L17

4102 10.24.2013

abhi shelat

Min Span Trees,
Shortest paths

MST

minimum spanning tree

looking for a set of edges that $T \subseteq E$

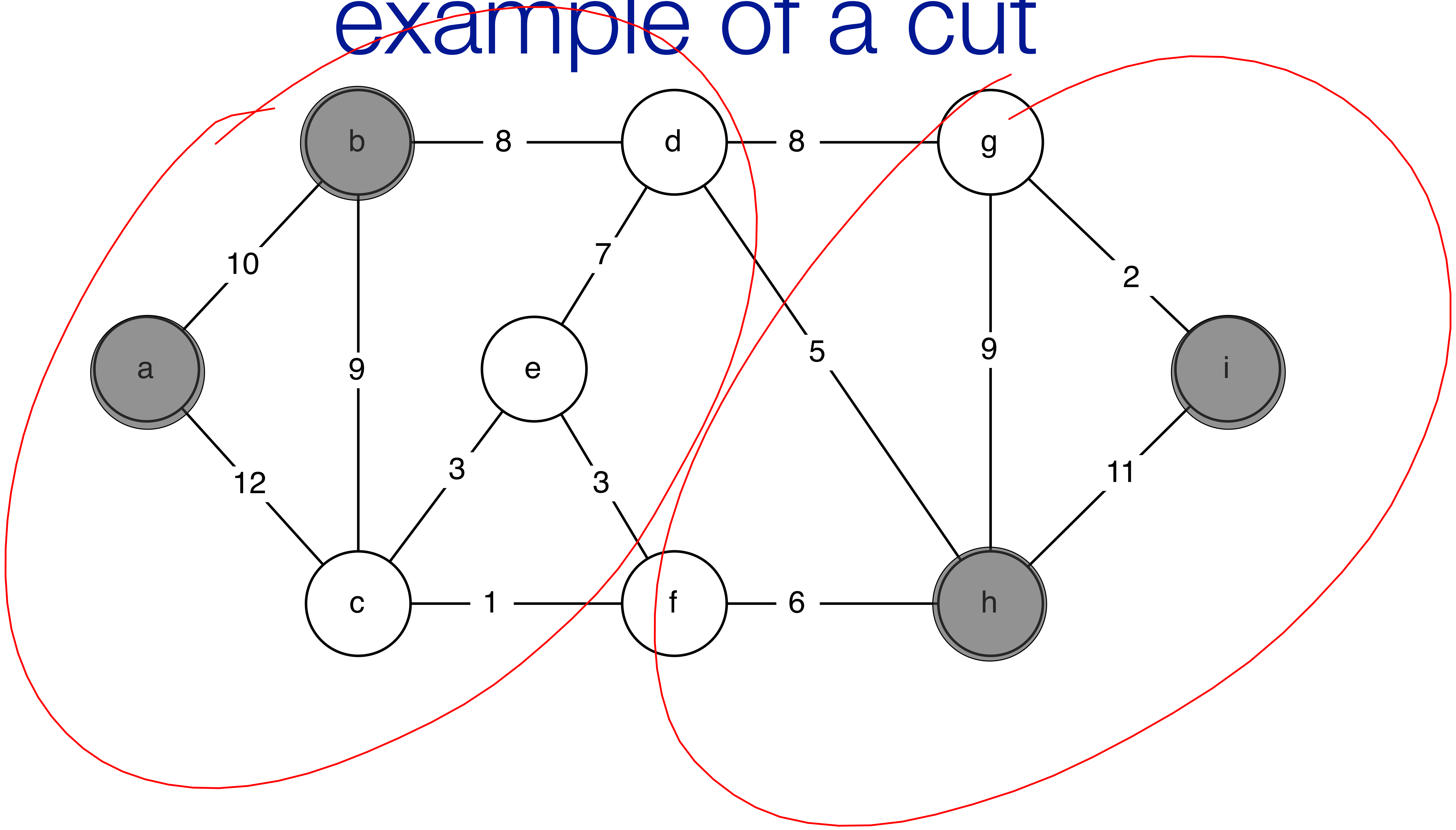
tree

(a) connects all vertices

(b) has the least cost

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

example of a cut

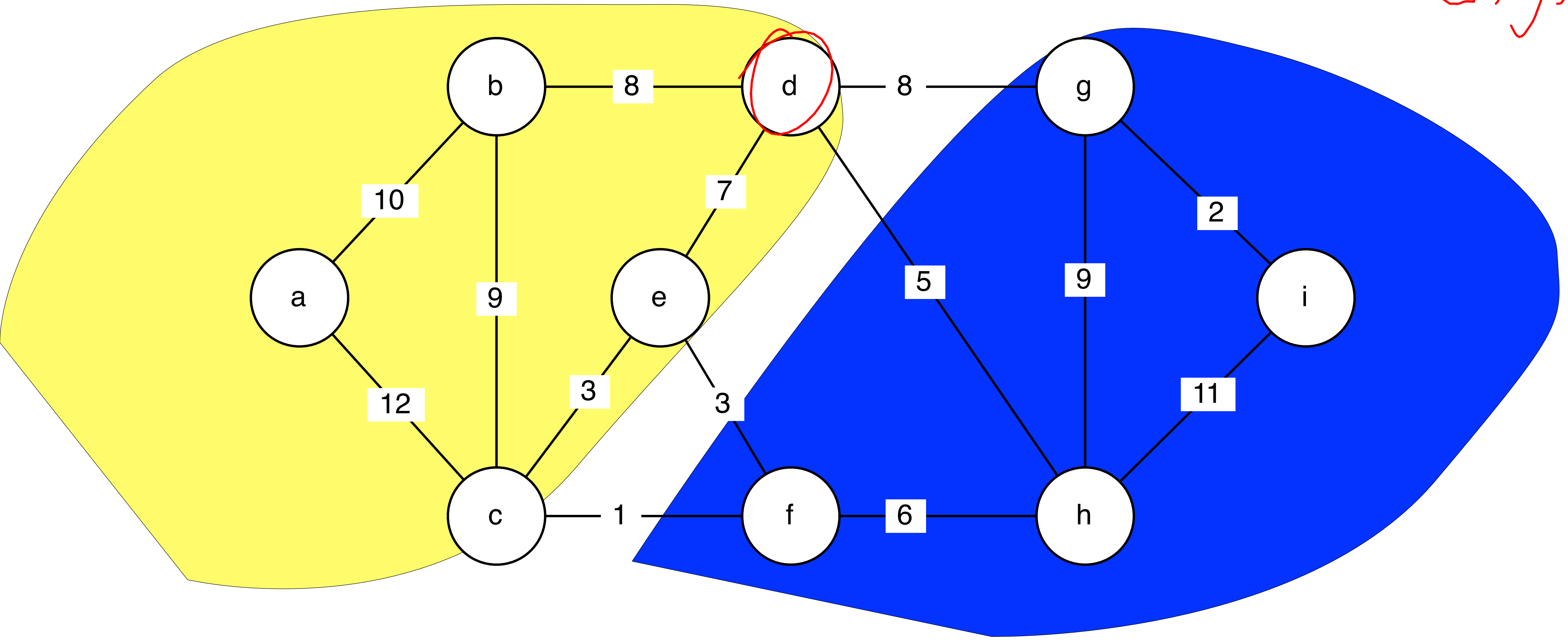


definition: crossing a cut

an edge $e = (u, v)$ crosses a graph cut (S,V-S) if

$u \in S$ $v \in V - S$

(d, g)



definition: respect

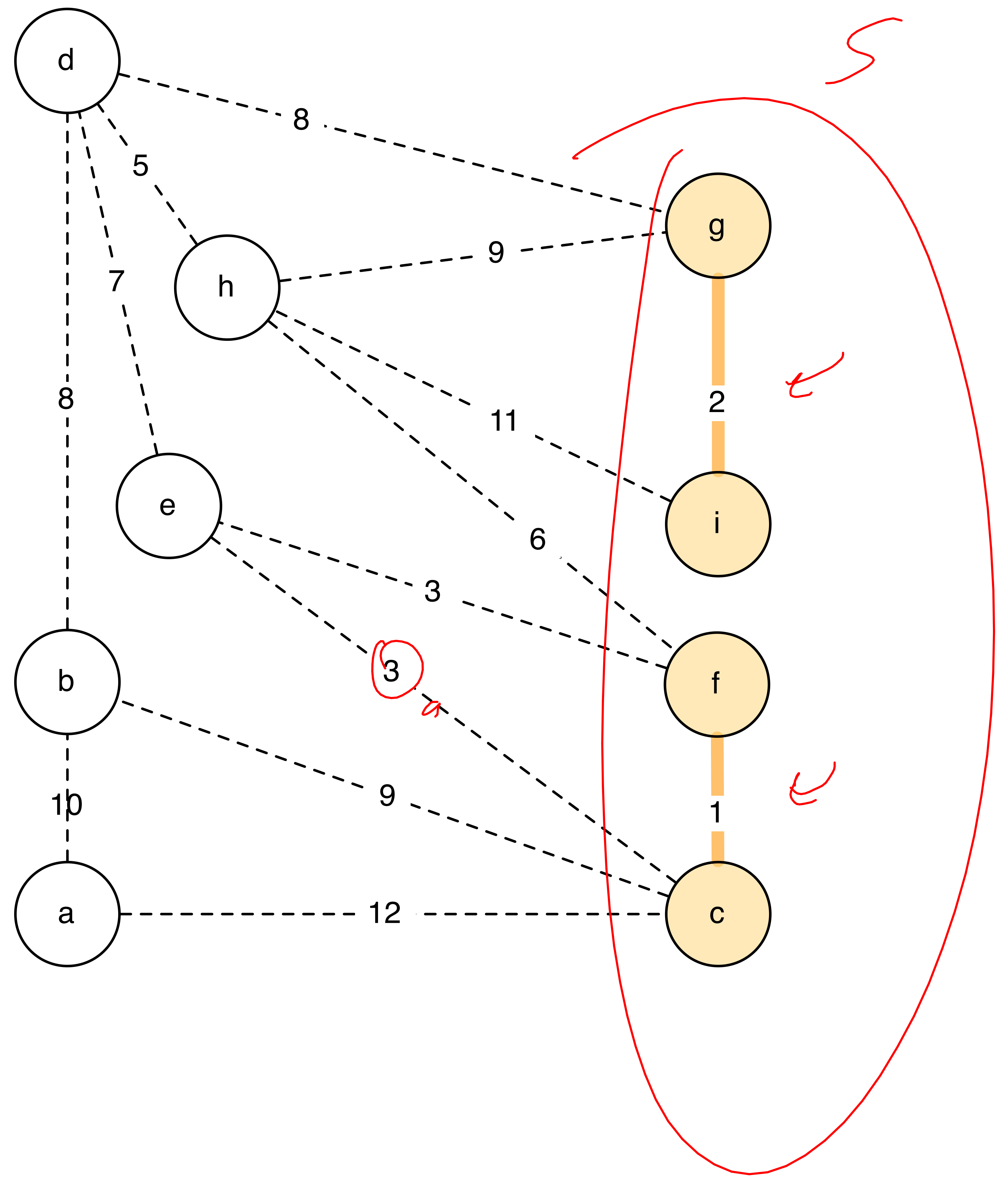
cut theorem

suppose the set of edges A is part of an m.s.t. of graph G

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.



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

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

Proven that this algorithm is correct.

By the cut theorem.

Prim's algorithm

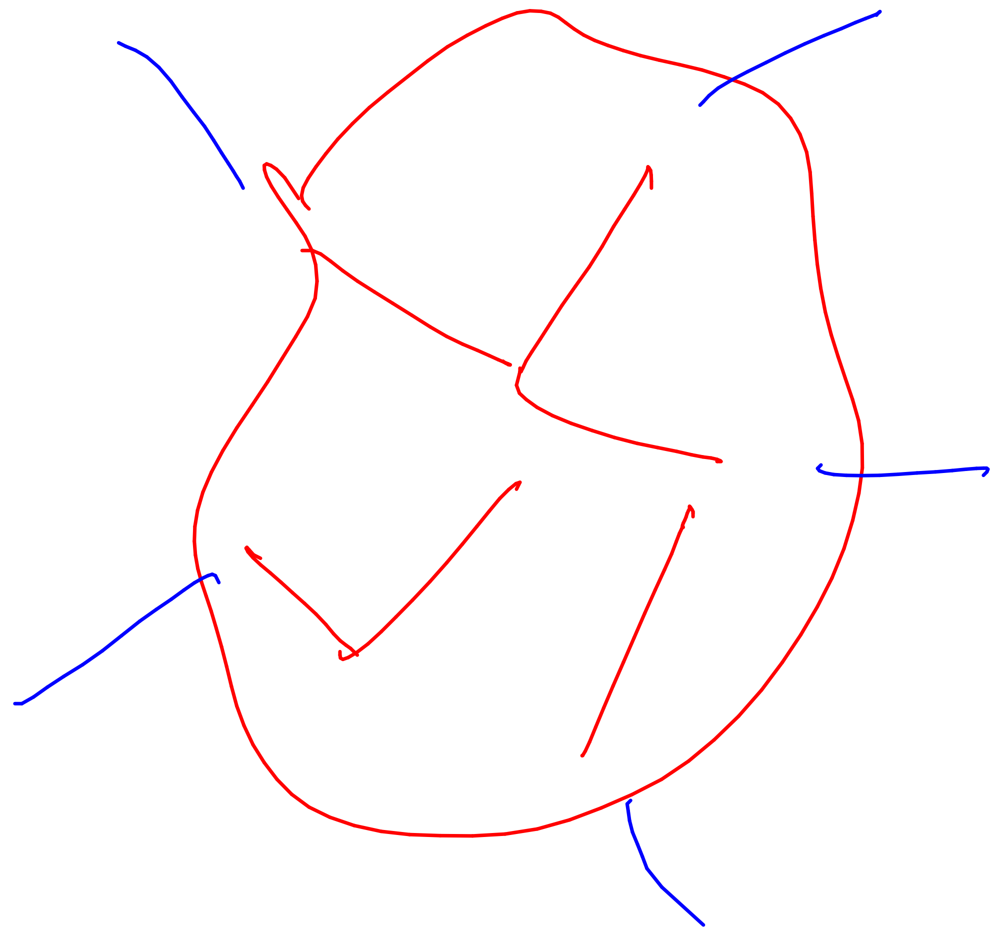
General-MST-Strategy$(G = (V, E))$

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

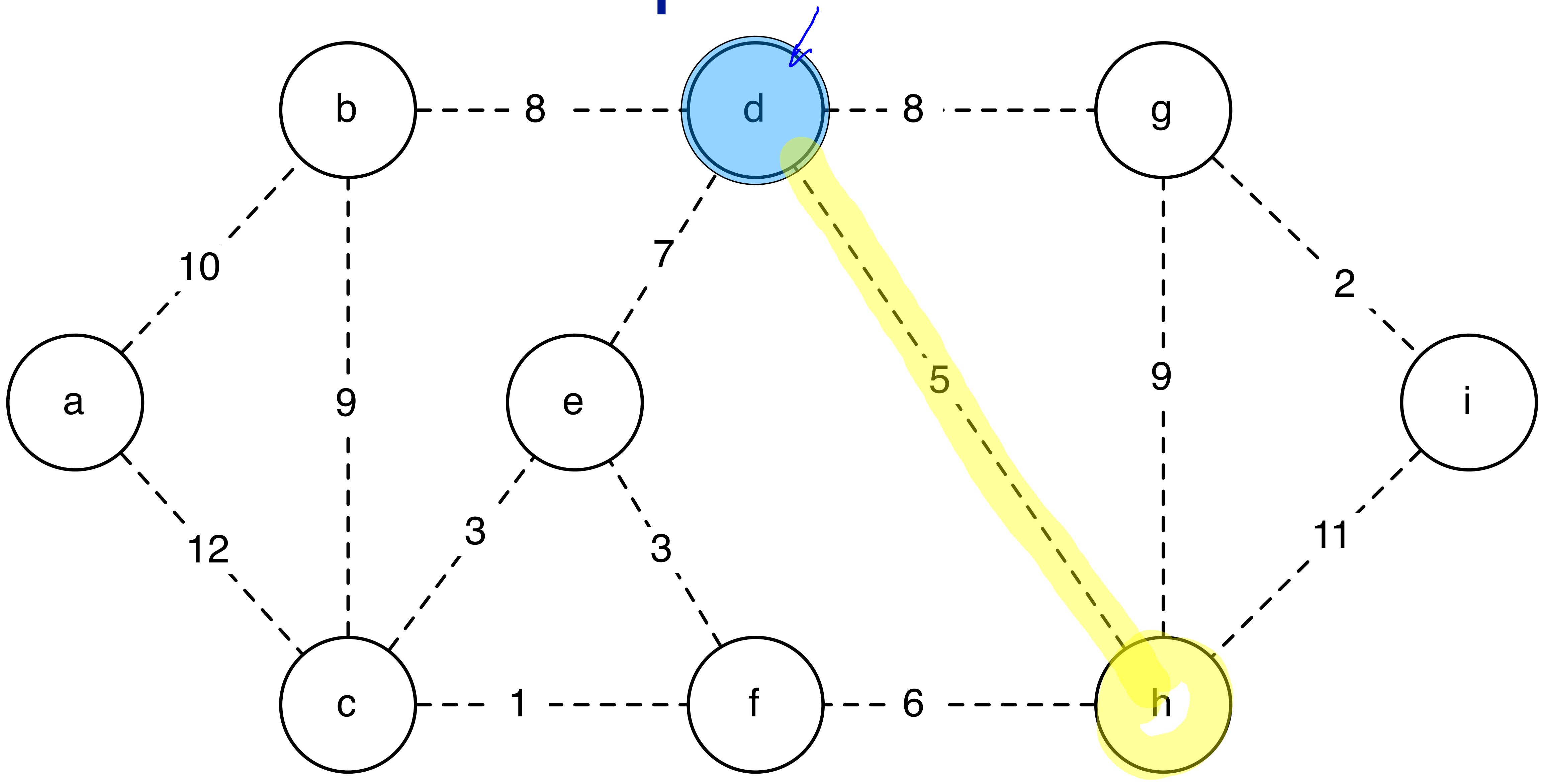
A is a subtree

$S = A$

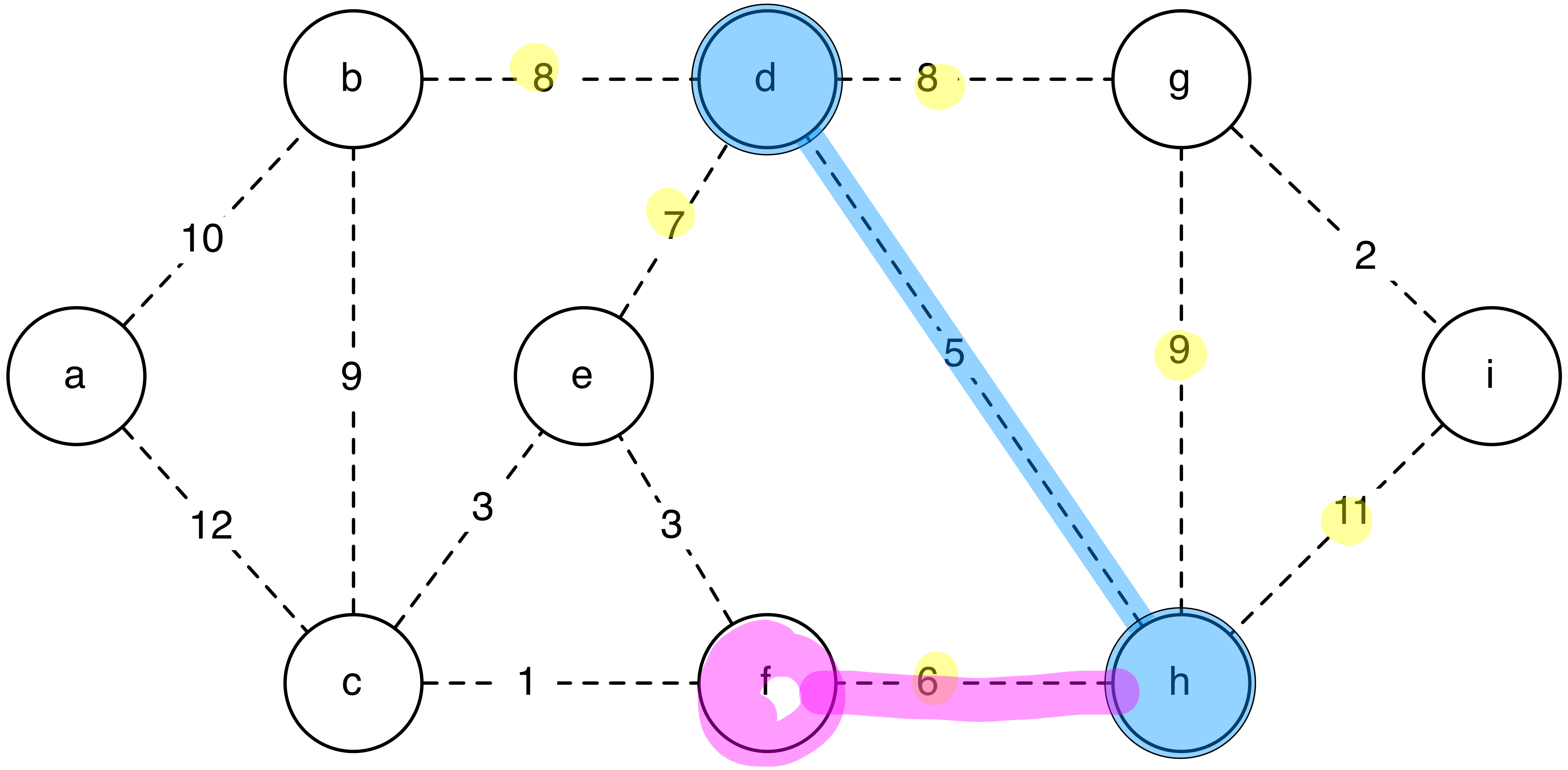
edge e is lightest edge that grows the subtree



