L13

4800

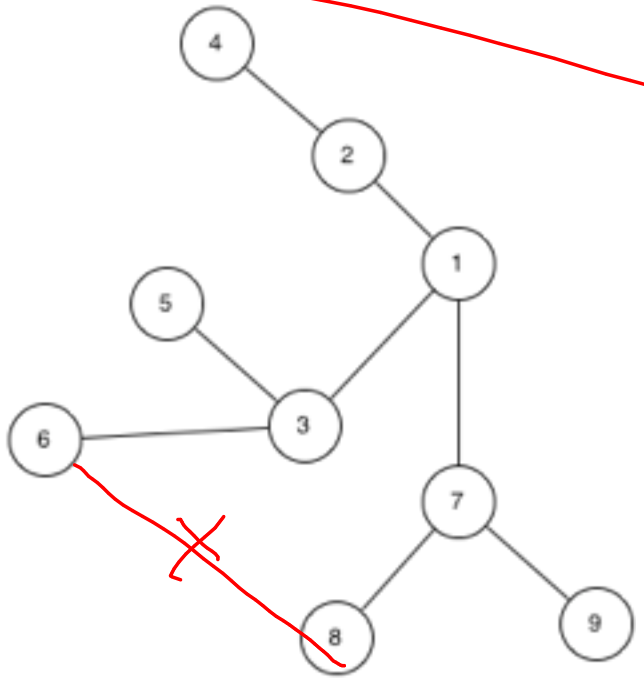
10.21.2016

abhi shelat

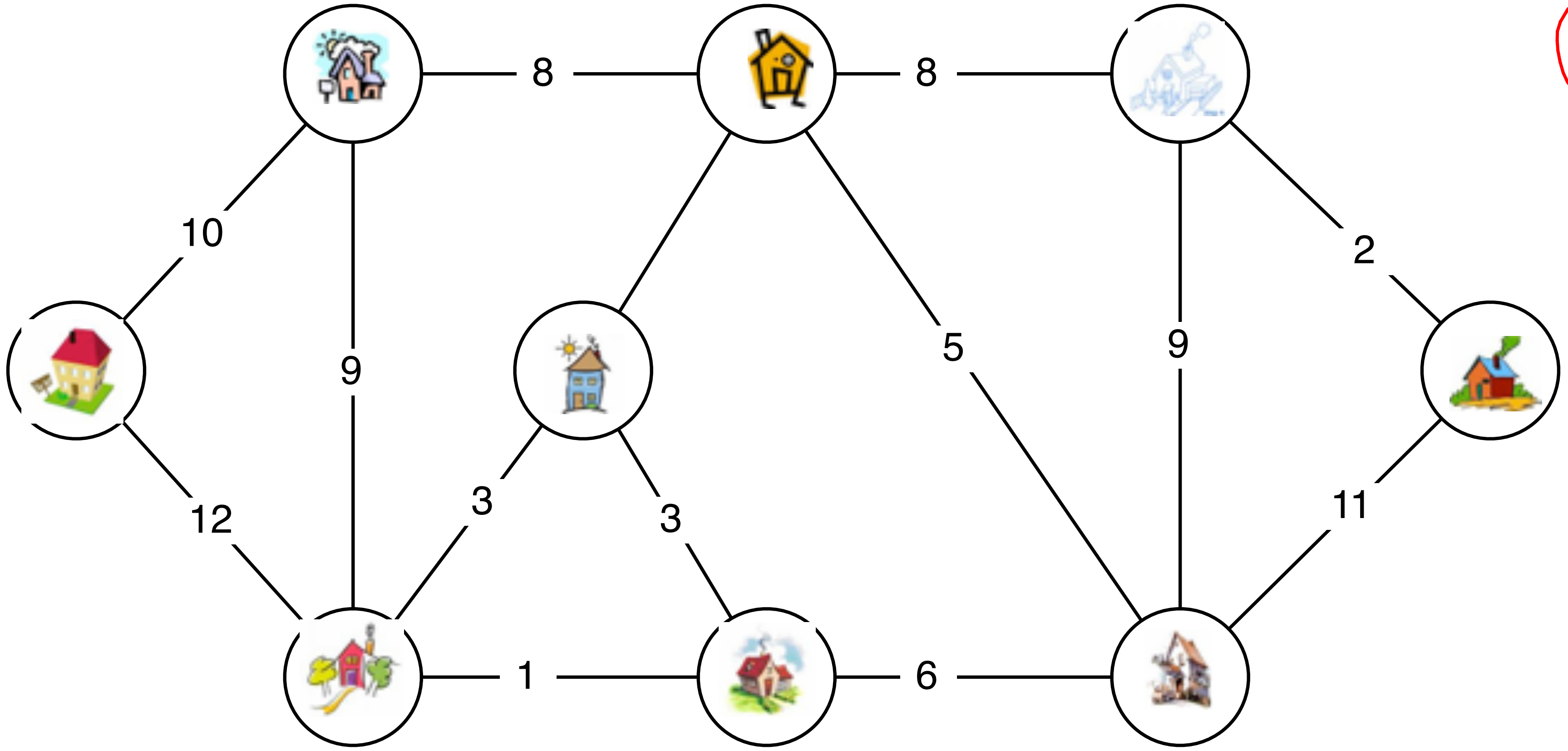
definition:tree

connected graph: for any pair $u, v \in V$, we have that there exists a path from u to v in G,

a tree is connected graph that has No cycles



what we want:



each edge has a cost

① set of edges A ⊆ E that connects all nodes in the graph

② minimize the cost of this set A

minimum spanning tree

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

(a) connects all vertices

(b) has the least cost $\min \sum_{(u,v) \in T} w(u,v)$

facts

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

(a) connects all vertices

(b) has the least cost $\min \sum_{(u,v)\in T} w(u,v)$

how many edges does solution have ? V-1

does solution have a cycle?

No cycle !!

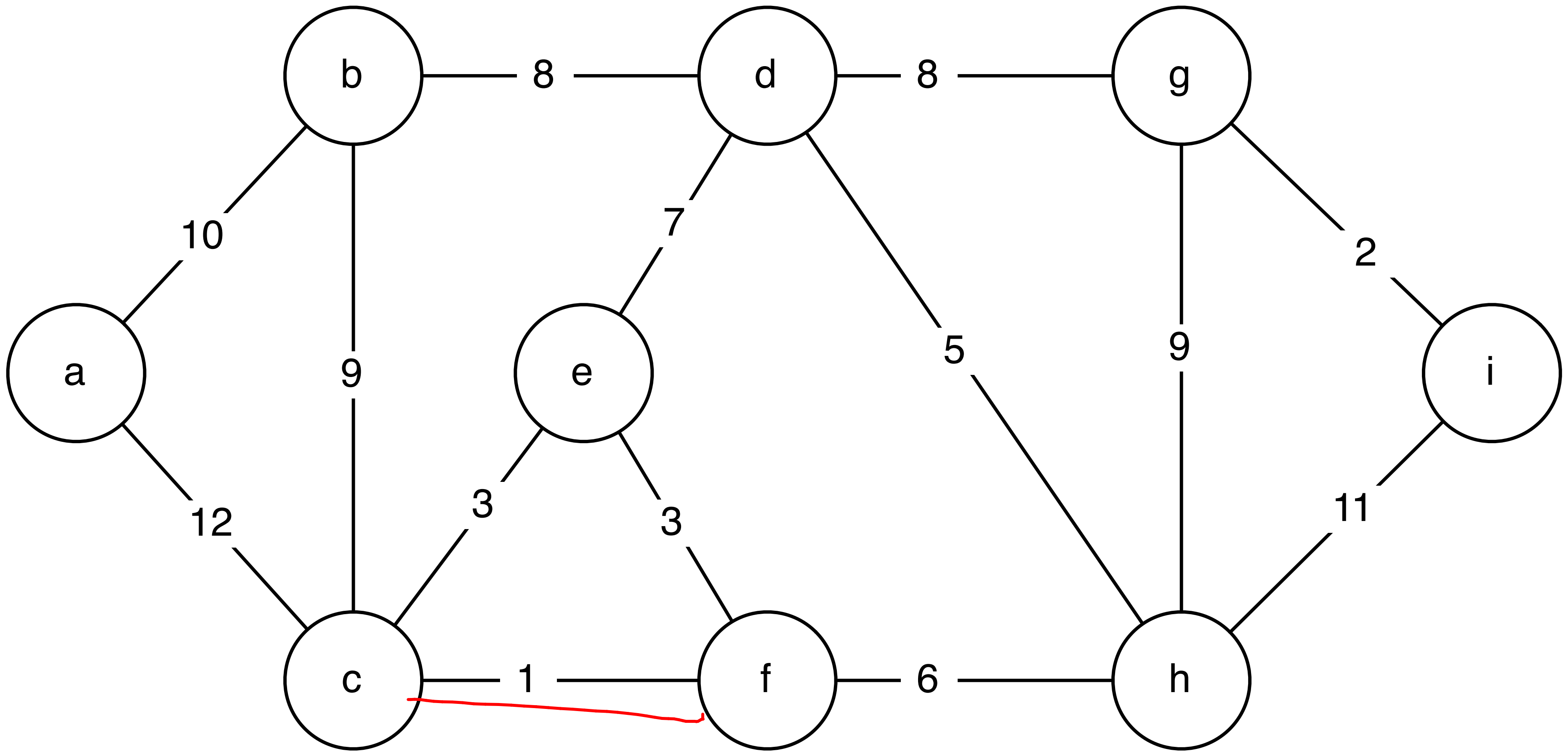
Greedy strategy

start with an empty set of edges A

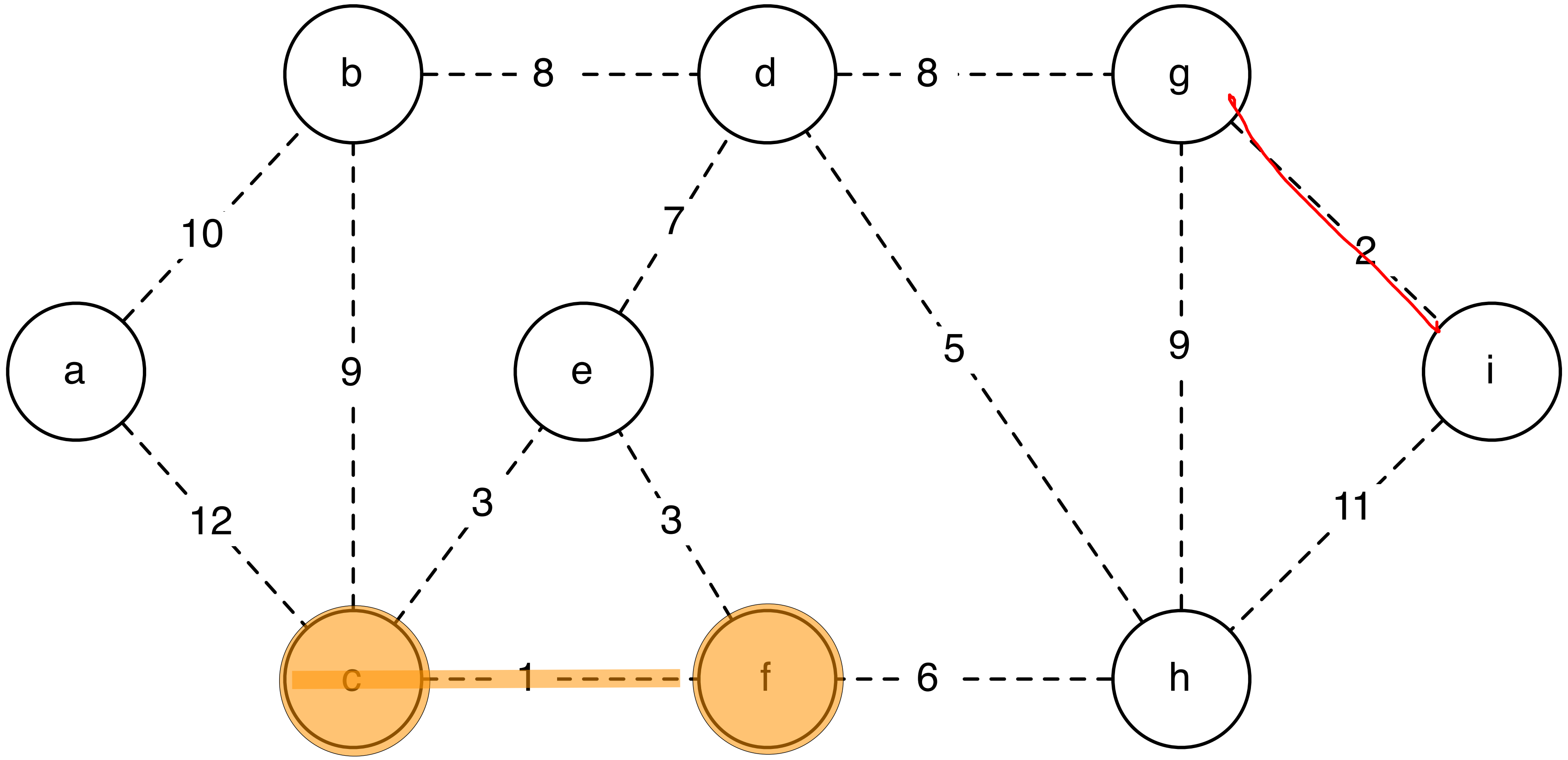
repeat for v-1 times:

