L16

4102

10.22.2013

abhi shelat

Greedy Alg:
Min Span Trees

MST

connecting houses

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

connecting houses

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

connecting houses

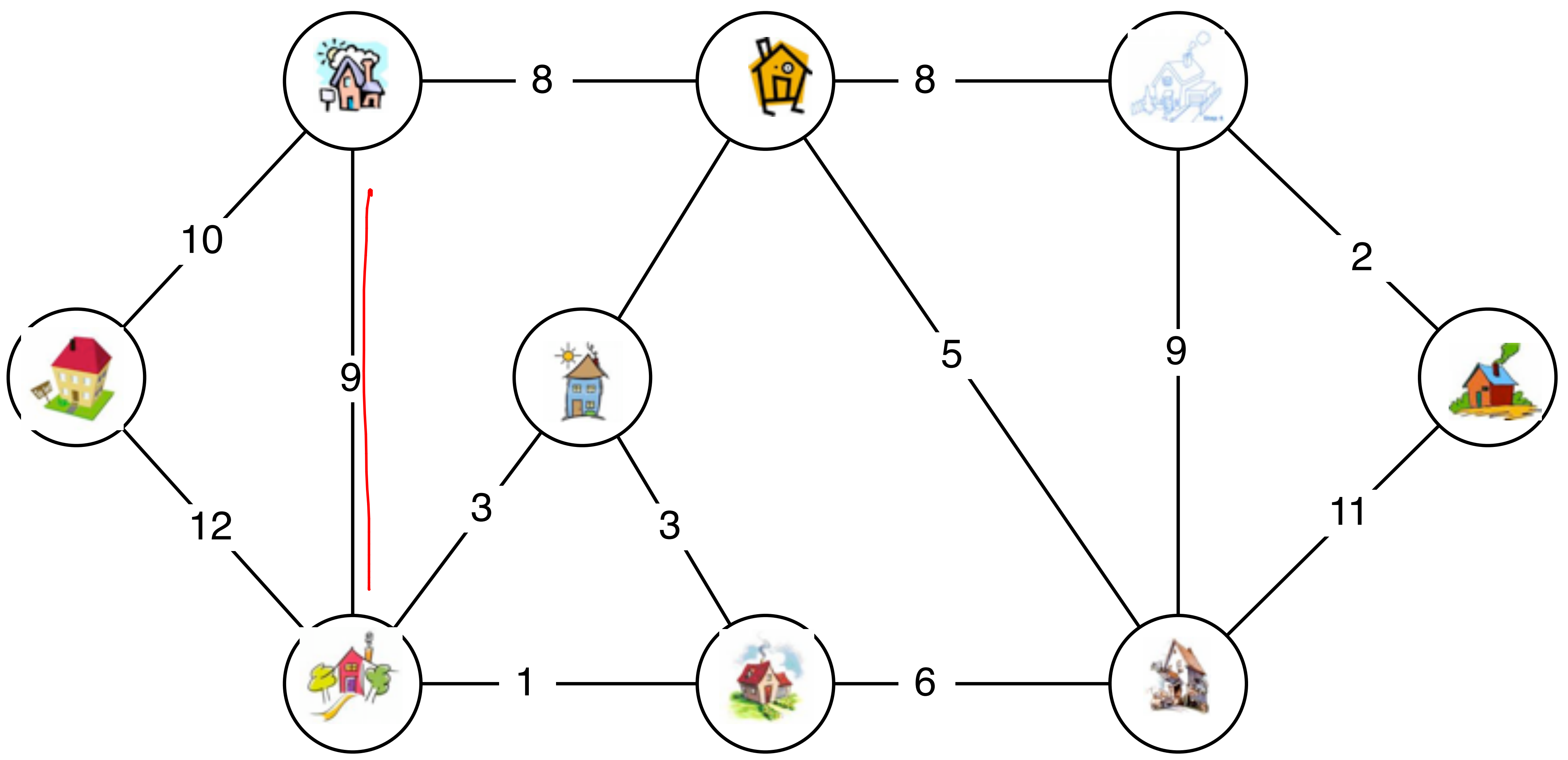
image:www.princegeorgeva.org, thefranciscofamily.org, www.rightdriveacademy.co.uk, www.ccscambridge.org, www.drawingcoach.com, www.pastoral.org.uk, www.daasgallery.com

connecting houses

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

connecting houses

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



8
8
10
2
9
5
9
12
3
3
11
1
6

graphs

clrs [ch 22]