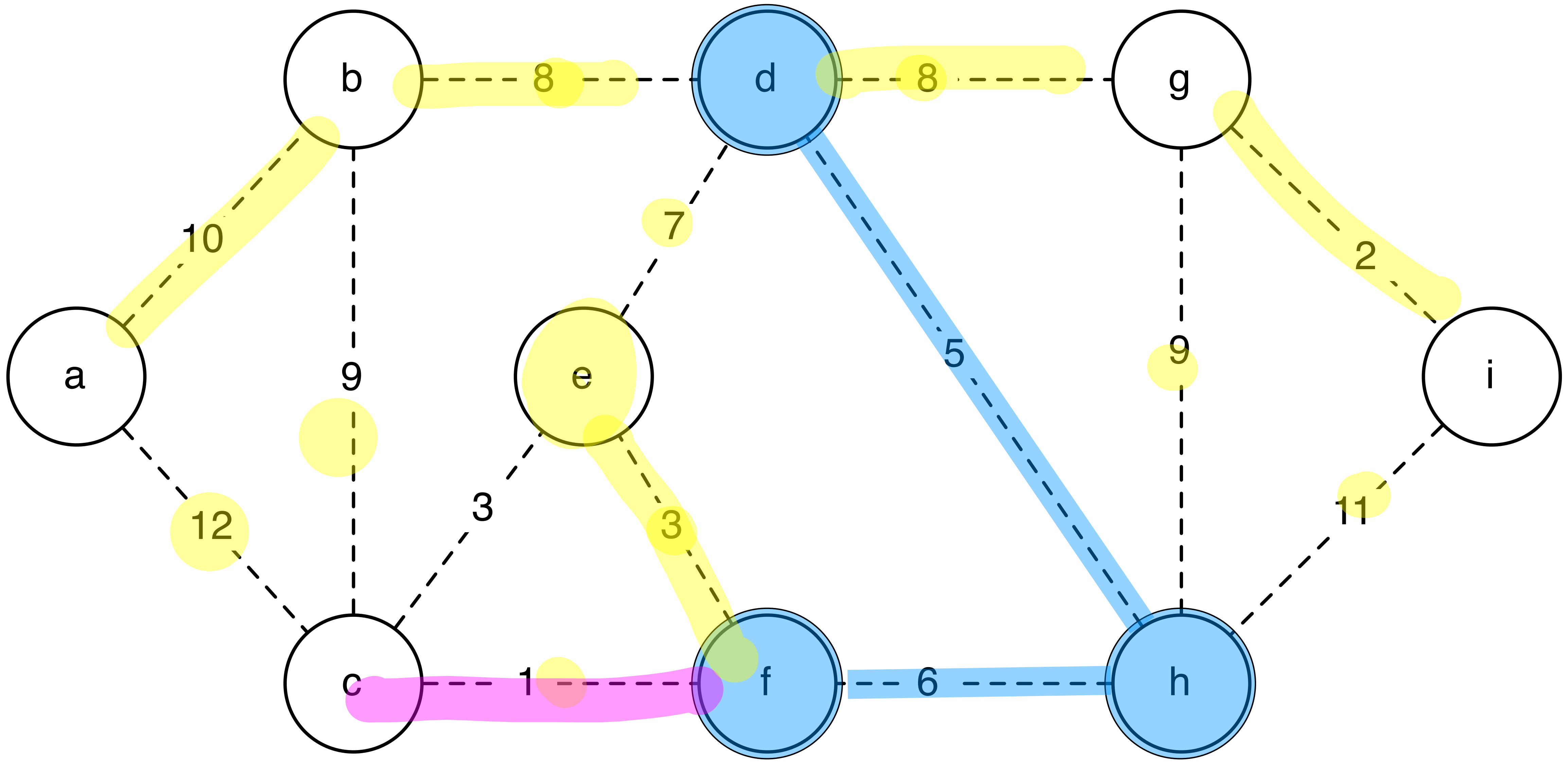
prim



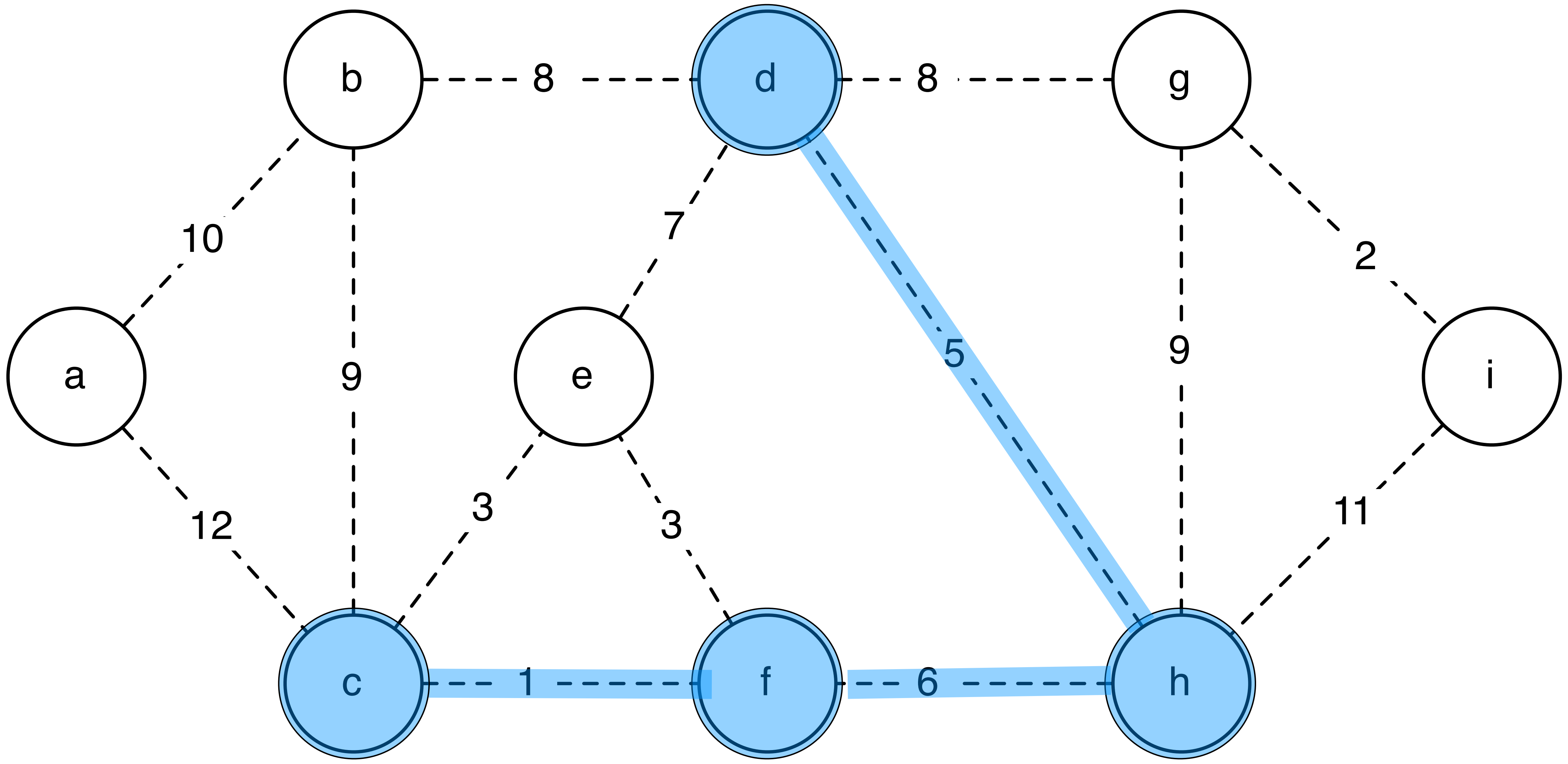
prim



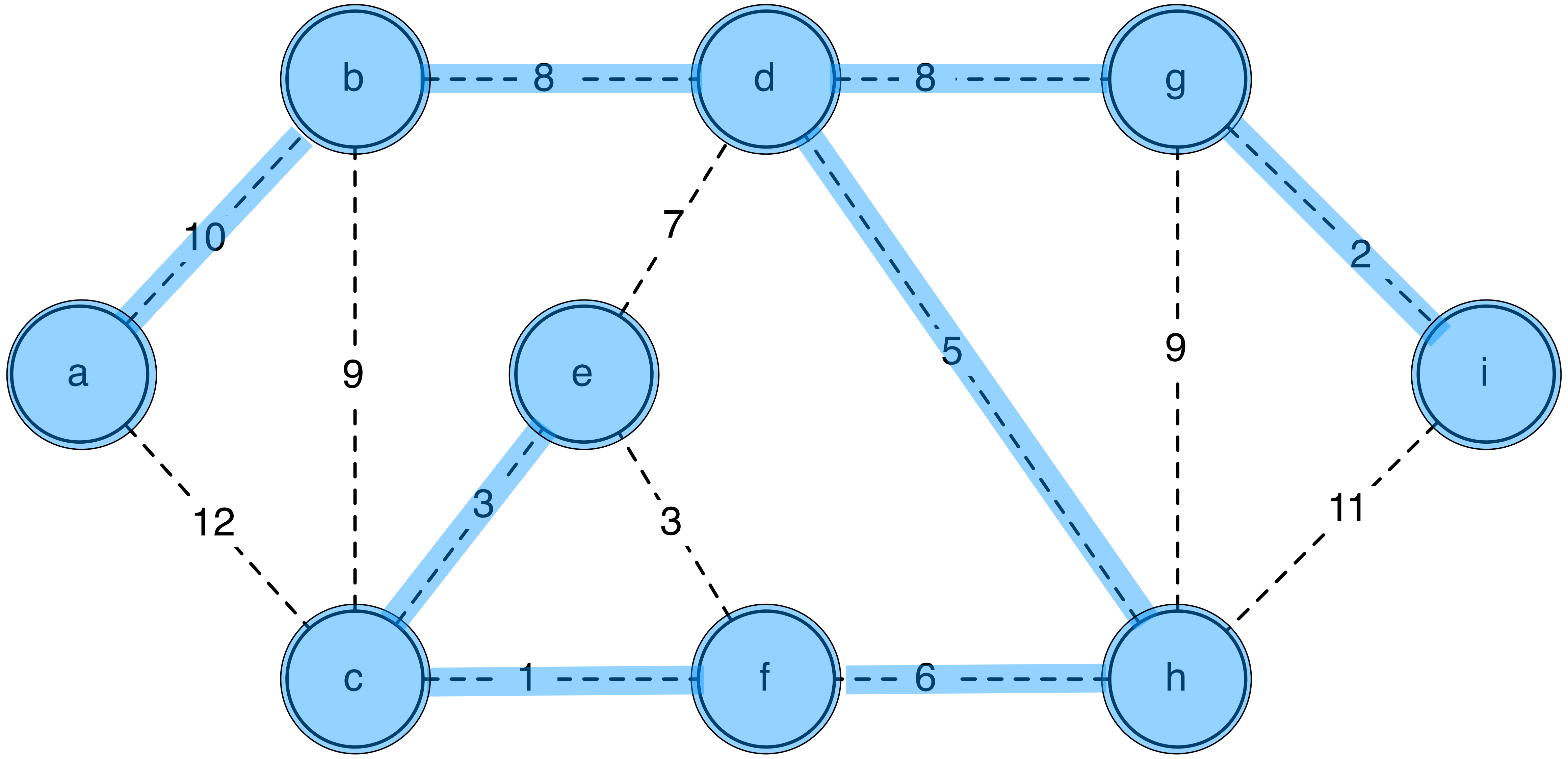
prim



prim



prim



implementation

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

Set S = A.

idea: – use a priority queue to maintain the set of edges that "emanate" from A.

– By cut theorem, this lightest edge is part of our solution.

implementation

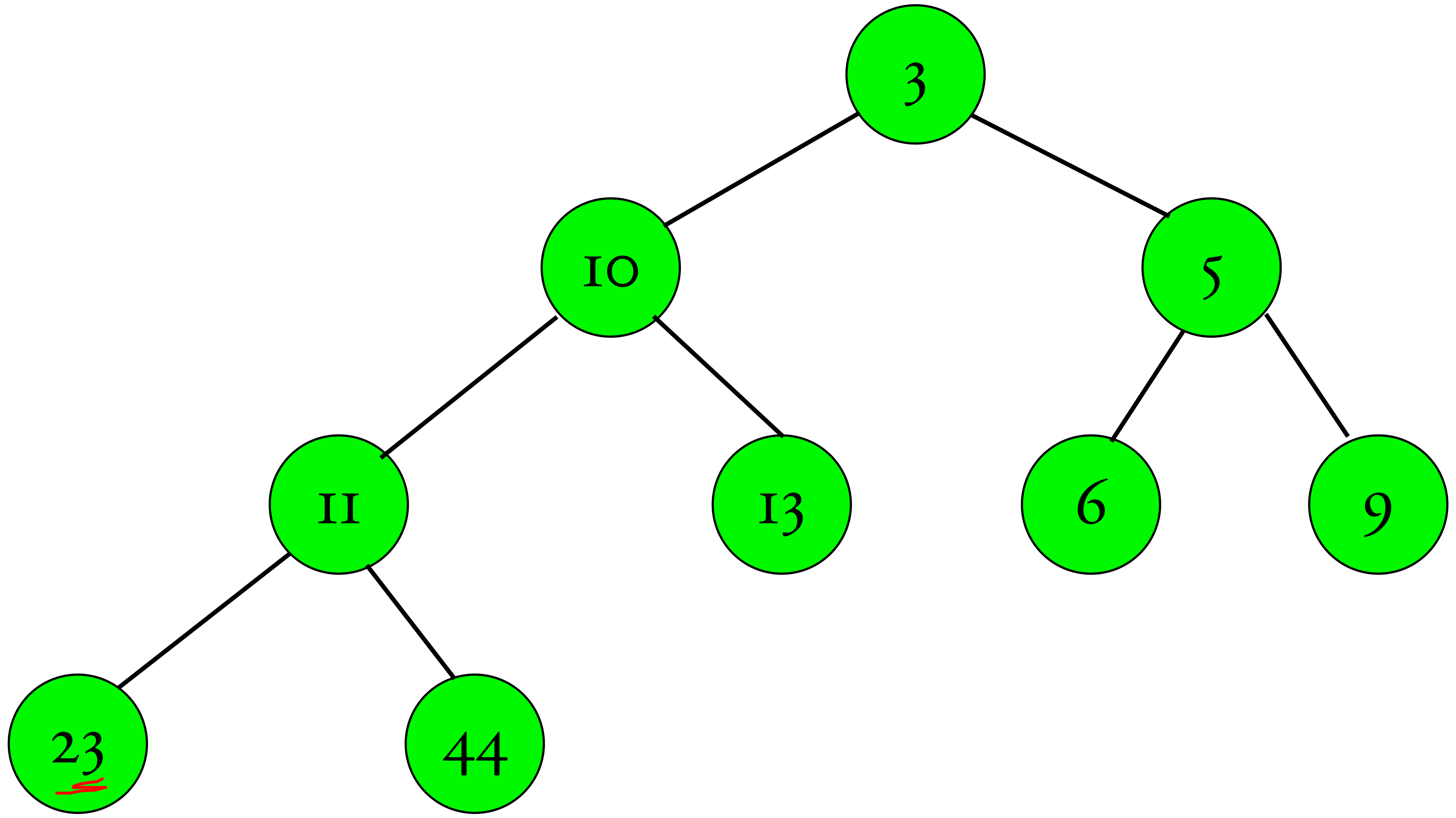
use a priority queue to keep track of light edges

insert:

makequeue:

extractmin:

decreasekey:



algorithm

$\forall\, v \in V \quad k_v \leftarrow \infty \quad \pi_v \leftarrow nil$

pick some $r \in V$ $\quad k_r \leftarrow 0$.

$Q \leftarrow \text{Makequeue}(V, k_v)$

while (Q is not empty)

$\quad u \leftarrow \text{extractmin}(Q)$

$\quad$ for each neighbor $v \in Adj(u)$

$\quad\quad$ if $\underline{v \in Q}$ and $\underline{k_v} > \underline{w(u,v)}$

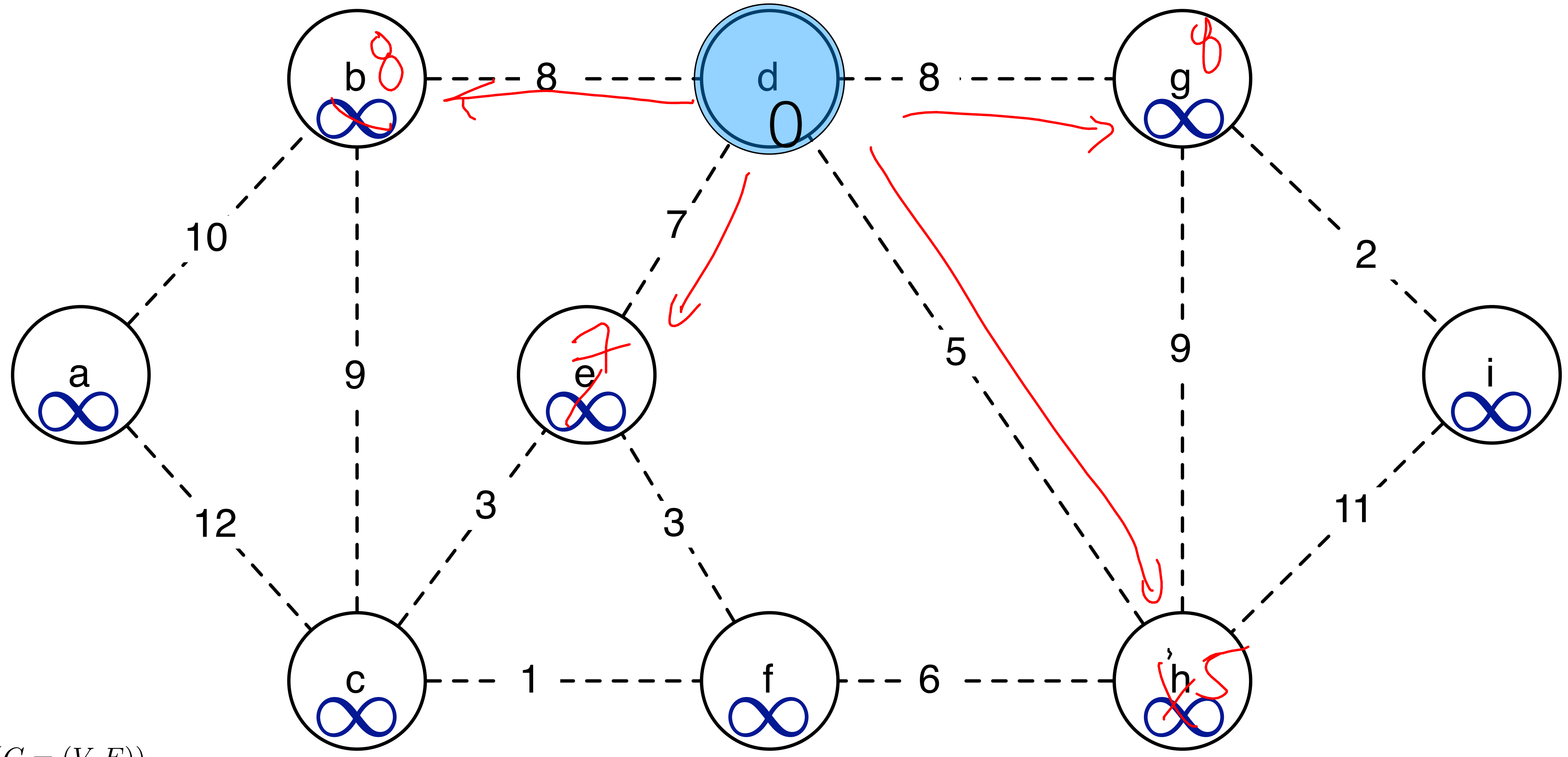
$\quad\quad\quad \underline{\text{DecreaseKey}(Q, v, w(u,v))}$