    add lightest edge that does not create a cycle

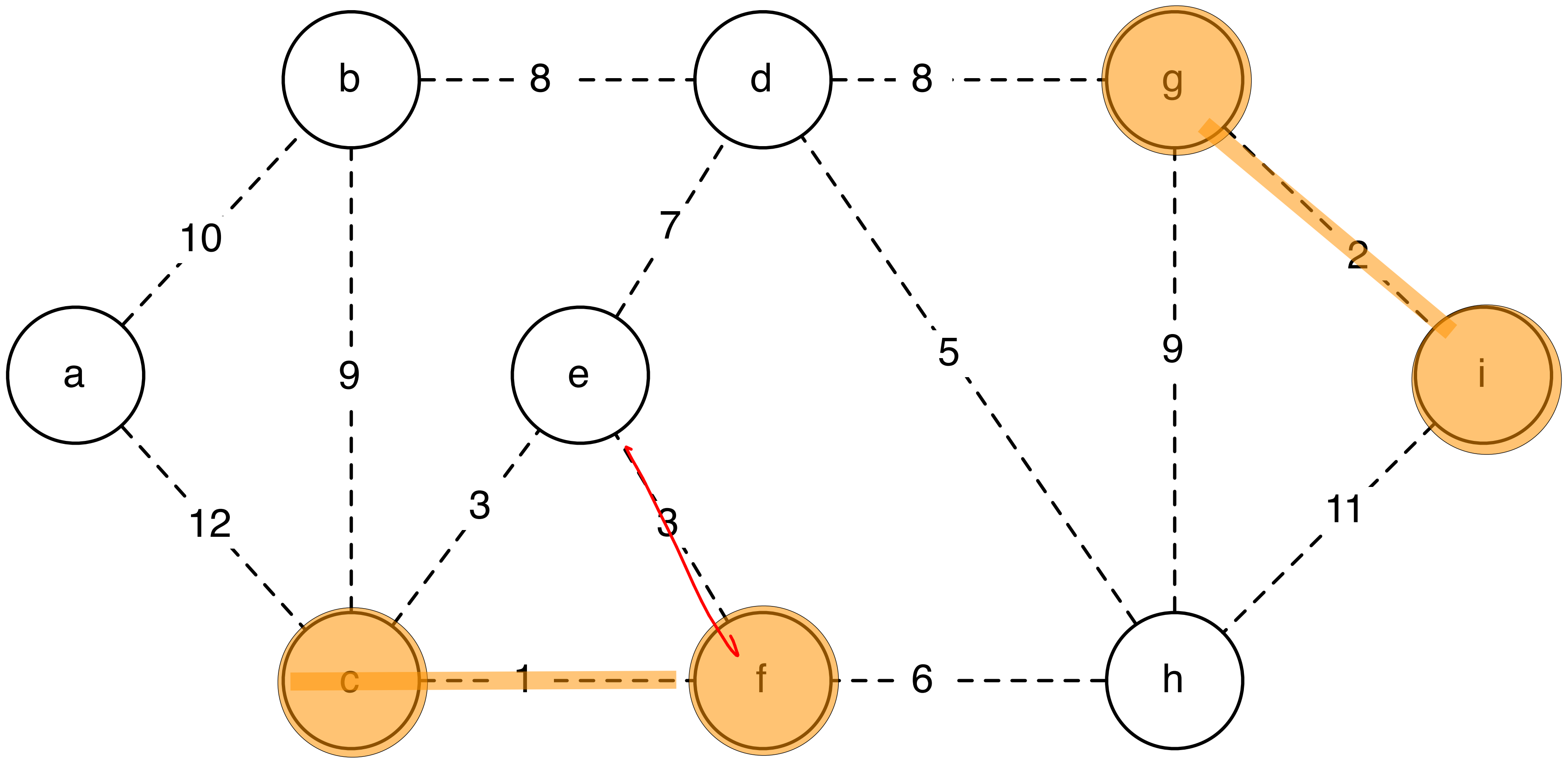
example



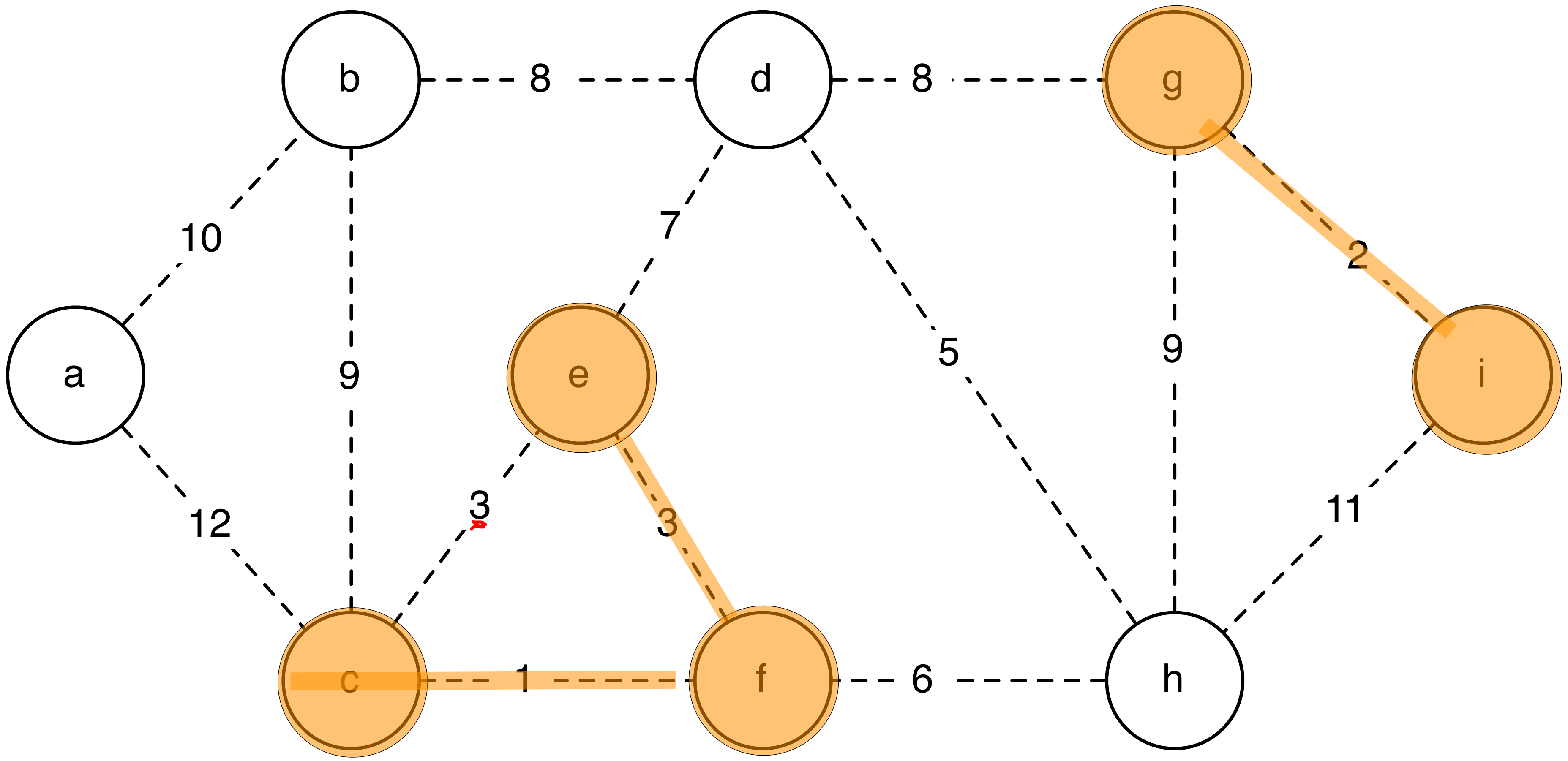
Kruskal



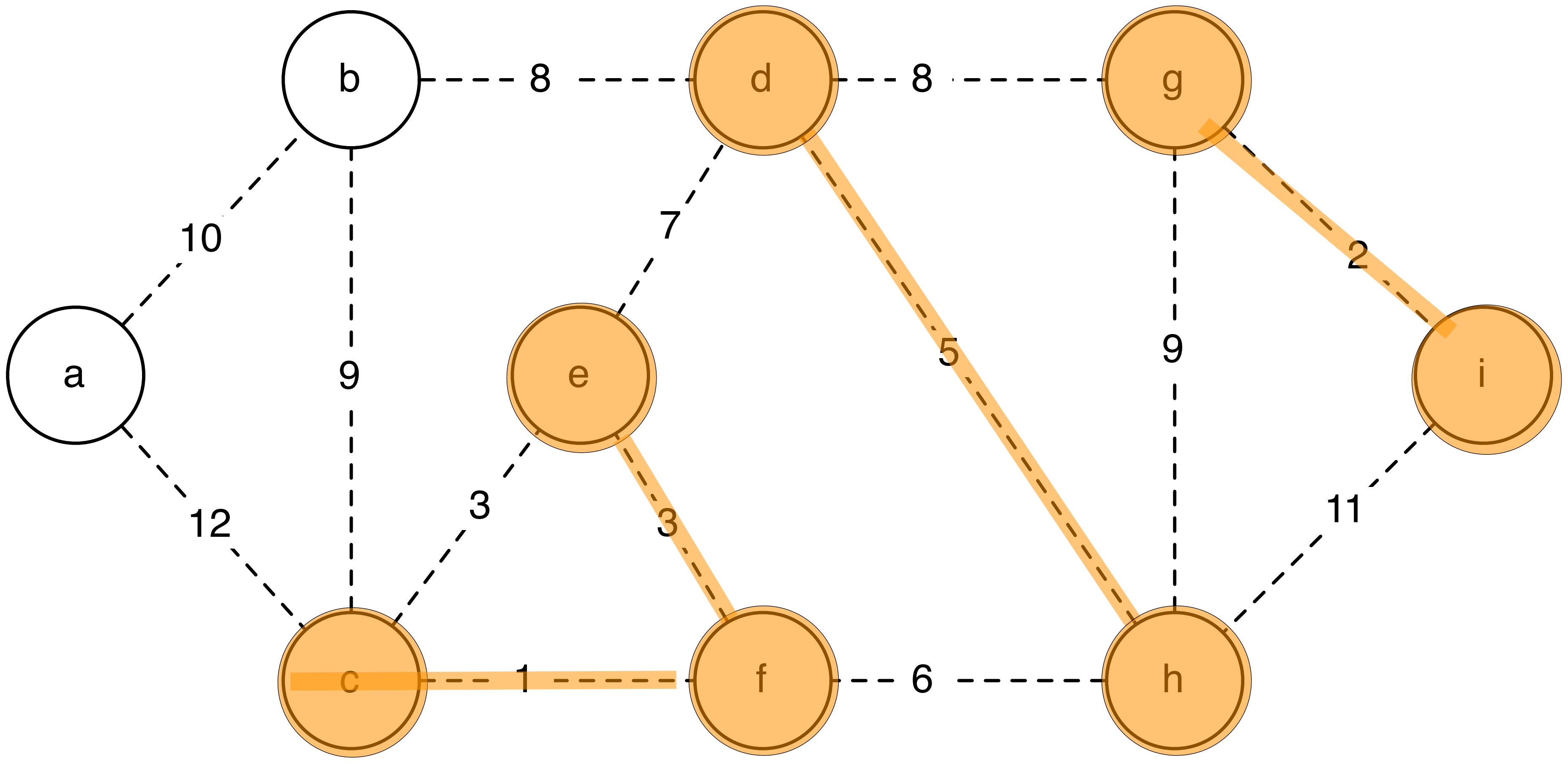
Kruskal



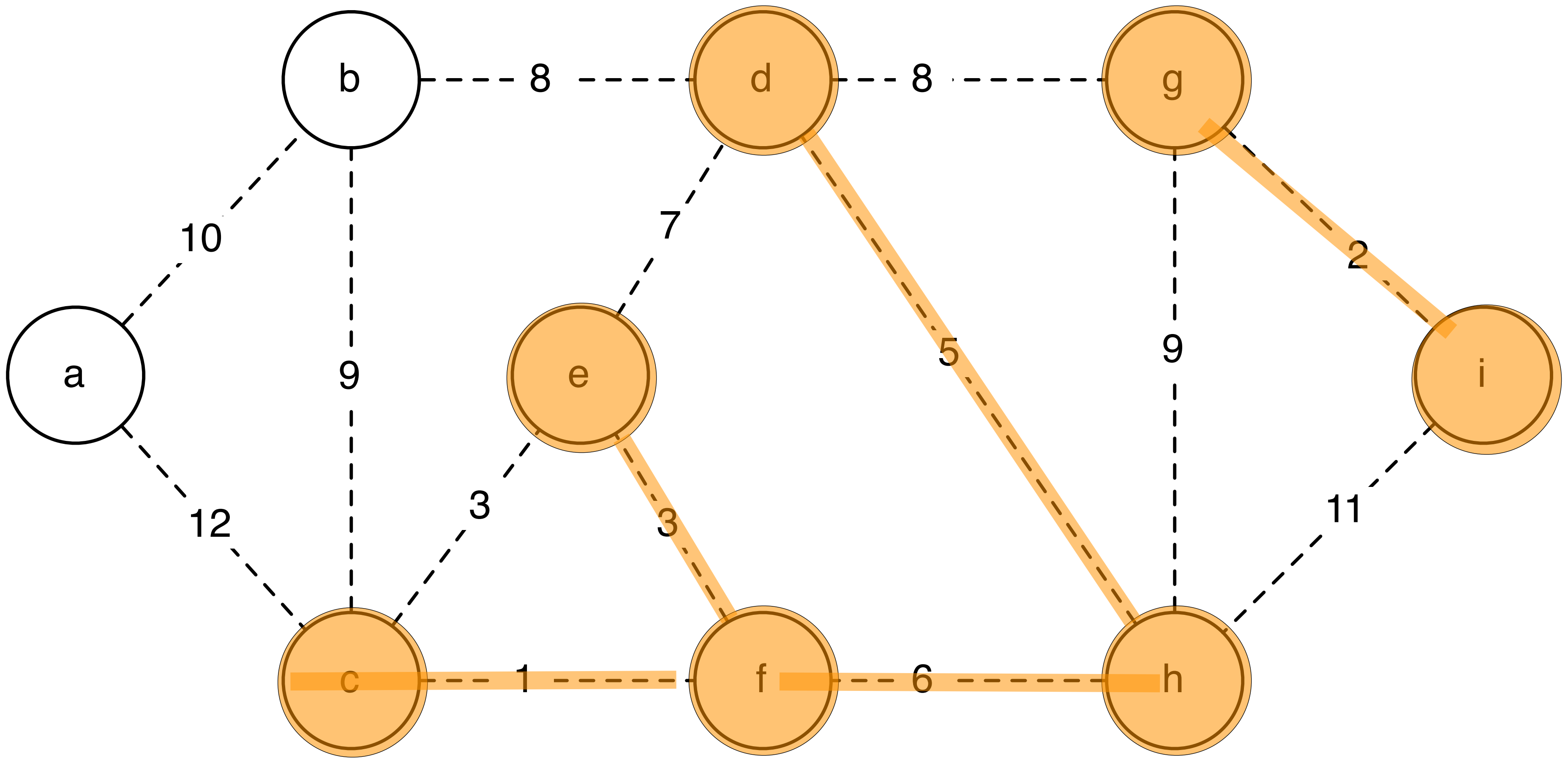
Kruskal



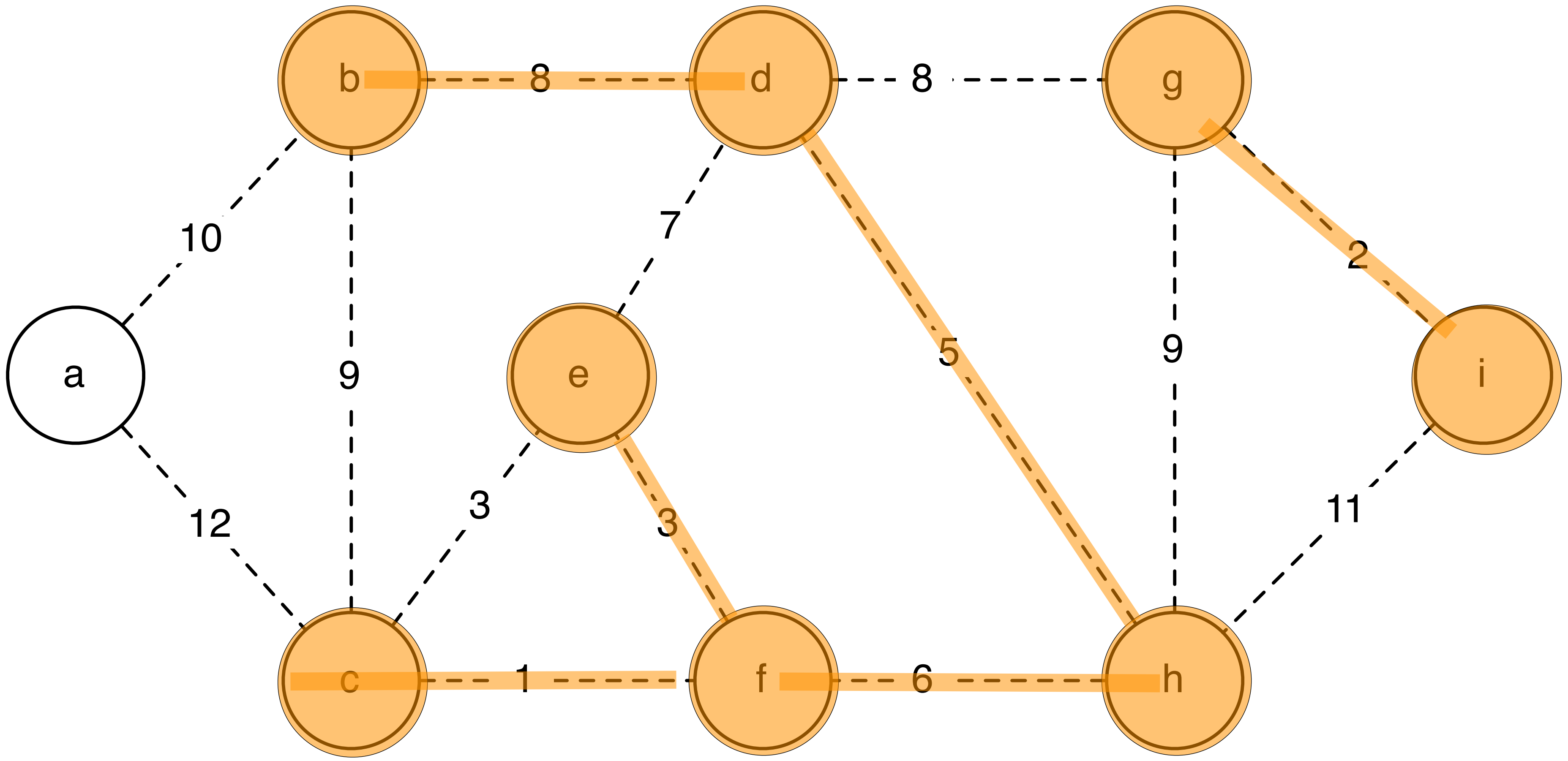
Kruskal



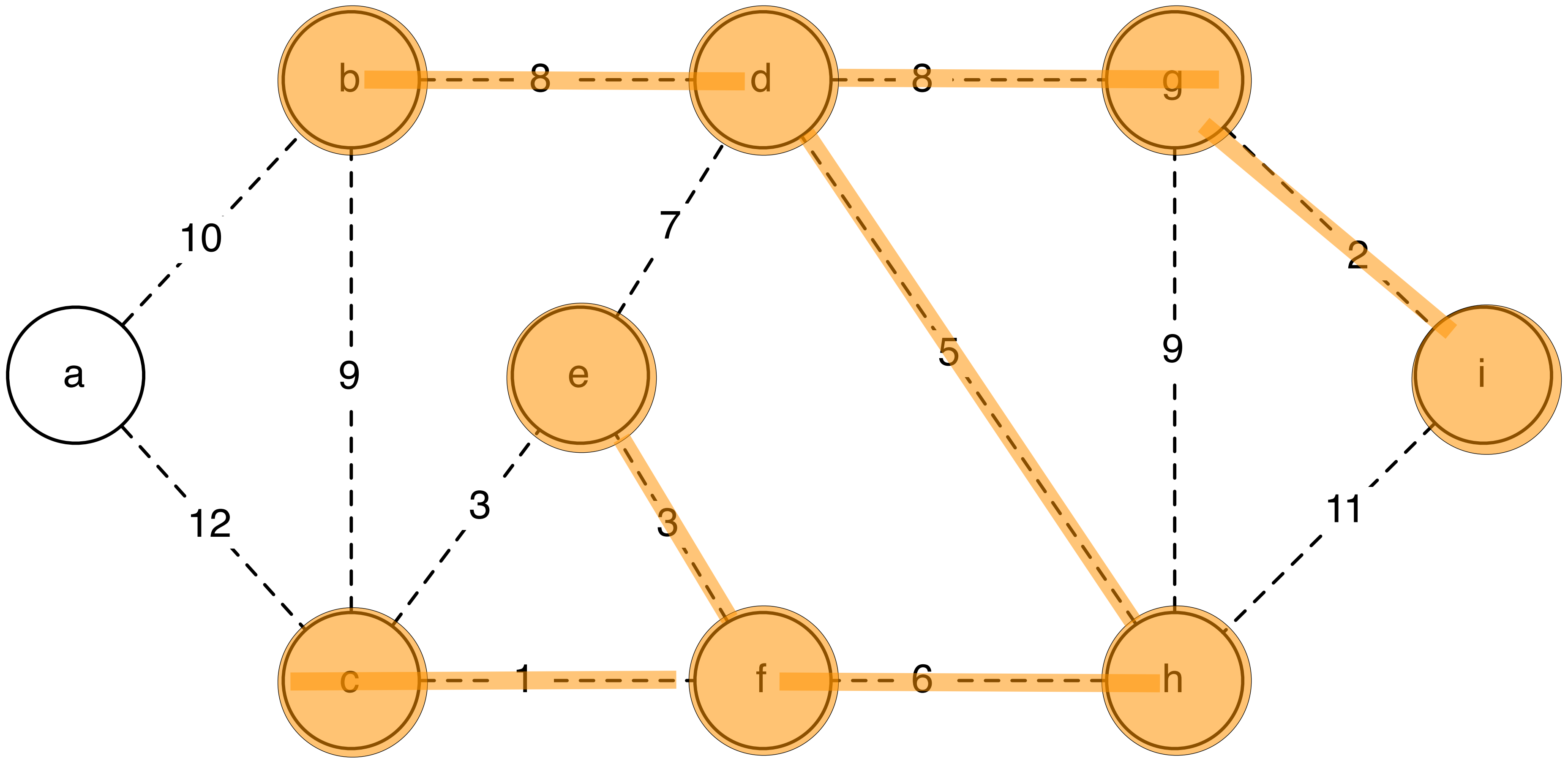
Kruskal



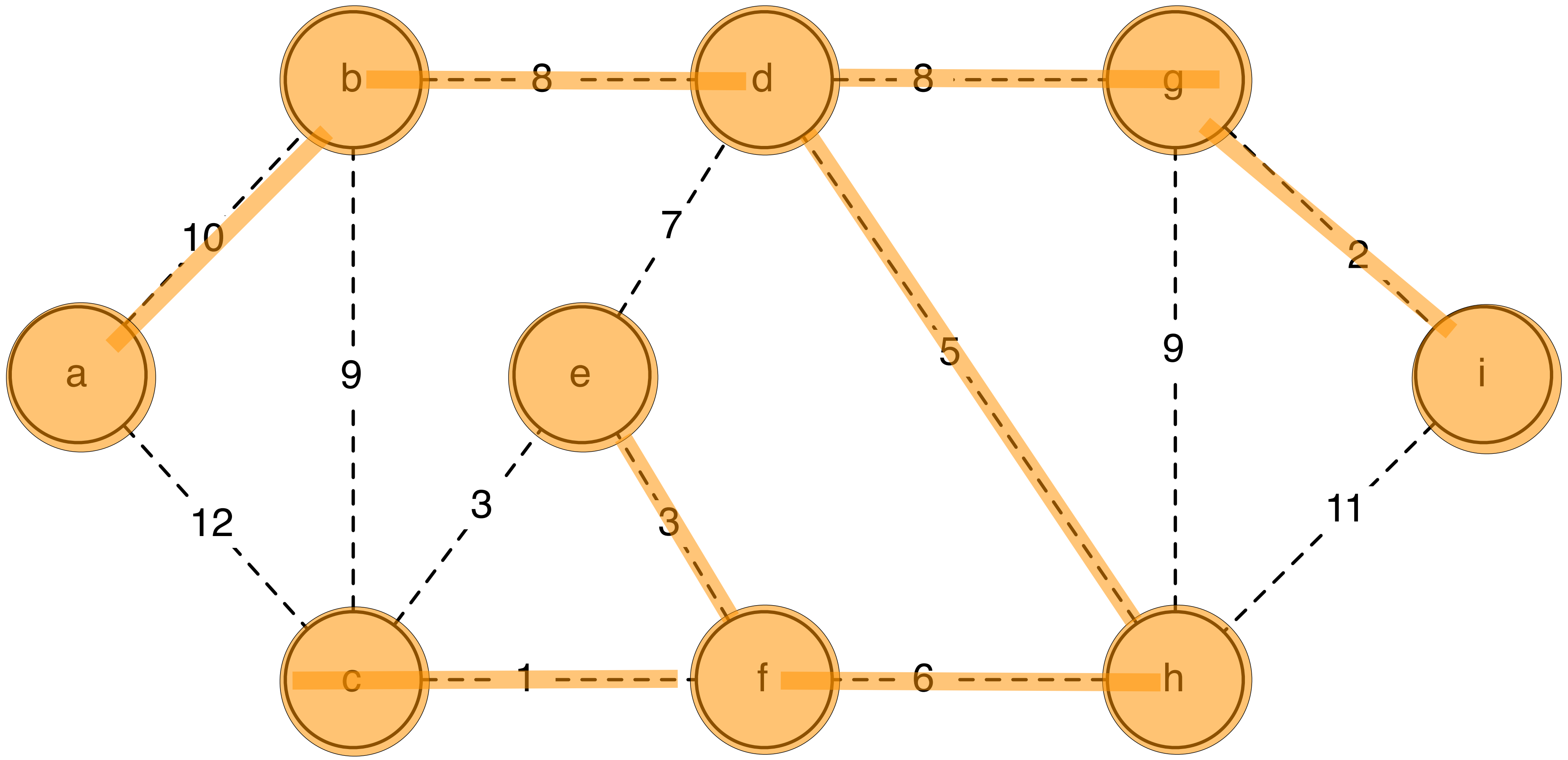
Kruskal



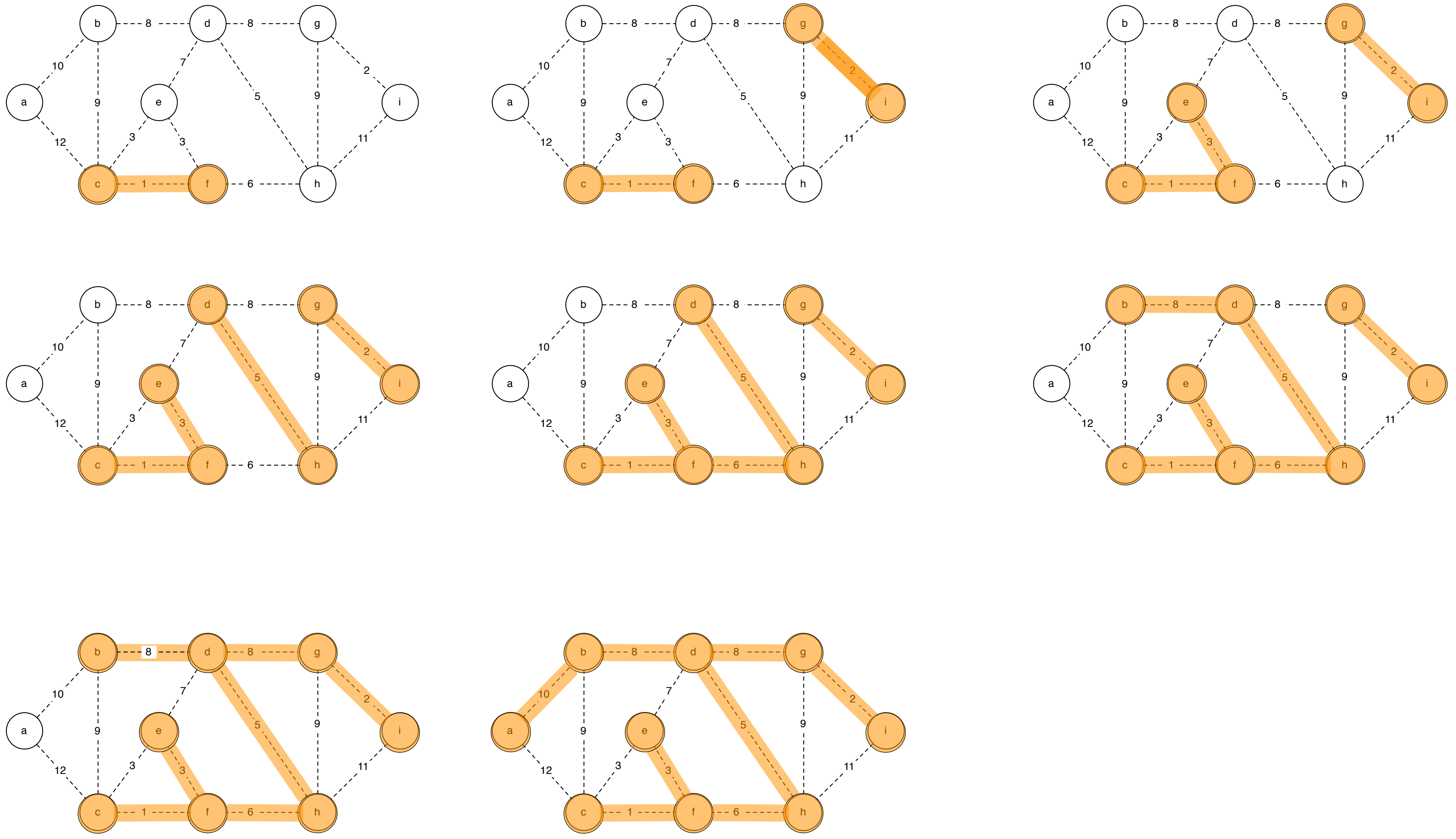
Kruskal



Kruskal



Kruskal



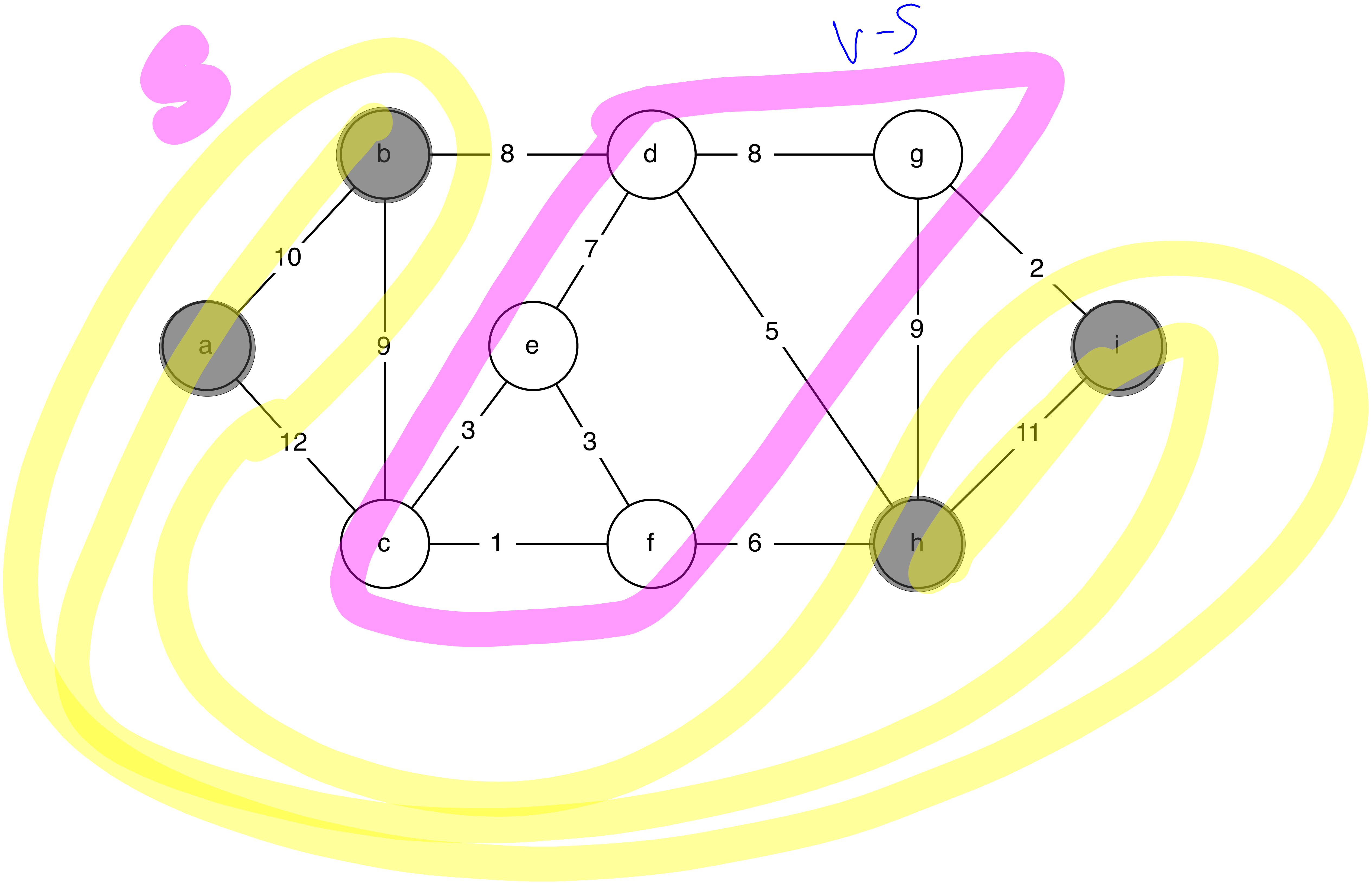
why does this work?

$T \leftarrow \emptyset$
repeat $V - 1$ times:
add to T the lightest edge $e \in E$ that does not create a cycle

definition: cut

A cut is a partition of the set V into 2 sets $(S, V-S)$.

example of a cut



definition: crossing a cut

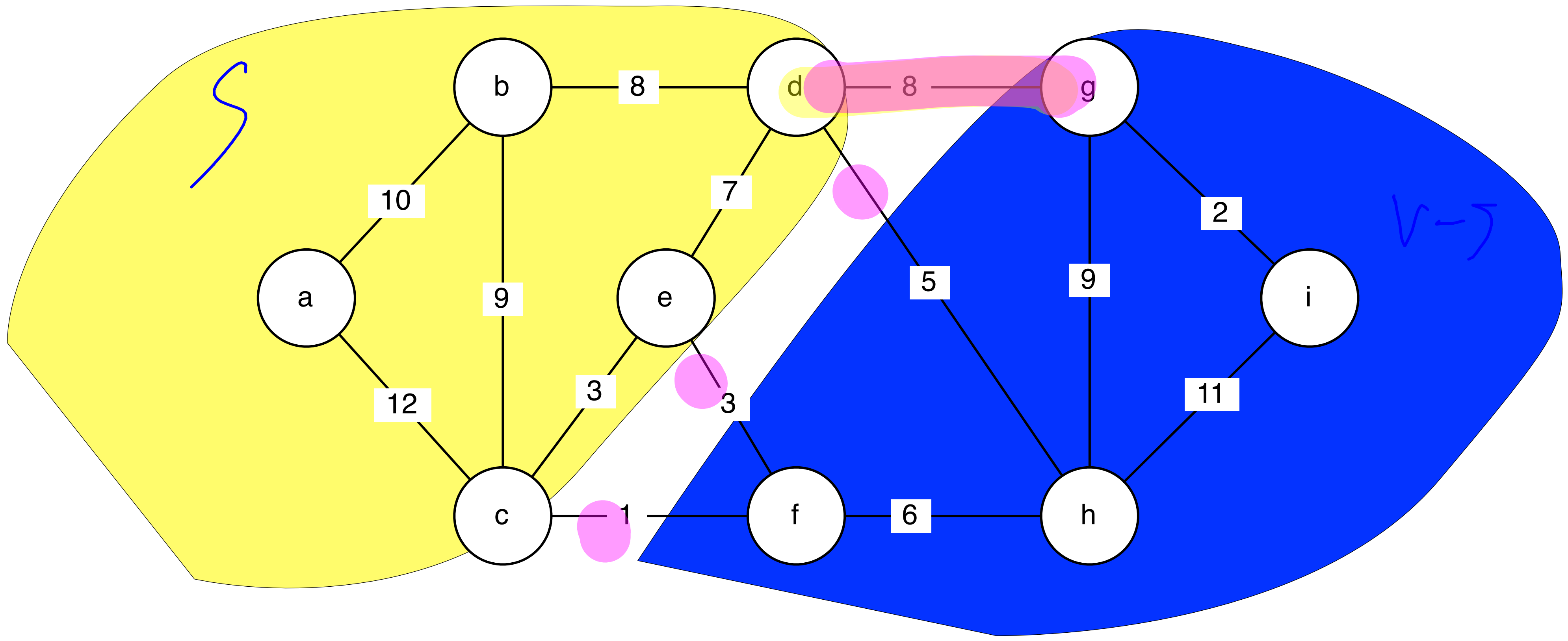
An edge $e = (u, v)$ crosses a cut $(S, V - S)$

if $u \in S$ and $v \in V - S$.