$$G = (V, E)$$

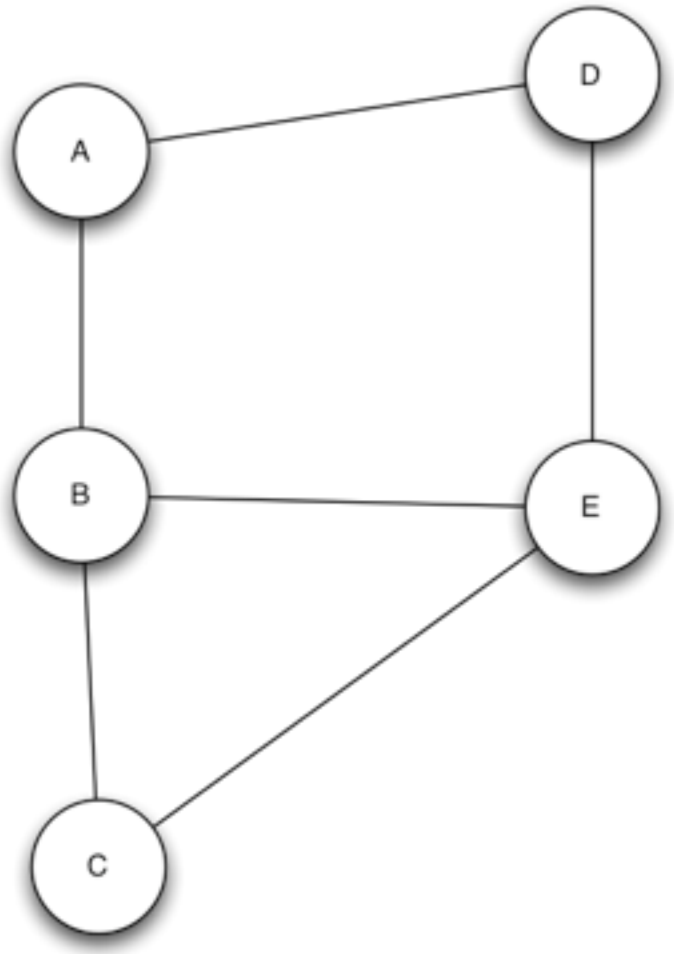
$(u,v) \in V^2$

$, \; w : E \to \mathbb{R}^+$

vertices

edges

edge weight function



definition: path

a sequence of nodes $v_1, v_2, \ldots, v_k$

with the property that $(v_i, v_{i+1}) \in E$

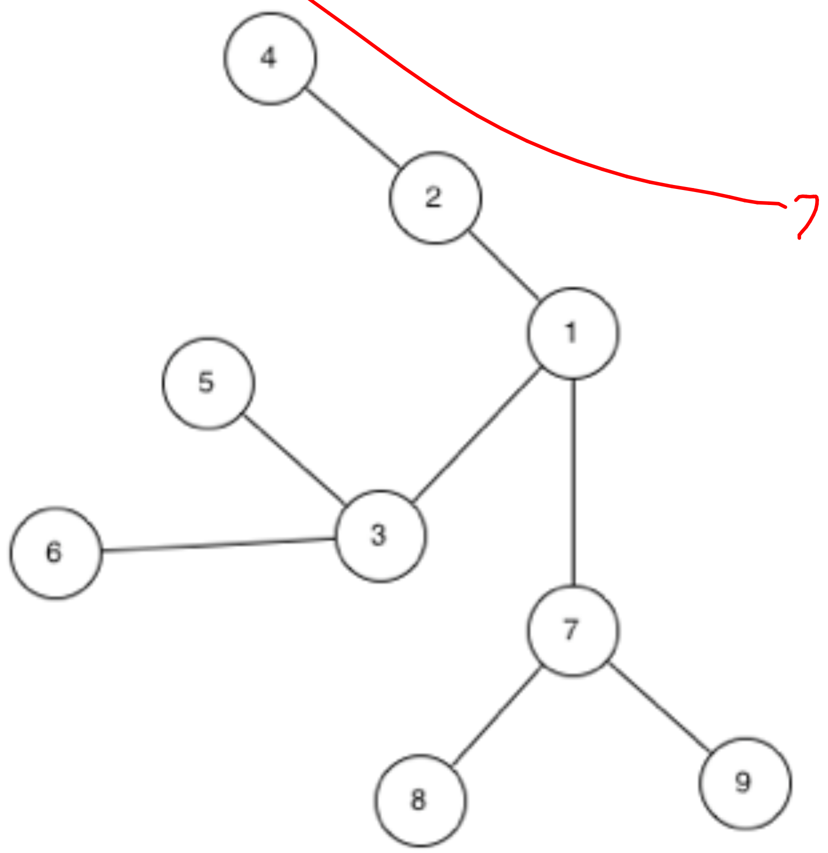
simple path: path in which each v_i occurs atmost once in the path