$\quad\quad\quad \underline{\pi_v \leftarrow u}$

implementation

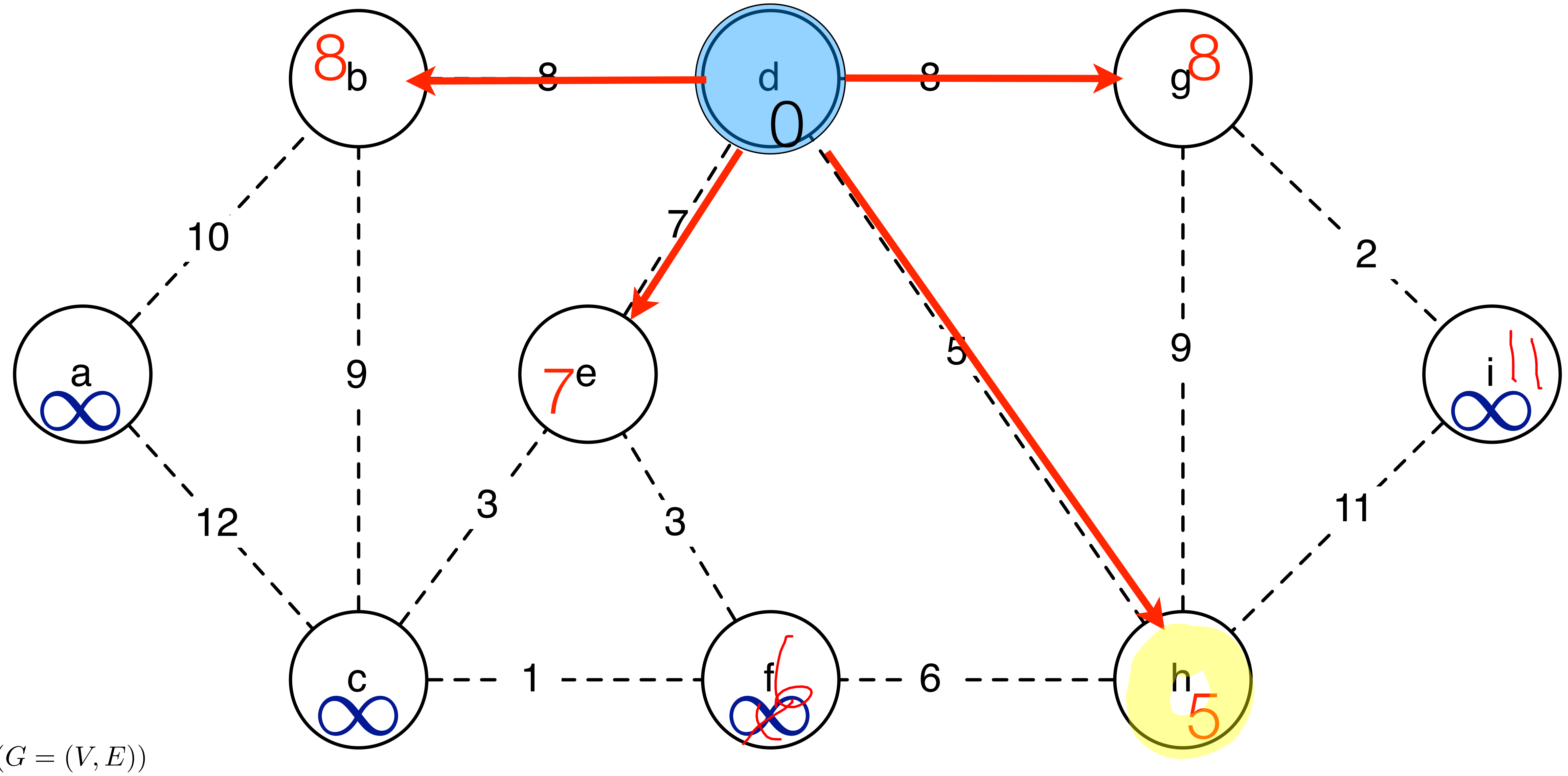
```
PRIM(G = (V, E))
Q ← ∅        ▷ Q is a Priority Queue
Initialize each v ∈ V with key k_v ← ∞, π_v ← NIL
Pick a starting node r and set k_r ← 0
Insert all nodes into Q with key k_v.
while Q ≠ ∅
      do u ← EXTRACT-MIN(Q)
         for each v ∈ Adj(u)
             do if v ∈ Q and w(u, v) < k_v
                   then π_v ← u
                        DECREASE-KEY(Q, v, w(u, v))    ▷ Sets k_v ← w(u, v)
```

prim



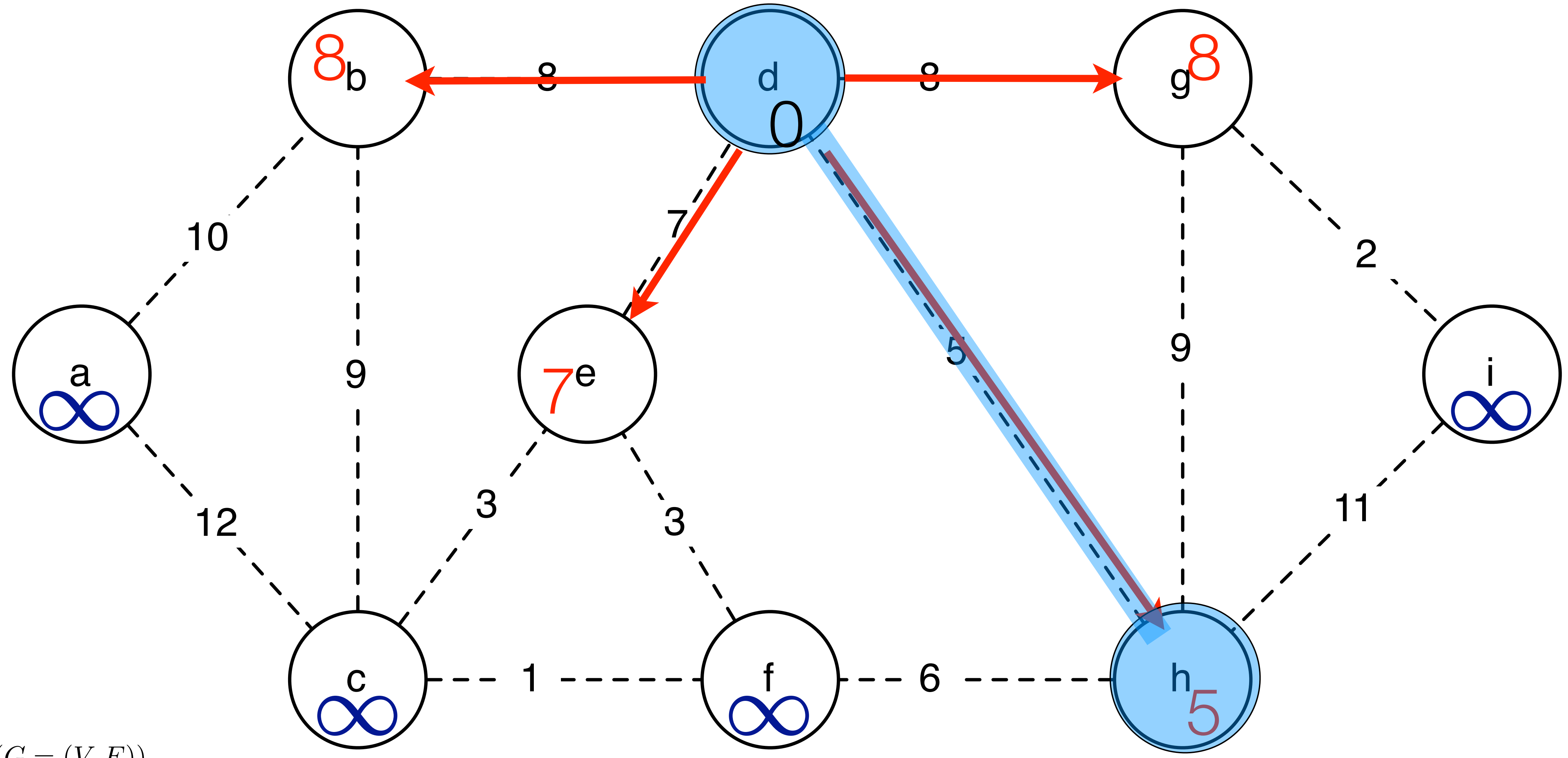

```
PRIM(G = (V, E))
Q ← ∅        ▷ Q is a Priority Queue
Initialize each v ∈ V with key k_v ← ∞, π_v ← NIL
Pick a starting node r and set k_r ← 0
Insert all nodes into Q with key k_v.
while Q ≠ ∅
    do u ← EXTRACT-MIN(Q)
       for each v ∈ Adj(u)
           do if v ∈ Q and w(u, v) < k_v
                 then π_v ← u
                      DECREASE-KEY(Q, v, w(u, v))    ▷ Sets k_v ← w(u, v)
```

prim



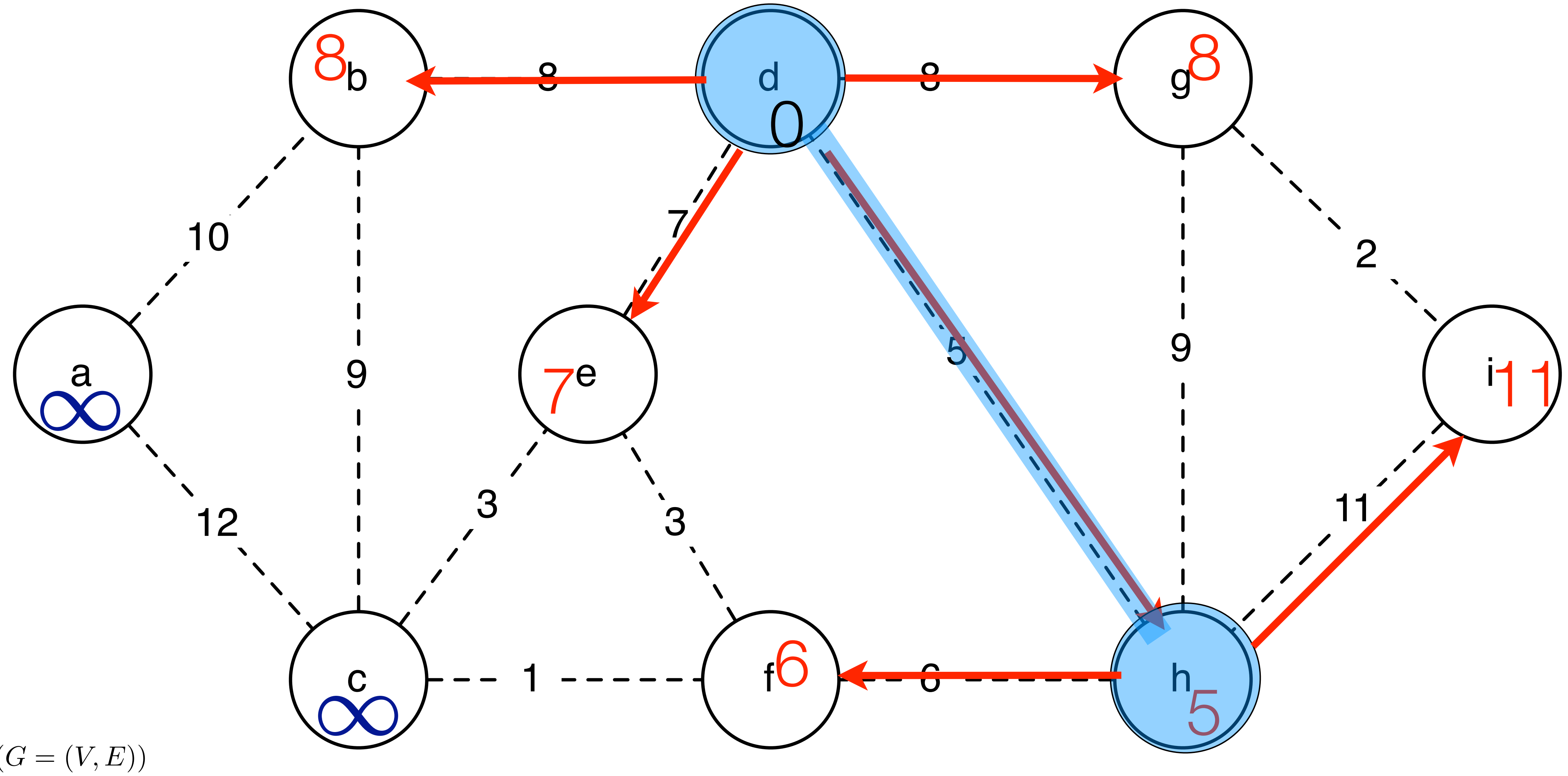

```
PRIM(G = (V, E))
Q ← ∅    ▷ Q is a Priority Queue
Initialize each v ∈ V with key k_v ← ∞, π_v ← NIL
Pick a starting node r and set k_r ← 0
Insert all nodes into Q with key k_v.
while Q ≠ ∅
      do u ← EXTRACT-MIN(Q)
         for each v ∈ Adj(u)
             do if v ∈ Q and w(u, v) < k_v
                  then π_v ← u
                       DECREASE-KEY(Q, v, w(u, v))    ▷ Sets k_v ← w(u, v)
```

prim




```
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)
```

prim

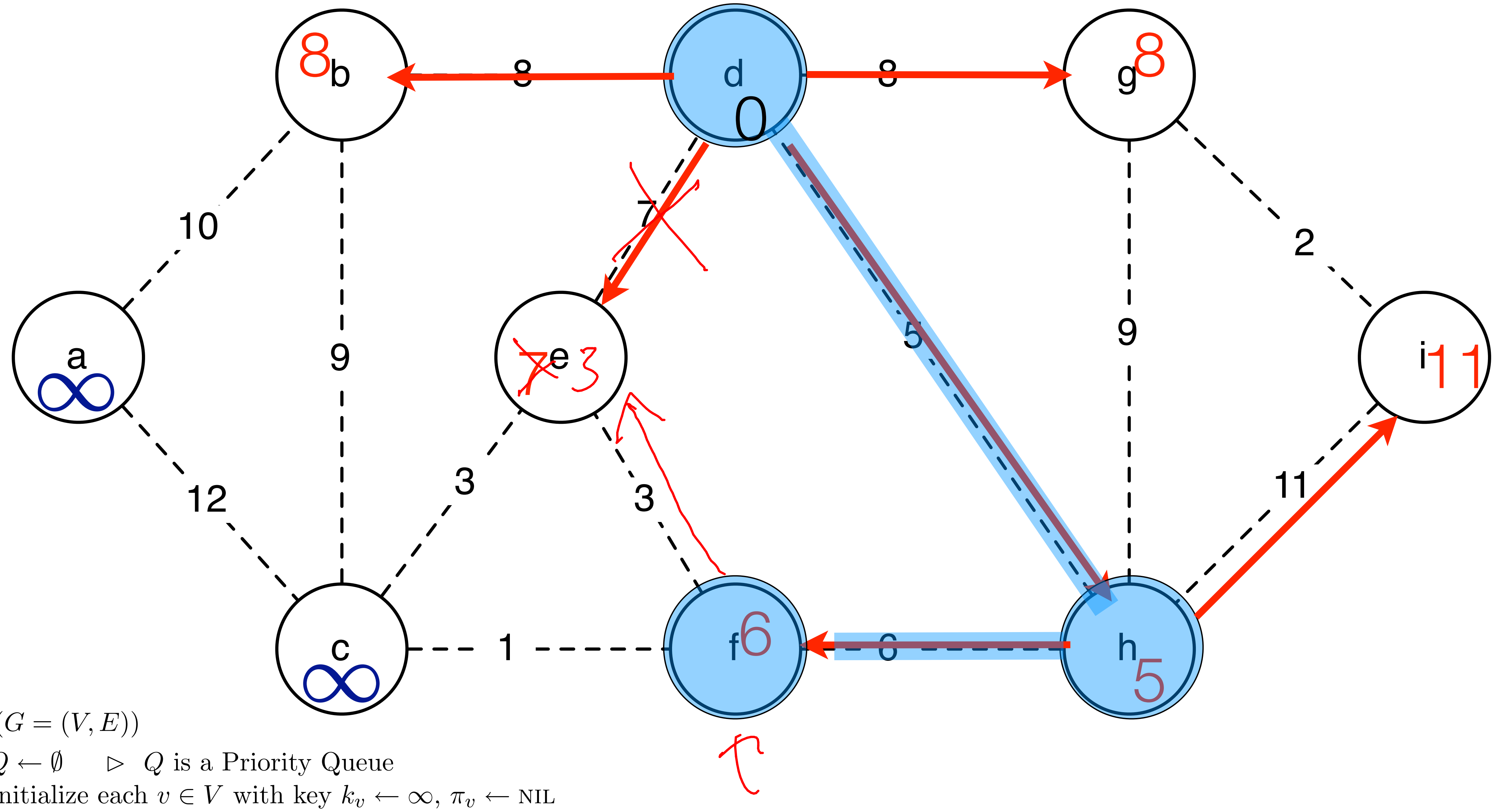


PRIM($G = (V, E)$)

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

π_v: "parent of node v in the MST"

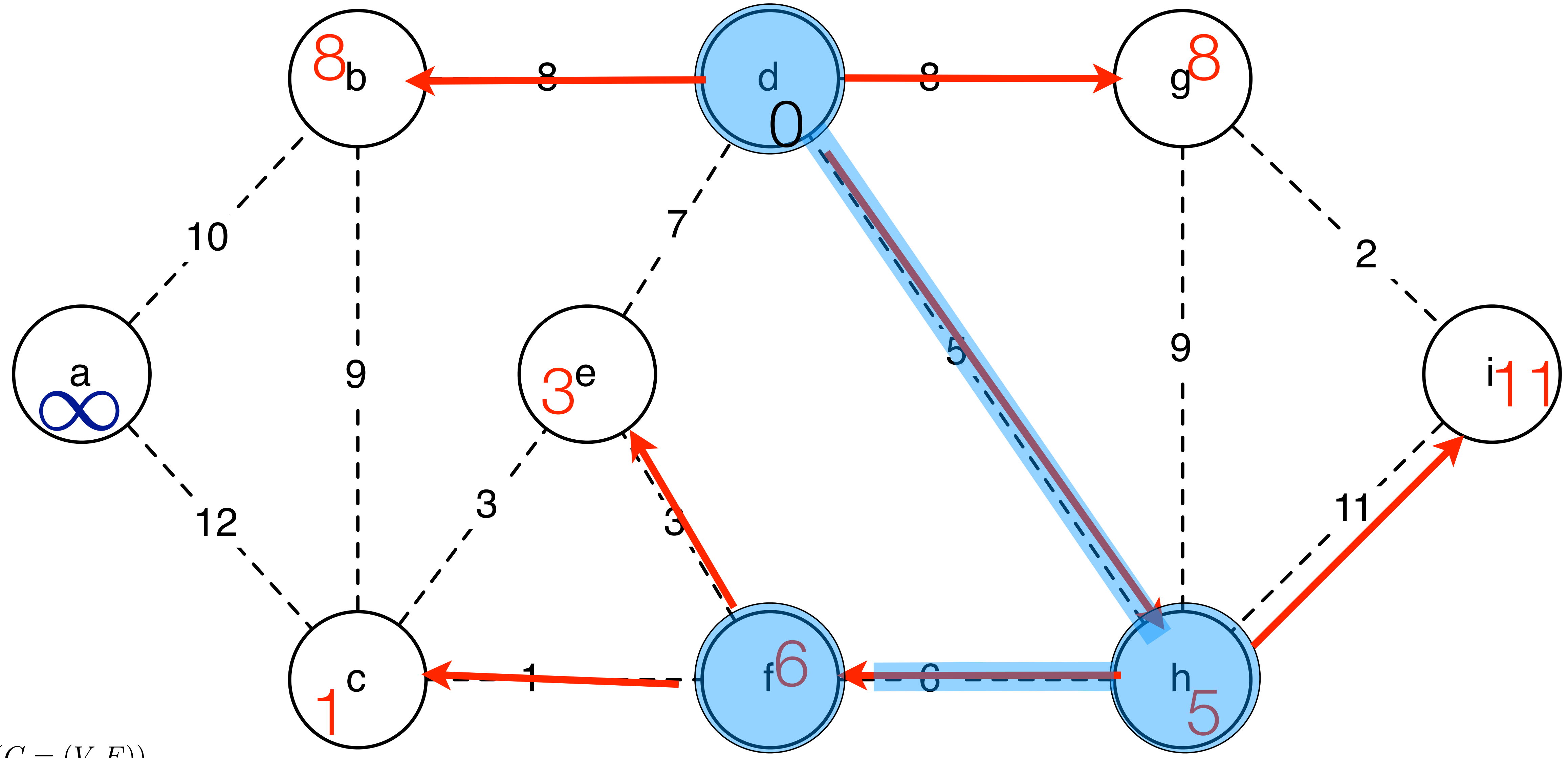
prim



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

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

prim




```
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)
```

running time

```
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)
```

Lines 1–4 → $\Theta(V)$

EXTRACT-MIN → $\Theta(V \cdot \log V)$ — repeated V times; each op takes $\Theta(\log V)$

if test: $\Theta(1)$

DECREASE-KEY: $\Theta(\log V)$ → $\Theta(E \cdot \log V)$

The for loop runs at most $\Theta(E)$ times throughout the execution of the algorithm

Overall: RUN TIME $\Theta(E \log V + V \log V) = \Theta(E \log V)$

implementation

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

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

$$O(V \log V + E \log V) = O(E \log V)$$

implementation

use a priority queue to keep track of light edges

	priority queue	fibonacci heap	
insert:	O(log n)	log n	
makequeue:	n	n	
extractmin:	O(log n)	log n	amortized
decreasekey:	O(log n)	O(1)	amortized