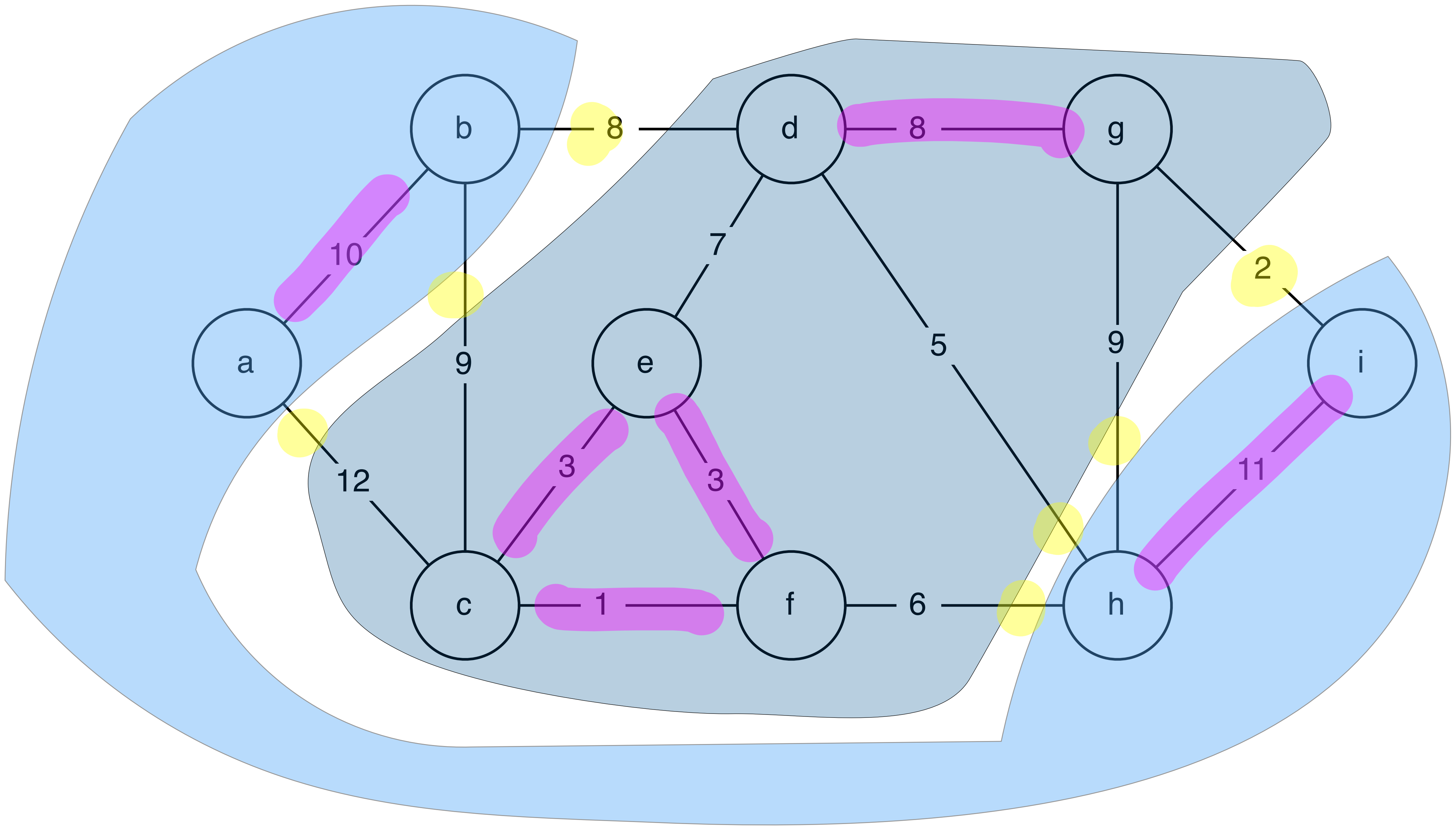
definition: crossing a cut

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

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



example of a crossing



definition: respect

A set A respects the cut $(S, V-S)$ if

No edge $e \in A$ crosses $(S, V-S)$.

Cut theorem

Let T be an MST for (G,V) and let $A \subseteq T$.

Let $(S, V-S)$ be some cut that A respects and

let e be the lightest edge that crosses $(S, V-S)$.

$\Rightarrow A \cup \{e\}$ is a subset of some MST of G.

Cut theorem

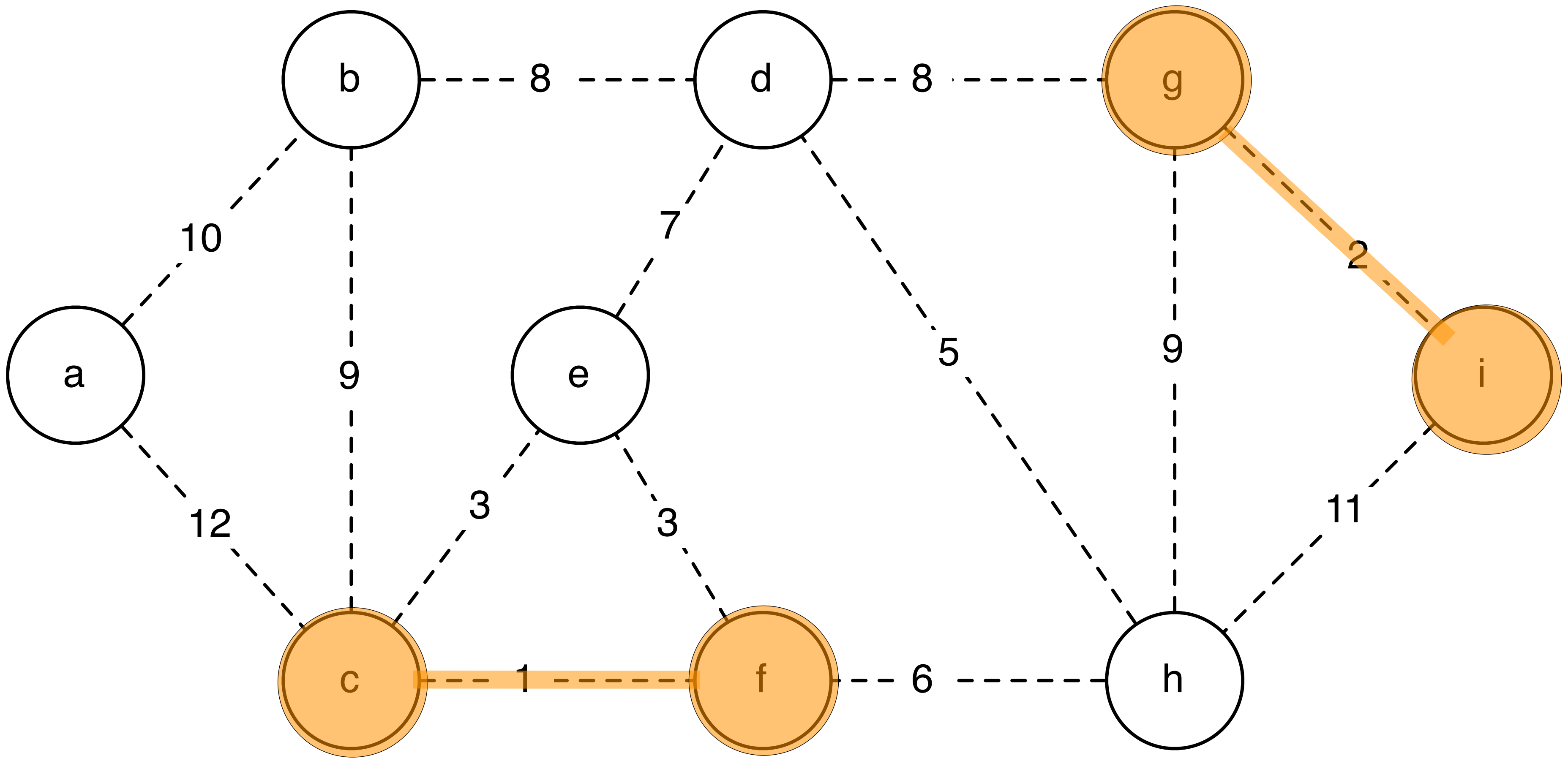
Suppose the set of edges A is part of an m.s.t.

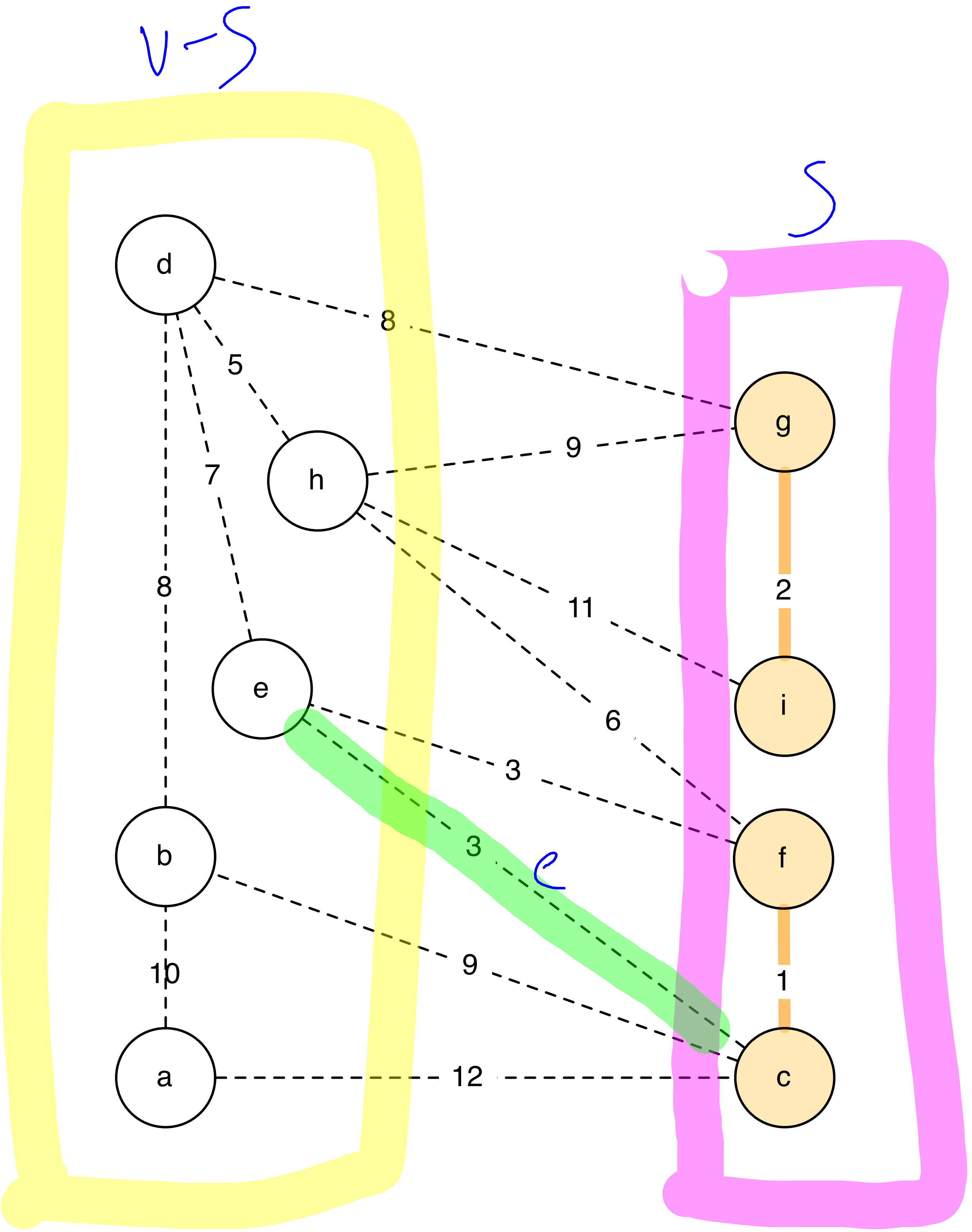
Let $(S, V-S)$ be any cut that A respects .

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





$A = \{(i,g)\ (c,f)\}$

proof of cut theorem

Theorem 2 *Suppose the set of edges A is part of a minimum spanning tree of* $G = (V, E)$*. Let* $(S, V - S)$ *be any cut that respects A and let e be the edge with the minimum weight that crosses* $(S, V - S)$*. Then the set* $A \cup \{e\}$ *is part of a minimum spanning tree.*

Proof: By hypothesis $A \subseteq T$ where T is an MST of G.

If $A \cup \{e\}$ is already $\subseteq T$, then the theorem follows.

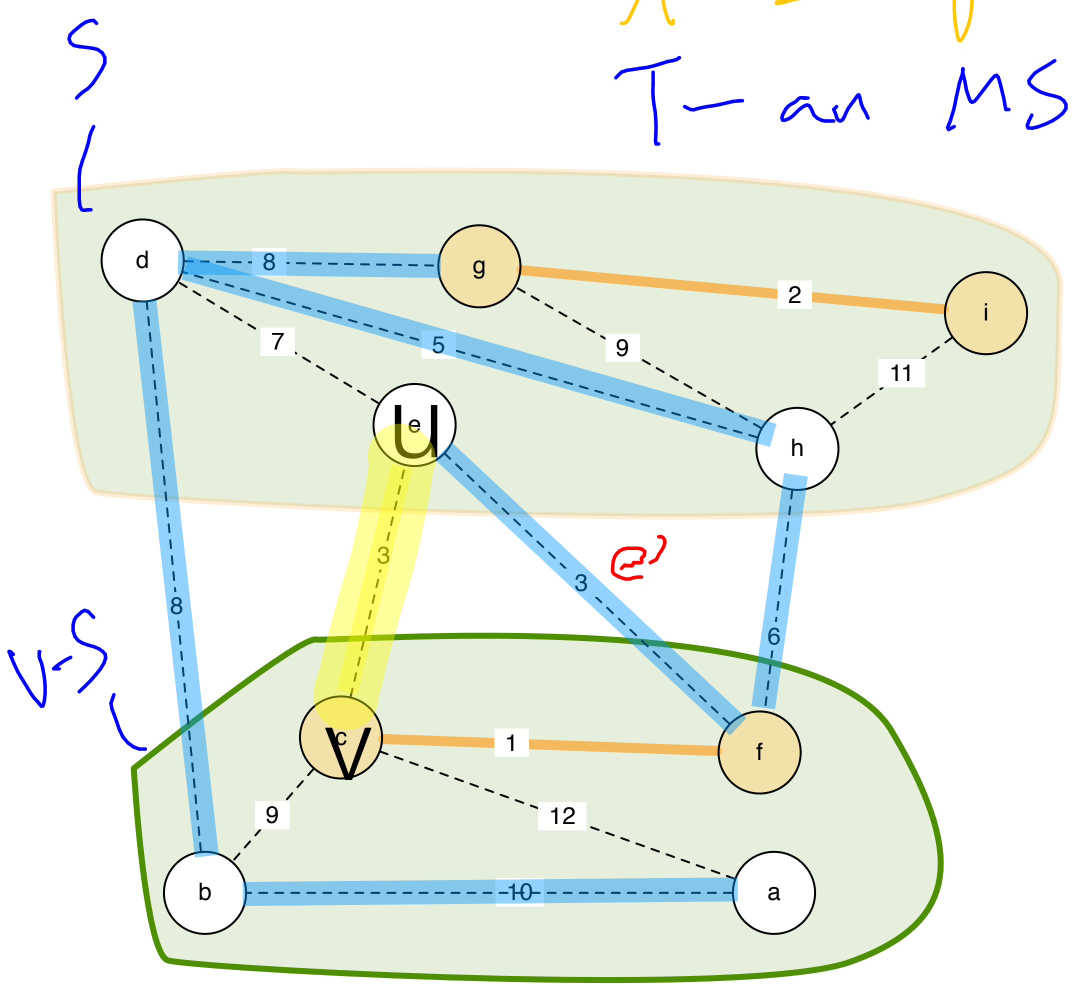
If not, then we need to construct another T' tree such that $A \cup \{e\} \subseteq T'$ and T' is also an MST.

How??

proof of cut thm

A – set of orange edges

T – an MST of G (with A)



Let $e=(u,v)$ be the lightest edge that crosses $(S,V-S)$.

① e is not part of T, but since T is an MST, it connects all nodes in G. So follow the path from u to v and let e' be the first edge to cross $(S,V-S)$. why does e' exist?? b/c e crosses $(S,V-S)$ so $u\in S$ and $v\in V-S$.

Consider the tree $T' = T\cup\{e\} - \{e'\}$. It has $(V-1)$ edges.

① $w(e) \le w(e') \Rightarrow w(T') \le w(T)$. But T was MST, so T' is an MST.

② $A\cup\{e\} \subseteq T'$. ∎

correctness

```
KRUSKAL-PSEUDOCODE(G)
→ 1  A ← ∅
  2  repeat V − 1 times:
  3        add to A the lightest edge e ∈ E that does not create a cycle
```

Proof: By induction. A is part of some MST T of G at line 1.

Spse A is part of an MST after k iteration of the main loop.

Show in the next iteration that A remains part of an MST.

→ e was the lightest edge that didn't create a cycle.

$e = (u, v)$.

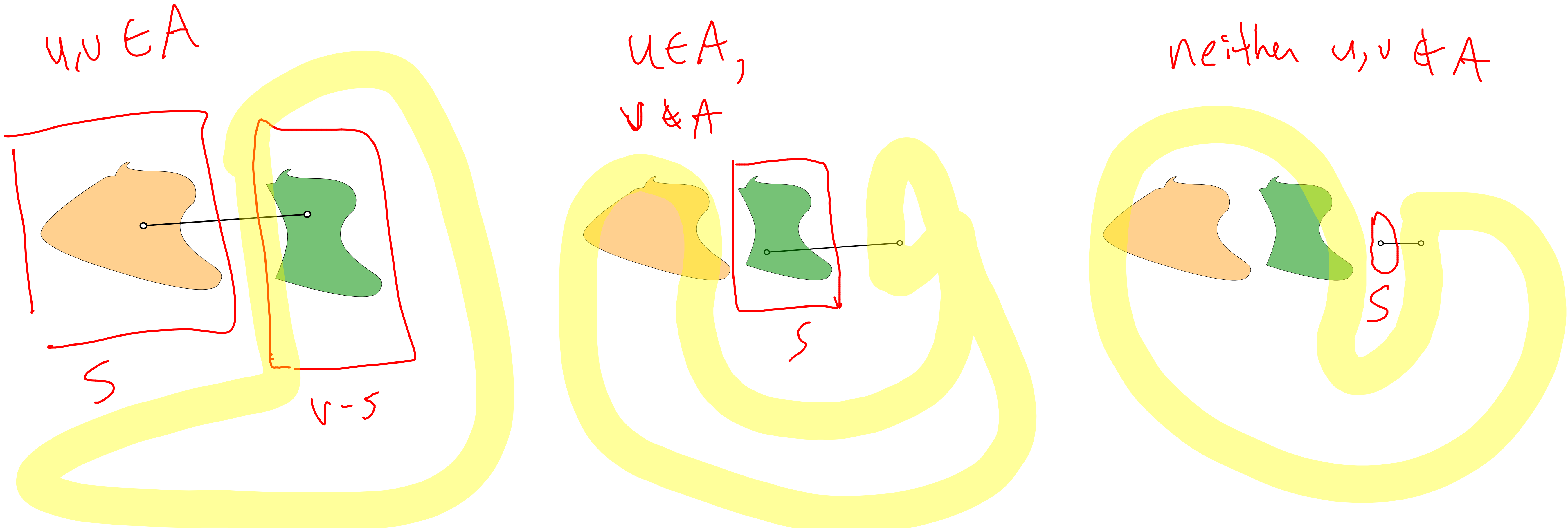
correctness

```
KRUSKAL-PSEUDOCODE(G)
1  A ← ∅
2  repeat  V − 1 times:
3            add to A the lightest edge e ∈ E that does not create a cycle
```

Proof: by induction. in step 1, A is part of some MST. Suppose that after k steps, A is part of some MST (line 2). In line 3, we add an edge e=(u,v).

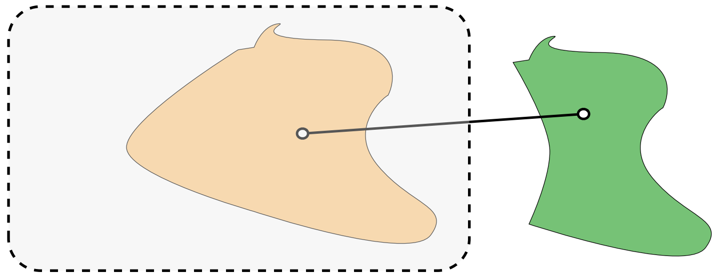
S to be the set of edges of A "connected" to u

e crosses (S, V−S)



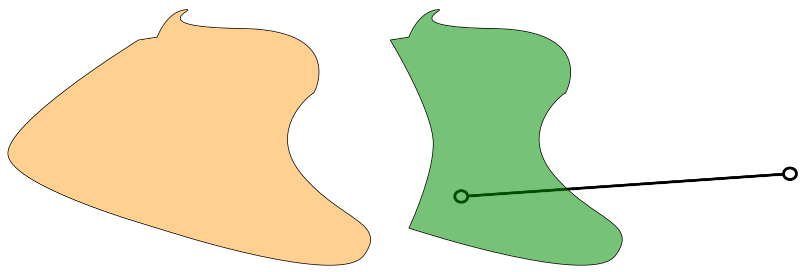
3 cases for edge e.

Case 1: e=(u,v) and both u,v are in A.



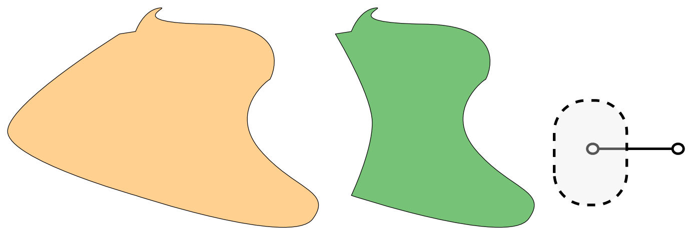
3 cases for edge e.

Case 2: e=(u,v) and only u is in A.



3 cases for edge e.