cycle: → path of length 2 or greater such that $v_1 = v_k$

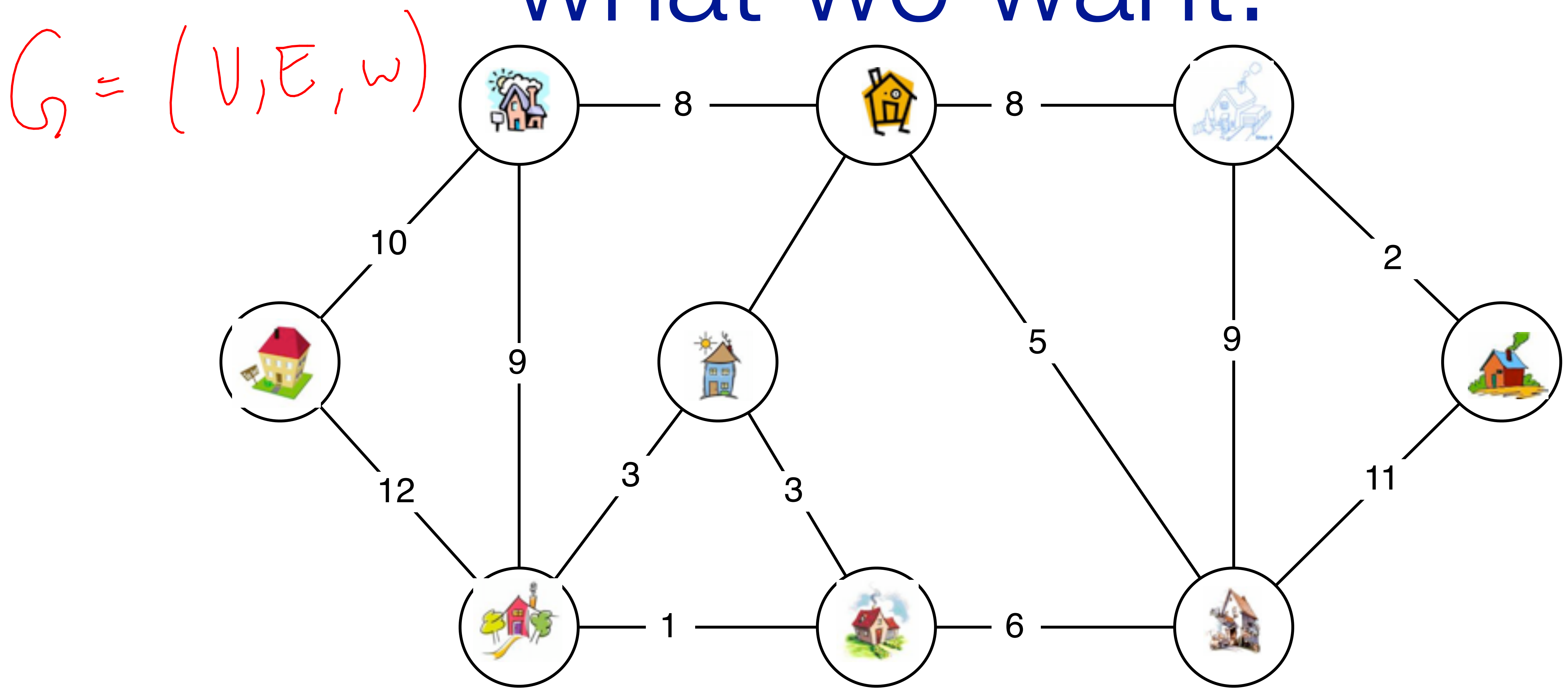
definition:tree

connected graph: G such that for any $u, v \in V$, there exists a path $v_1 = u$ and $v_k = v$.

a tree is → connected graph with no cycles.



what we want:



we want to find a tree $T \subseteq G$

that has "minimum cost" or "min weight"

minimum spanning tree

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

(a) connects all vertices

(b) has the least cost

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

looking for a set of edges that $T \subseteq E$
(a) connects all vertices
(b) has the least cost

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

Boruvka 1926

facts

how many edges does solution have ? → V−1

does solution have a cycle?

No cycles in our solution

(all edge weights > 0)

strategy

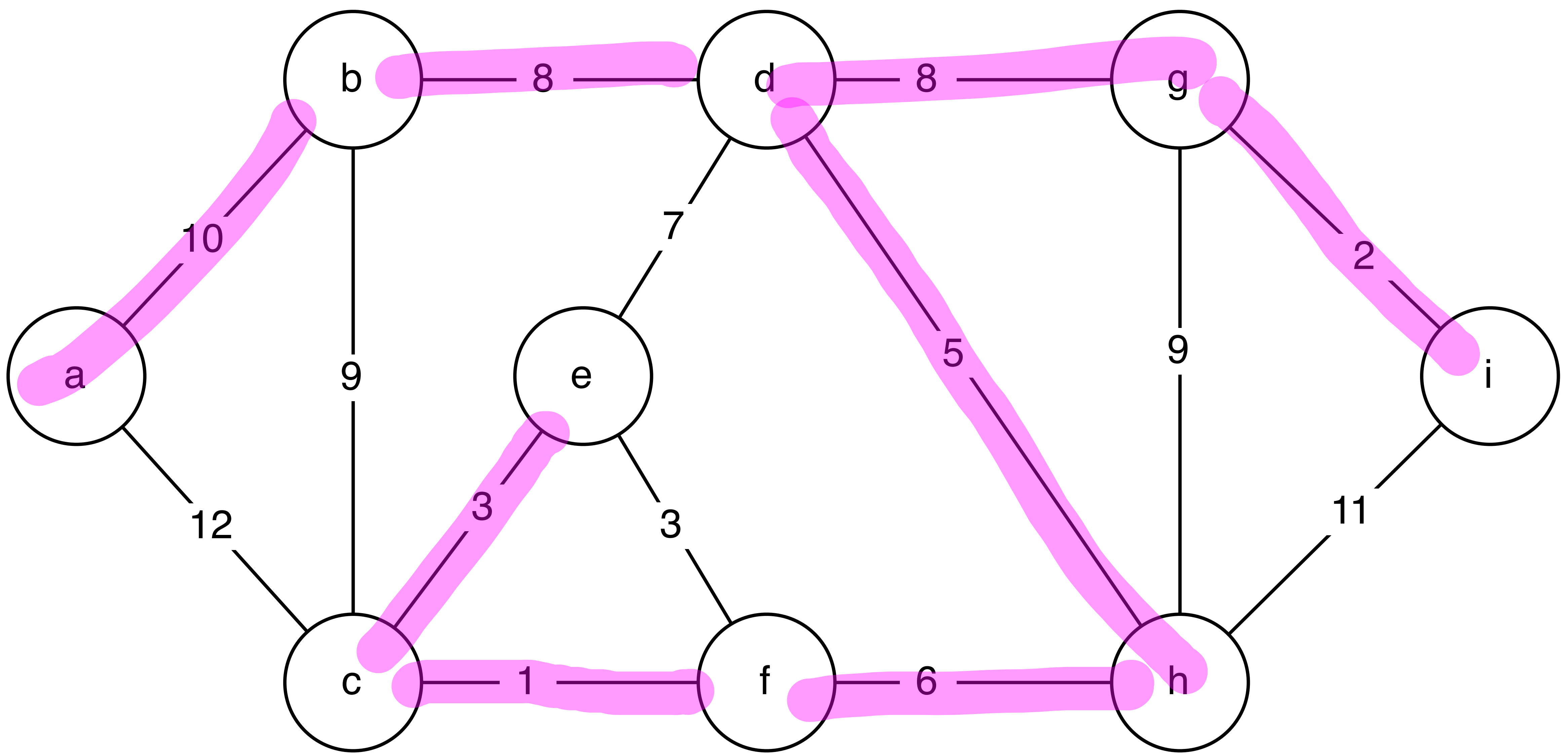
KRUSKAL'S Algorithm.