faster implementation

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)
```

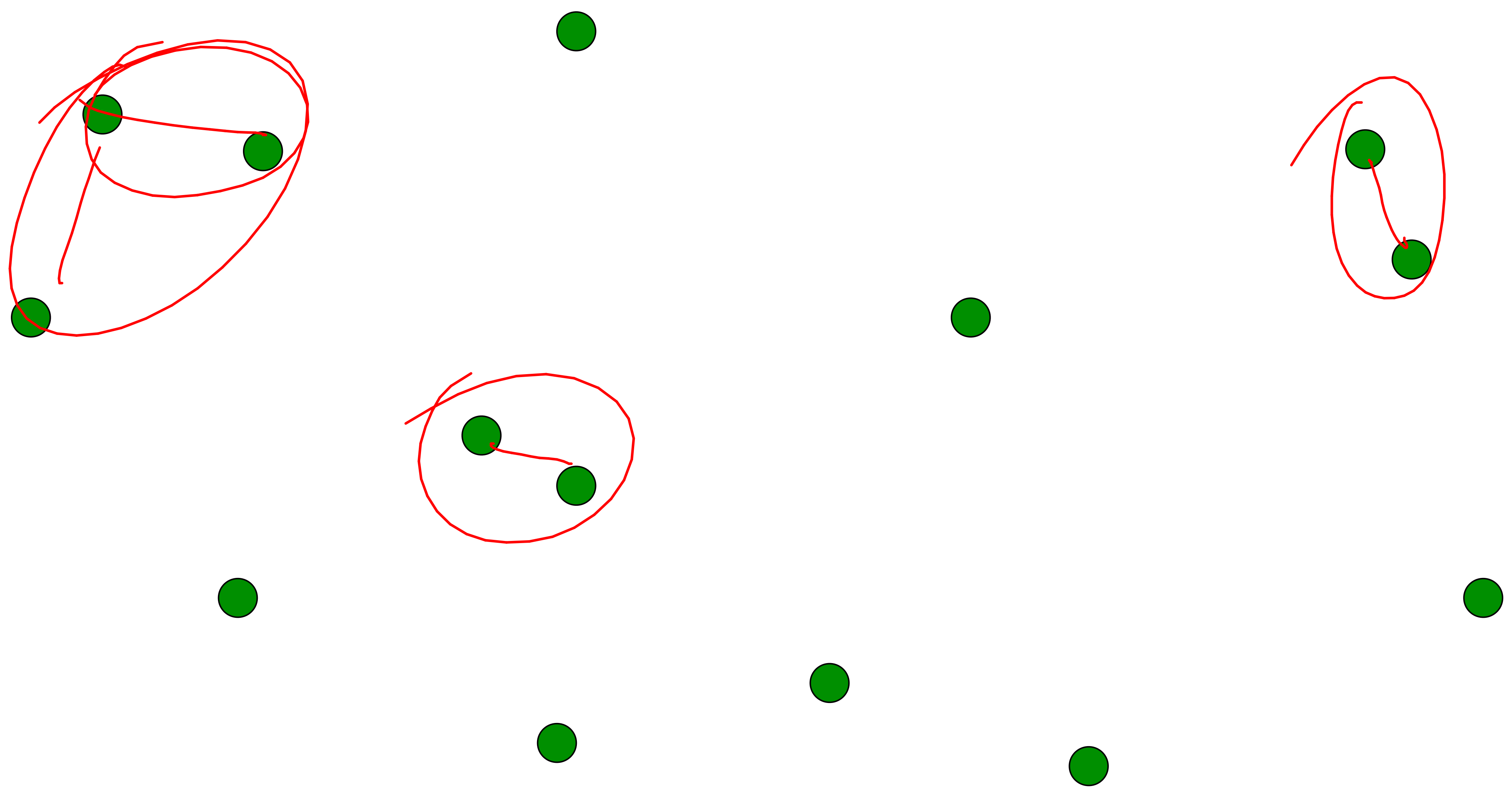
$O(1)$

$$O(E + V \log V)$$

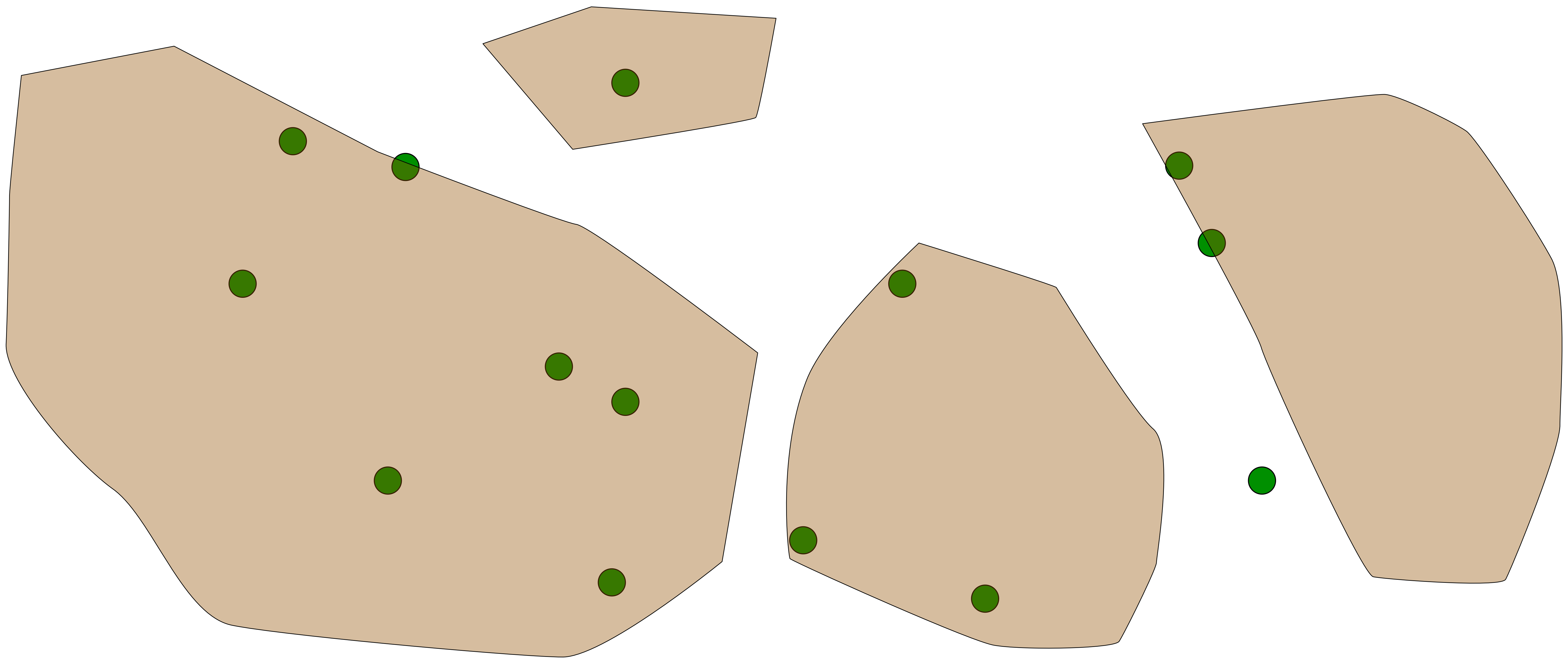
research in mst

fredman-tarjan 84:	$E + V \log V$
gabow-galil-spencer-tarjan 86:	$E \log(\log^* V)$
chazelle 97	$E\alpha(V) \log \alpha(V)$
chazelle 00	$E\alpha(V)$
pettie-ramachandran 02:	(optimal)
karger-klein-tarjan 95: (randomized)	E
euclidean mst:	$V \log V$

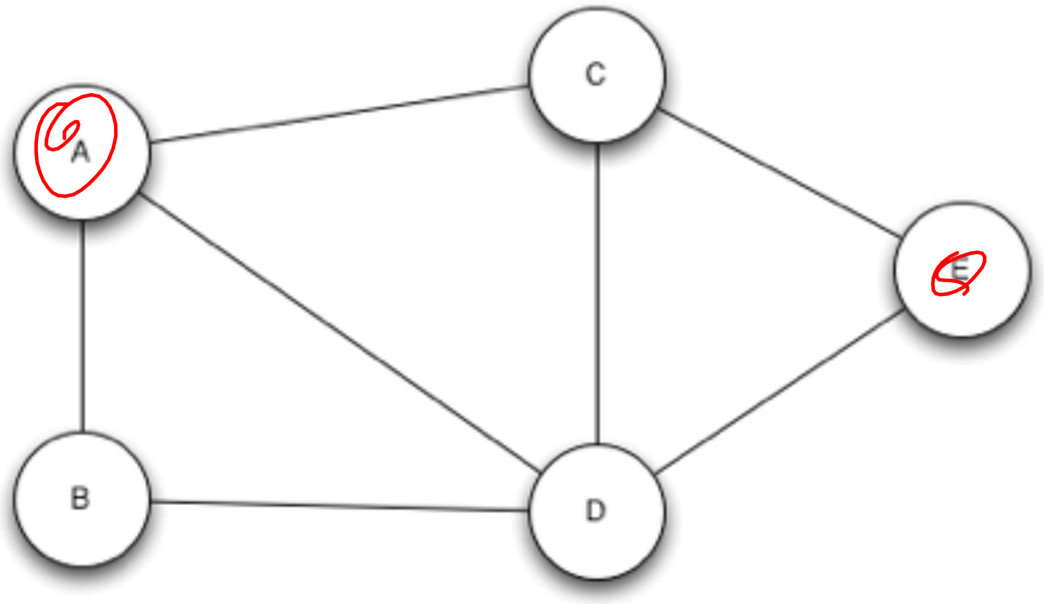
application of mst



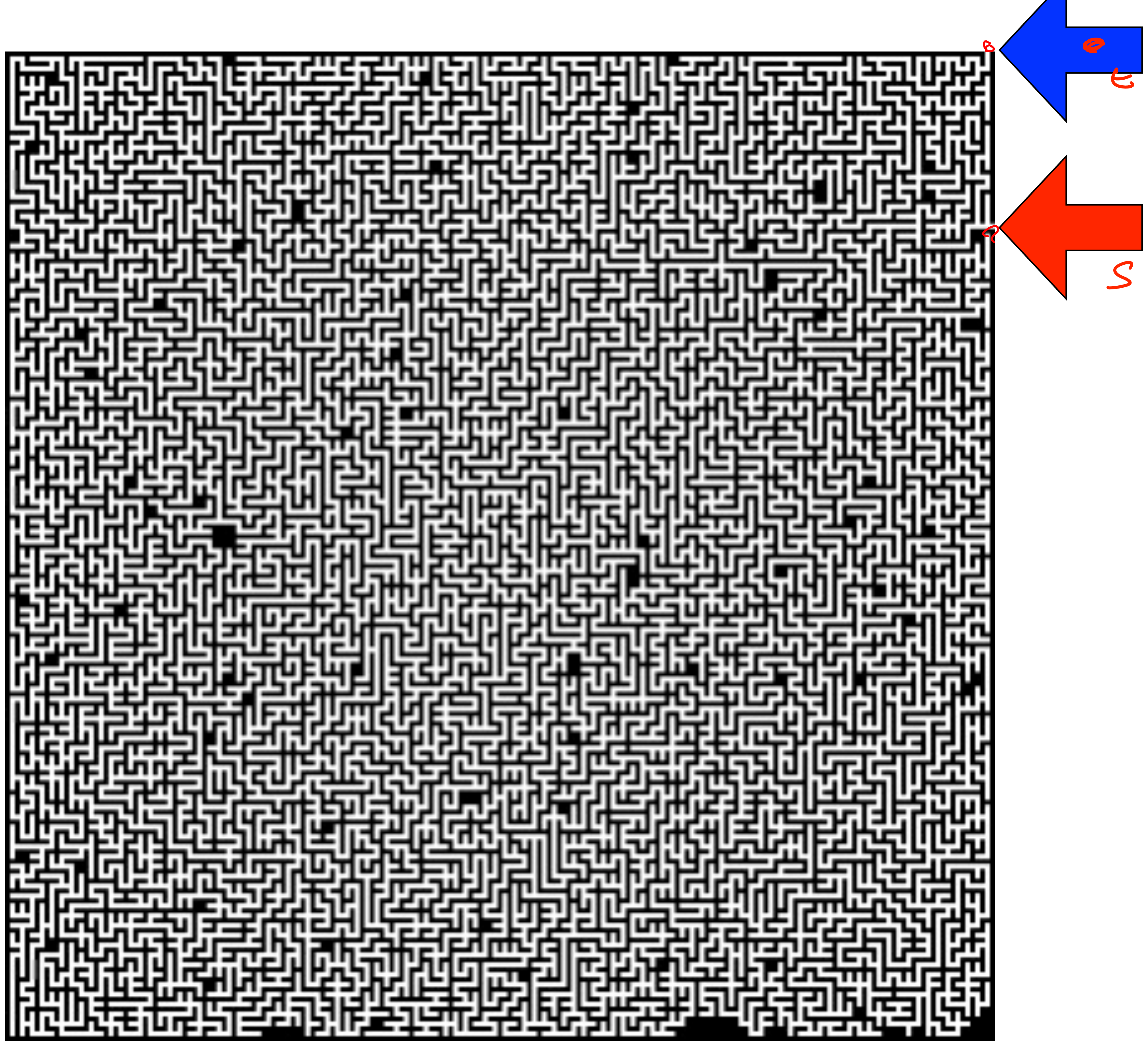
application of mst



SIMPLE QUESTIONS ON GRAPHS



WHAT IS THE LENGTH OF THE PATH FROM A TO E?


E
S

SHORTEST PATH PROPERTY

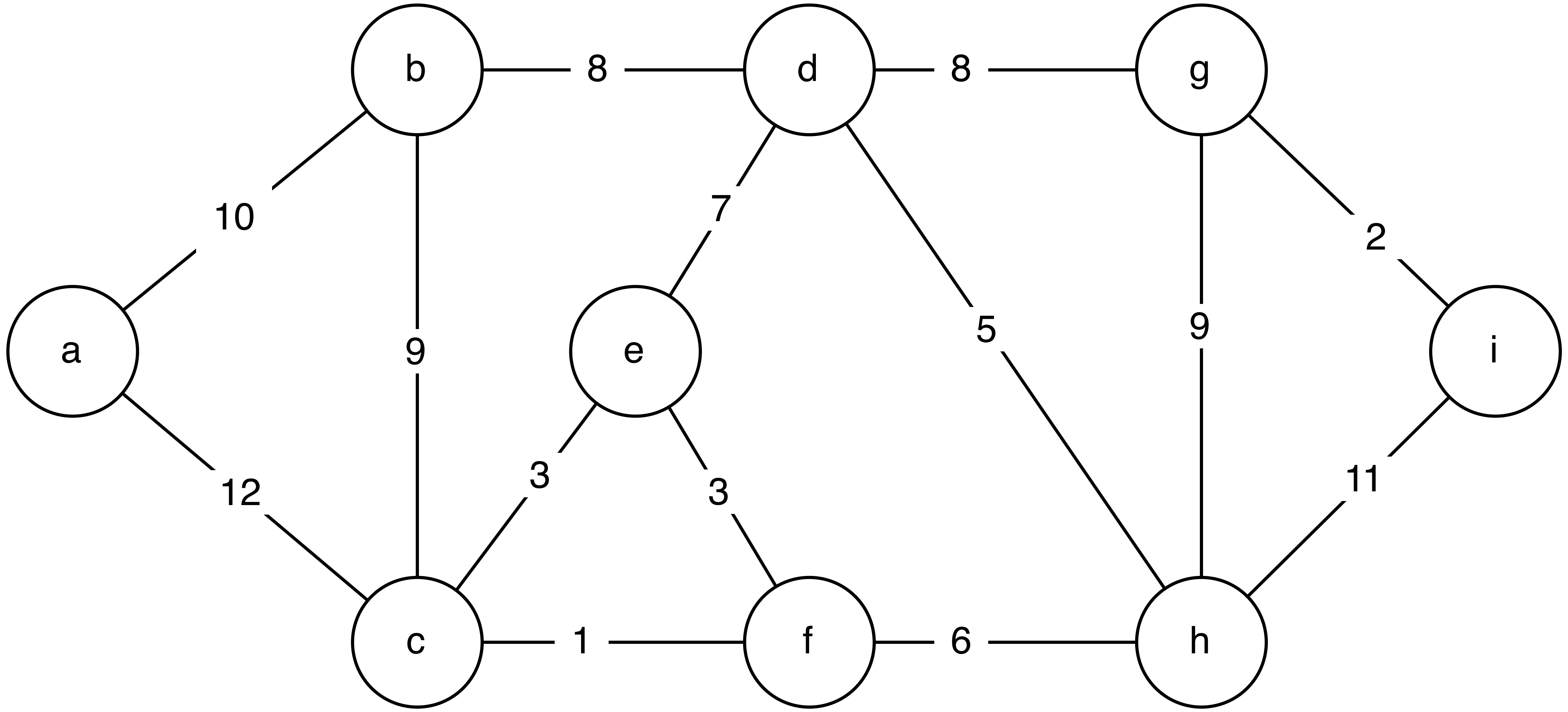
DEFINITION:

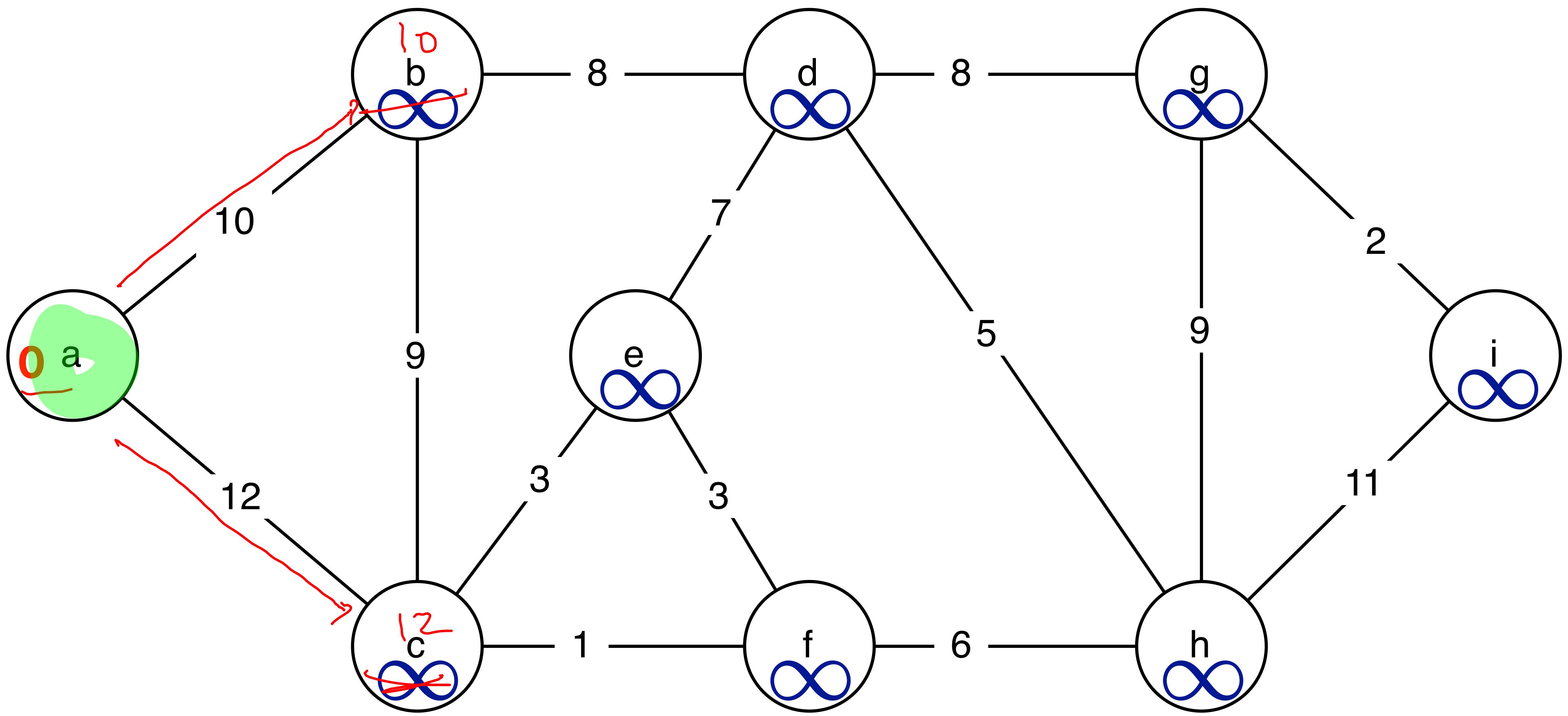
$\delta(s, v)$ – length of the shortest path from s to v in

$G = (E, V, w)$.

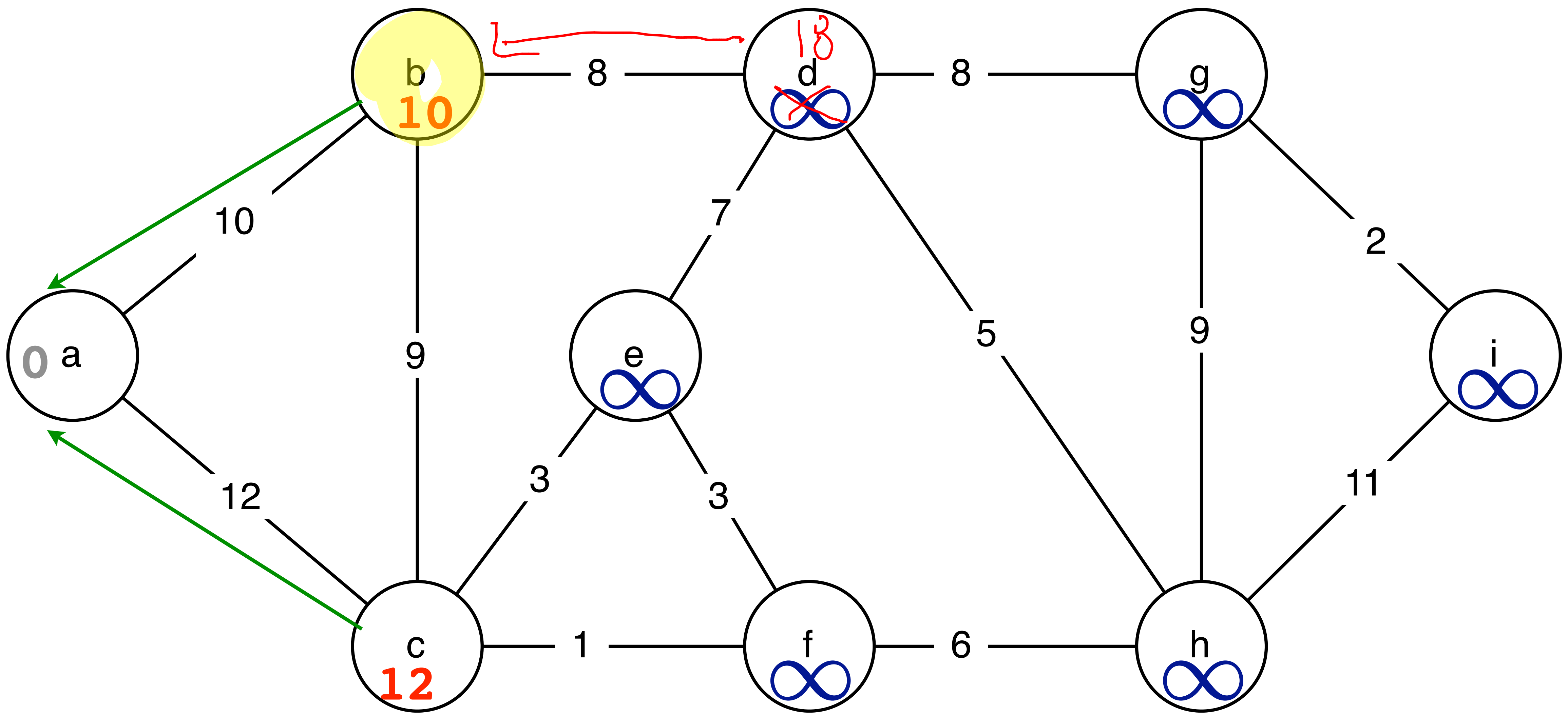
– all weights are positive, $w : E \rightarrow \mathbb{R}^+$

SHORTEST PATHS