Case 3: e=(u,v) and neither u nor v are in A.



analysis?

```
KRUSKAL-PSEUDOCODE(G)
A ← ∅
repeat  V − 1 times:
        add to A the lightest edge e ∈ E that does not create a cycle
```

```
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}
```

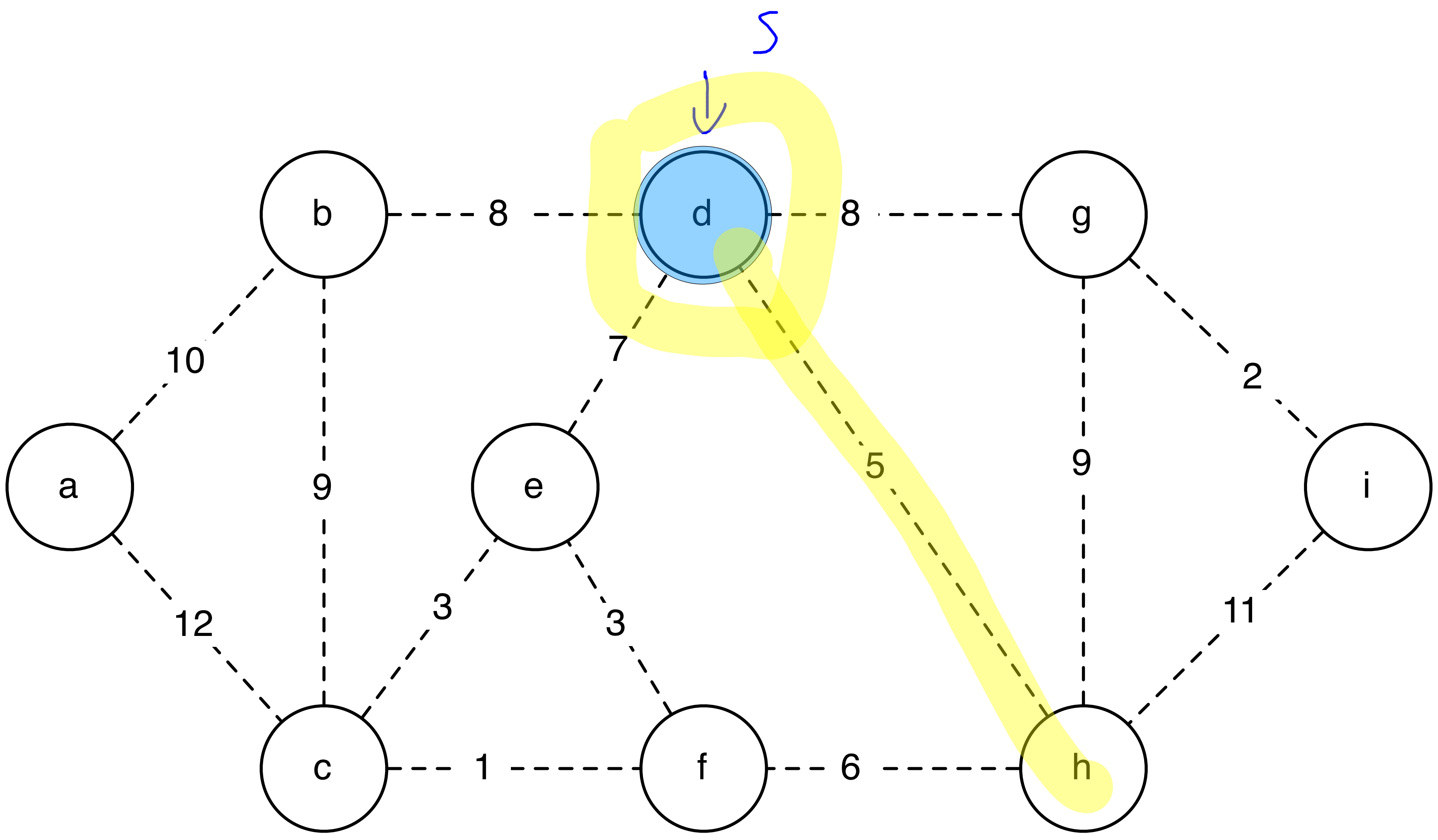
Prim's algorithm

```
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}
```

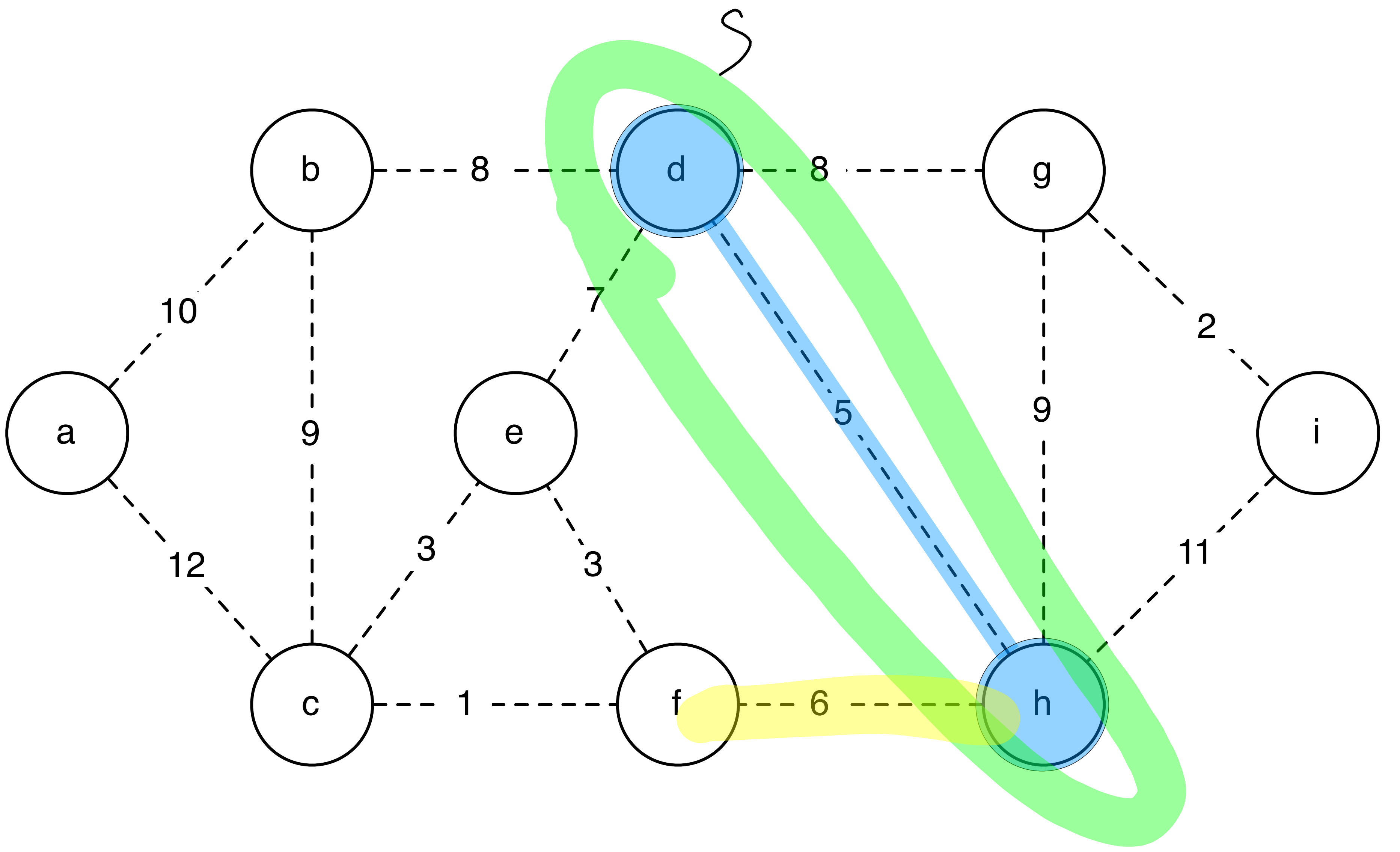
A is a subtree

edge e is lightest edge that grows the subtree

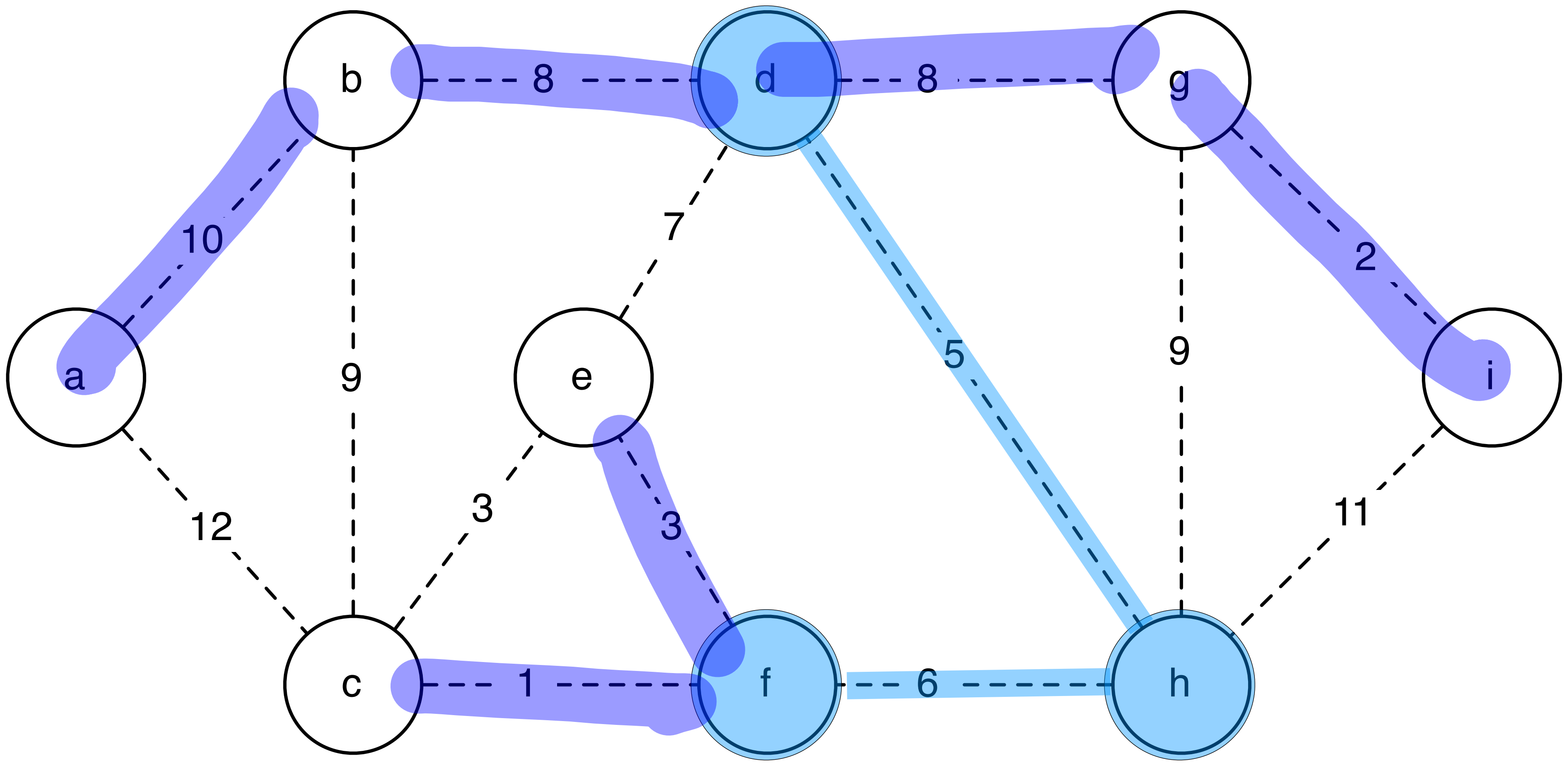
prim



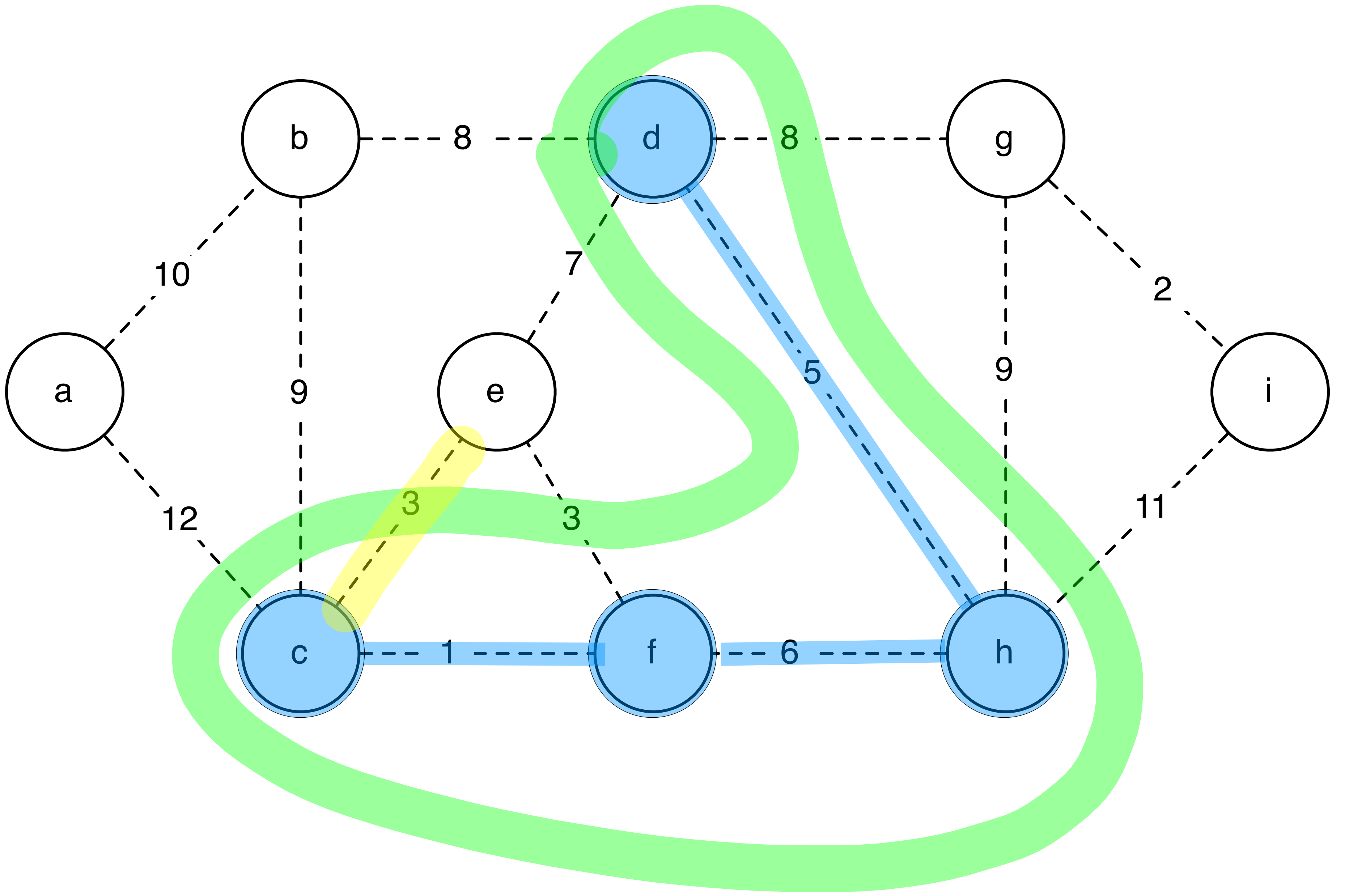
prim



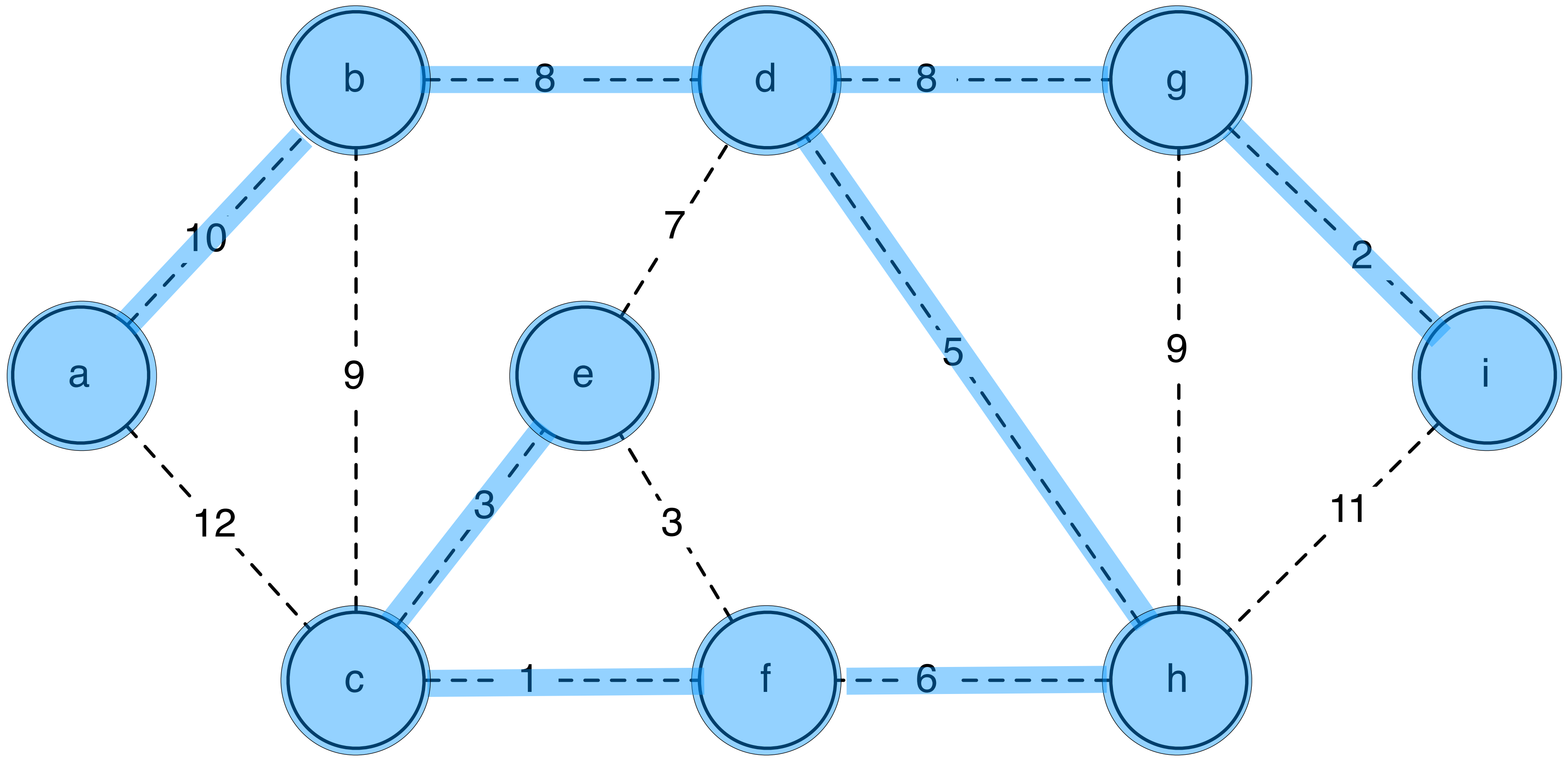
prim



prim



prim



implementation

idea: ~~At~~ each step, we need to identify the "lightest edge" which augments our tree —

— use priority queue

implementation

new data structure

<u>Priority queue</u> –

- make($q_1 \ldots q_n$) & creates a queue w/ these n elements

- <u>extractmin</u> – produces smallest element in queue

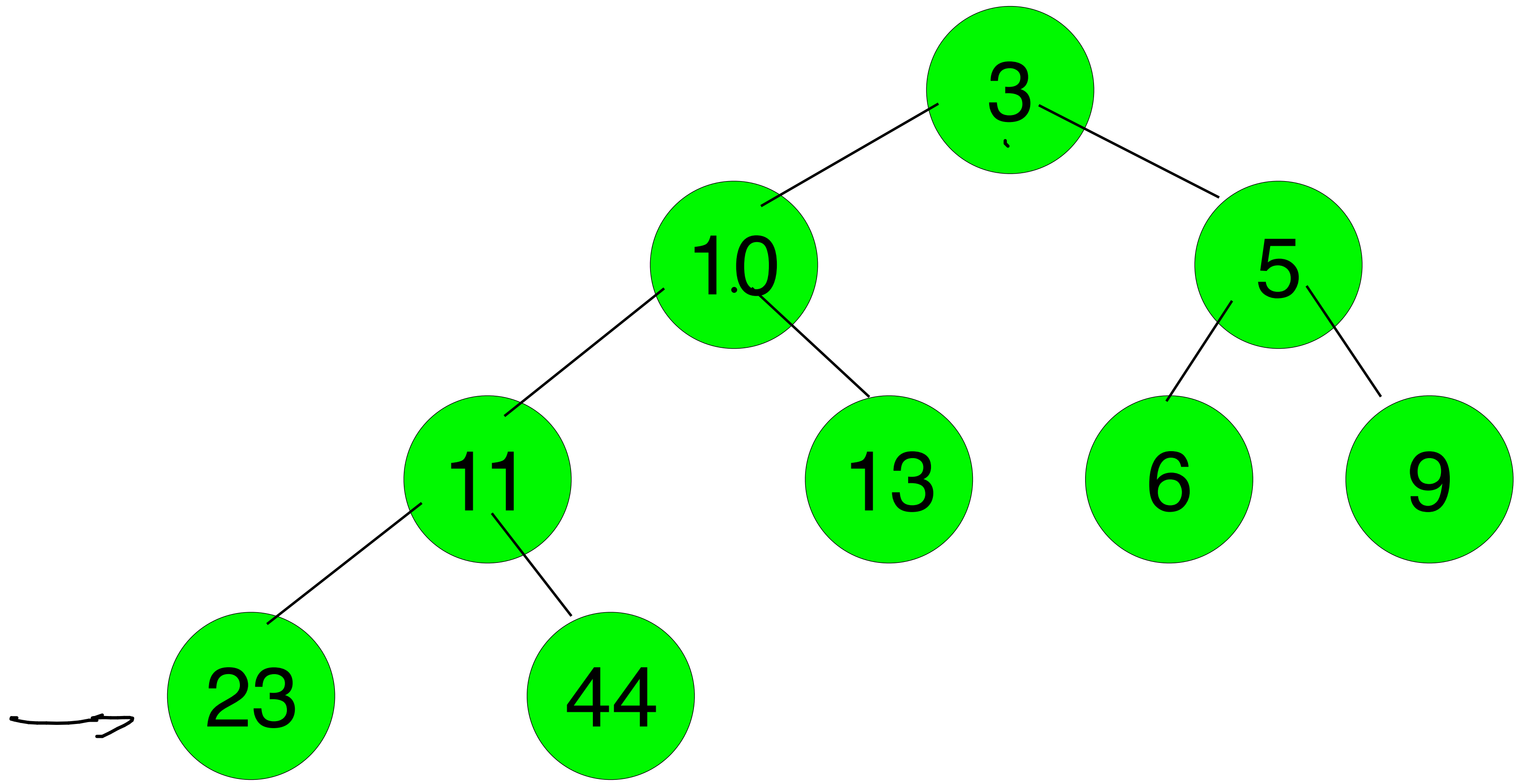
- <u>decrease key</u> – reduces the key value for some item.

binary heap

full tree, key value <= to key of children

binary heap

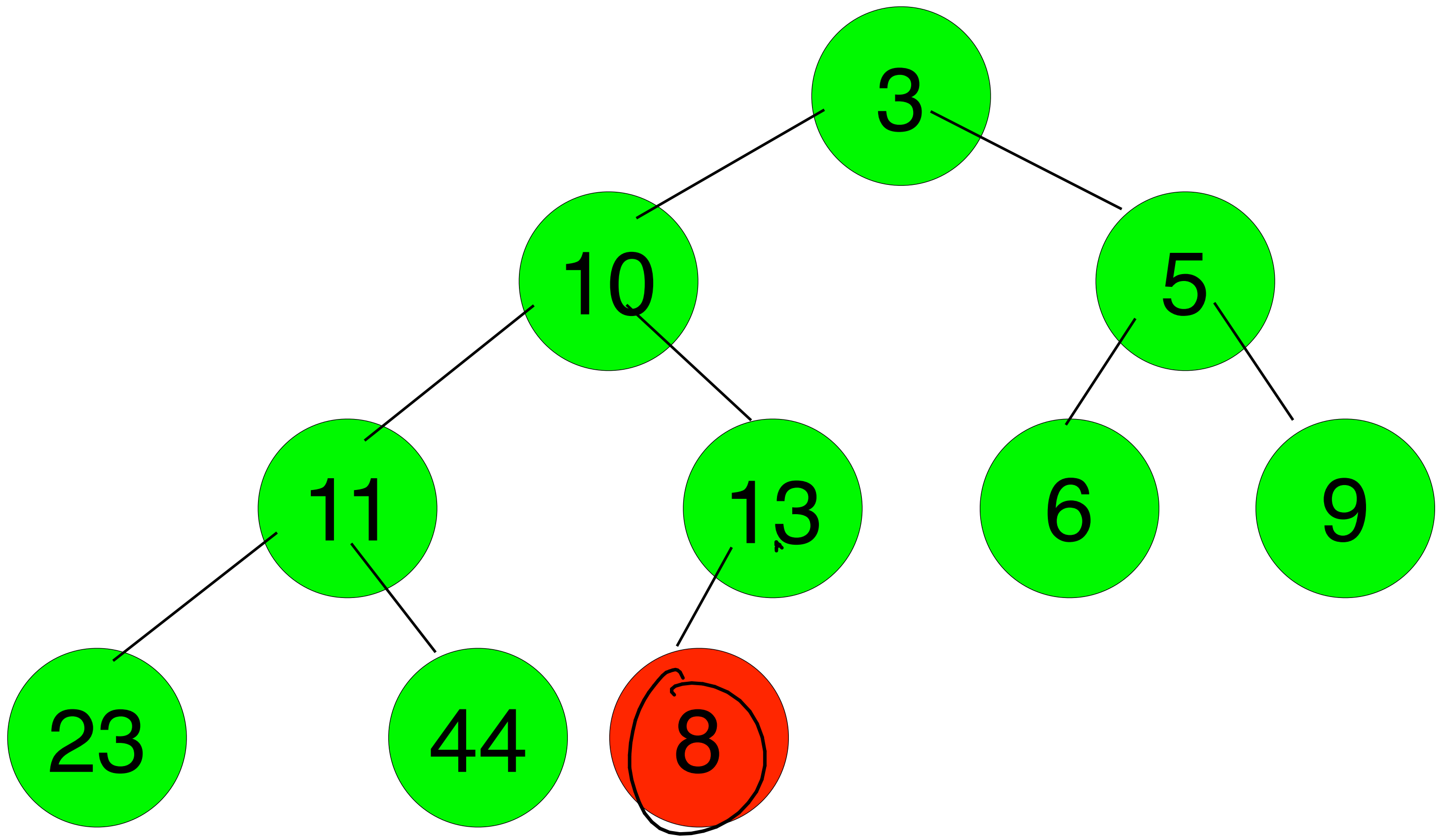
full tree, key value <= to key of children



