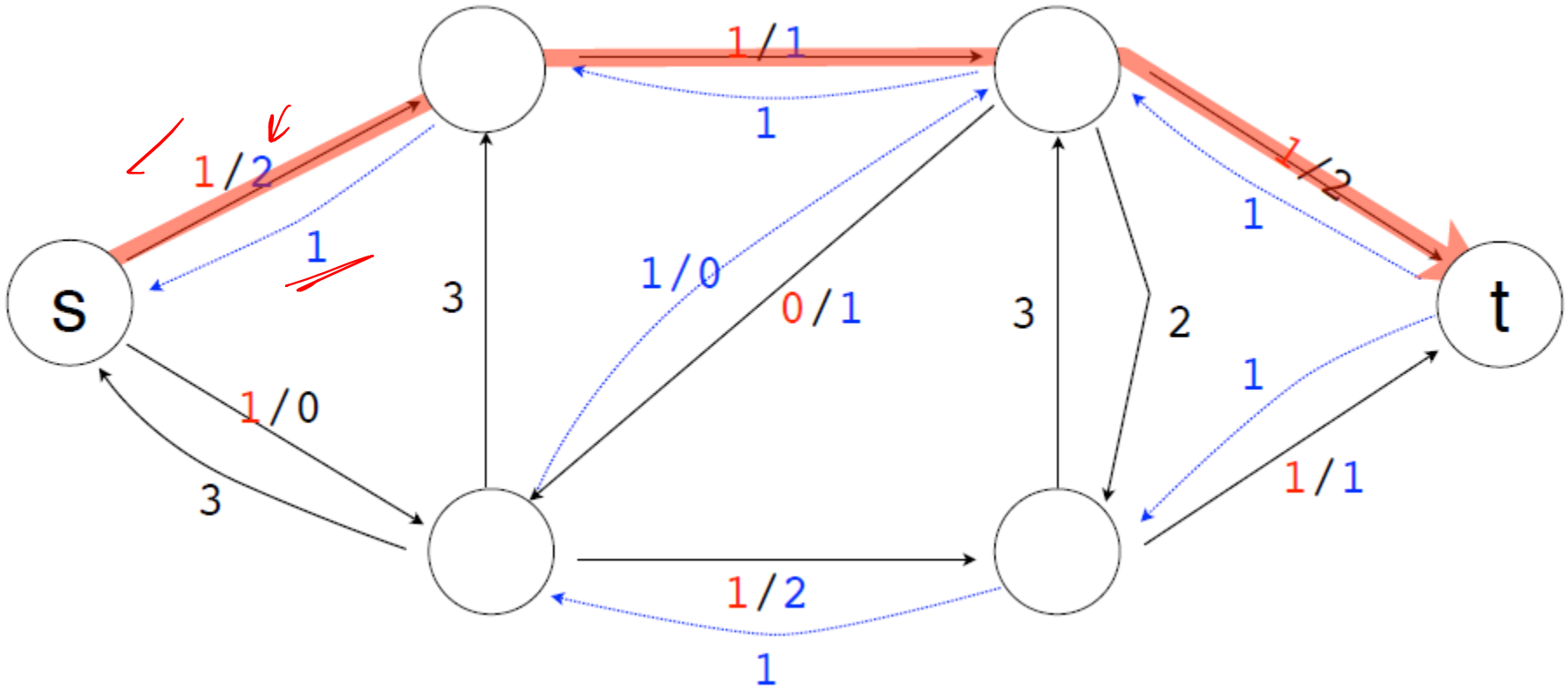

s
t
3
2
3
1
3
2
1
2
3
2

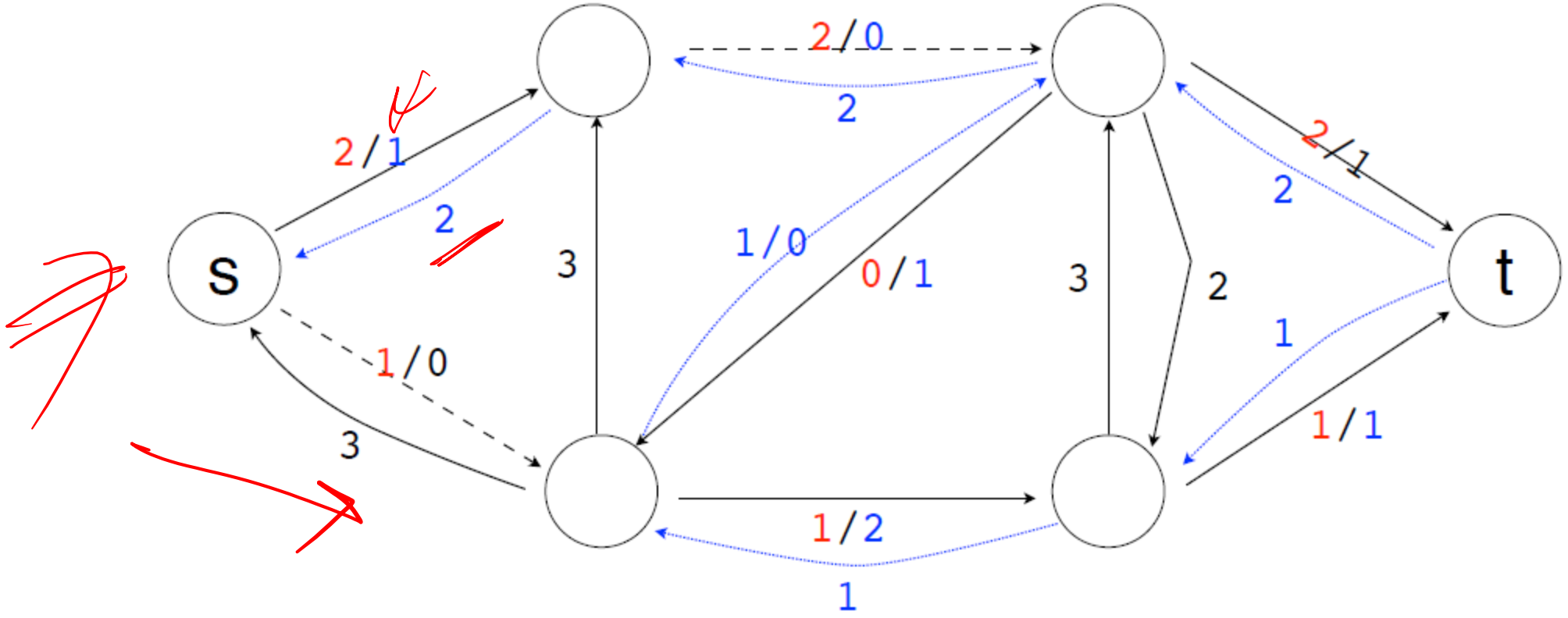

s
t
3
2
1
2
3
3
1
3
2
3
2


s
t
1/1
1
1/2
1
3
1
1/0
3
2
3
1
1/1
2
1/2
1


s
t
1 / 1
1
1 / 2
1
3
1
1 / 0
3
2
3
1
1 / 1
1
2
1 / 2
1


s
t
1/2
1
1/1
1
1/2
1
1/0
0/1
3
3
2
1/0
3
1
1/1
1/2
1


1/1
1
1/2
1
1/2
1
s
3
1/0
0/1
3
2
1
t
1/0
3
1/1
1/2
1


s
t
2/1
2
2/0
2
1/0
0/1
3
3
2
2/1
2
1
1/1
1/0
3
1/2
1

ford-fulkerson

initialize $f(u,v) \leftarrow 0 \; \forall u,v$

while exists an augmenting path p in G_f

augment f with $c_f(p) = \min_{(u,v) \in p} c_f(u,v)$

time to find an augmenting path: $O(V+E)$

number of iterations of while loop: $|f|$