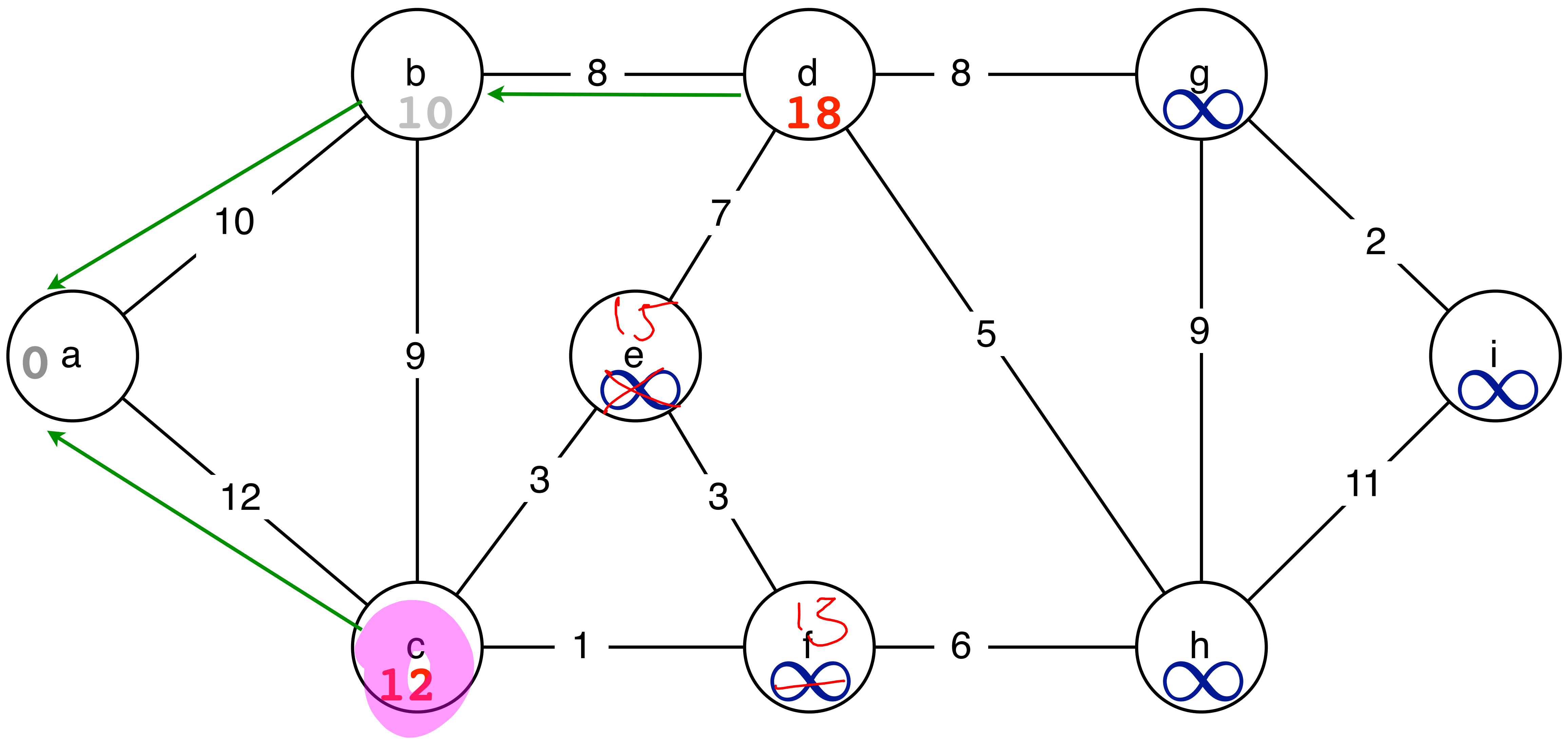


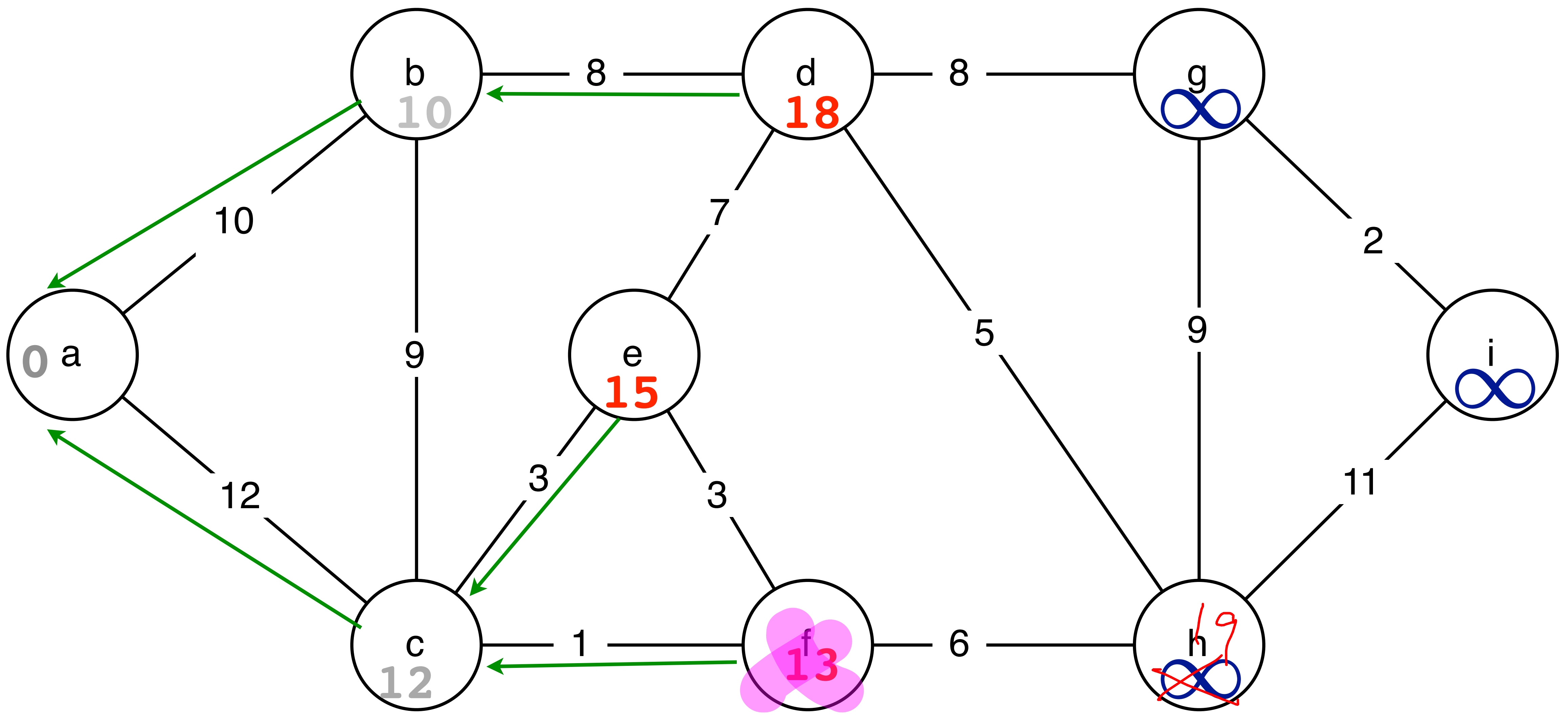


a
b
c
d
e
f
g
h
i
10
12
8
9
7
5
3
3
1
6
8
2
9
11
0
10
12

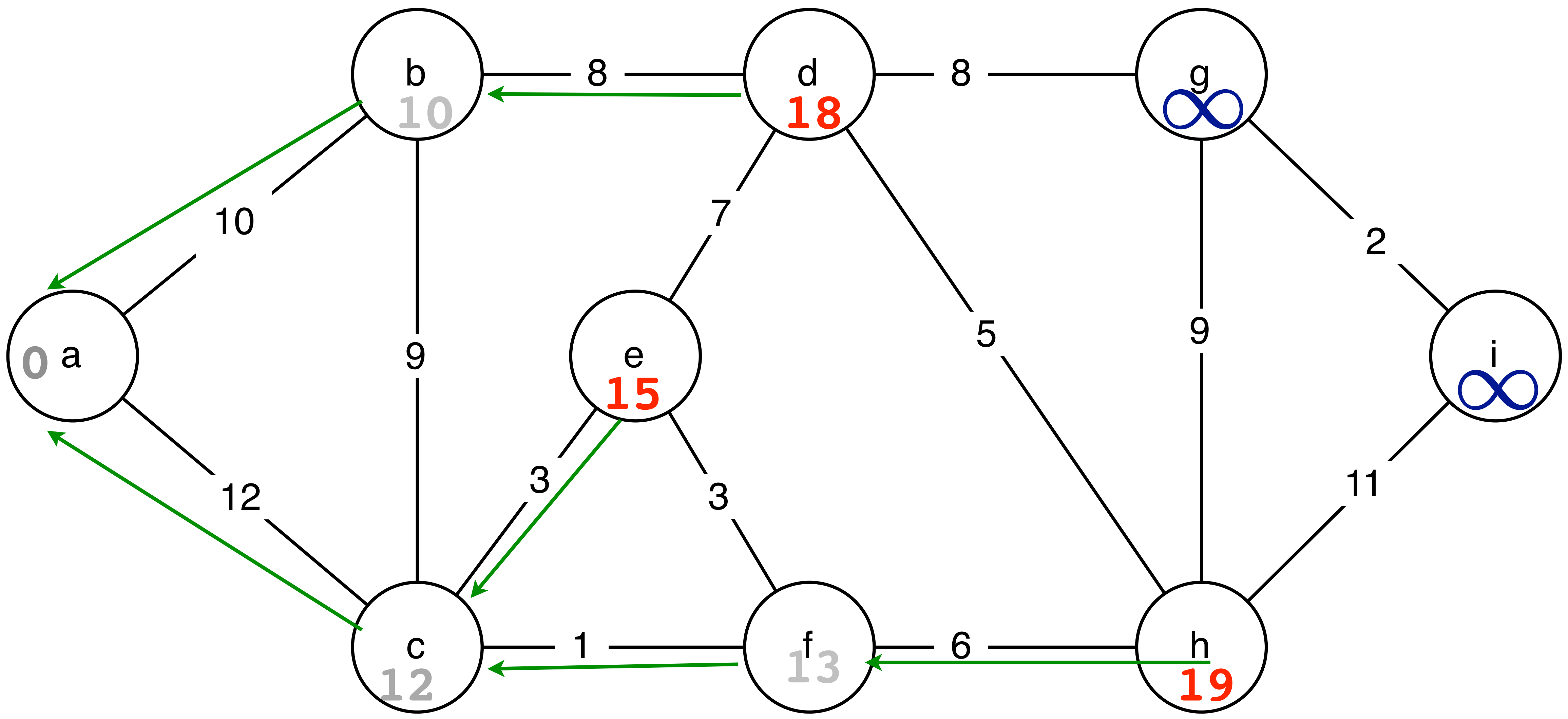


b
10
8
d
18
8
g
10
7
5
9
2
0
a
9
e
i
12
3
3
11
c
12
1
f
6
h

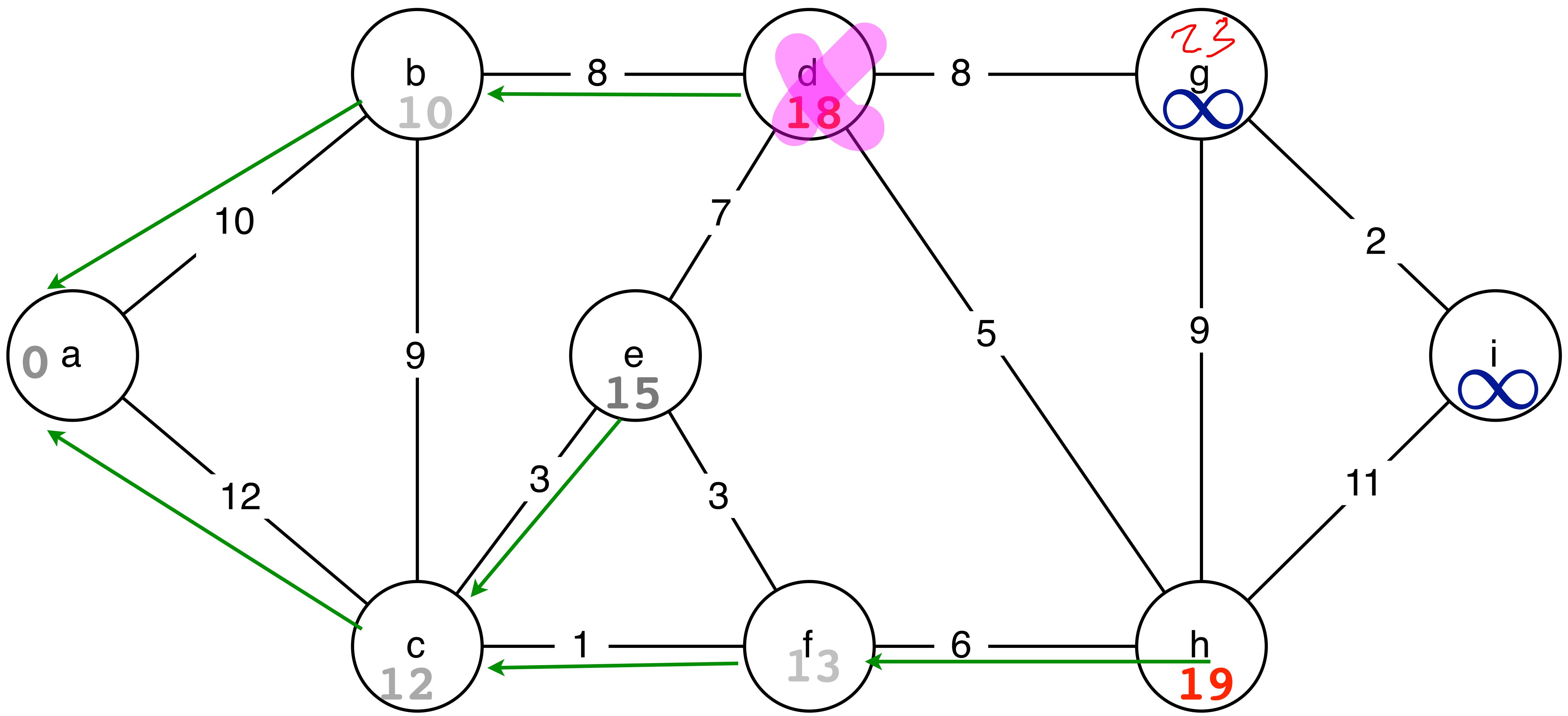

a
0
b
10
c
12
d
18
e
15
f
13
g
h
i
8
8
10
9
7
5
9
2
12
3
3
11
1
6



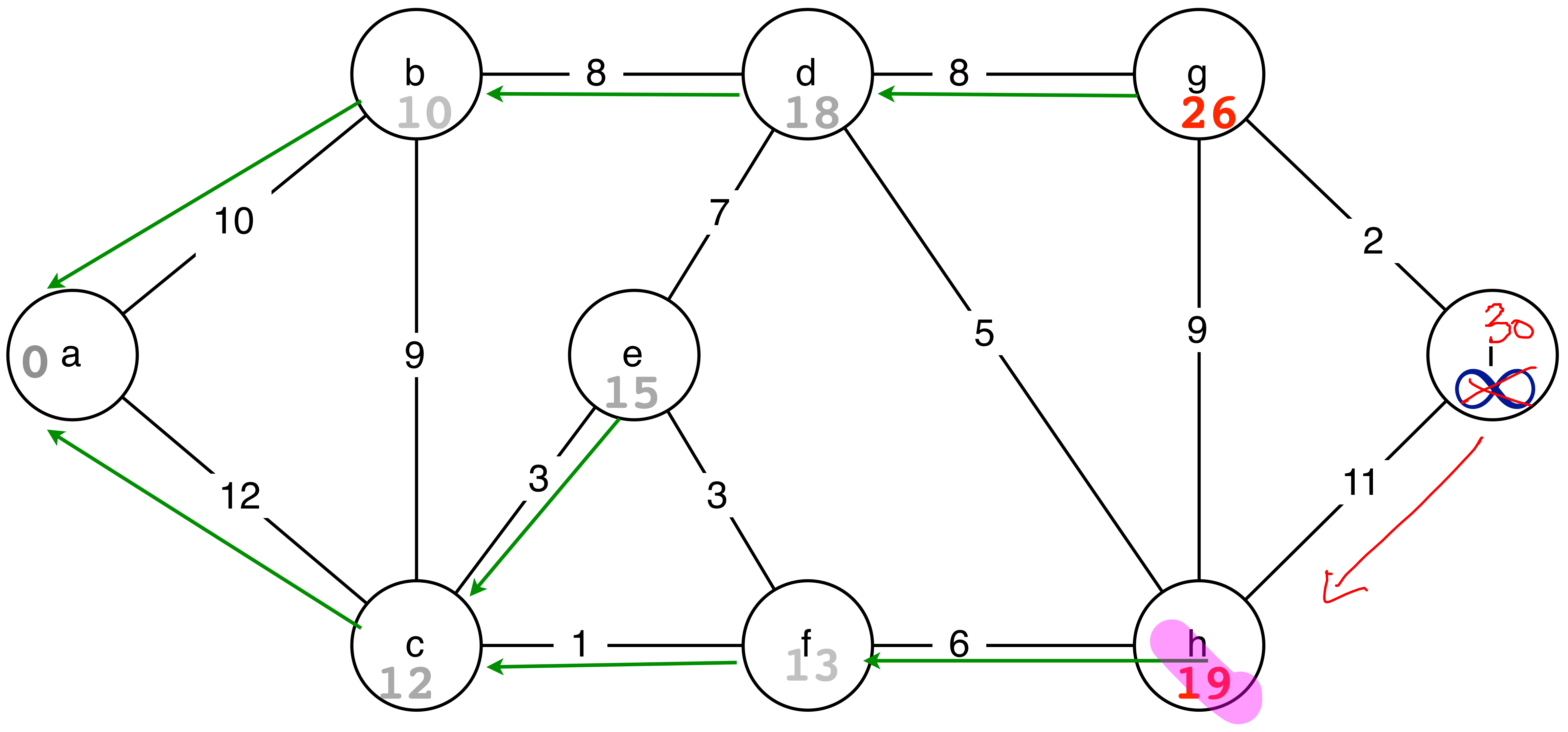
b
10
d
18
g
8
8
10
7
2
a
0
9
e
15
5
9
i
12
3
3
11
c
12
1
f
13
6
h
19



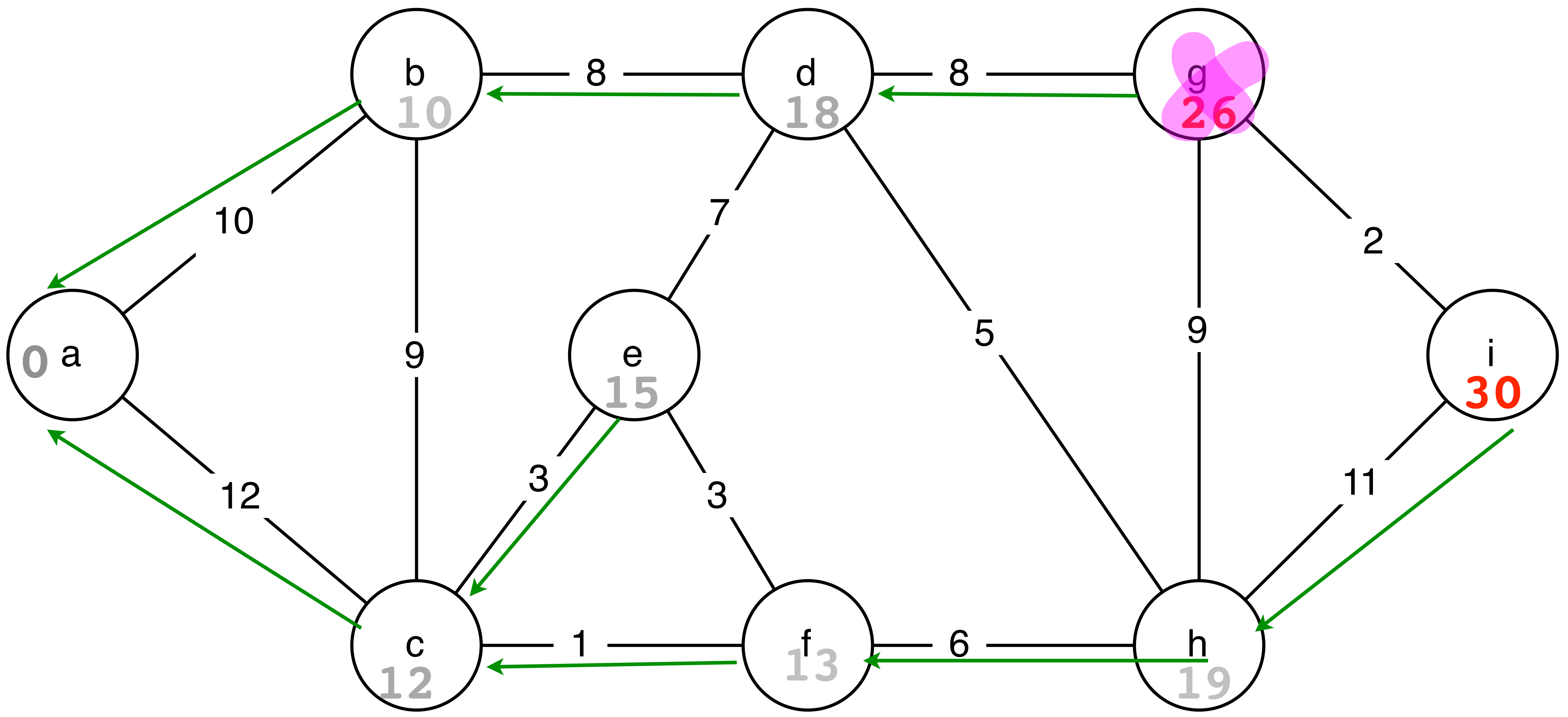
a
0
b
10
c
12
d
18
e
15
f
13
g
∞
h
19
i
∞
8
8
10
9
7
5
9
2
12
3
3
11
1
6



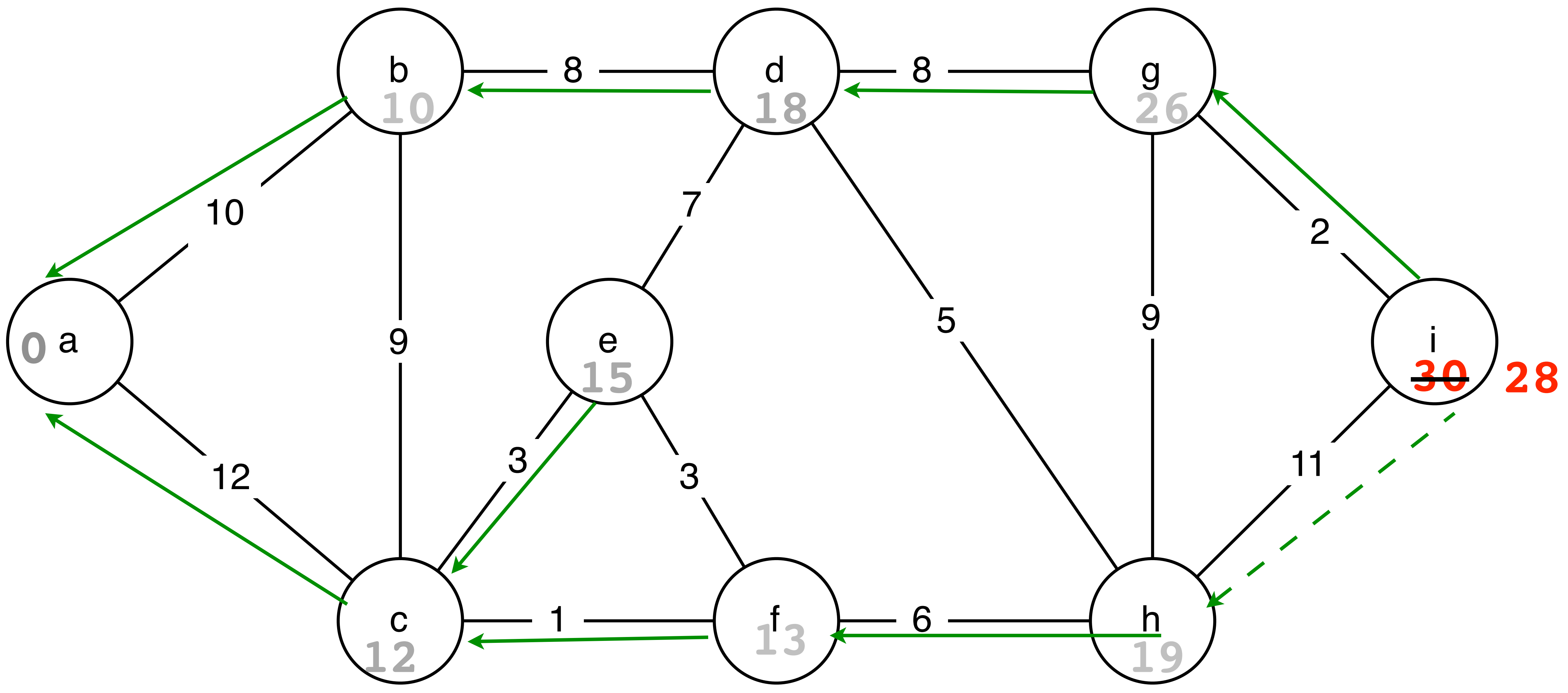
b
10
8
d
18
8
g
23
∞
10
7
2
0 a
9
e
15
5
9
i
∞
12
3
3
11
c
12
1
f
13
6
h
19



a
0
b
10
c
12
d
18
e
15
f
13
g
26
h
19
i
30
8
8
10
9
7
5
9
2
12
3
3
11
1
6



a
0
b
10
c
12
d
18
e
15
f
13
g
26
h
19
i
30
8
8
10
7
9
5
9
2
12
3
3
11
1
6



$\forall v \in V$ d_v at each node is the value $\delta(a, v)$ and the green arrows represent the shortest path from a to v.

ALGORITHM