binary heap

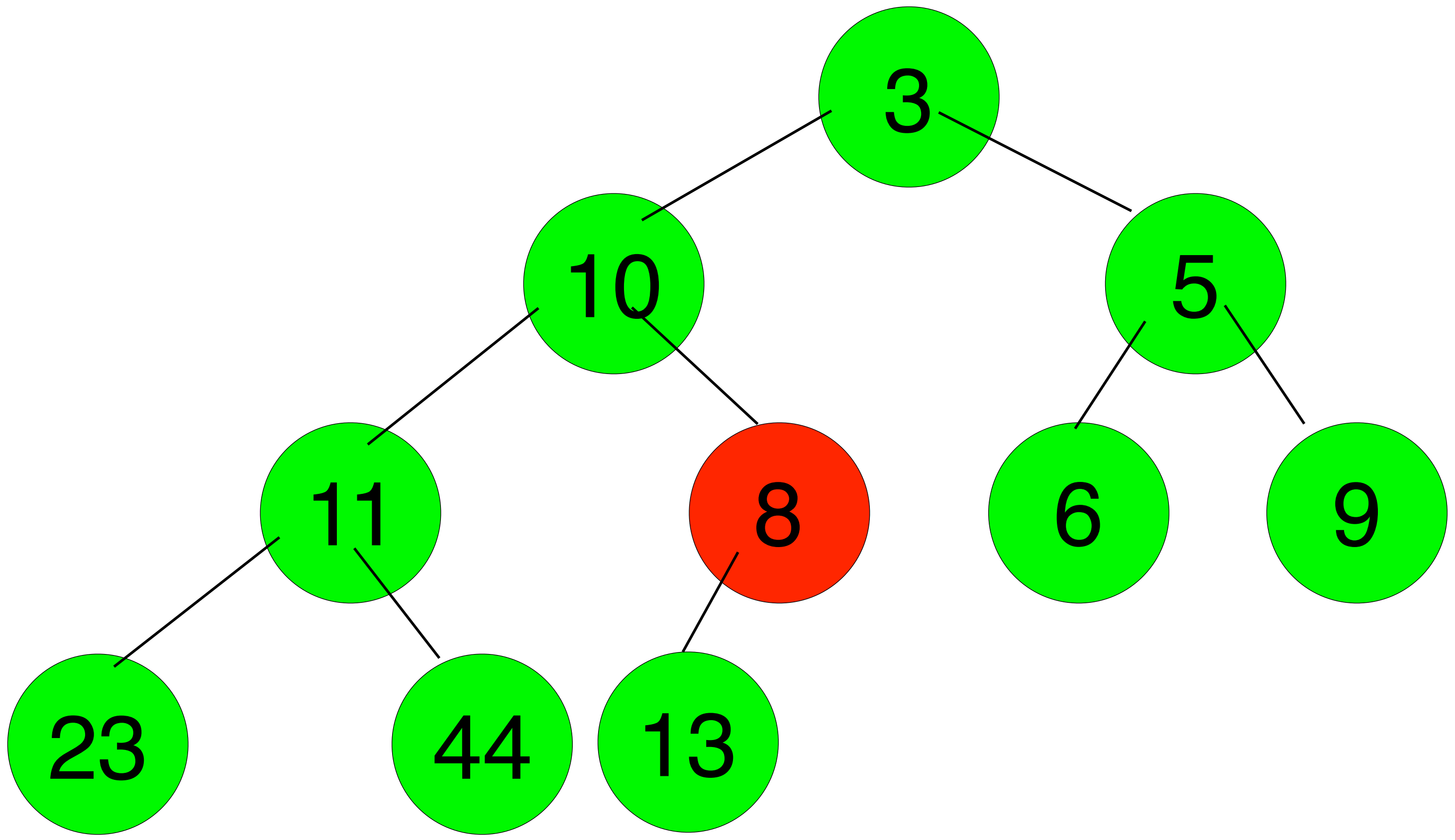
full tree, key value <= to key of children

insert(8)



binary heap

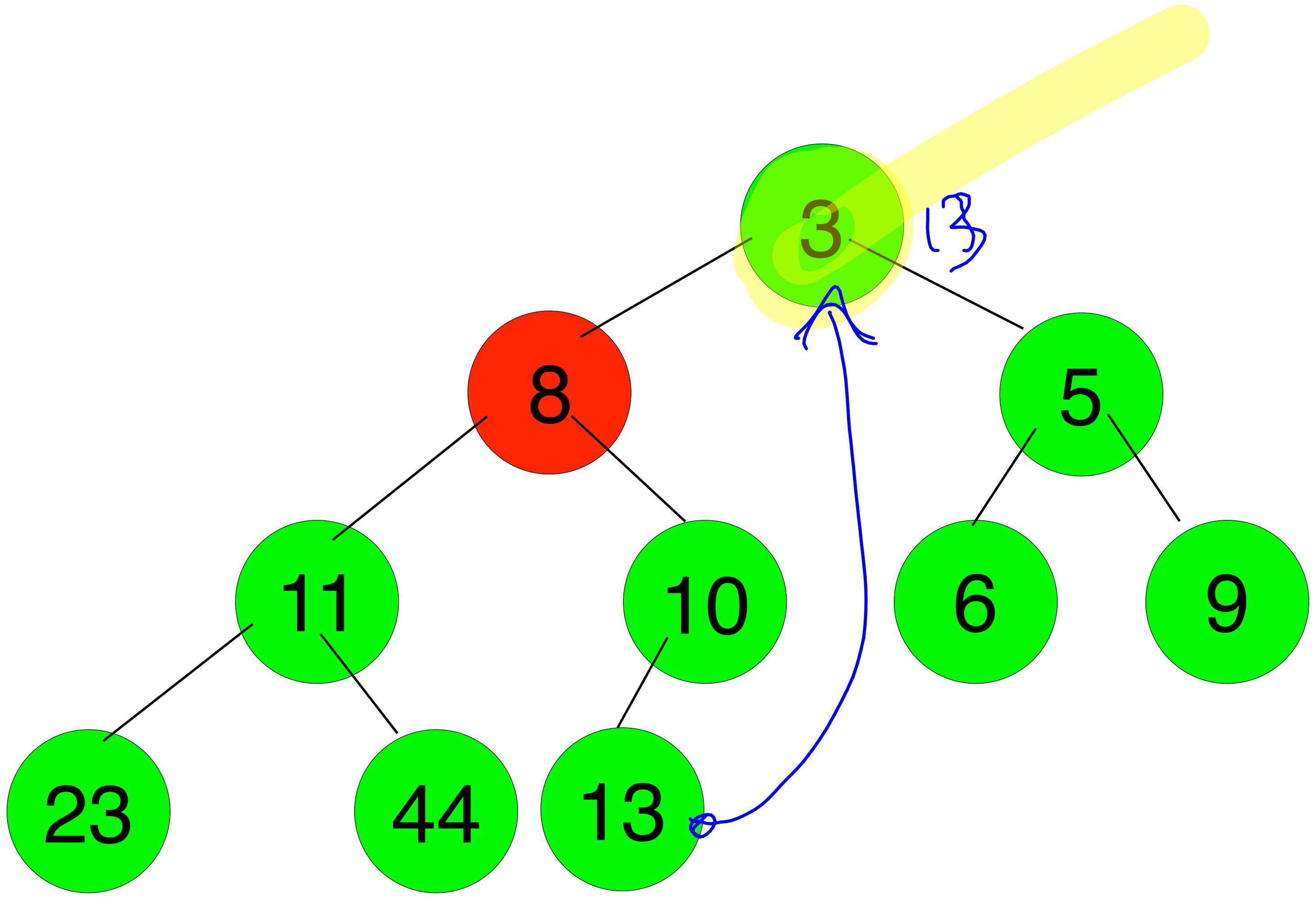
full tree, key value <= to key of children



binary heap

full tree, key value <= to key of children

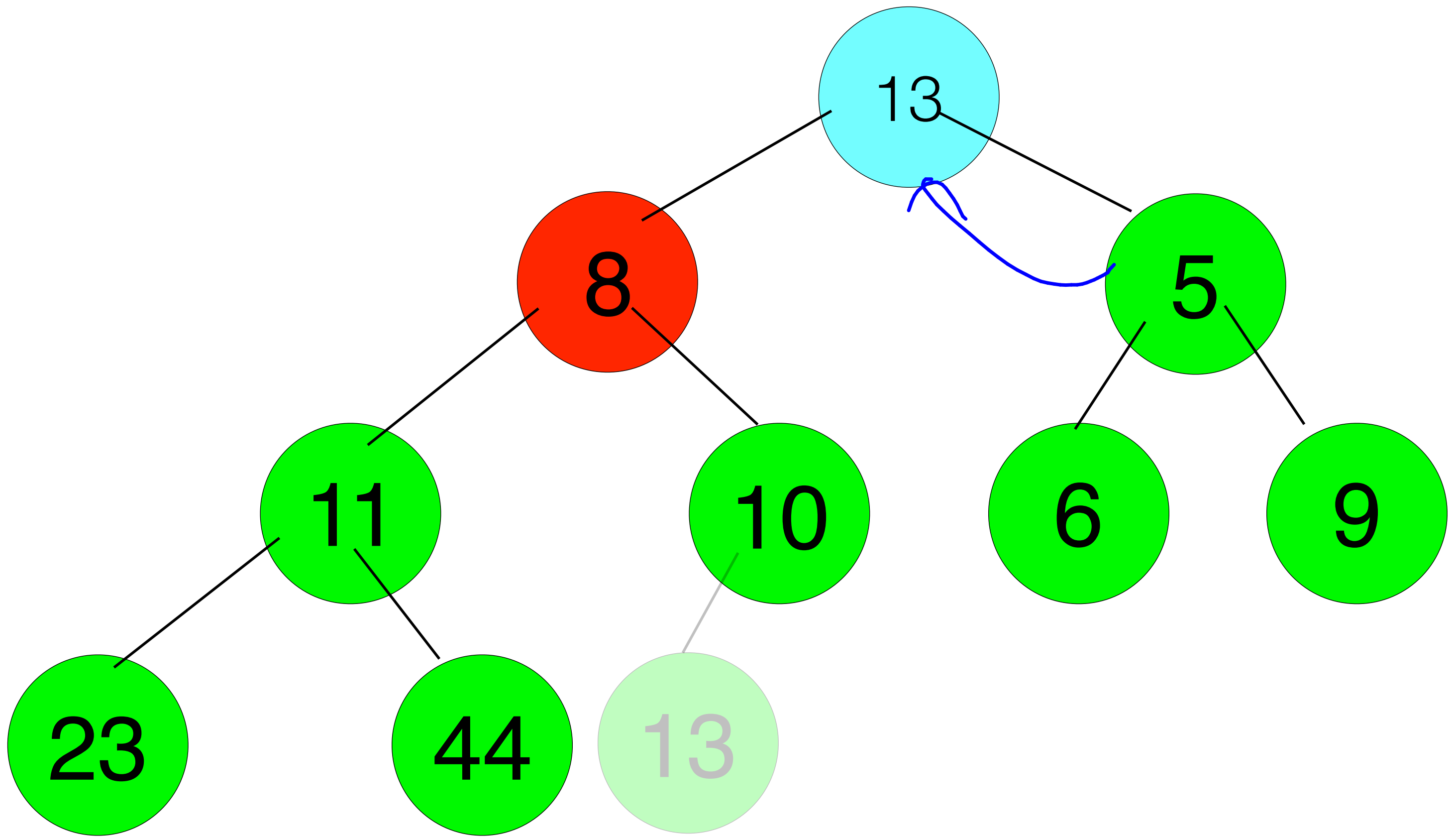
how to extractmin?



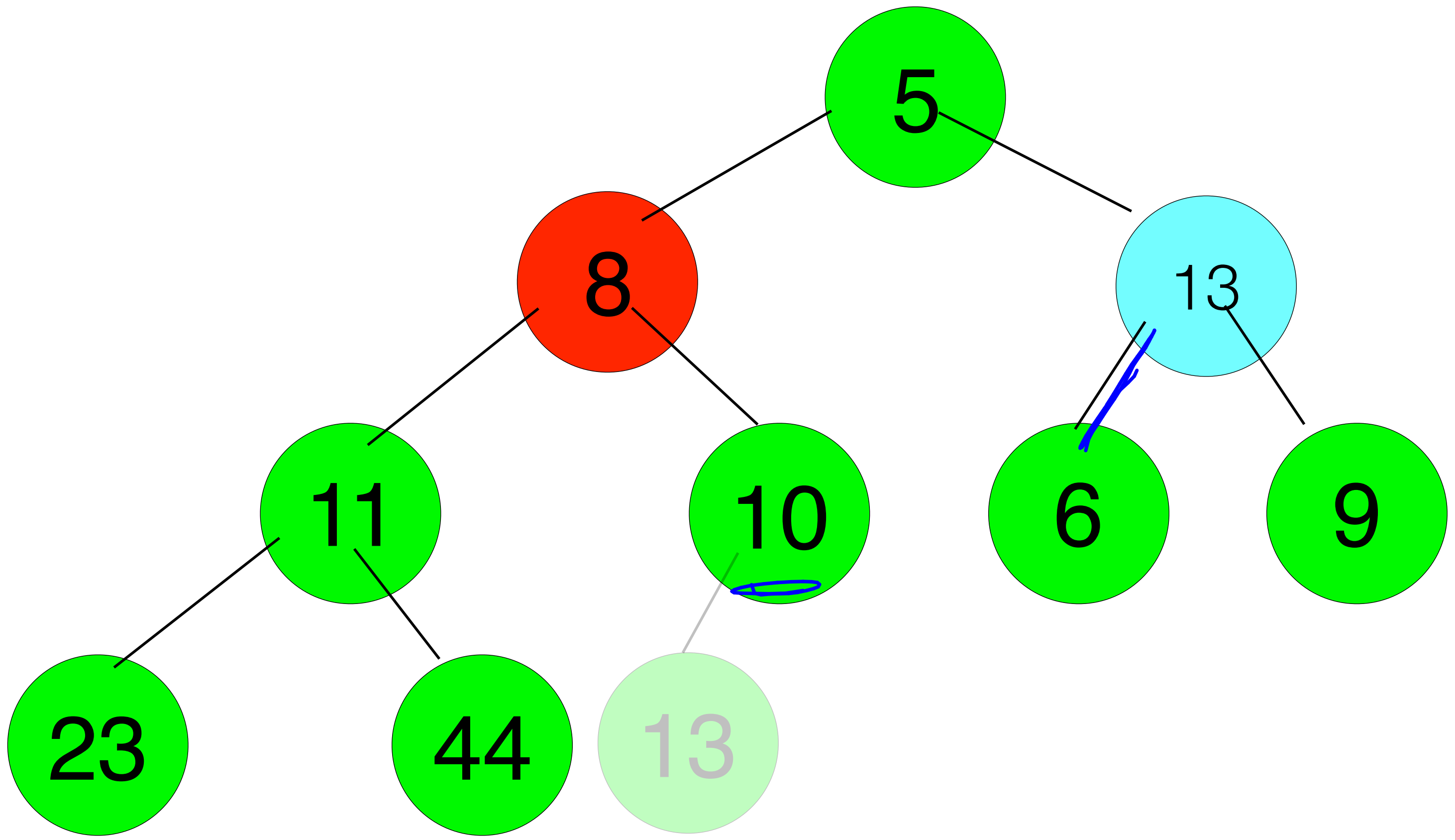
binary heap

full tree, key value <= to key of children

how to extractmin?



binary heap

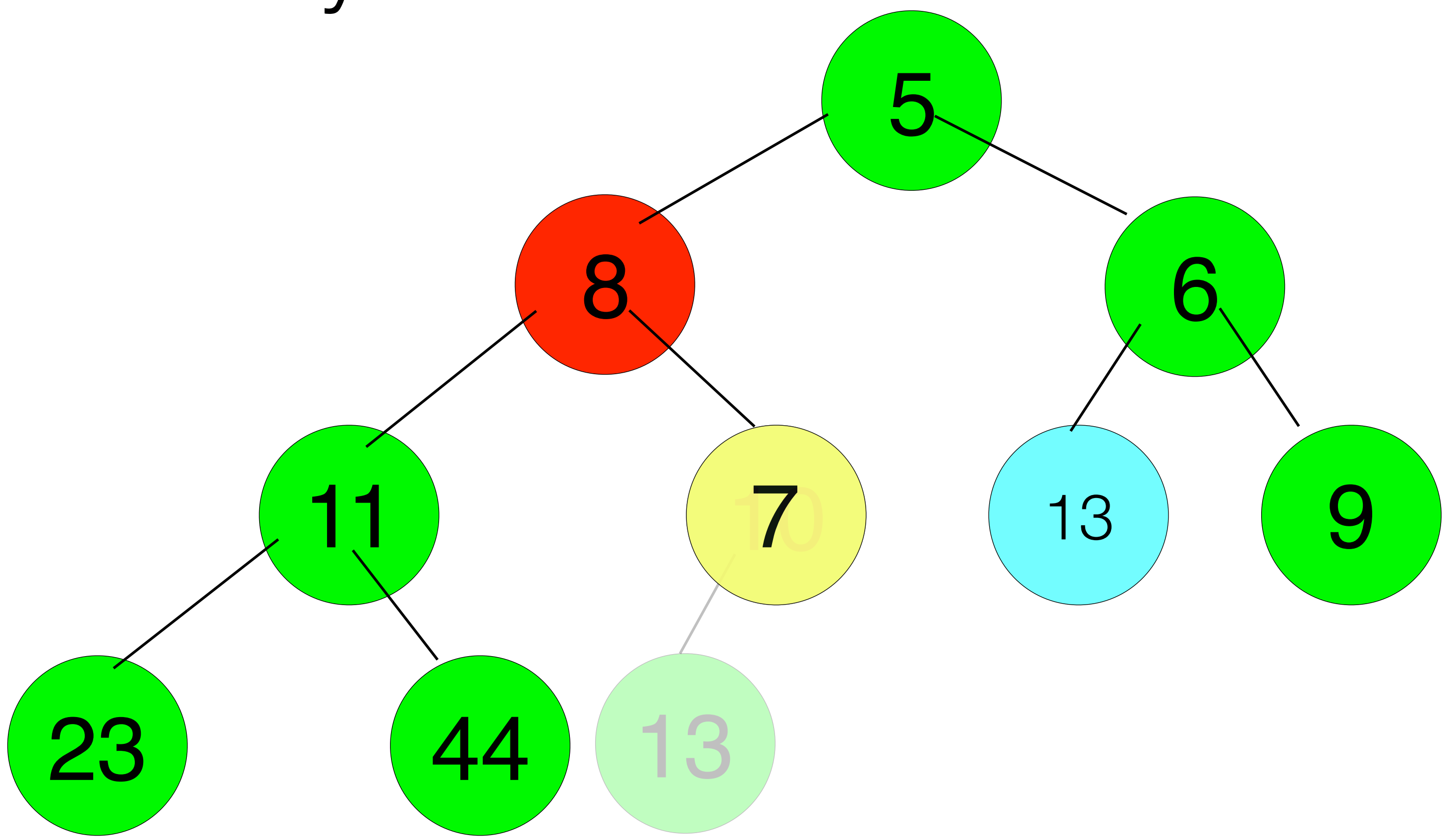


binary heap

full tree, key value <= to key of children

how to extractmin? → $\Theta(\log n)$

how to decreasekey?

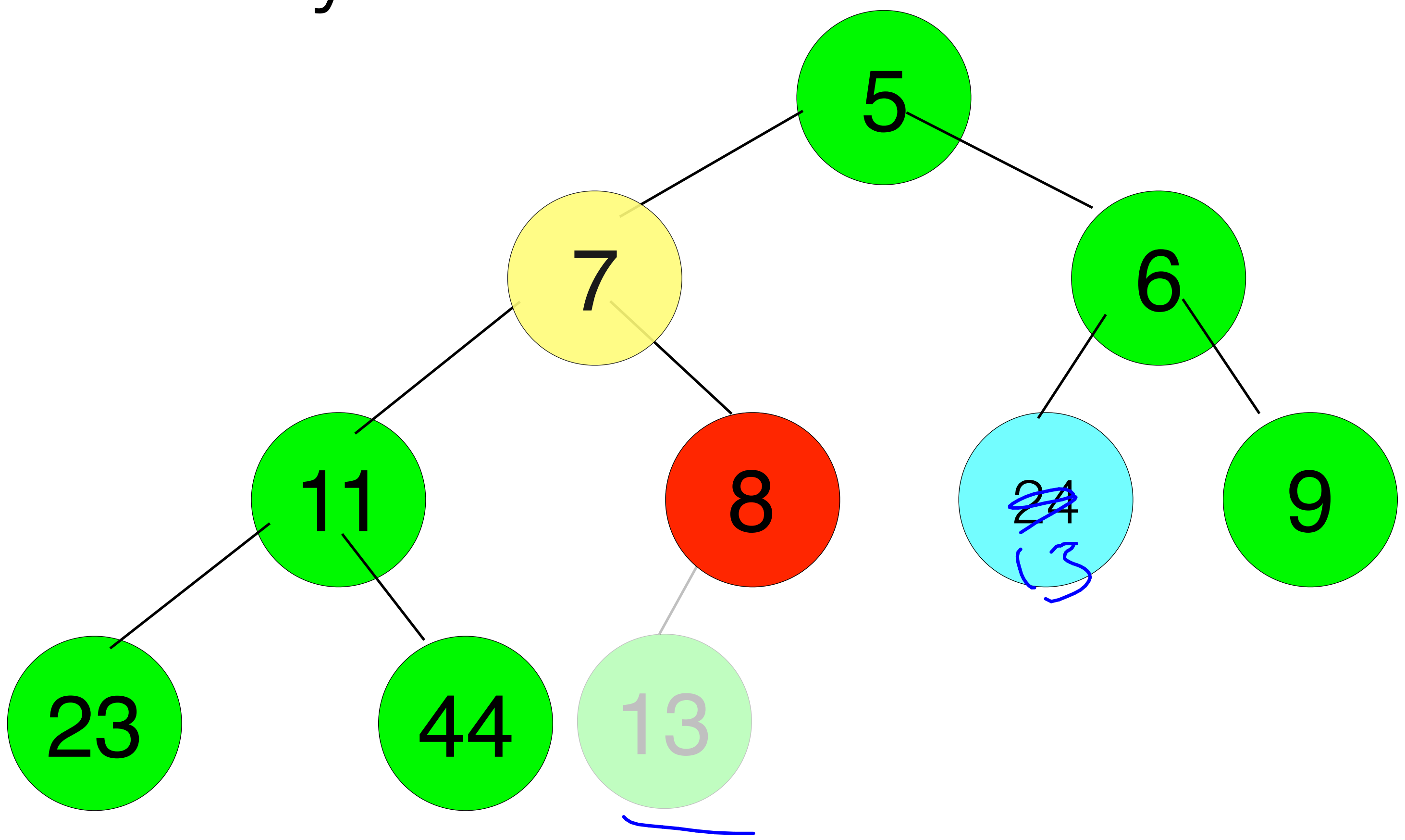


binary heap

full tree, key value <= to key of children

how to extractmin?