start with an empty set of edges A

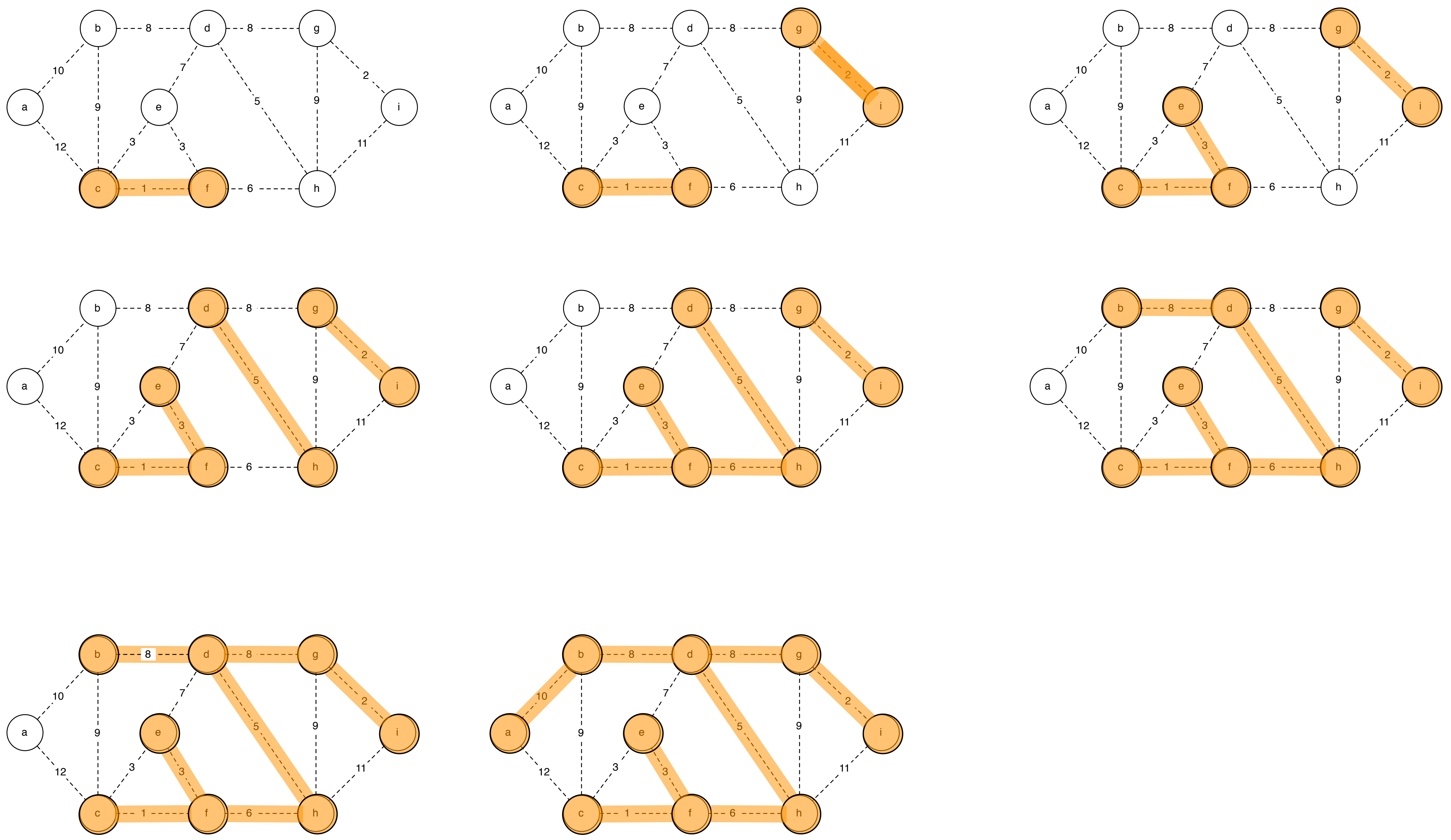
repeat for v-1 times:

add lightest edge that does not create a cycle

example



kruskal



why does this work?

```
T ← ∅
repeat  V − 1 times:
        add to T the lightest edge e ∈ E that does not create a cycle
```

how can we implement this check??

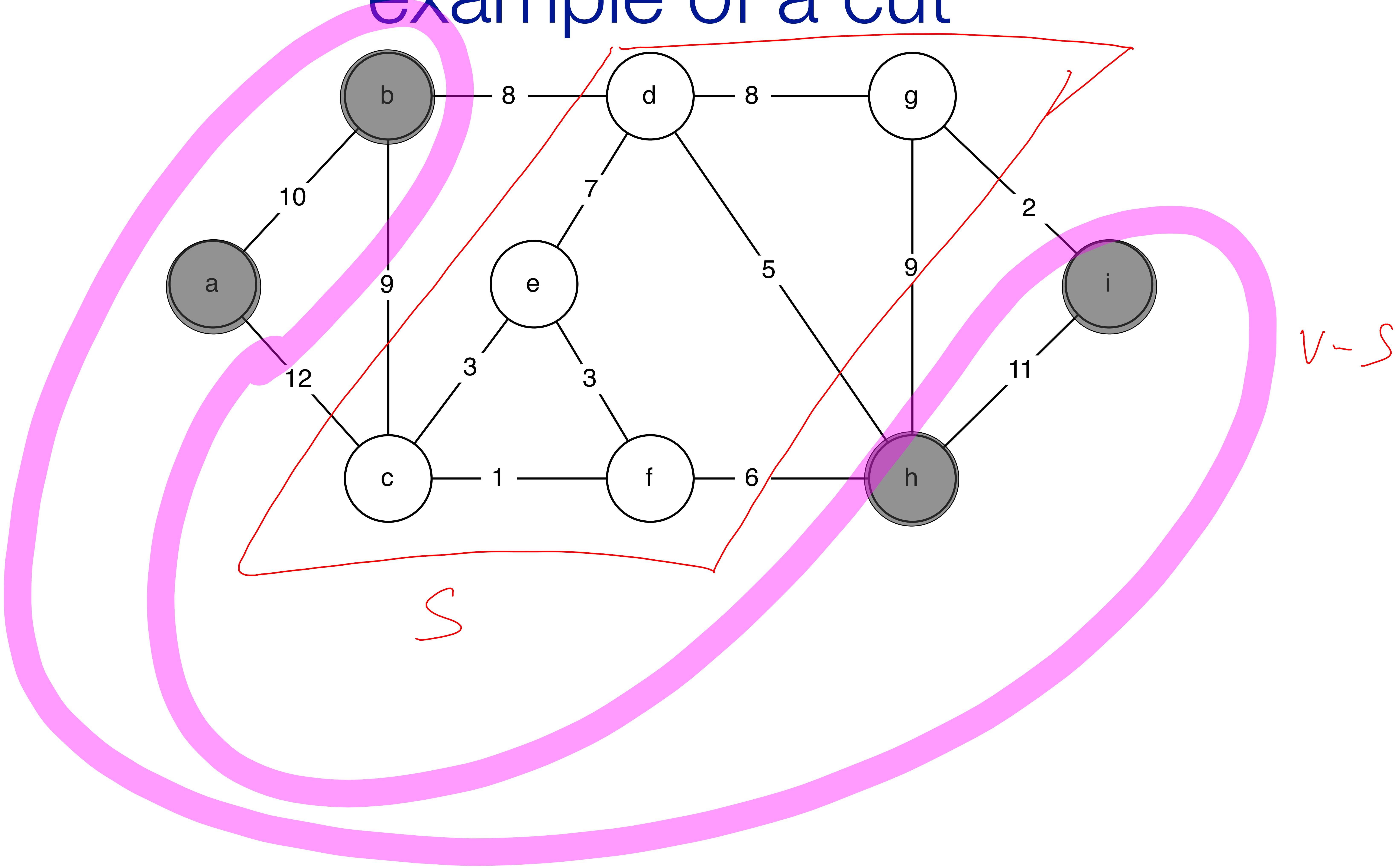
UNION-FIND data structure.