```
INIT: d_v ← ∞   π_v ← nil
d_s ← 0

while Q not empty
    u ← extractmin(Q)
    for each neighbor v ∈ Adj(u)
        if d_v > d_u + w(u,v) then
            DecreaseKey(v, d_u + w(u,v))
            π_v ← u
```

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

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

```
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(
```

RUNNING TIME

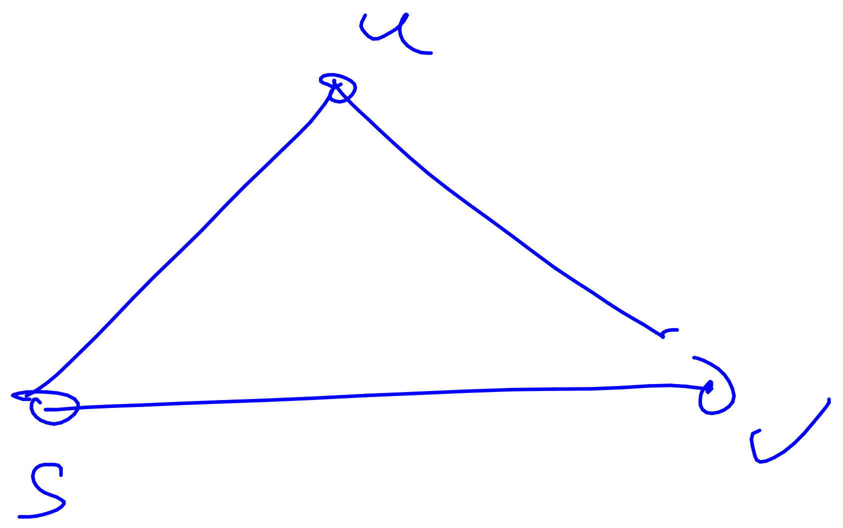
DIJKSTRA$(G = (V, E), s)$

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

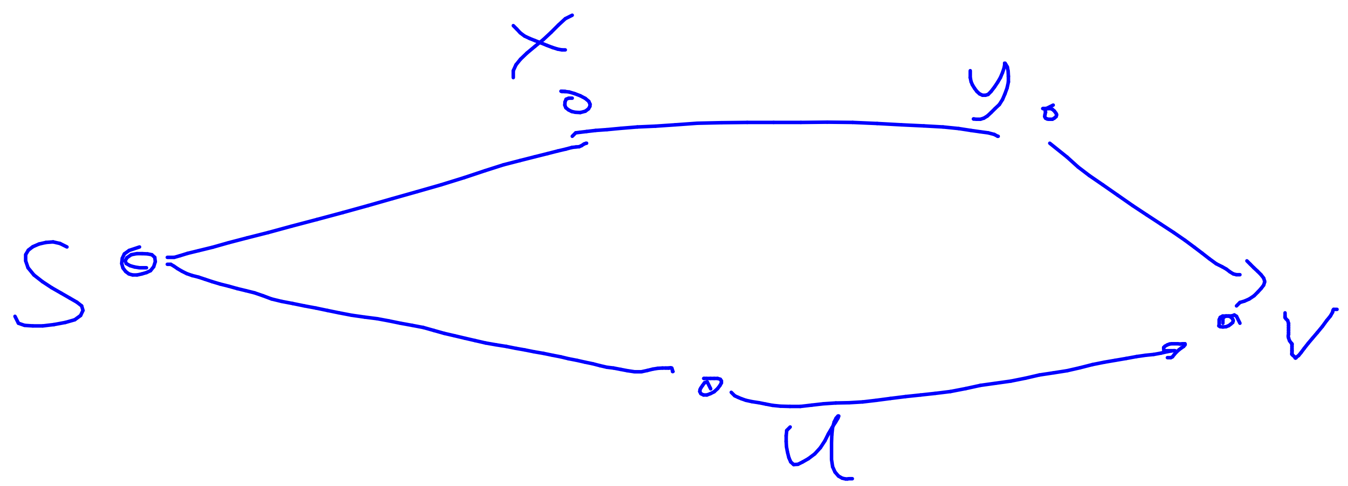
$\Theta(E \log V)$

WHY DOES DIJKSTRA WORK?

TRIANGLE INEQUALITY: $\forall (u, v) \in E,\ \delta(s, v) \leq \delta(s, u) + w(u, v)$



UPPER BOUND: $d_v \geq \delta(s, v)$



BREADTH FIRST SEARCH

INPUT: $G = (V, E), s$

OUTPUT:

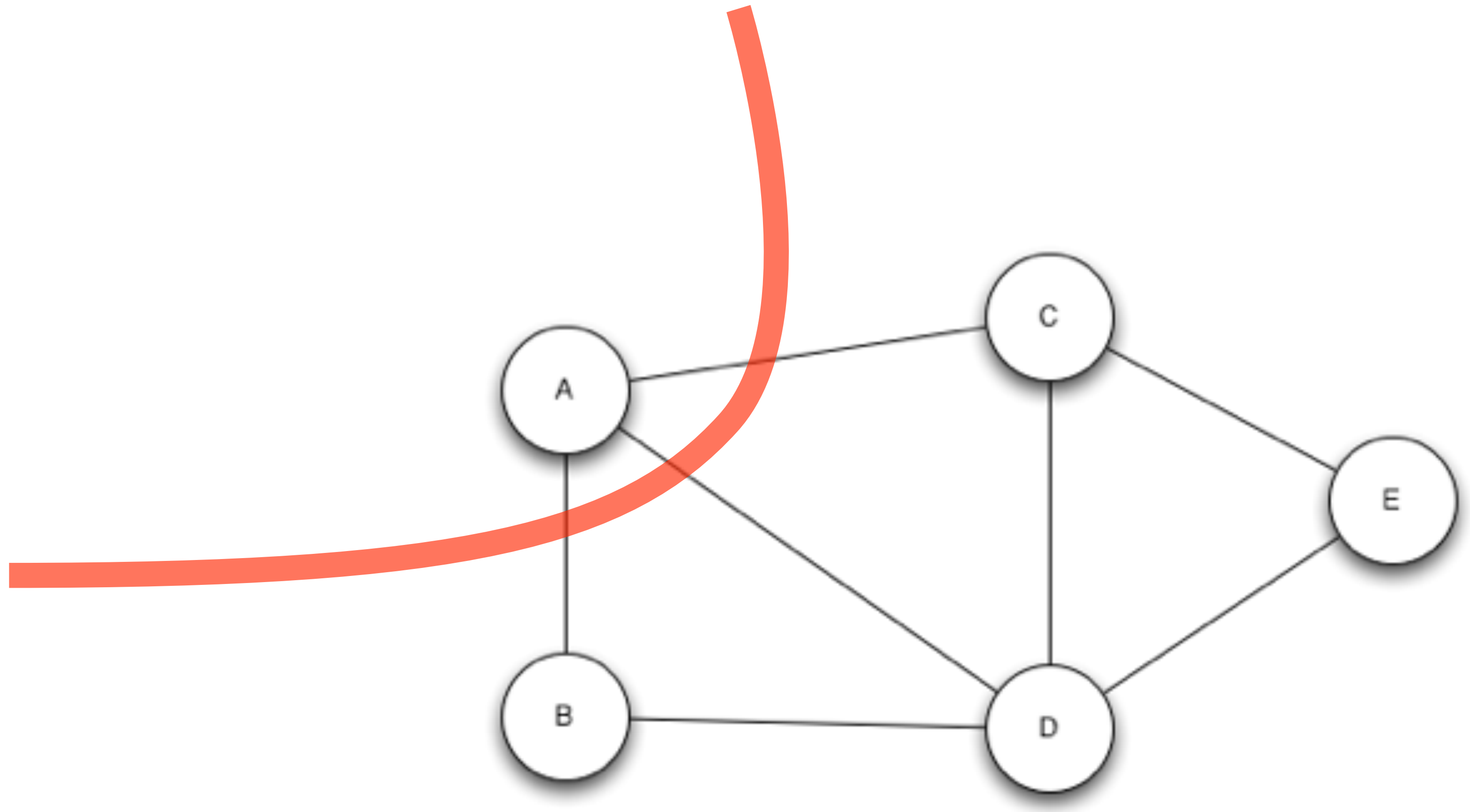
BREADTH FIRST SEARCH

INPUT: $G = (V, E), s$

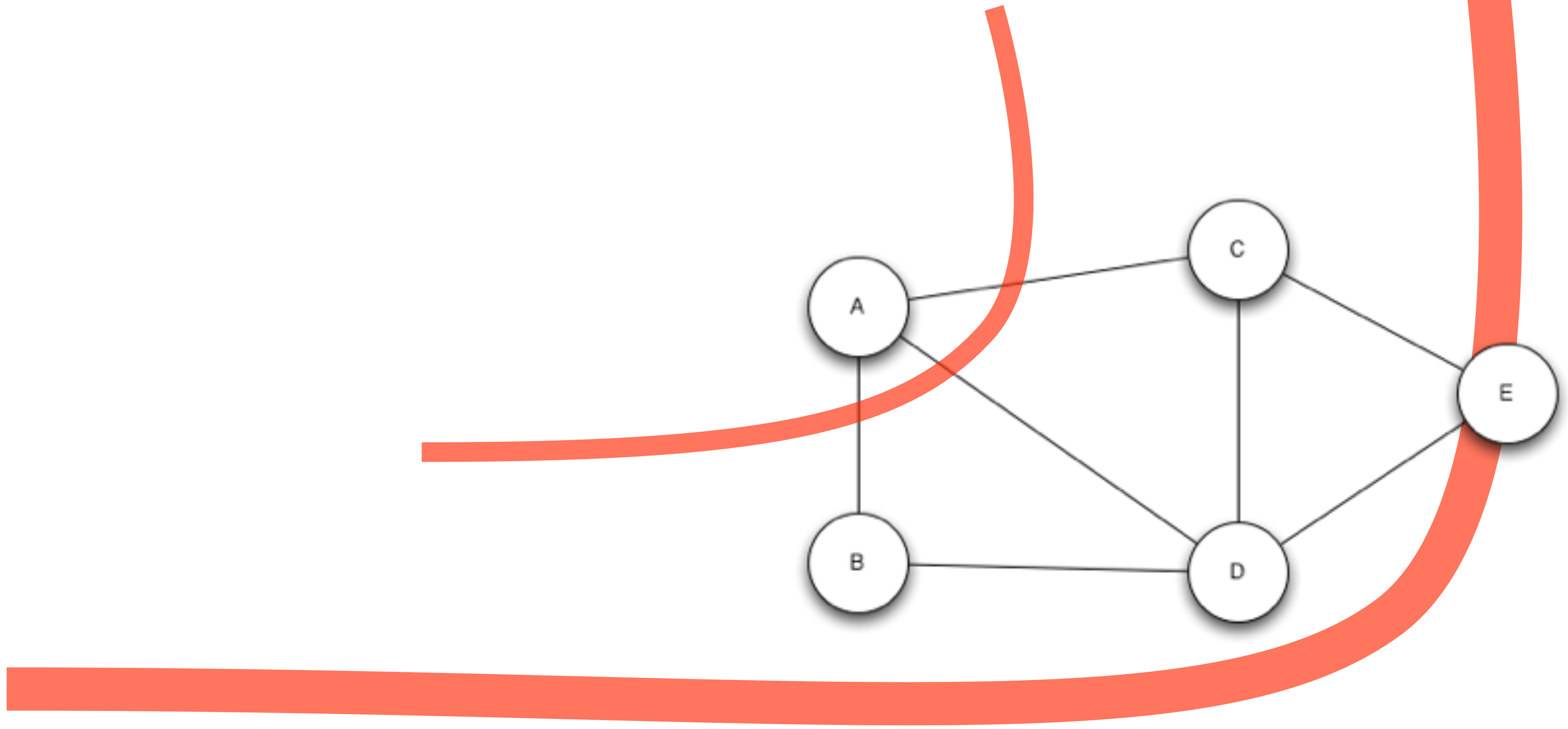
OUTPUT: $\forall v \in V \quad d_v = \delta(s, v)$

SMALLEST # OF EDGES FROM S TO V

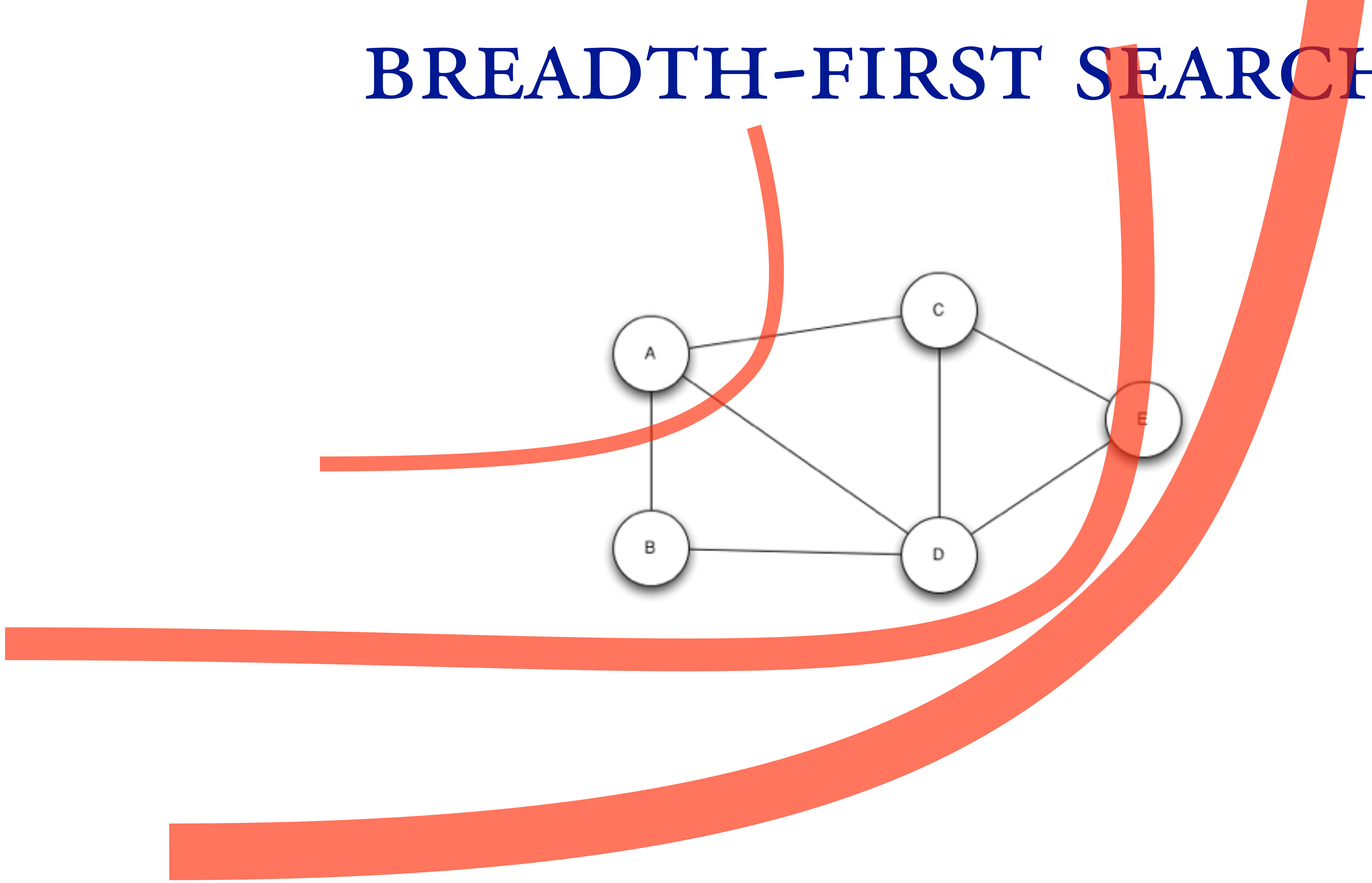
BREADTH-FIRST SEARCH



BREADTH-FIRST SEARCH



BREADTH-FIRST SEARCH

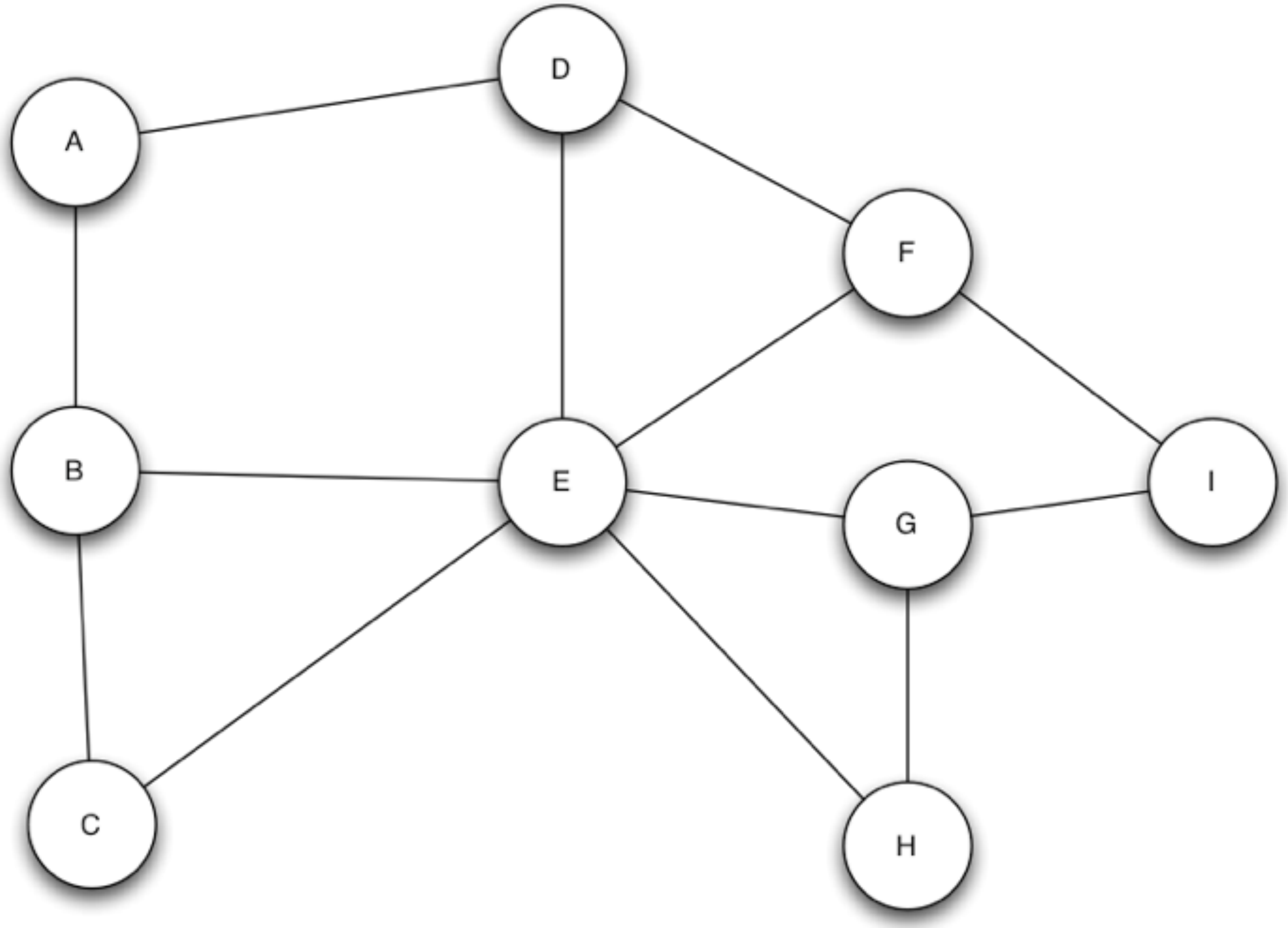


BREADTH FIRST SEARCH

INPUT: $G = (V, E), s$

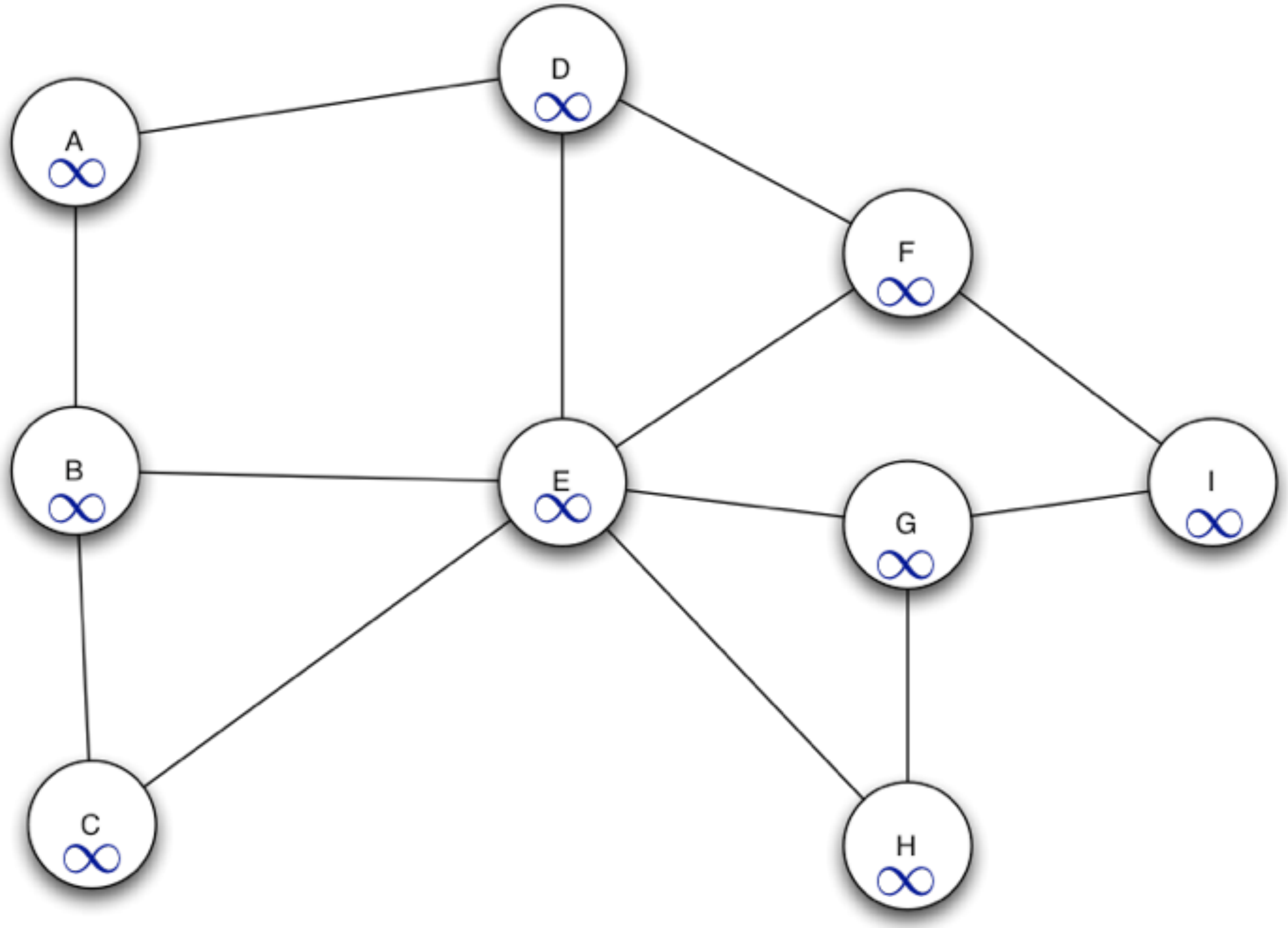
OUTPUT: SMALLEST # OF EDGES FROM S TO V d_v $\forall v \in V$

BFS(G, A)



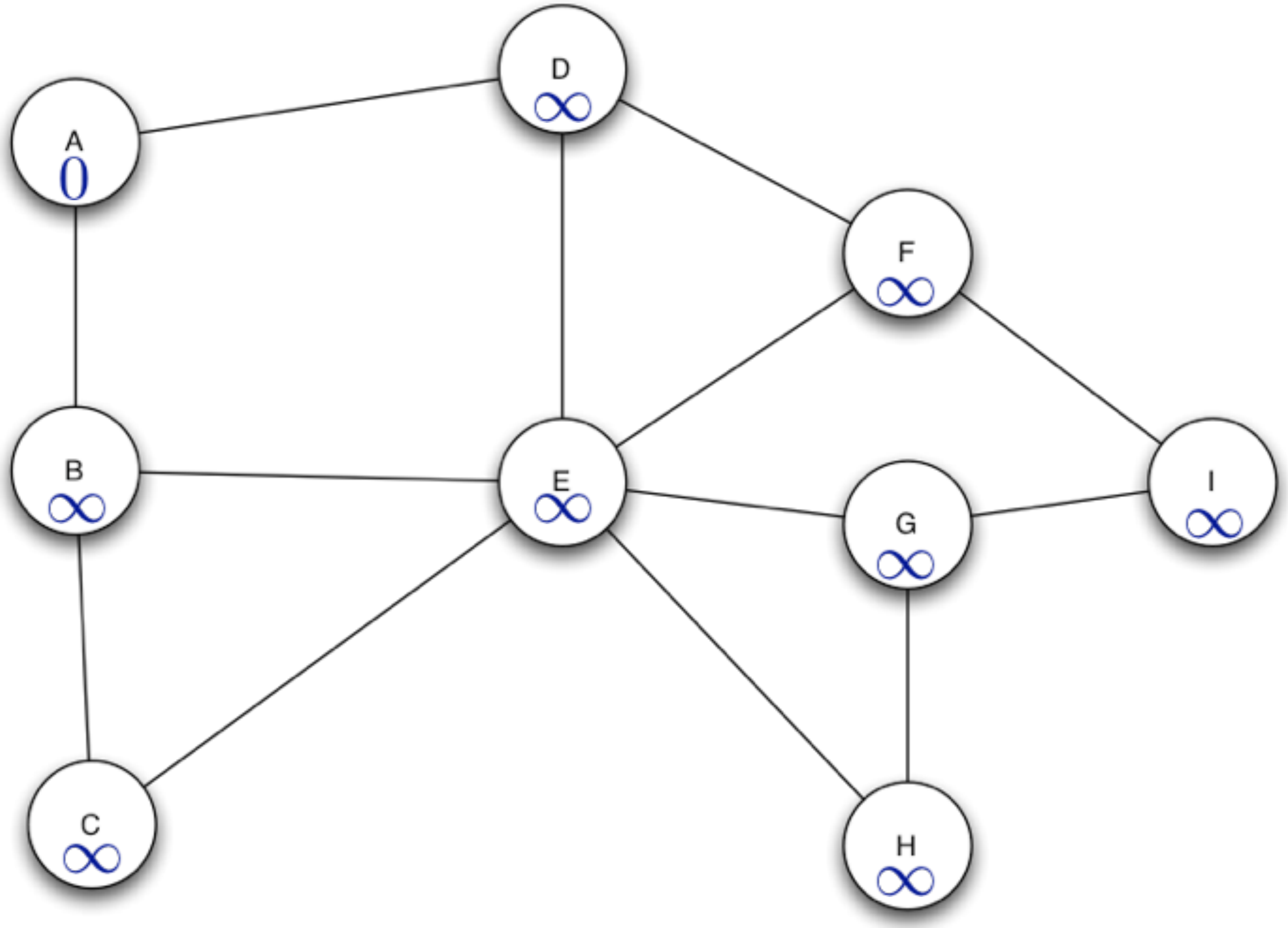
$\mathcal{Q}$

BFS(G, A)



Q

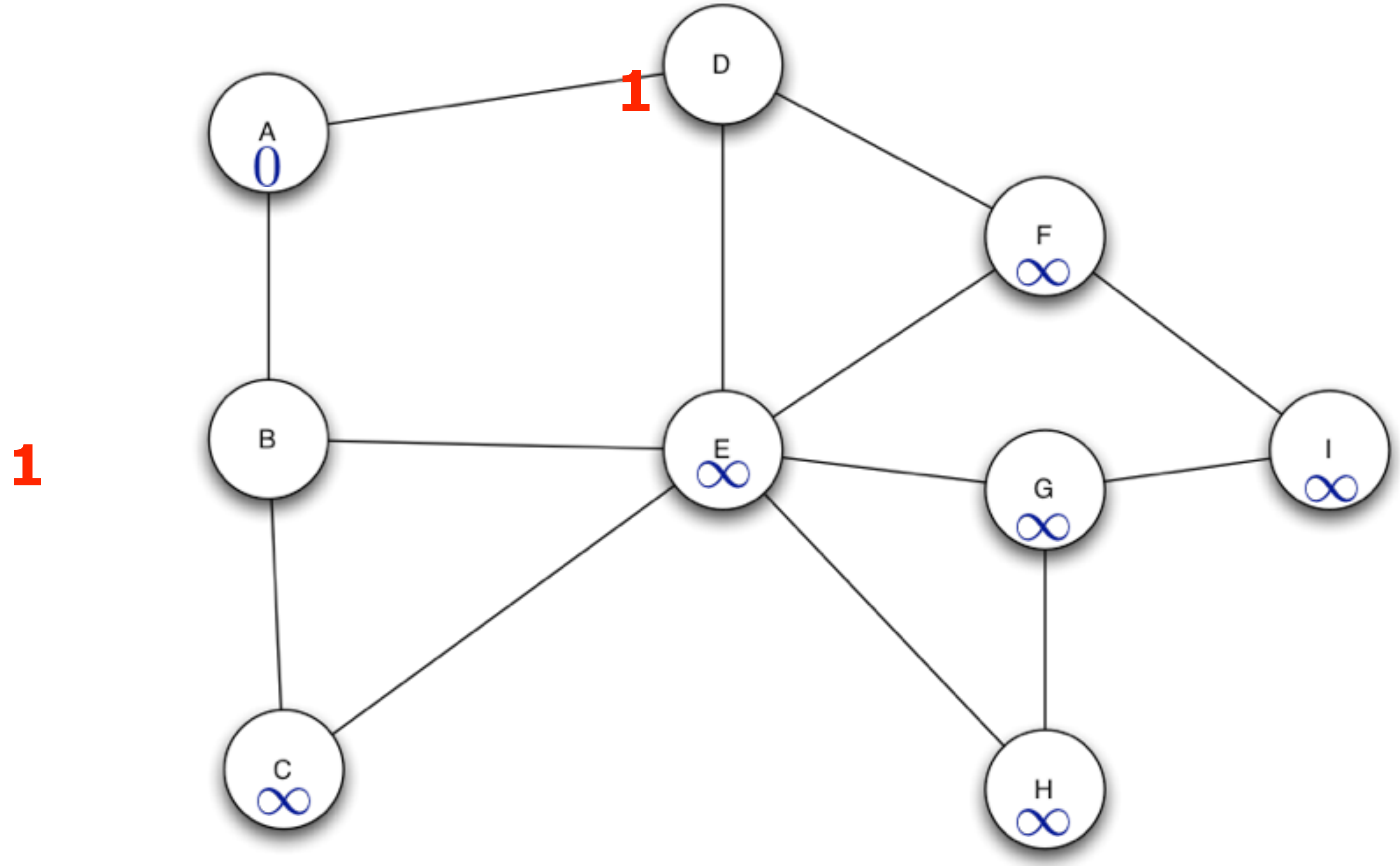
BFS(G, A)