how to decreasekey? O(log n)



implementation

use a priority queue to keep track of light edges

insert: →

makequeue: → $O(n)$

extractmin: →

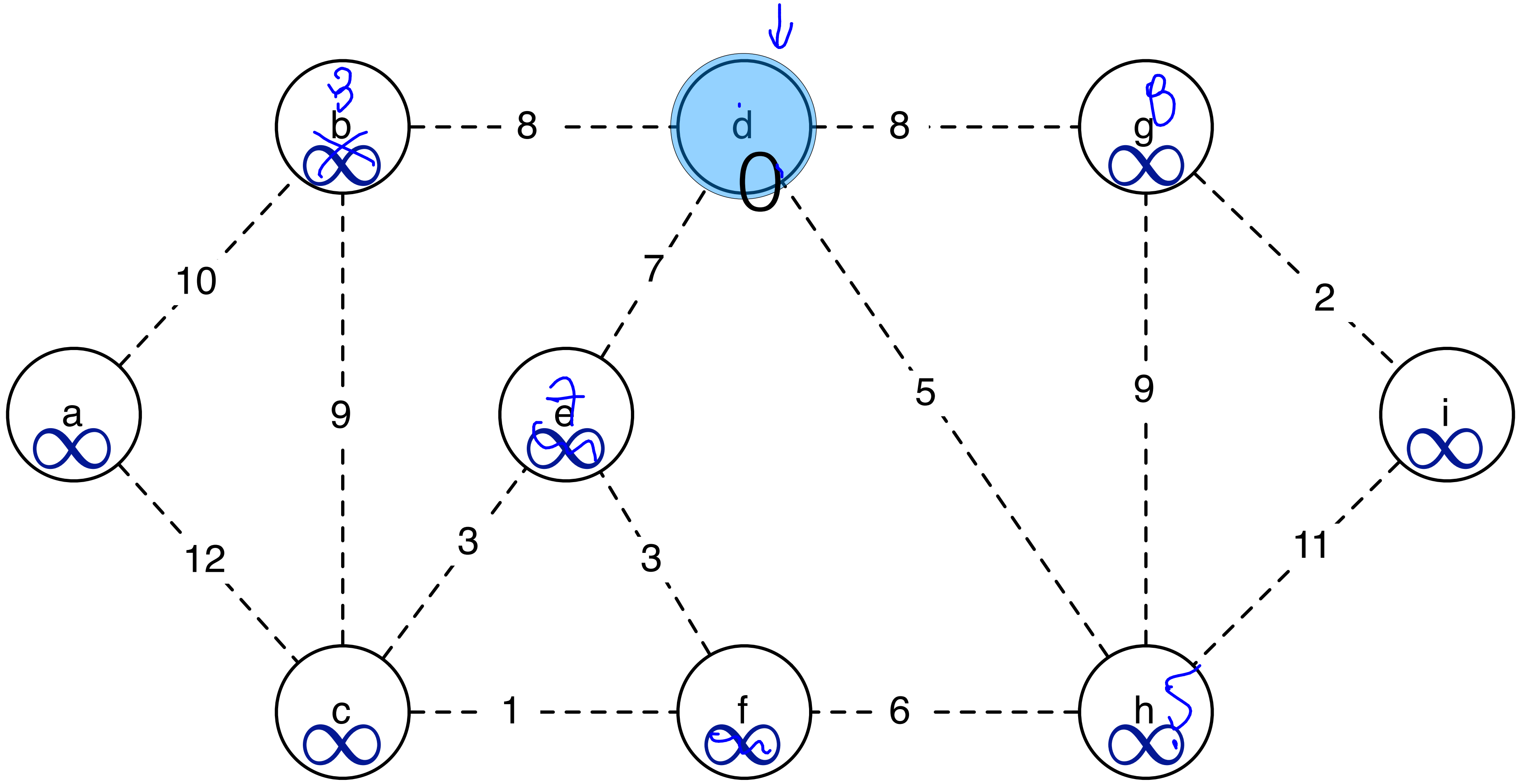
decreasekey: → $O(\log n)$

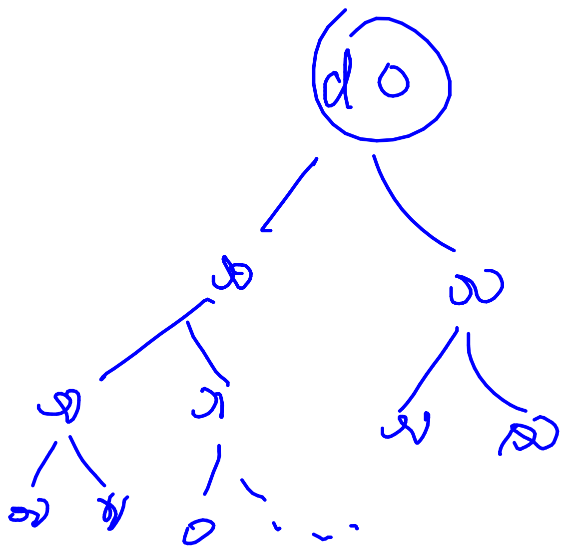
Prim's algorithm

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)
```

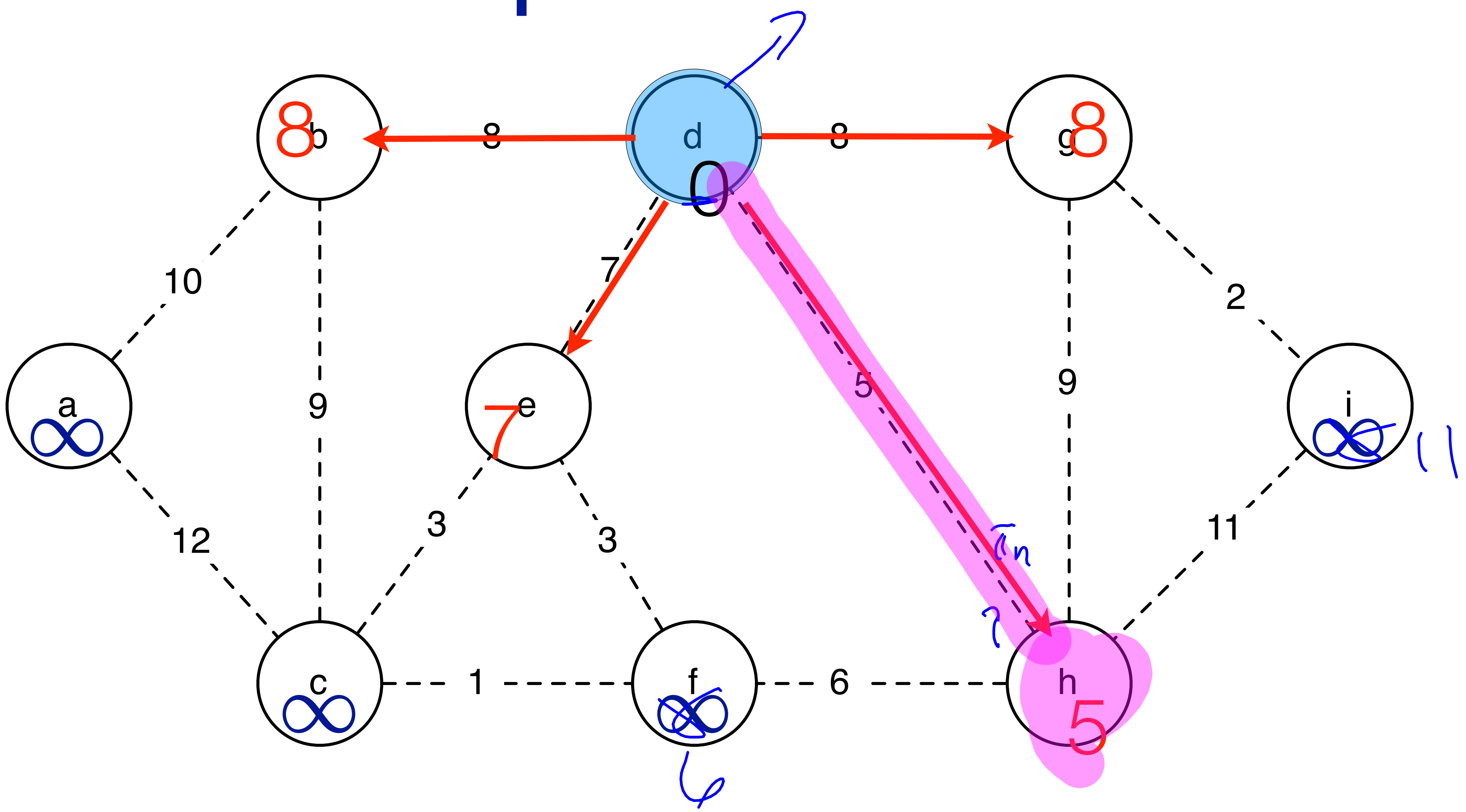
prim





```
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)
```

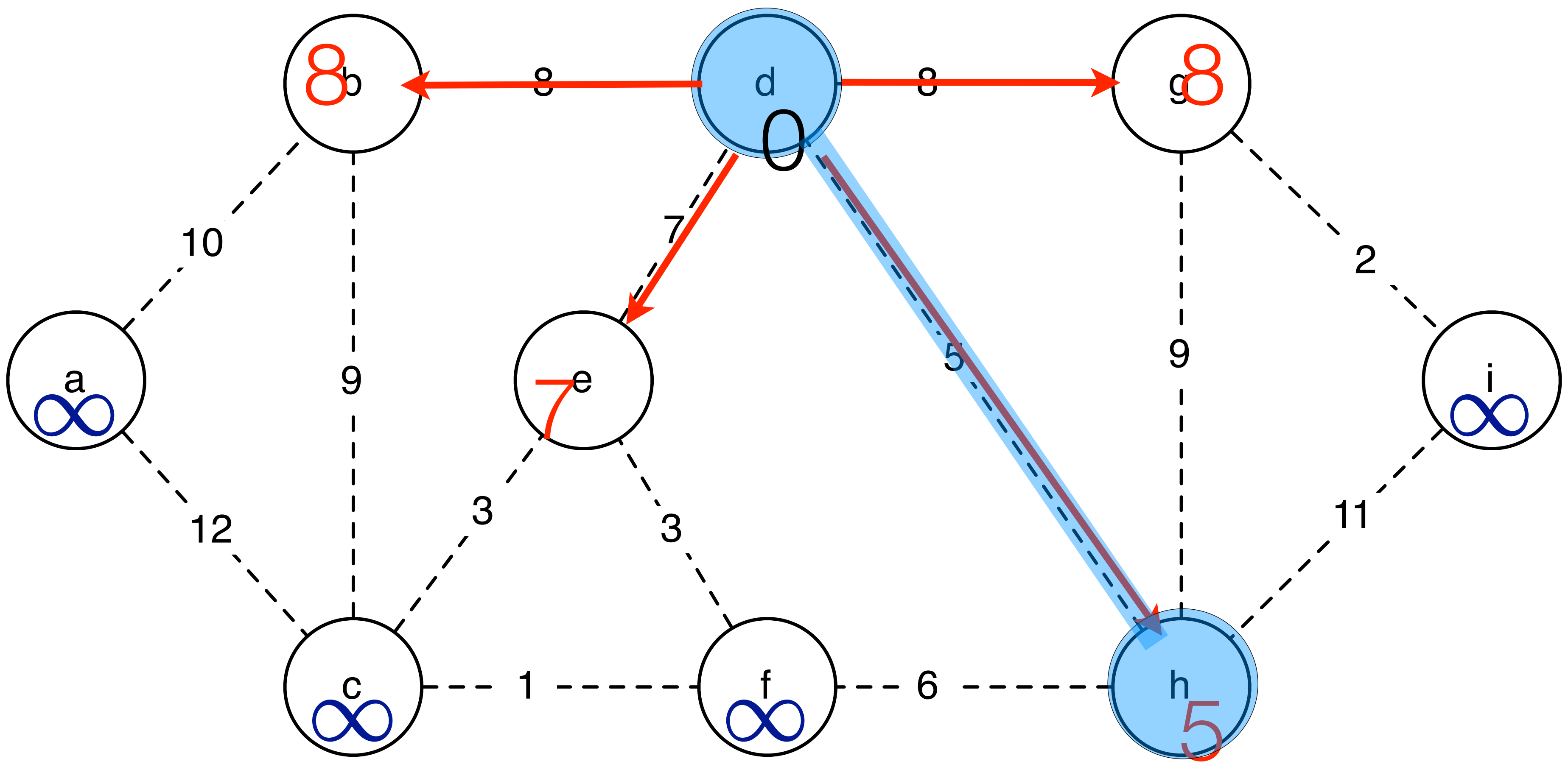
prim



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

$Q \leftarrow \emptyset$ ▷ 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 $k_v \leftarrow w(u, v)$

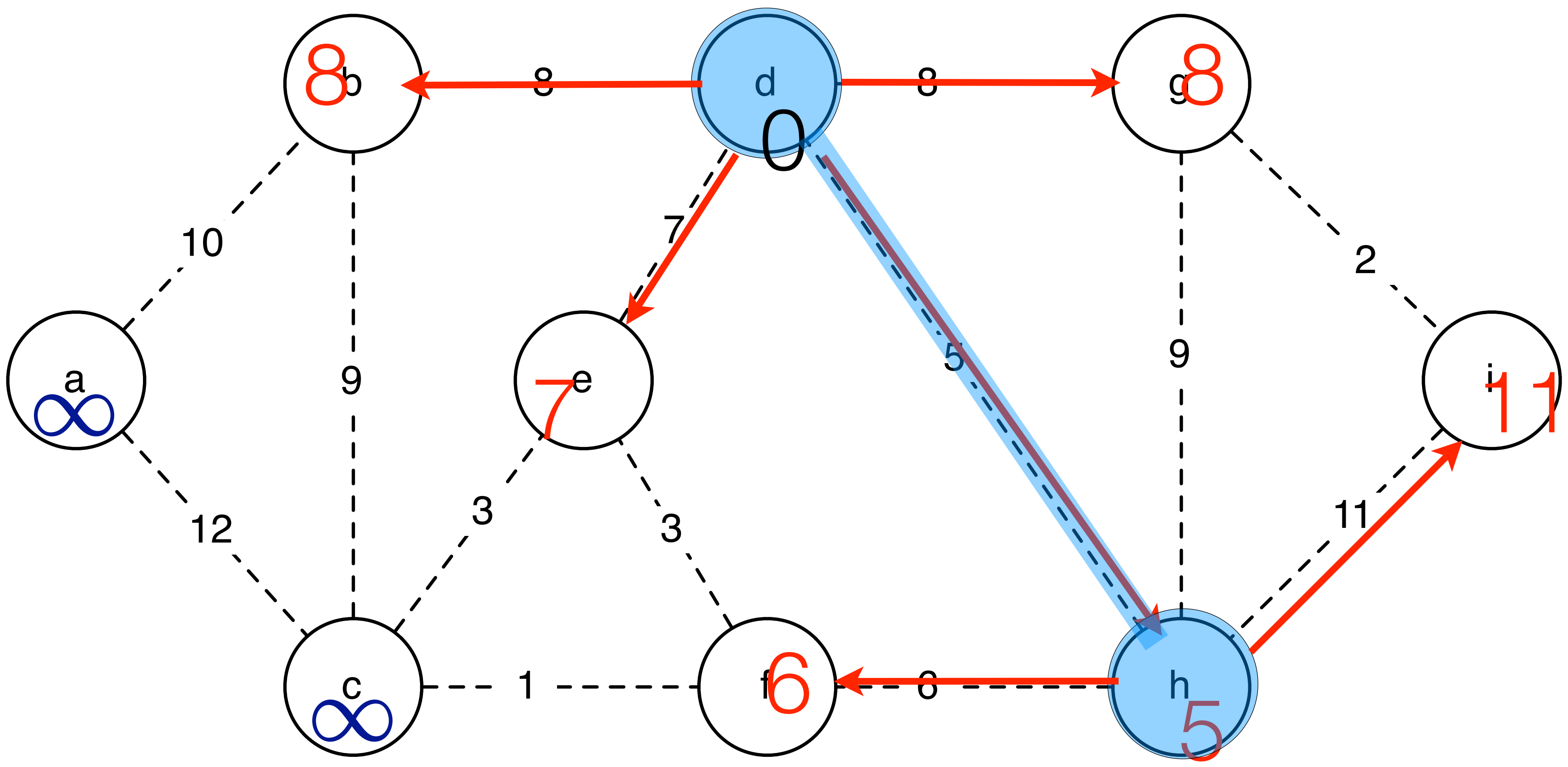
prim



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

$Q \leftarrow \emptyset$ ▷ 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$(G = (V, E))$

$Q \leftarrow \emptyset$ $\quad \triangleright$ Q is a Priority Queue

Initialize each $v \in V$ with key $k_v \leftarrow \infty$, $\pi_v \leftarrow$ NIL

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

Insert all nodes into Q with key k_v.

while $Q \neq \emptyset$

  do $u \leftarrow$ EXTRACT-MIN(Q)

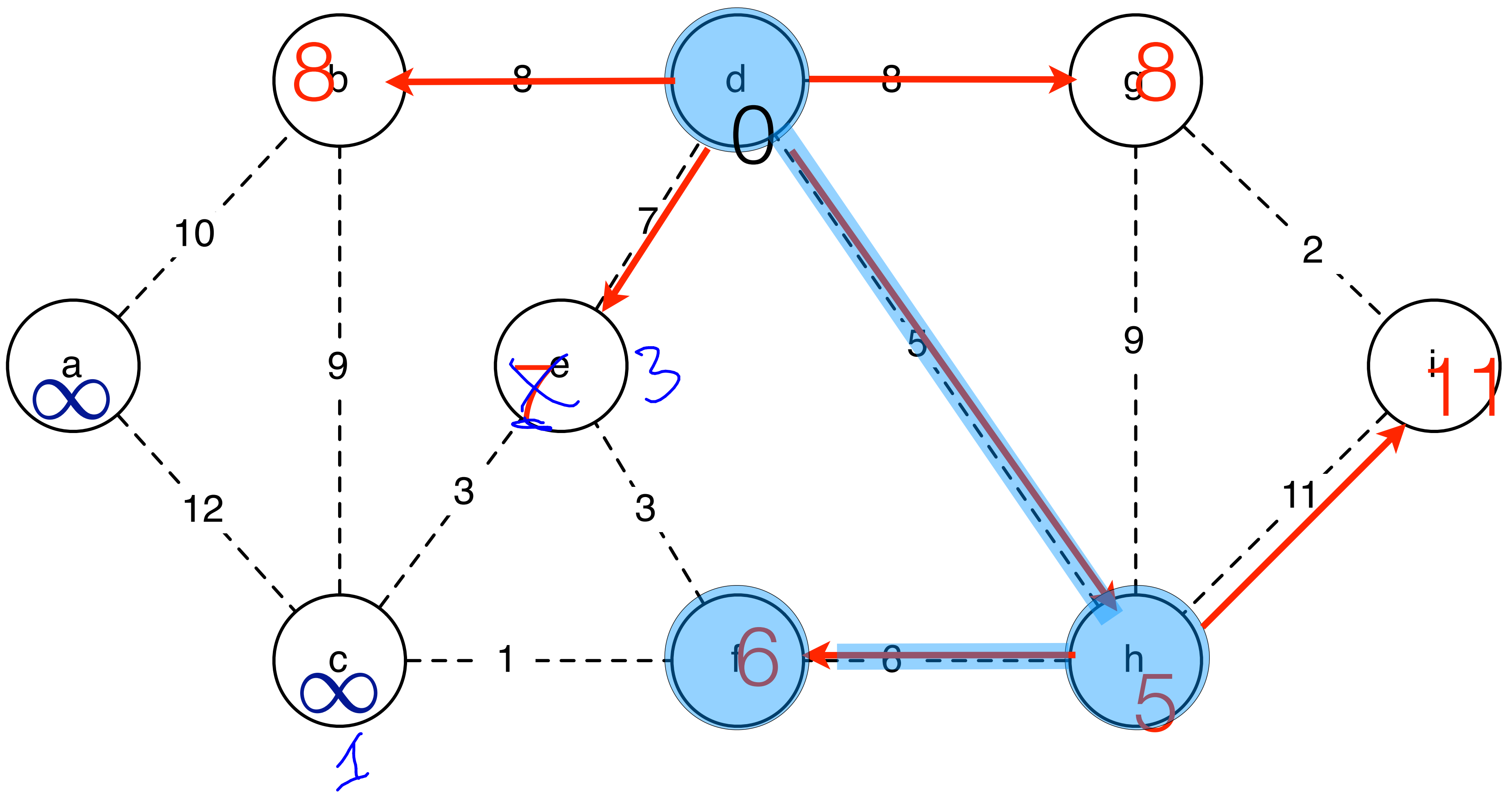
   for each $v \in Adj(u)$

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

     then $\pi_v \leftarrow u$

      DECREASE-KEY$(Q, v, w(u, v))$ $\quad \triangleright$ Sets $k_v \leftarrow w(u, v)$

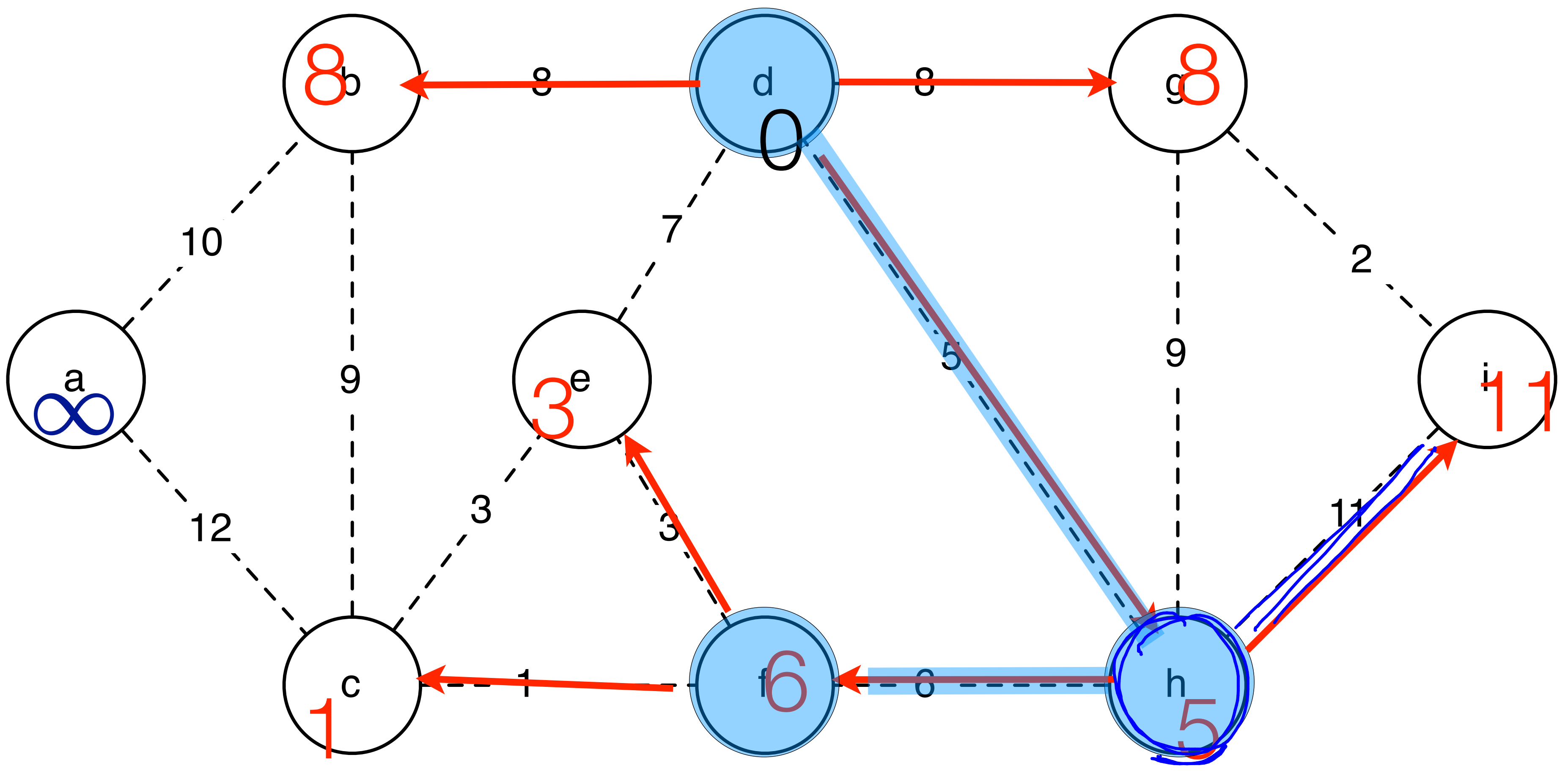
prim



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)
```