definition: cut

Cut: partition of the verticies into 2 sets

$(S, V-S)$.

example of a cut



definition: crossing a cut

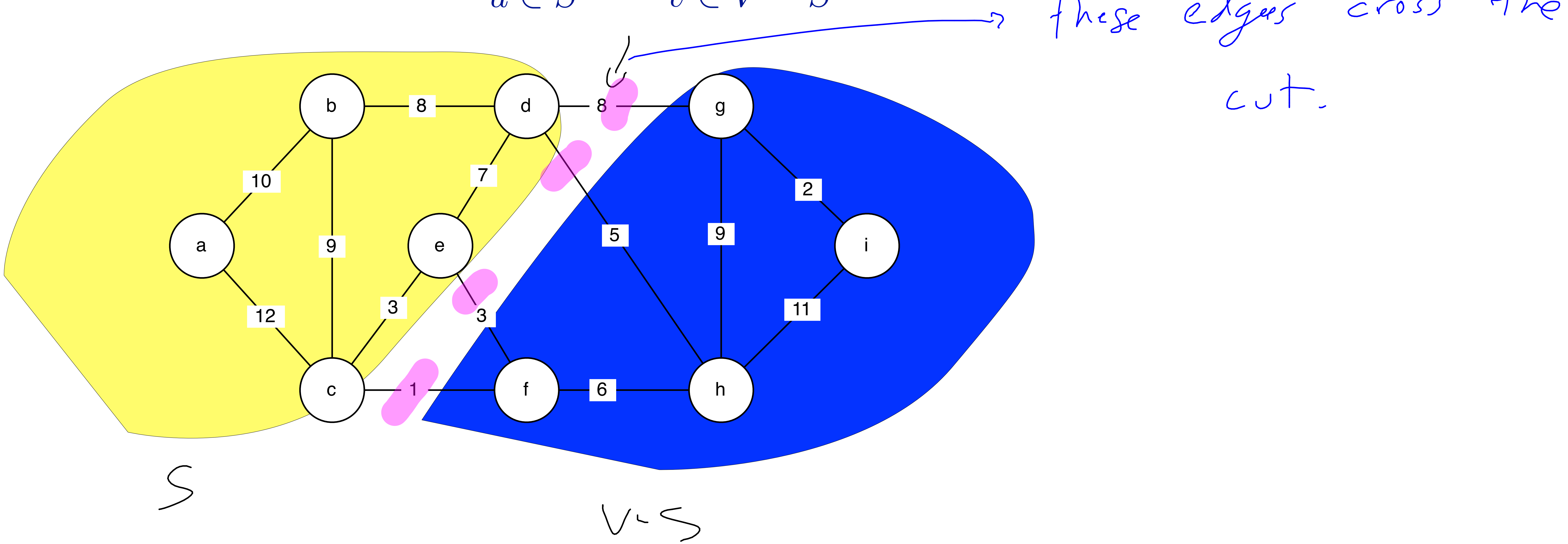
An edge $e = (u, v)$ crosses a cut $(S, V-S)$