Q

A

BFS(G, A)



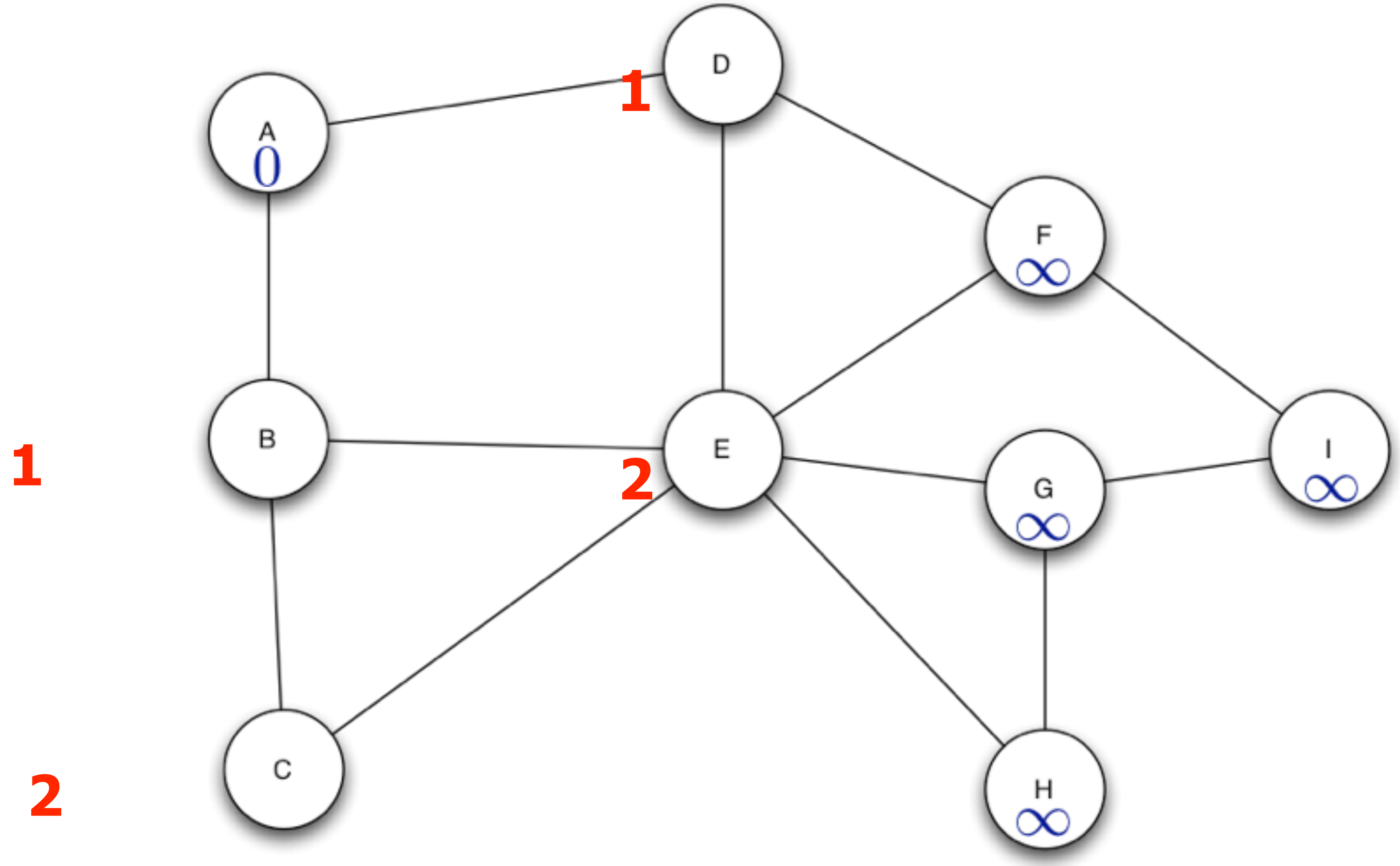
Q

A

B

D

BFS(G, A)



Q

A

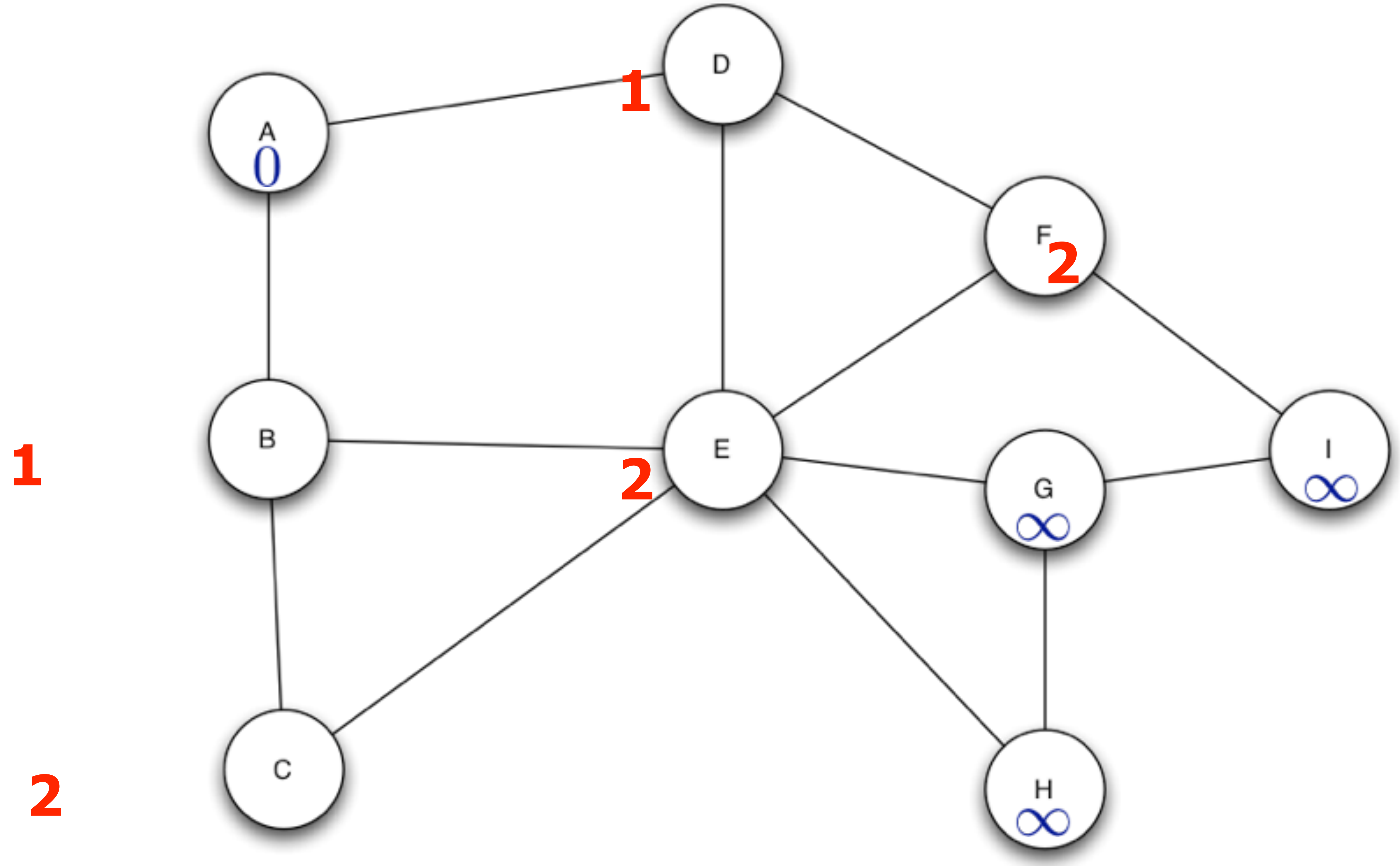
B

D

C

E

BFS(G, A)



Q

A

B

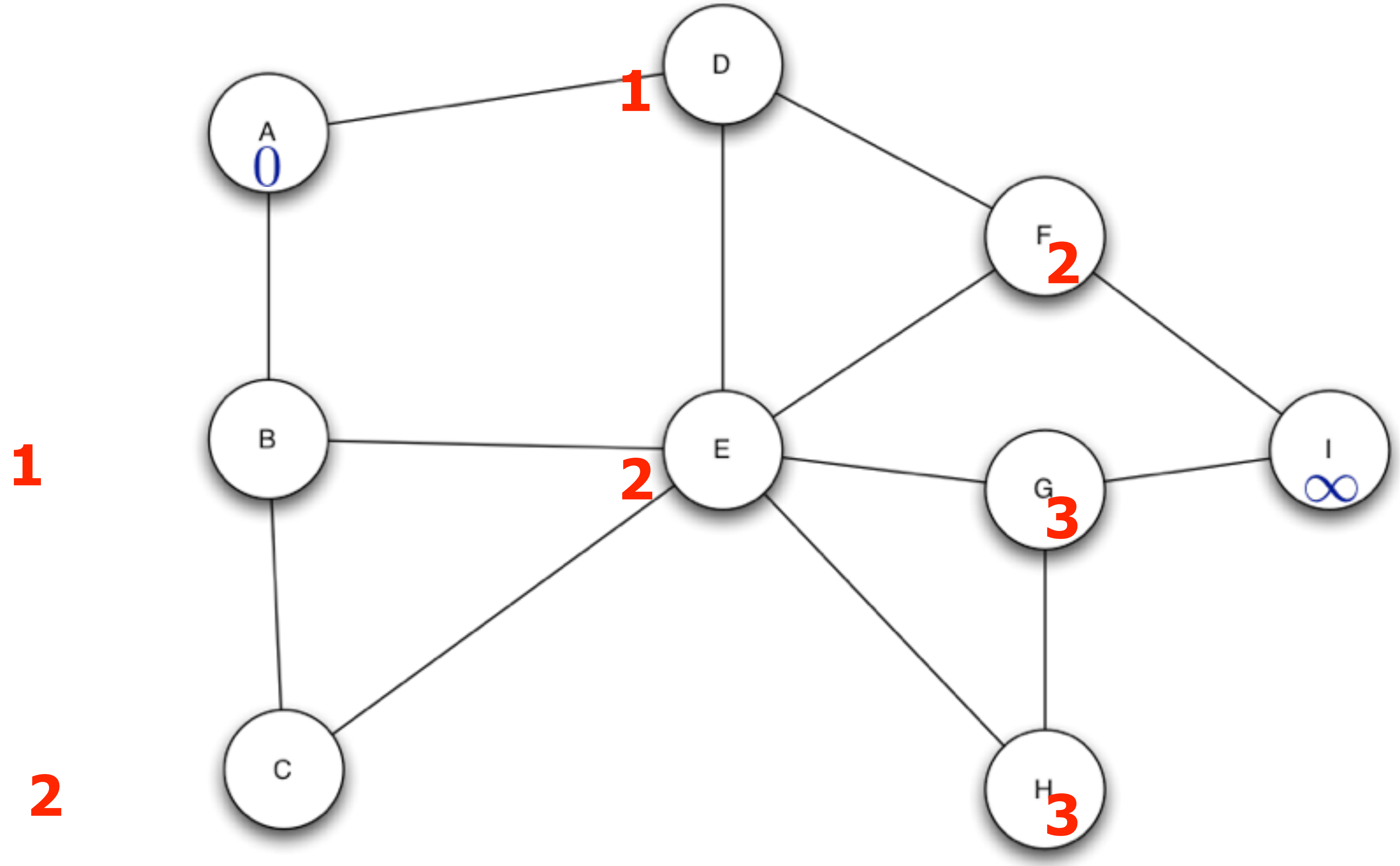
D

C

E

F

BFS(G, A)



Q

A

B

D

C

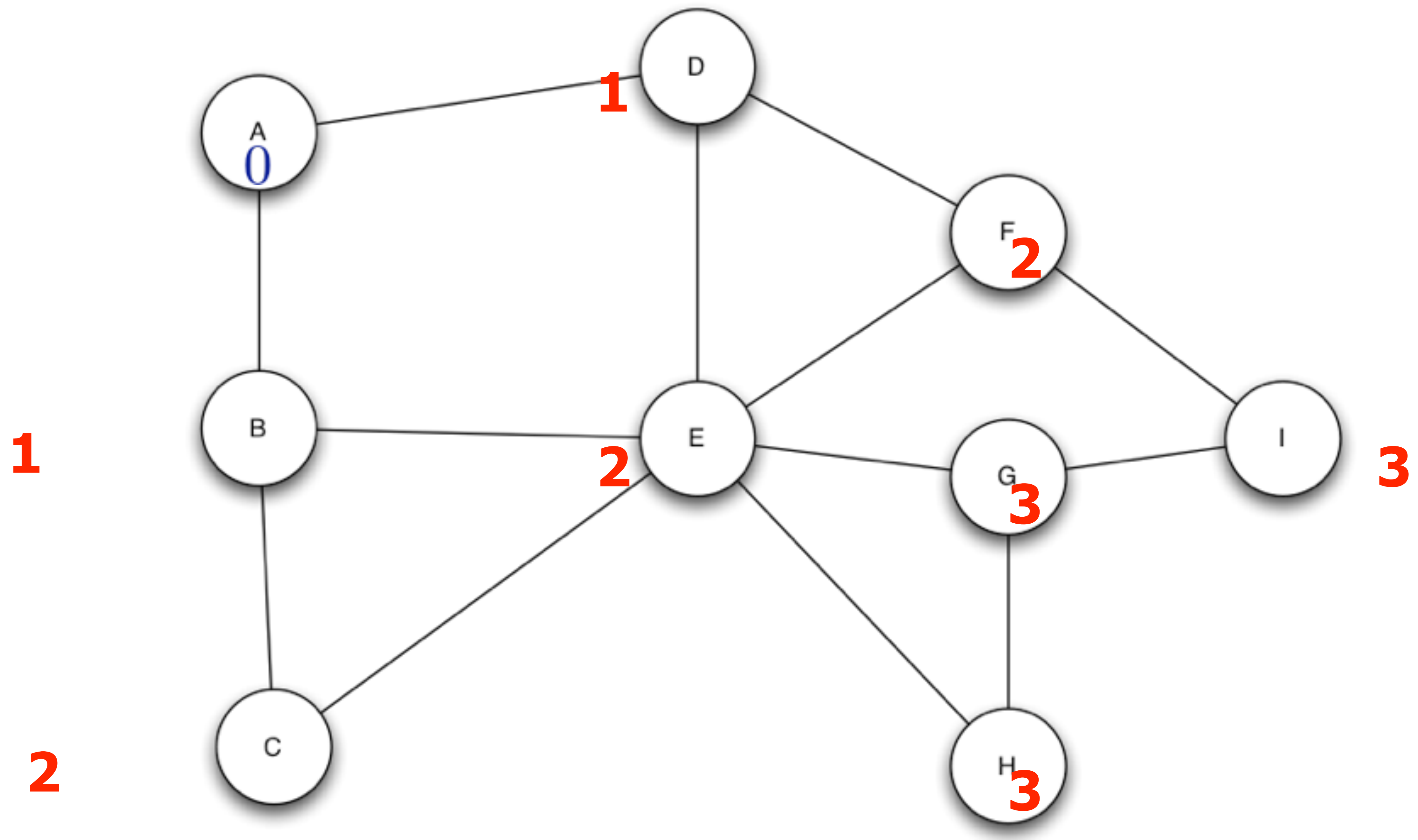
E

F

G

H

BFS(G, A)



Q

A

B

D

C

E

F

G

H

BFS(G, A)

BREADTH FIRST SEARCH

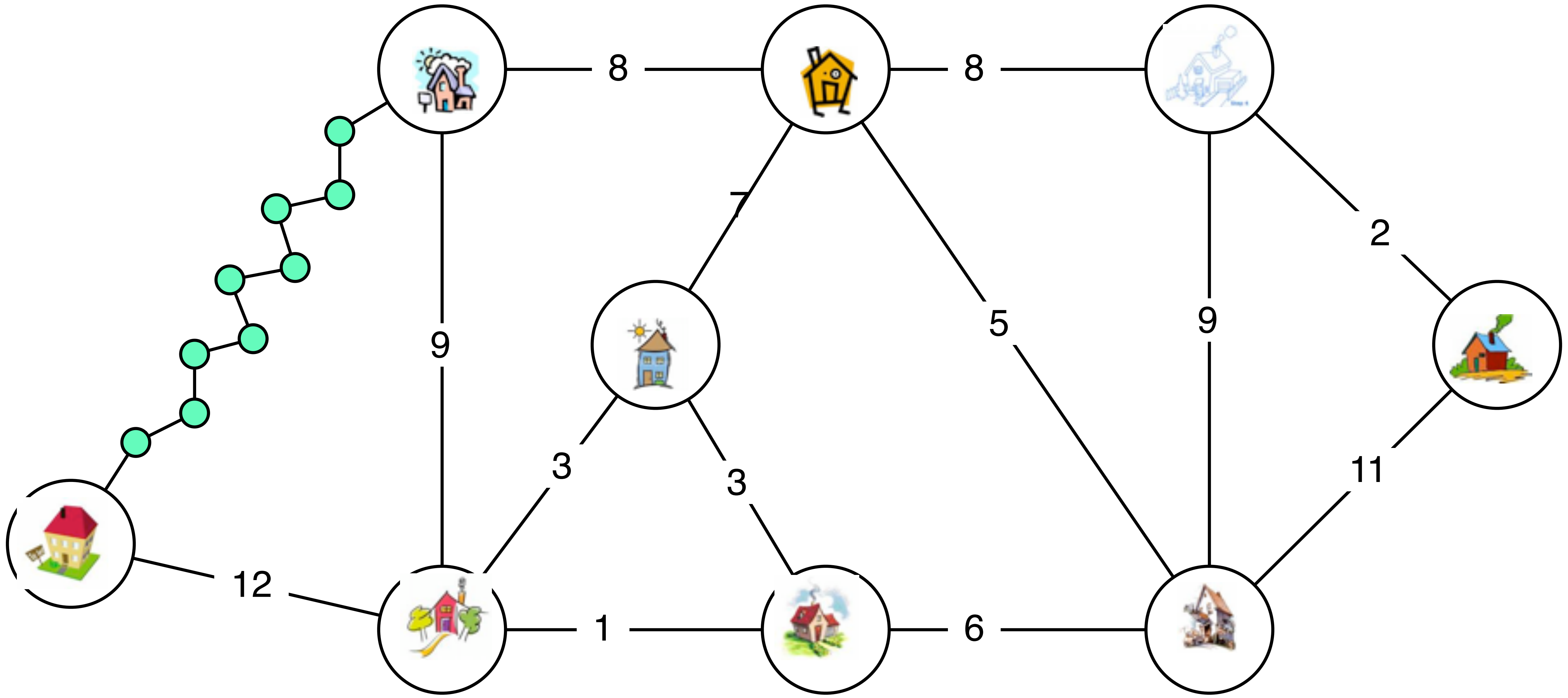
```
BFS(V, E, s)
for each u ∈ V − {s}
    do d[u] ← ∞
d[s] ← 0
Q ← ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    do u ← DEQUEUE(Q)
       for each v ∈ Adj[u]
           do if d[v] = ∞
                then d[v] ← d[u] + 1
                     ENQUEUE(Q, v)
```

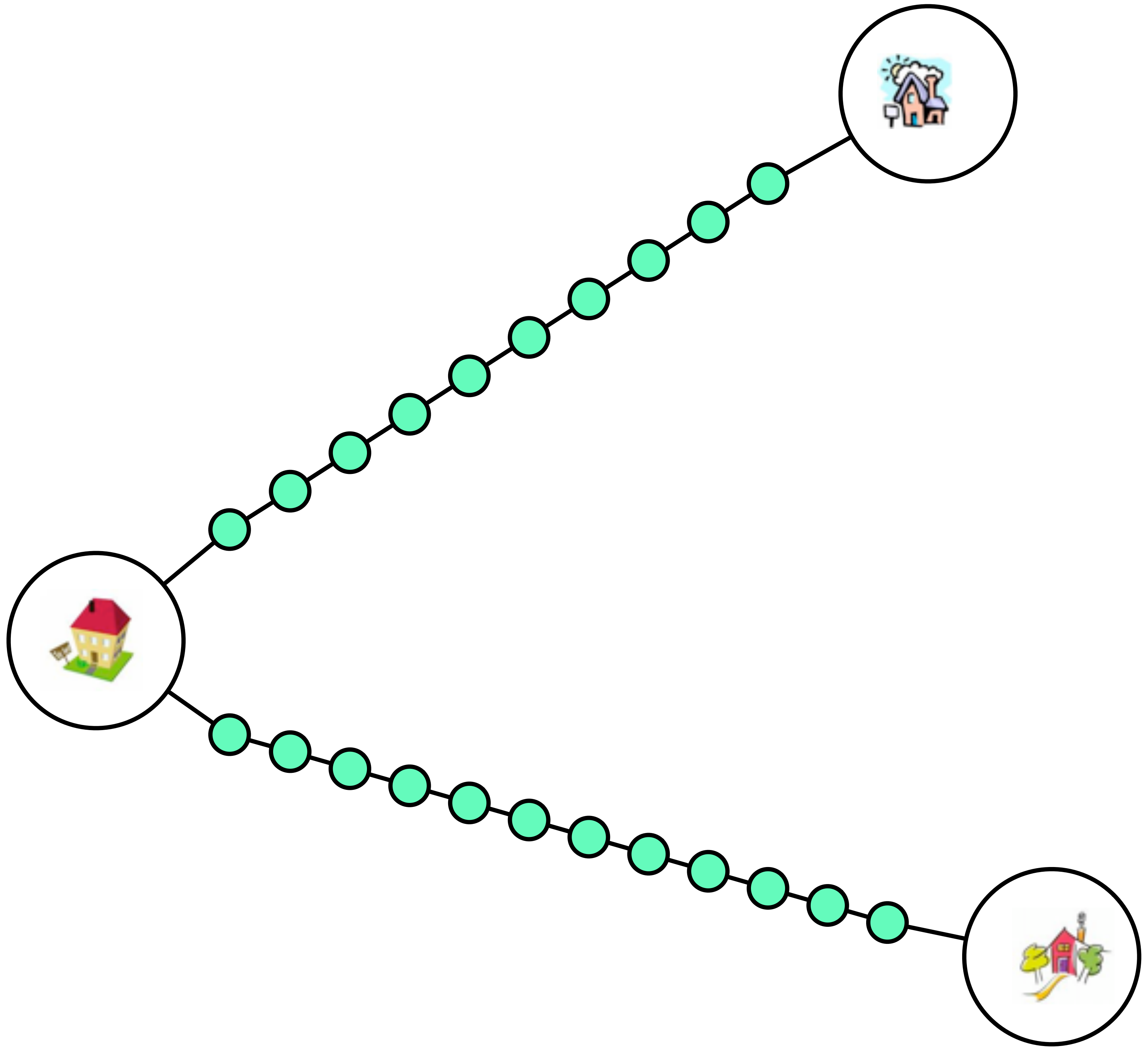
BFS THEOREM

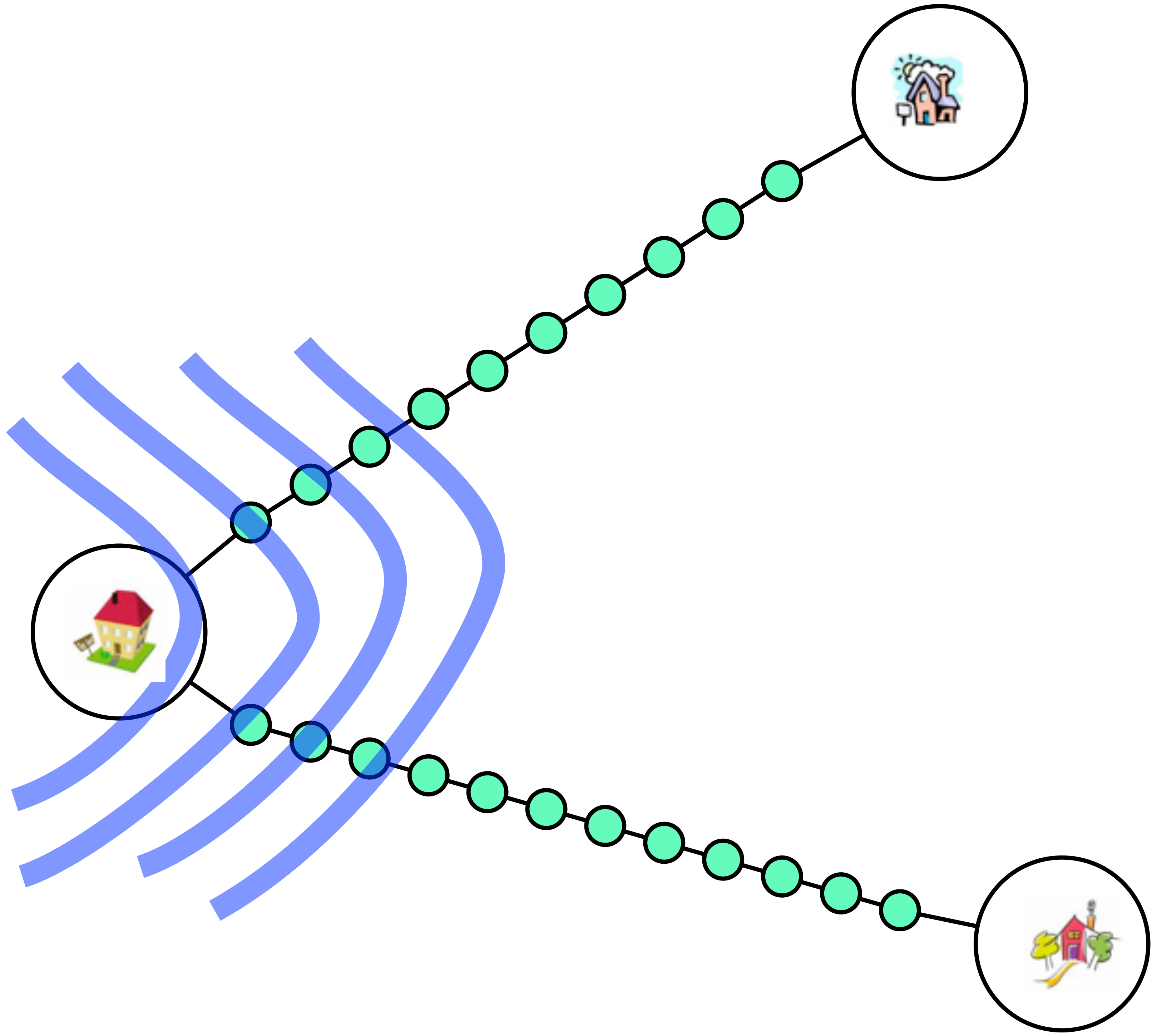
```
BFS(V, E, s)
for each u ∈ V − {s}
    do d[u] ← ∞
d[s] ← 0
Q ← ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    do u ← DEQUEUE(Q)
       for each v ∈ Adj[u]
           do if d[v] = ∞
                 then d[v] ← d[u] + 1
                      ENQUEUE(Q, v)
```

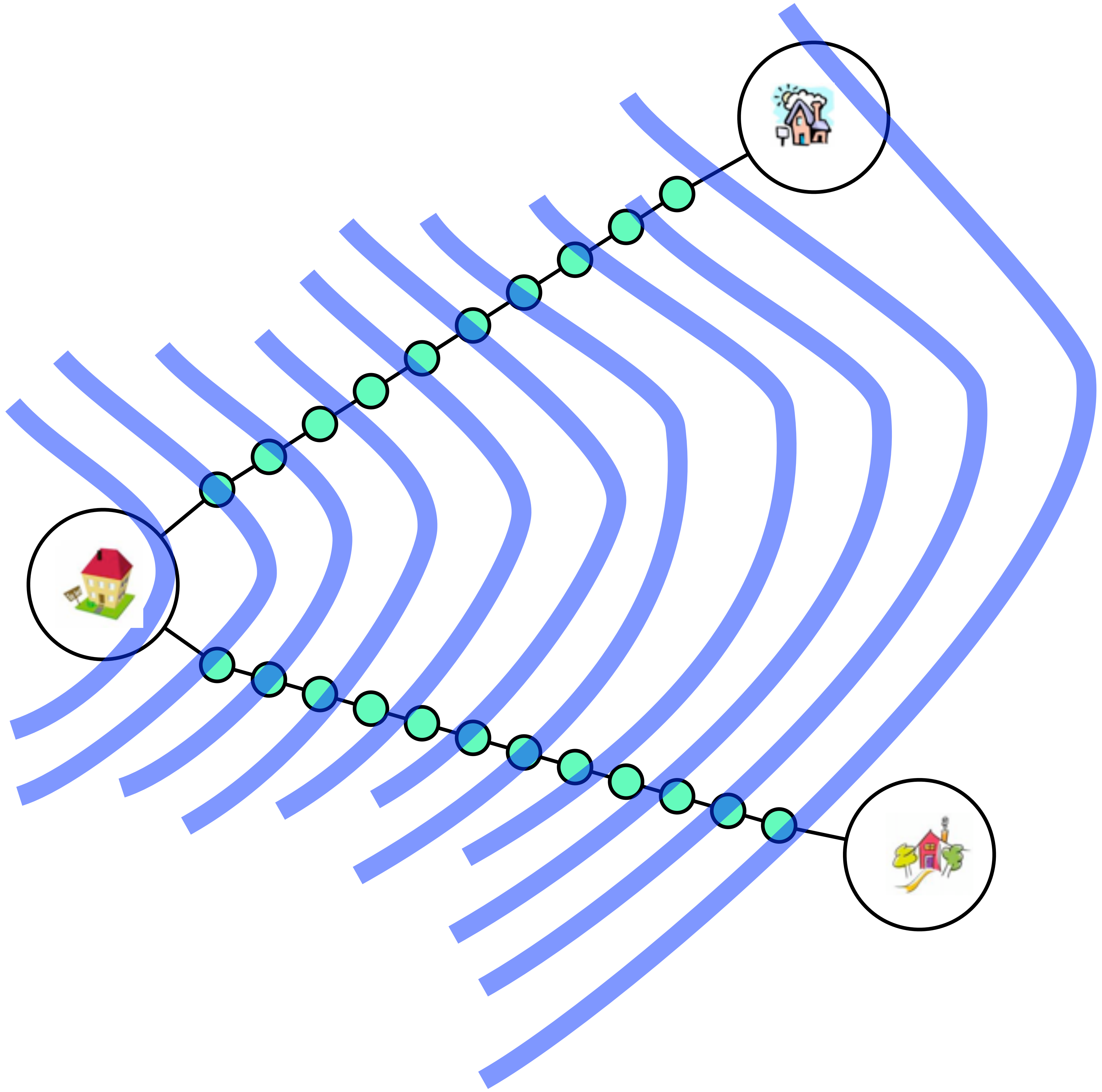
```
DIJKSTRA(G = (V, E), s)
for all v ∈ V
      do d_u ← ∞
         π_u ← NIL
d_s ← 0
Q ← MAKEQUEUE(V)    ▷ use d_u as key
while Q ≠ ∅
      do u ← EXTRACTMIN(Q)
         for each v ∈ Adj(u)
             do if d_v > d_u + w(u, v)
                   then d_v ← d_u + w(u, v)
                        π_v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