running time

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 $k_v \leftarrow w(u, v)$

make queue: $\Theta(V)$

$\Theta(V \log V)$ time

$\Theta(E \cdot \log V)$

$\log(V)$ time

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

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)
```

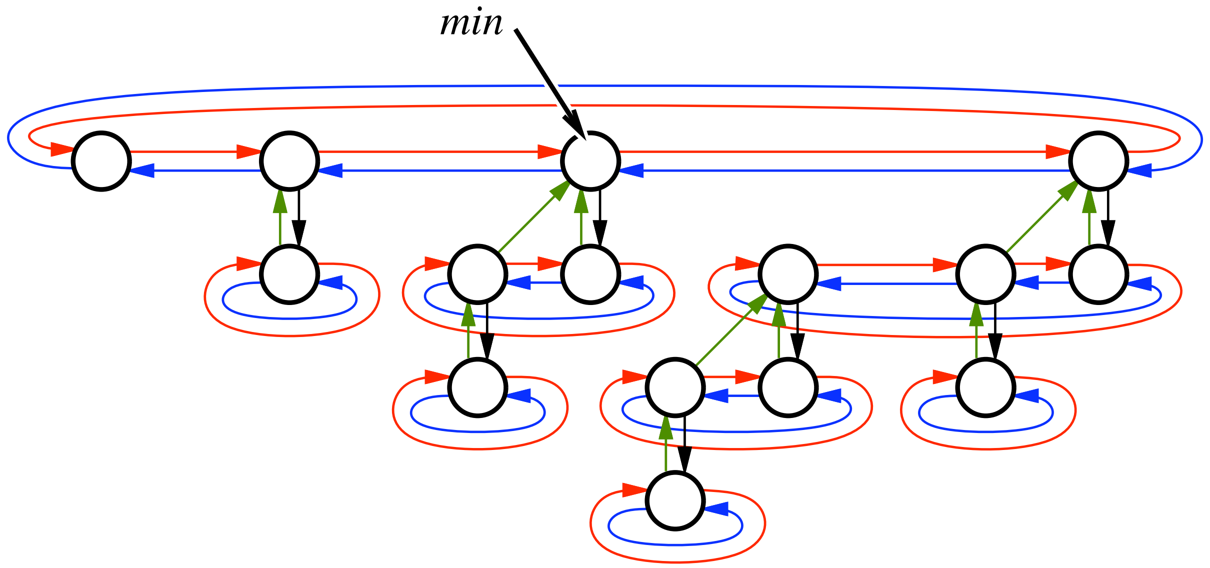
$$O(V \log V + E \log V) = O(E \log V)$$

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

fibonacci heap



faster implementation

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

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

$\Theta(1)$

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

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:	$V \log V$

Ackerman function

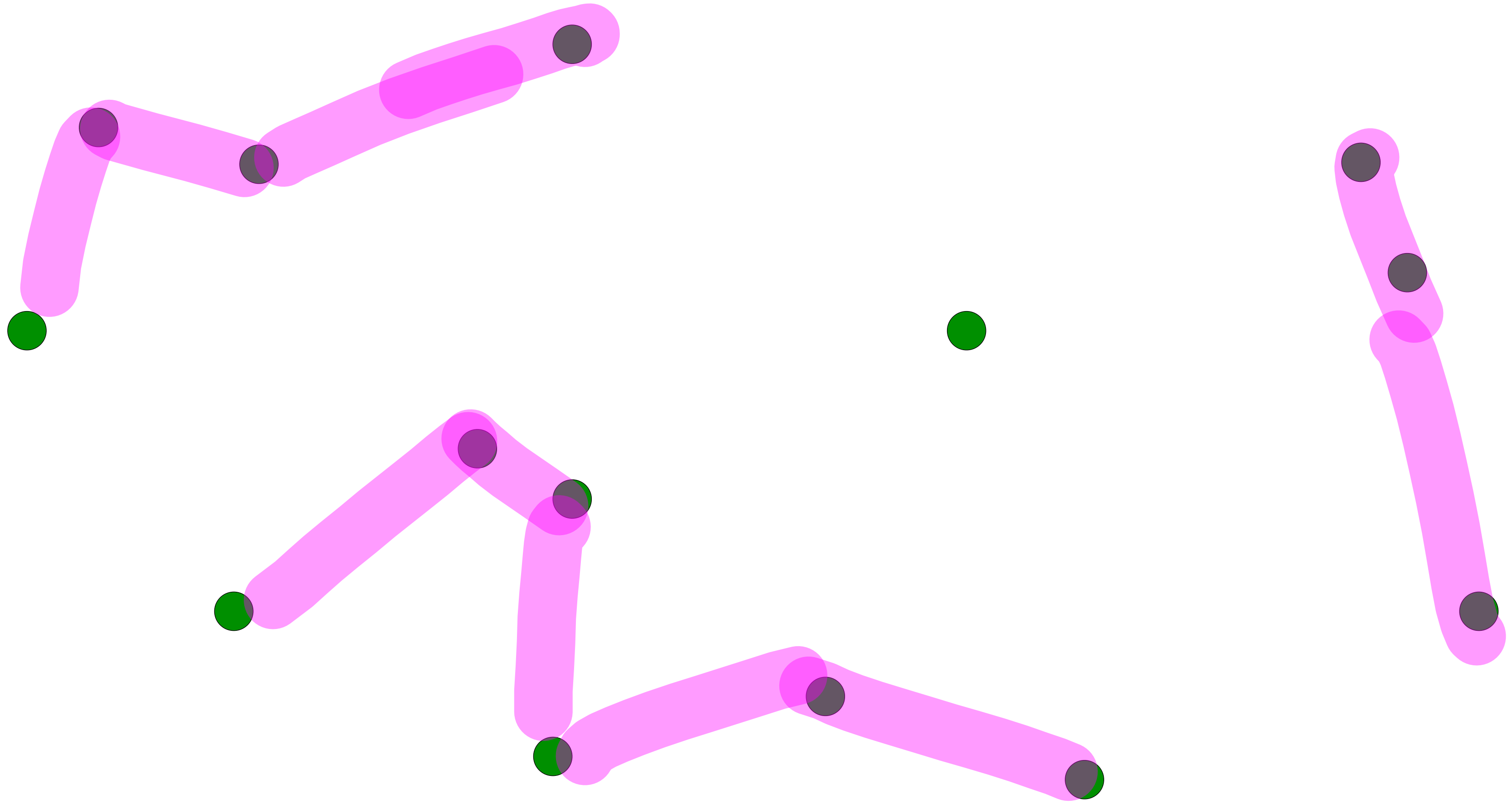
$$A(m,n) = \begin{cases} n+1 & m = 0 \\ A(m-1,1) & m > 0,\ n = 0 \\ A(m-1, A(m, n-1)) & m, n > 0 \end{cases}$$

$$A(4,2) =$$

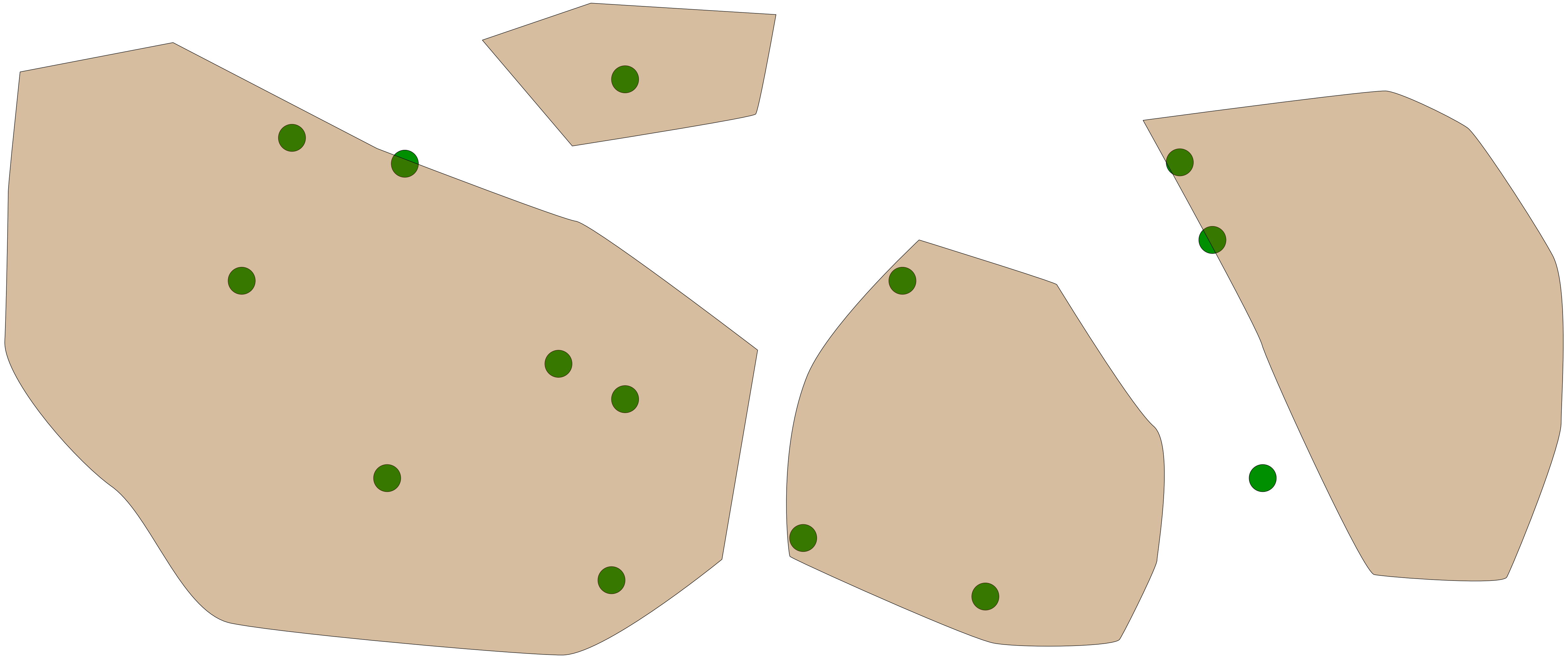
inverse ackerman

$\alpha(n) =$

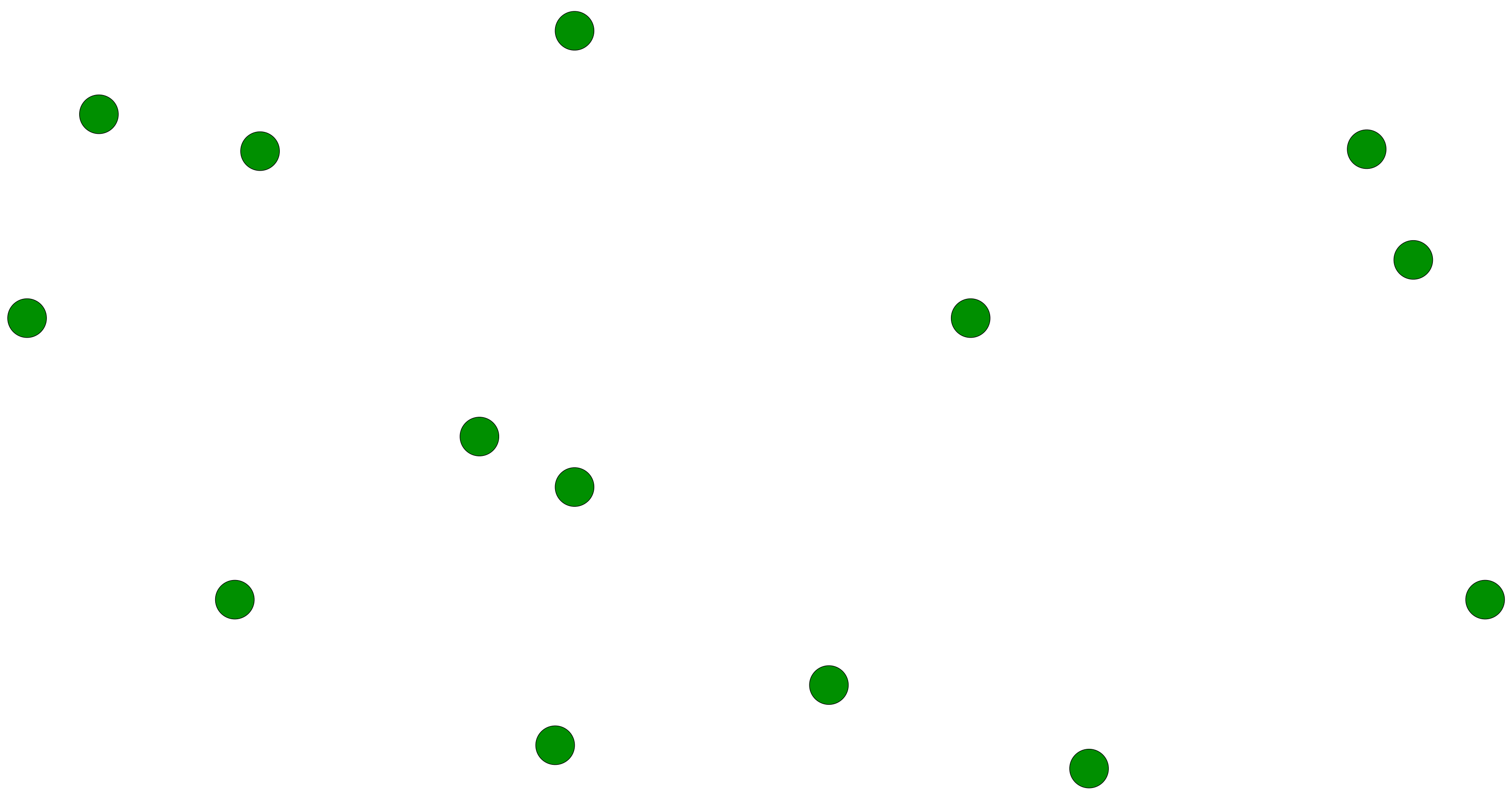
application of mst



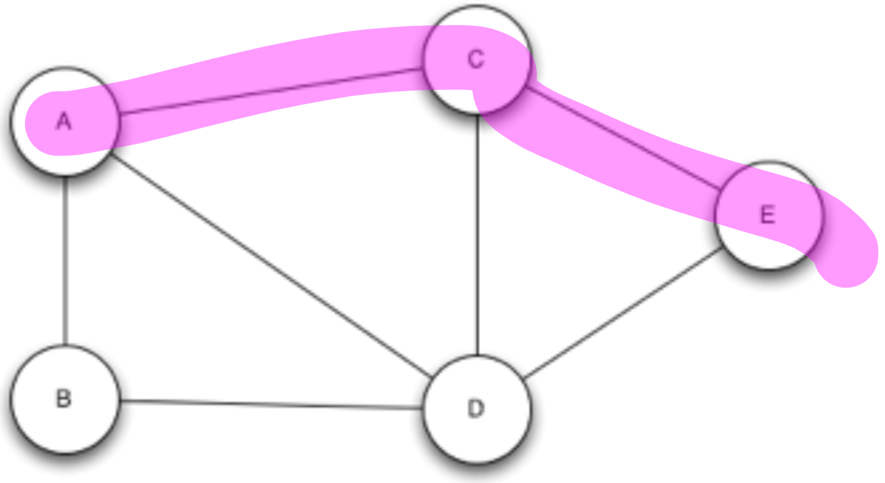
application of mst



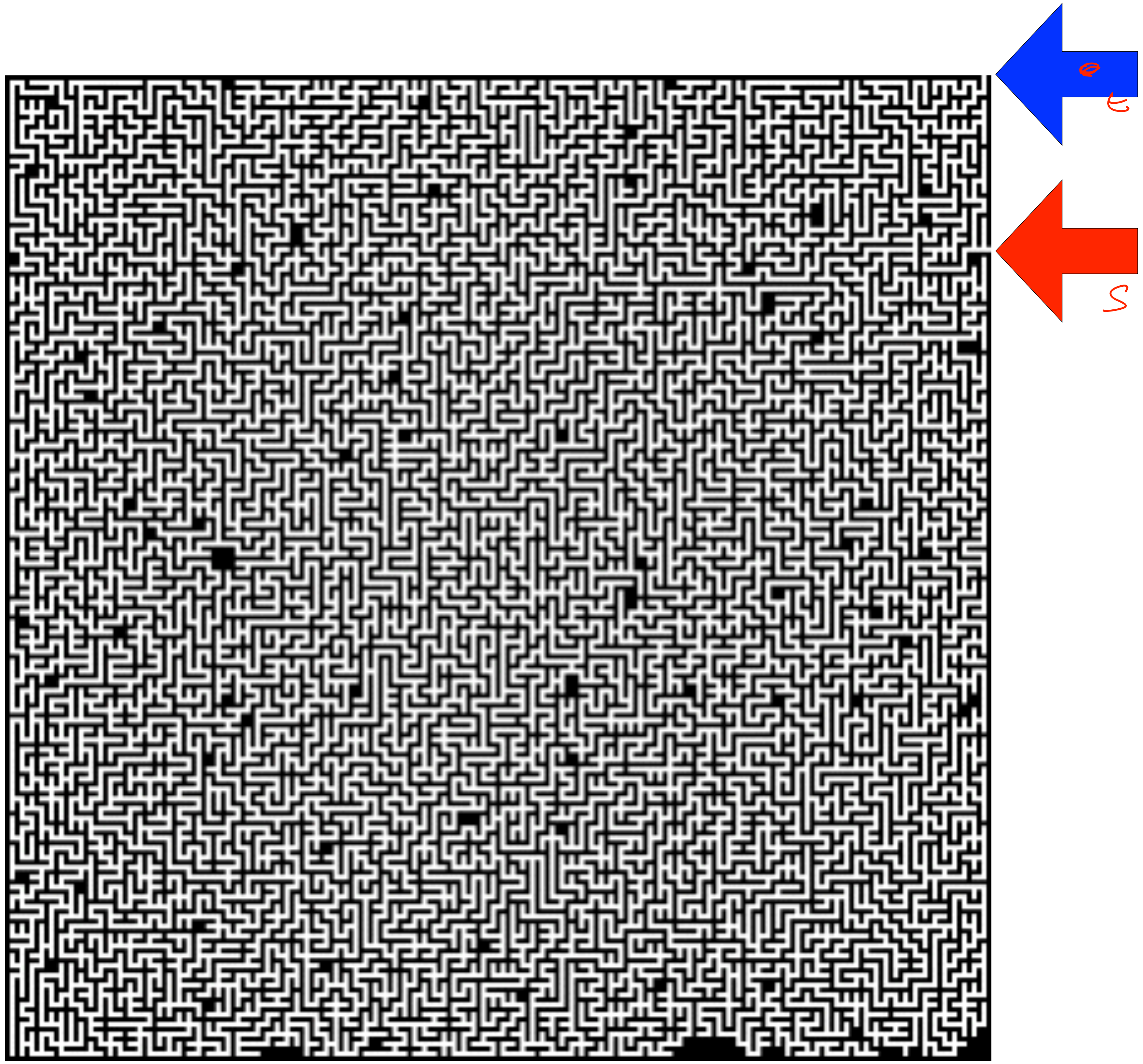
application of mst



simple graph questions