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

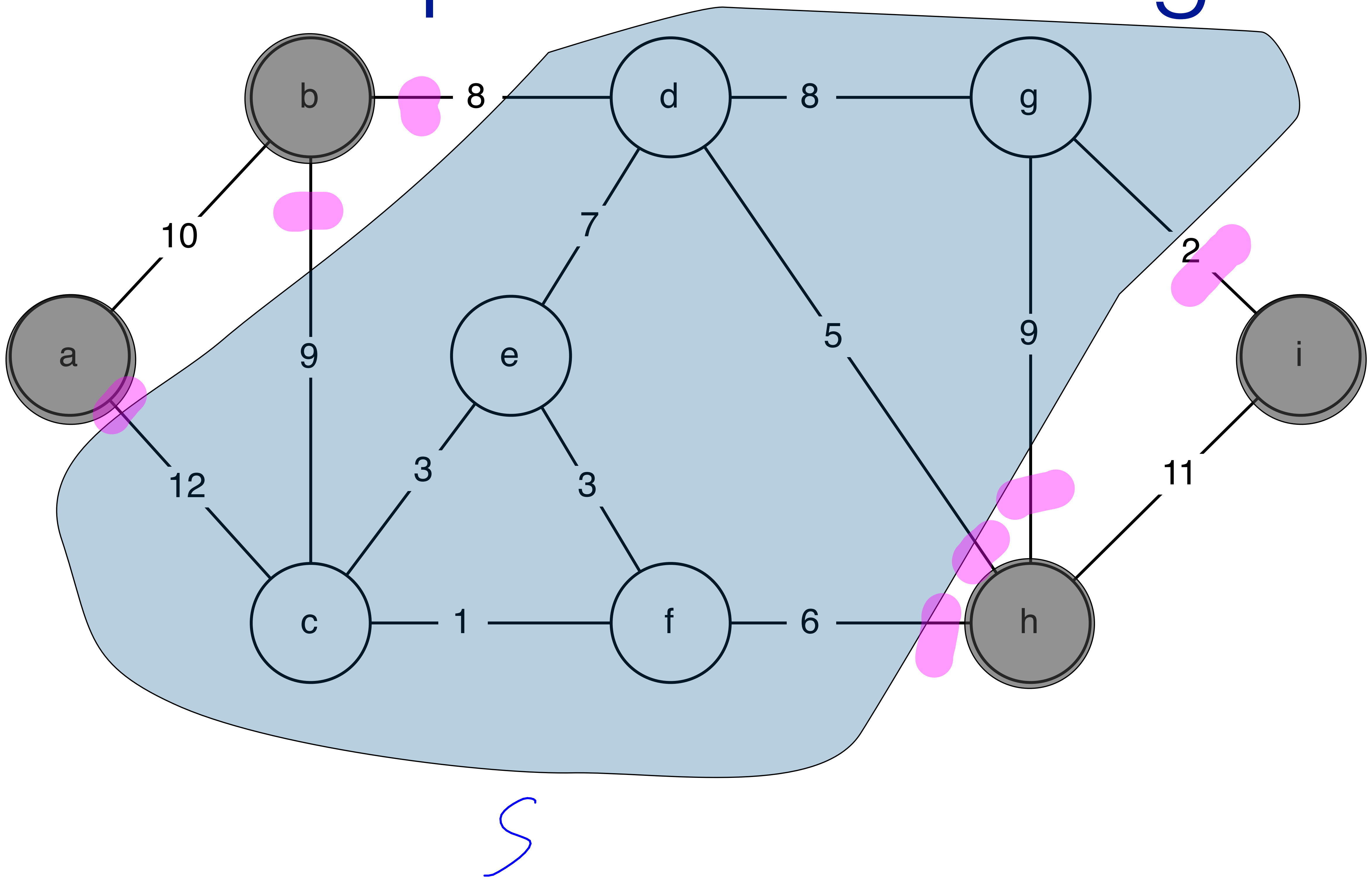
definition: crossing a cut

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

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



example of a crossing



definition: respect

The set of edges A respects the cut (S, V-S)

if No edge in A crosses (S, V-S)

cut theorem

MAIN IDEA BEHIND the simple MST algorithms

Let A be some subset of an MST T.
Let $(S, V-S)$ be any cut such that A respects $(S, V-S)$.
Let e be the lightest edge that crosses $(S, V-S)$.

Then $A \cup \{e\}$ is part of some MST.

cut theorem

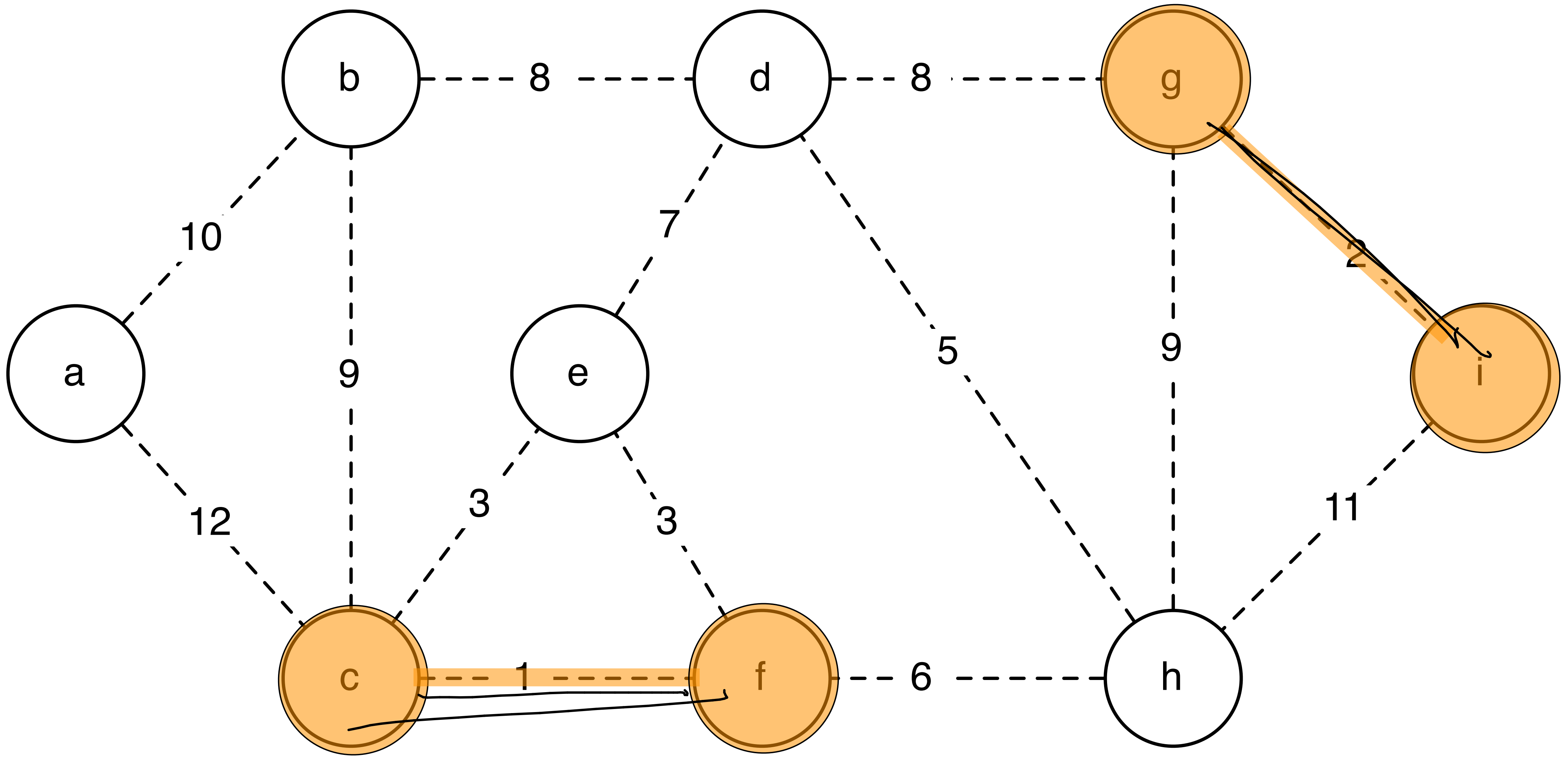
suppose the set of edges A is part of an m.s.t. T. of graph $G=(V,E)$