BFS









SHORTEST PATHS

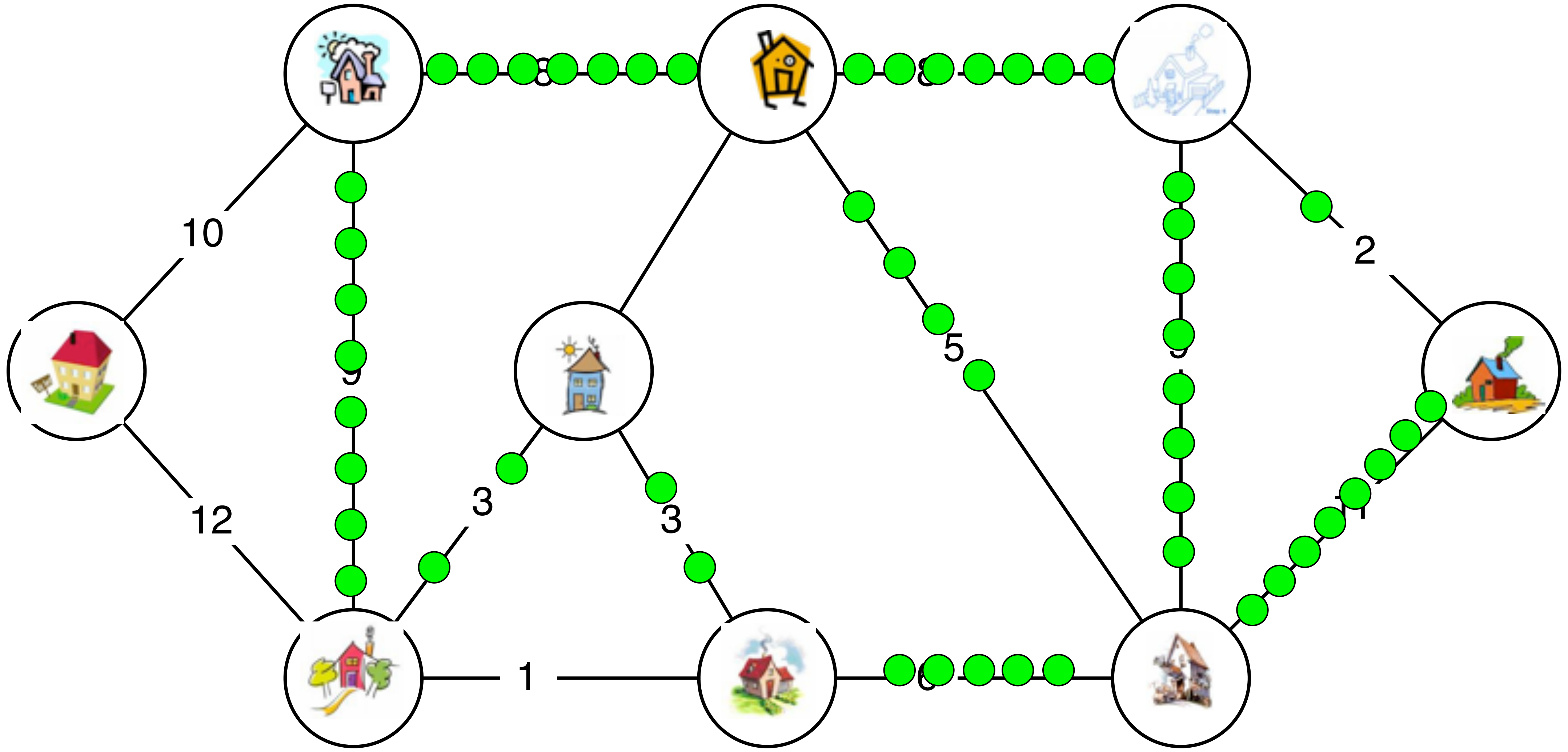


image:www.princegeorgeva.org, thefranciscofamily.org, www.rightdriveacademy.co.uk, www.ccscambridge.org, www.drawingcoach.com, www.pastoral.org.uk, www.daasgallery.com