what is the length of the path from a to e?

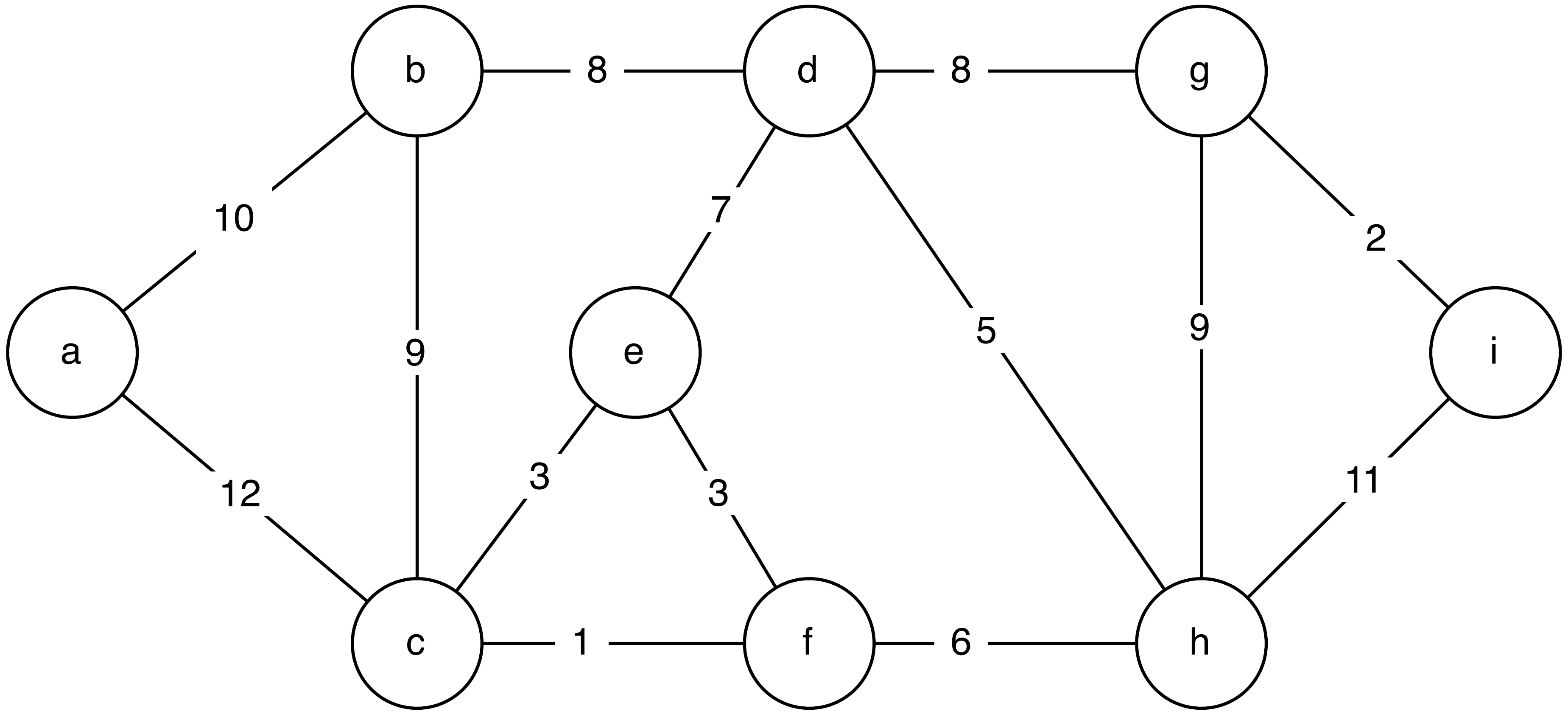

e
S

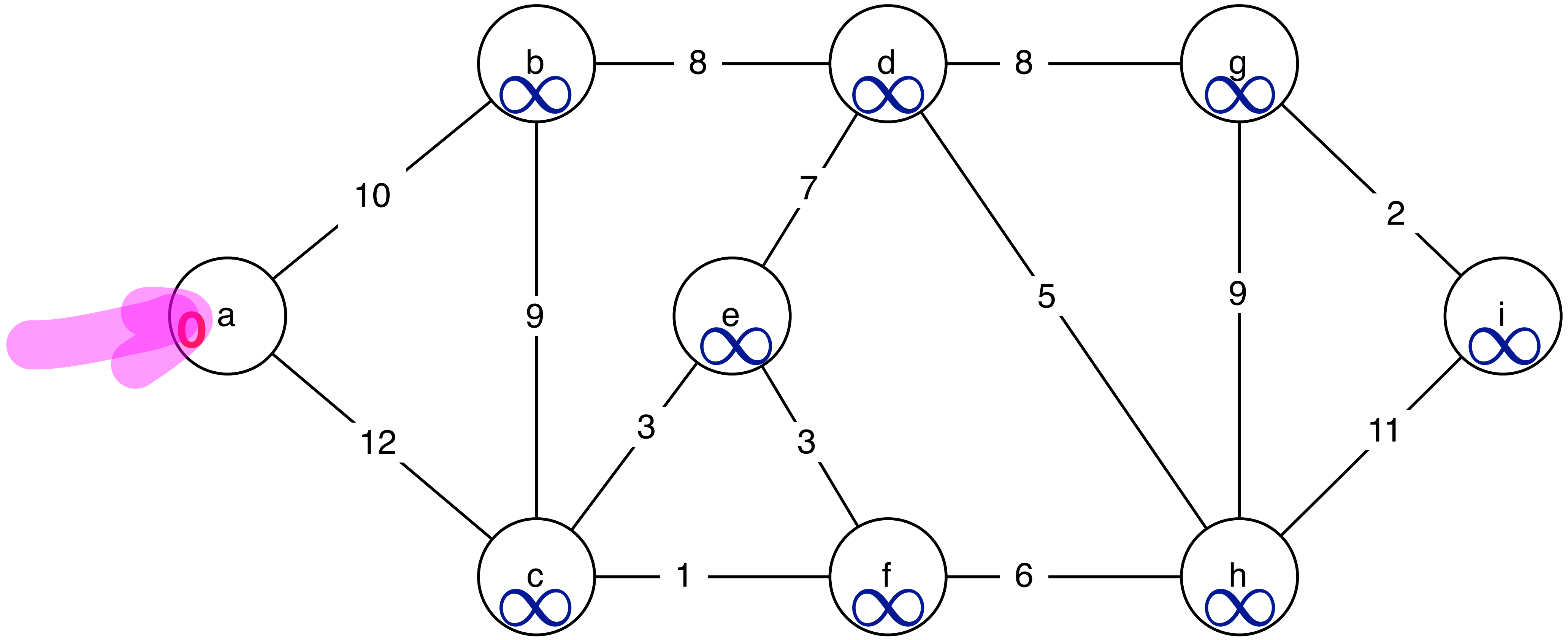
shortest path property

definition:

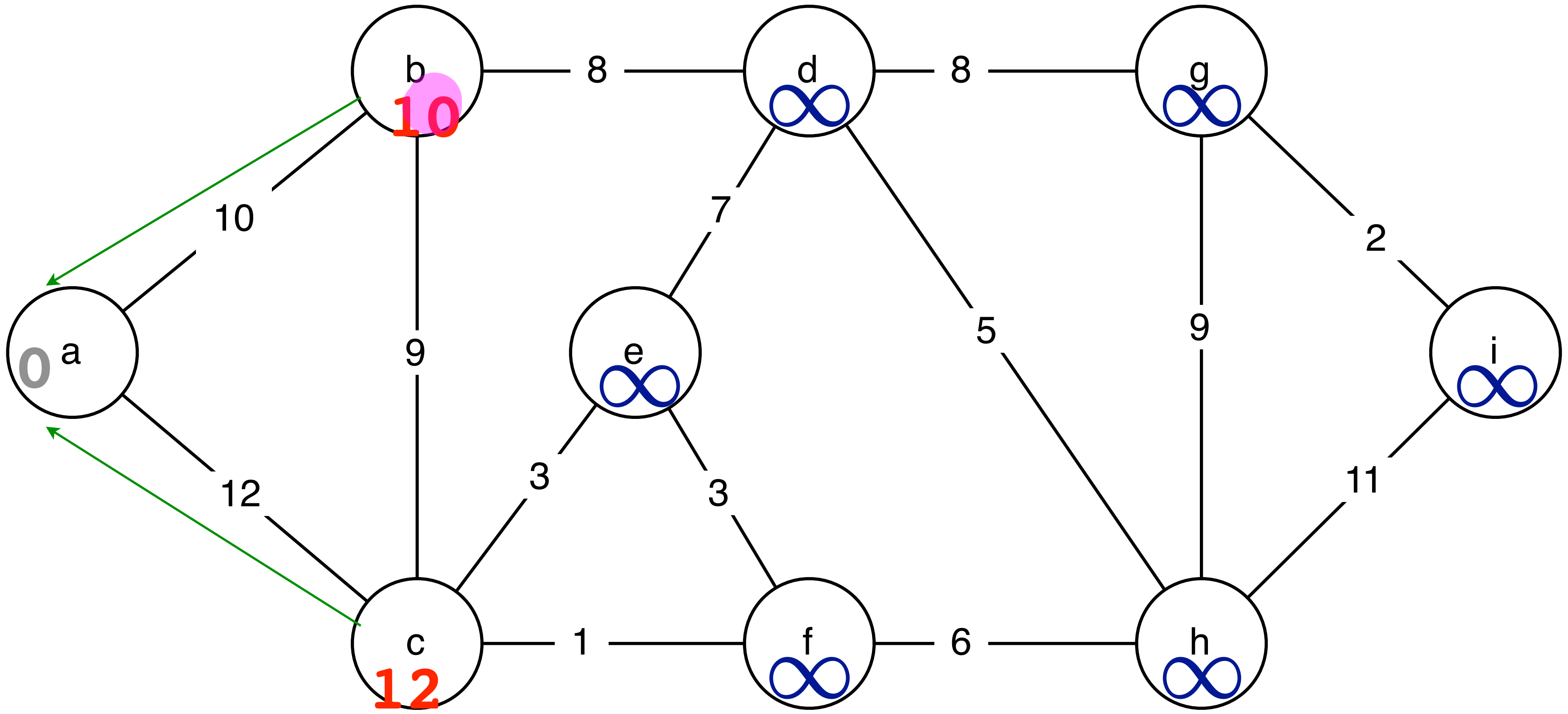
$$\delta(s, v)$$

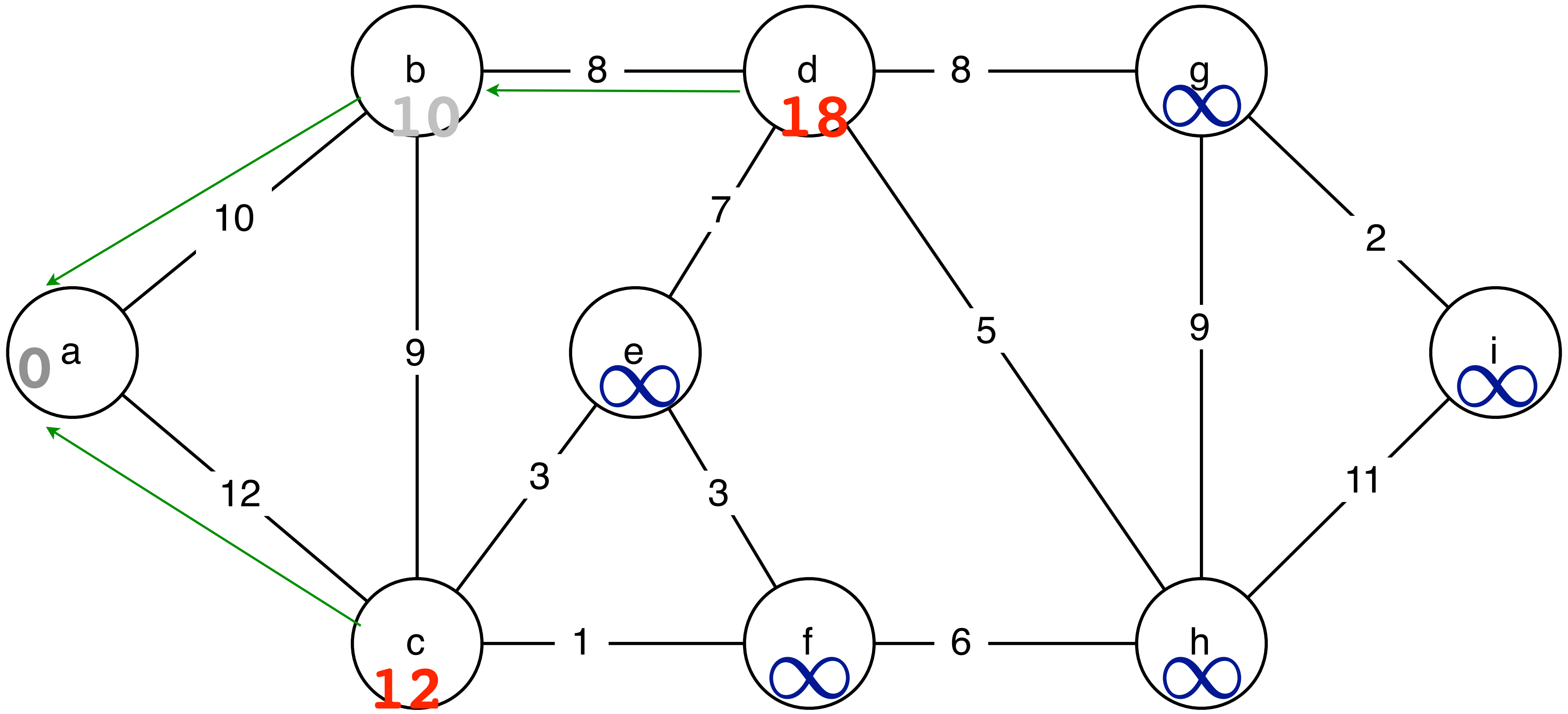
shortest paths

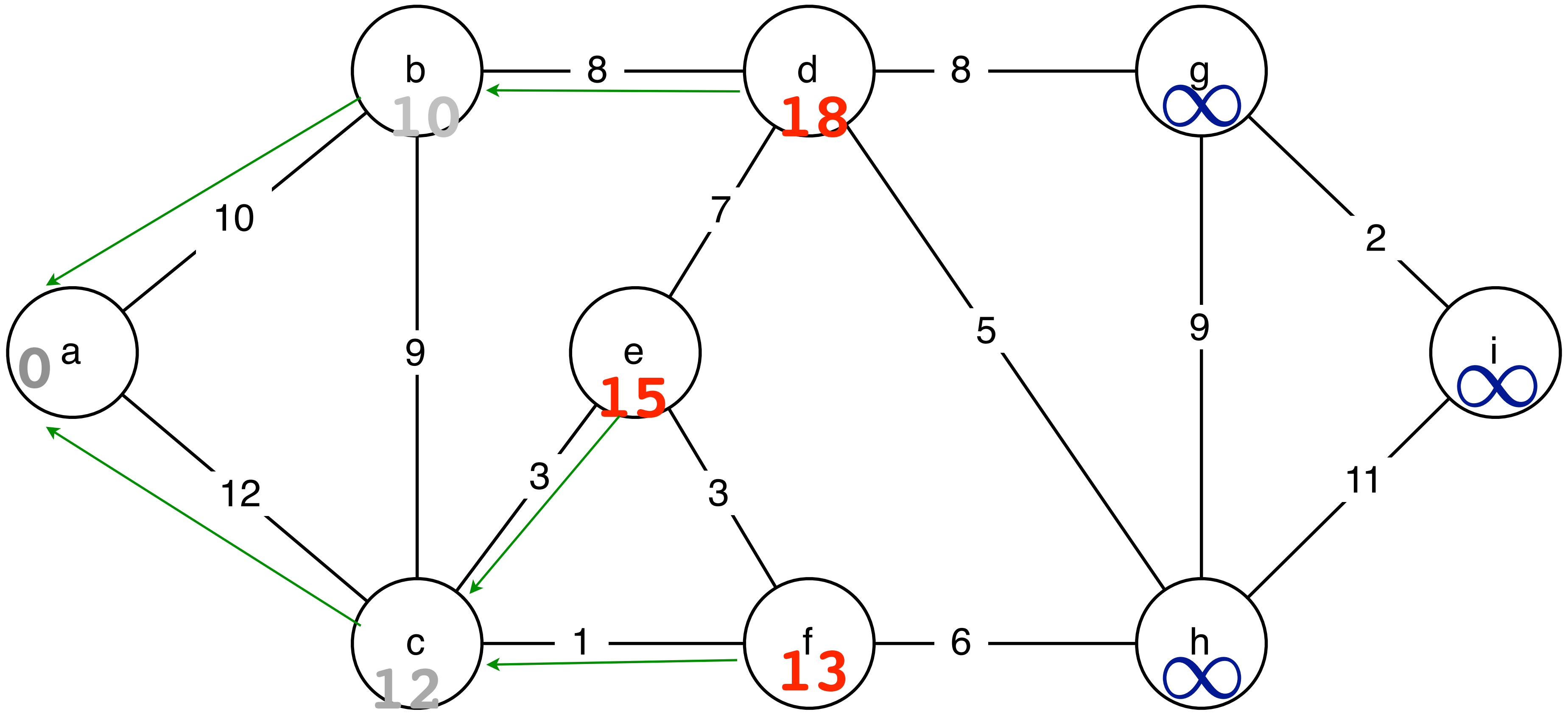




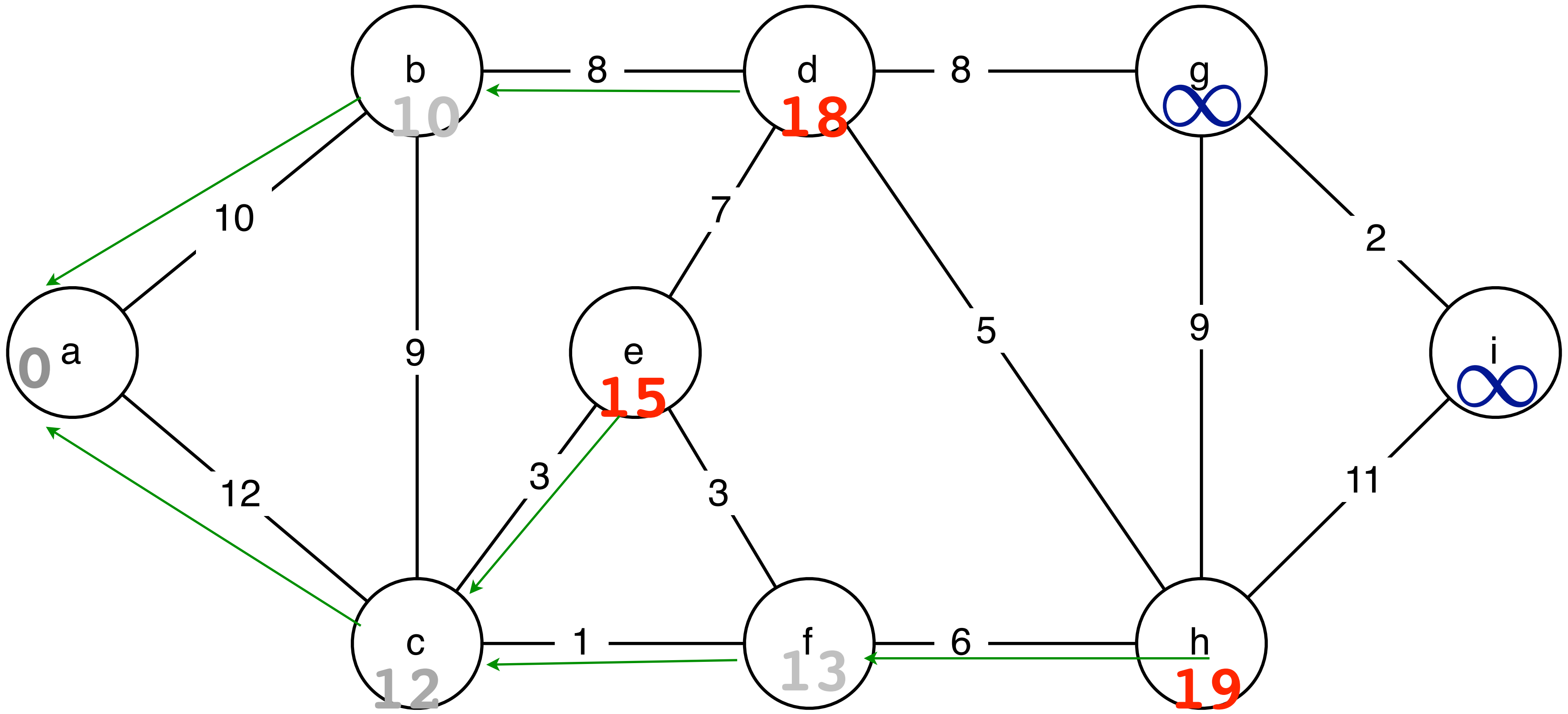
a
0
b
c
d
e
f
g
h
i
∞
10
12
9
8
8
7
3
3
1
6
5
9
2
11


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

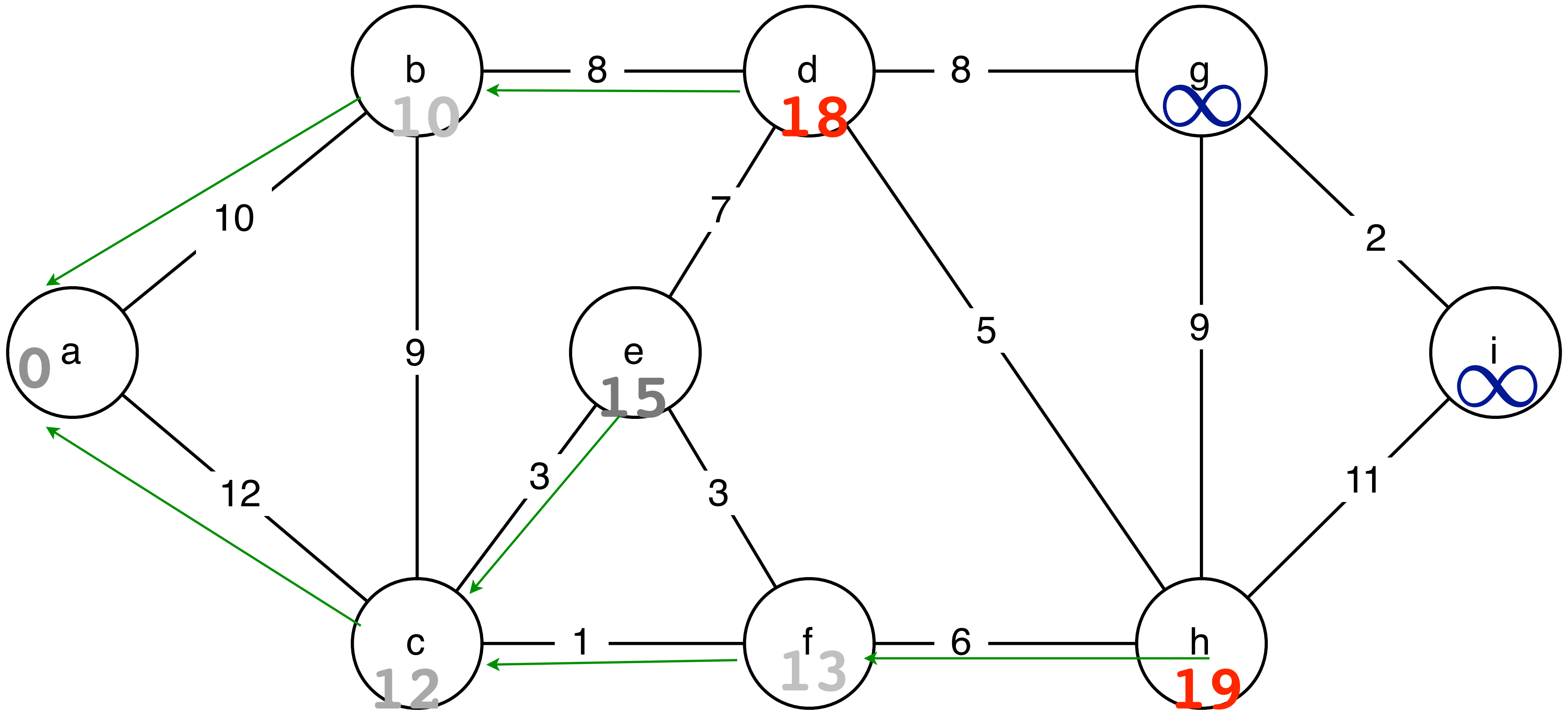

a
0
b
10
c
12
d
18
e
f
g
h
i
10
12
8
9
7
3
3
1
8
5
9
6
2
11



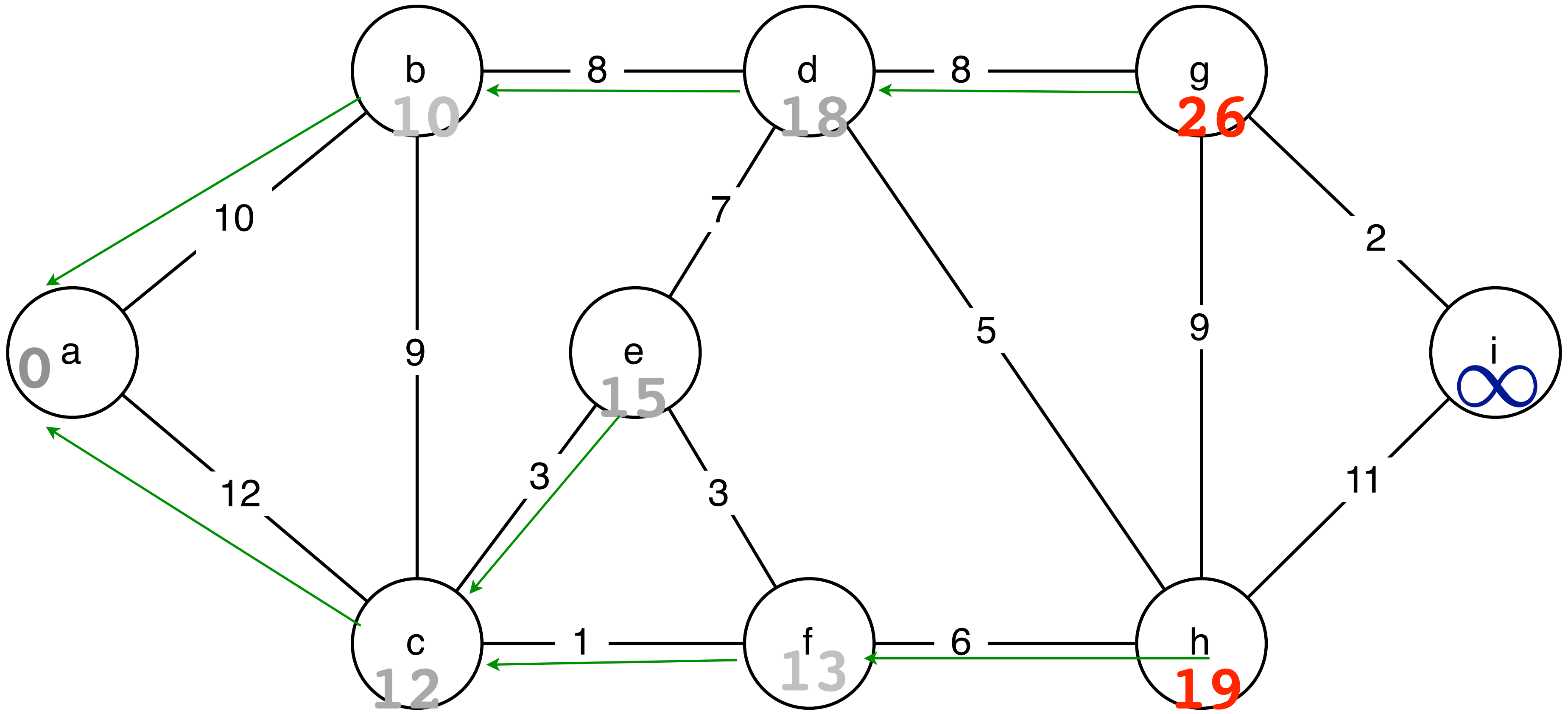
a
0
b
10
c
12
d
18
e
15
f
13
g
∞
h
∞
i
∞
10
12
9
8
3
1
7
3
8
5
6
9
2
11



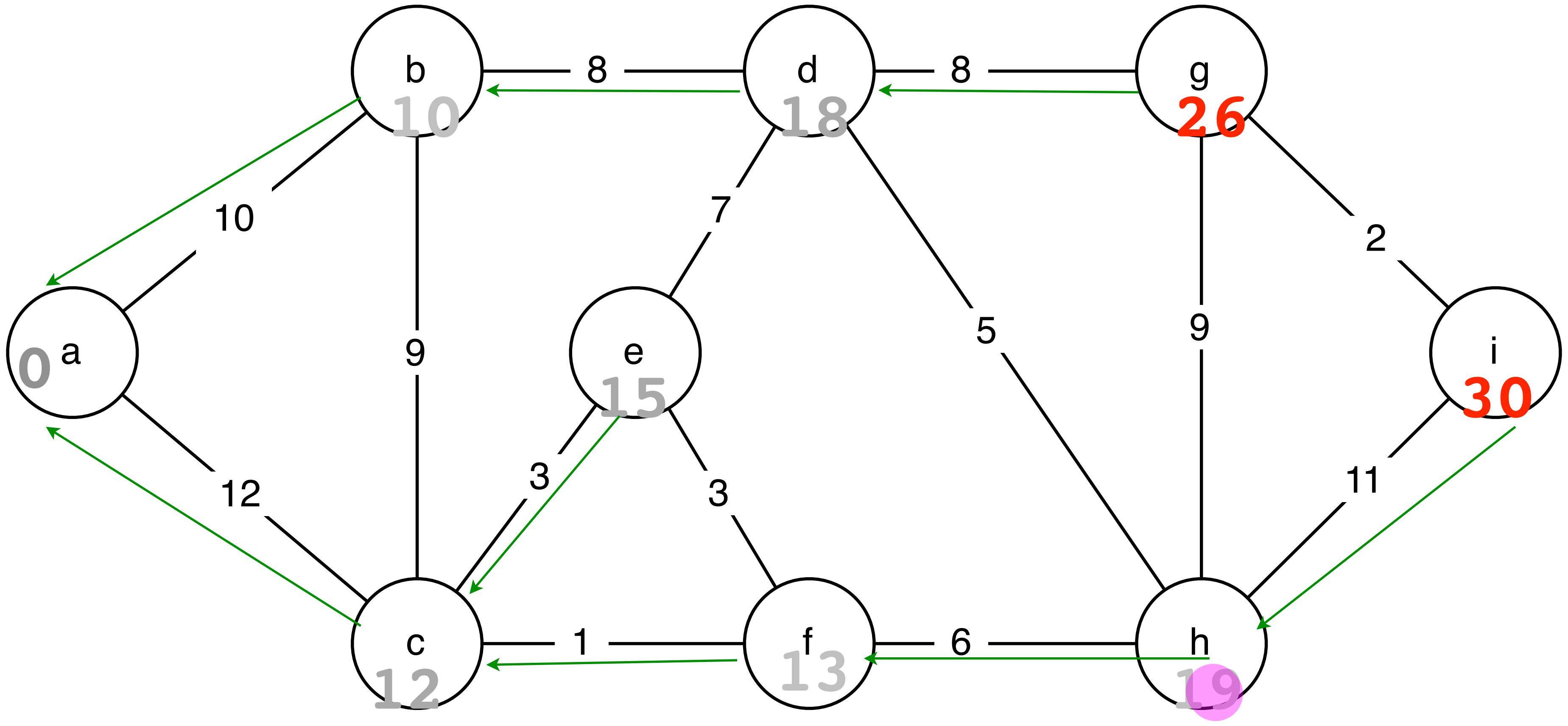
a
0
b
10
c
12
d
18
e
15
f
13
g
∞
h
19
i
∞
10
12
8
9
7
3
3
1
8
5
9
6
2
11



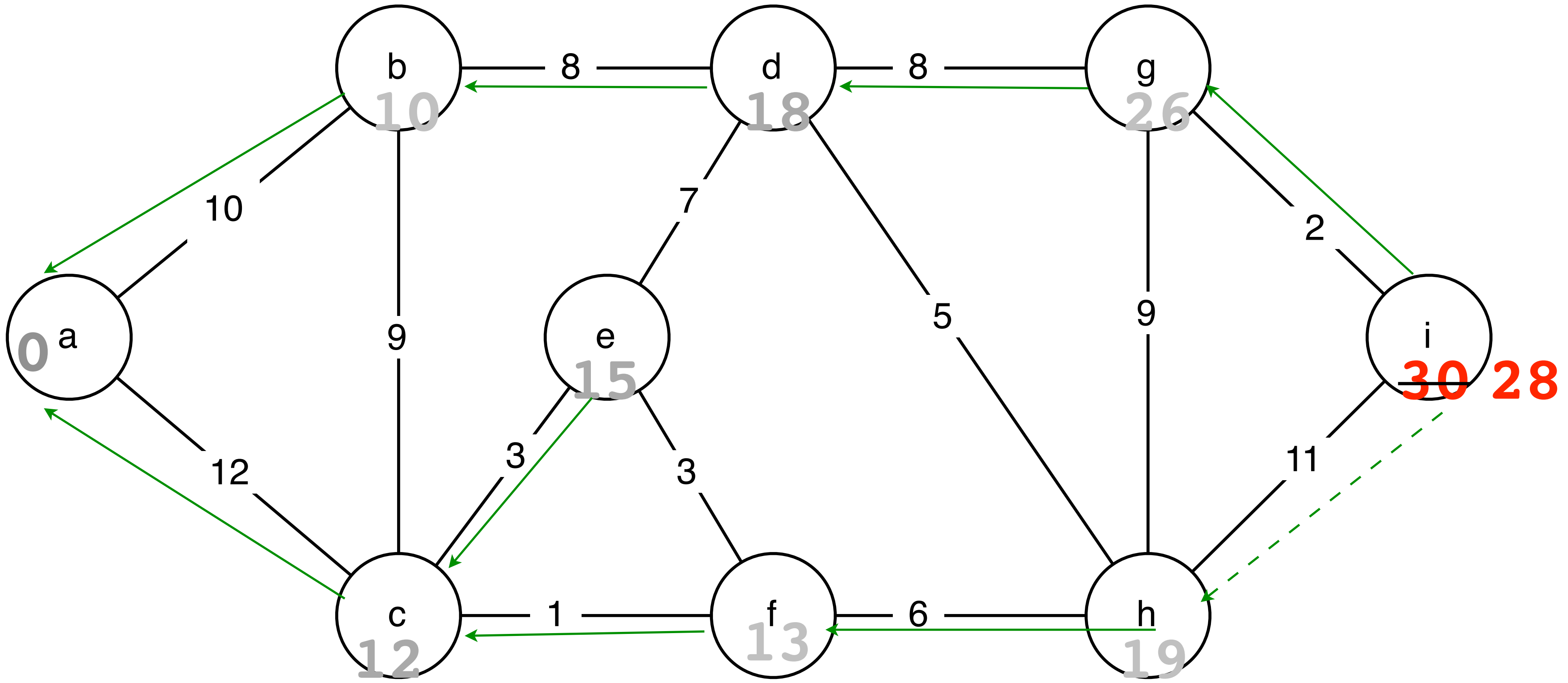
a
0
b
10
c
12
d
18
e
15
f
13
g
∞
h
19
i
∞
8
8
10
9
7
5
9
2
12
3
3
11
1
6



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



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
1
6
11



a
b
c
d
e
f
g
h
i
0
10
12
18
15
13
26
19
30
28
10
12
9
8
8
7
3
3
1
6
5
9
2
11

algorithm

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(
```