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

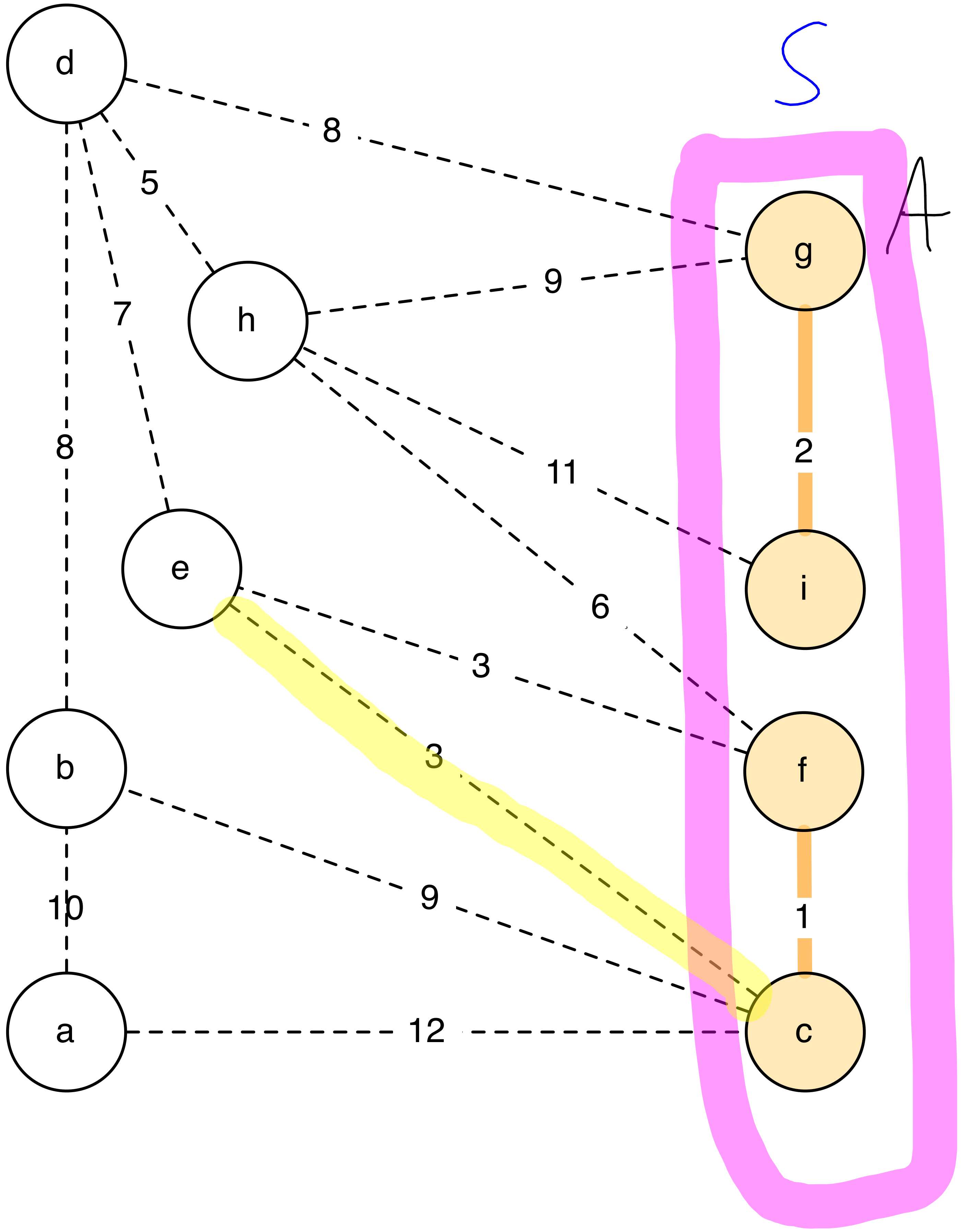
let edge e be the min-weight edge across $(S, V-S)$

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

example of theorem



A: edges in orange.



(1) pick some cut (S, V−S) s.t. A respects the cut.

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.*

an

an MST of G.

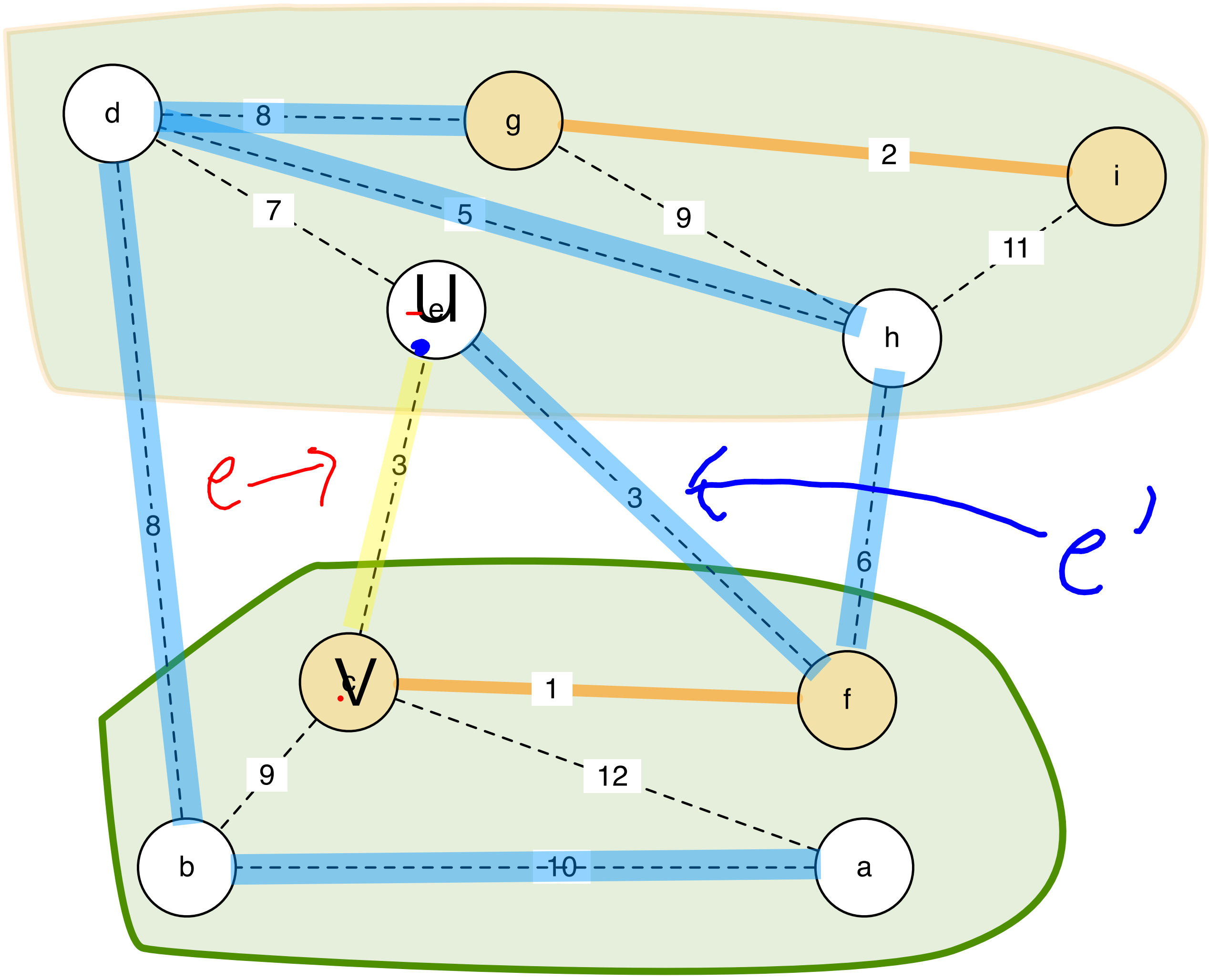
Proof: If $A \cup \{e\}$ is already part of T, then the thm follows.

Suppose that $A \cup \{e\}$ is not part of T.

We will construct another MST T' such that $A \cup \{e\} \subset T'$

Let $e = (u, v)$.

proof of cut thm



A: orange links. T (MST): blue + orange

(S, V−S) is a cut & A respects S.

① Add e to T. A cycle is created from u→v→u

Let e' be the first edge on the cycle from v → u that crosses (S, V−S)

② Consider the tree T' = T − {e'} + {e}

T' is an MST: ⓐ T' has no cycles and |T'| = V−1

$wt(T') = wt(T) - w(e') + w(e) \leq wt(T) \Rightarrow T'$ is an MST.

because $w(e) \leq wt(e')$

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
```

→ In step ①, we start with A as a subset of some MST.

Suppose after k steps, A is a subset of some MST.

Now consider one iteration the loop @ 2-3,

Let $e=(u,v)$ be the selected edge.

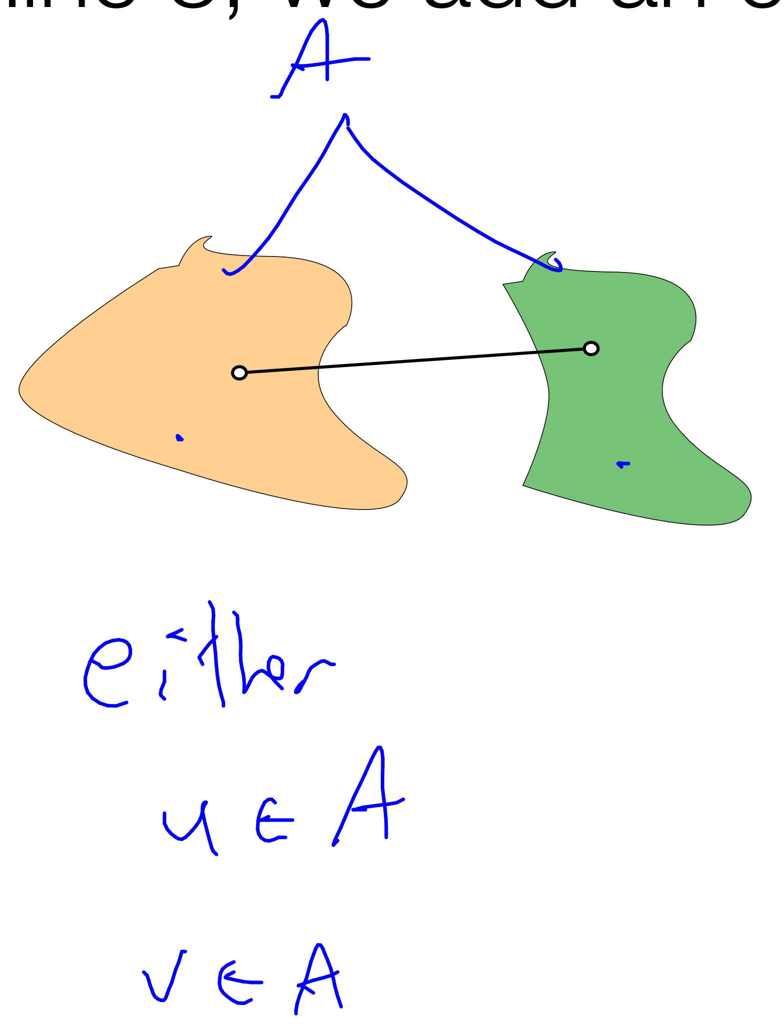
I claim 3 cases to consider.

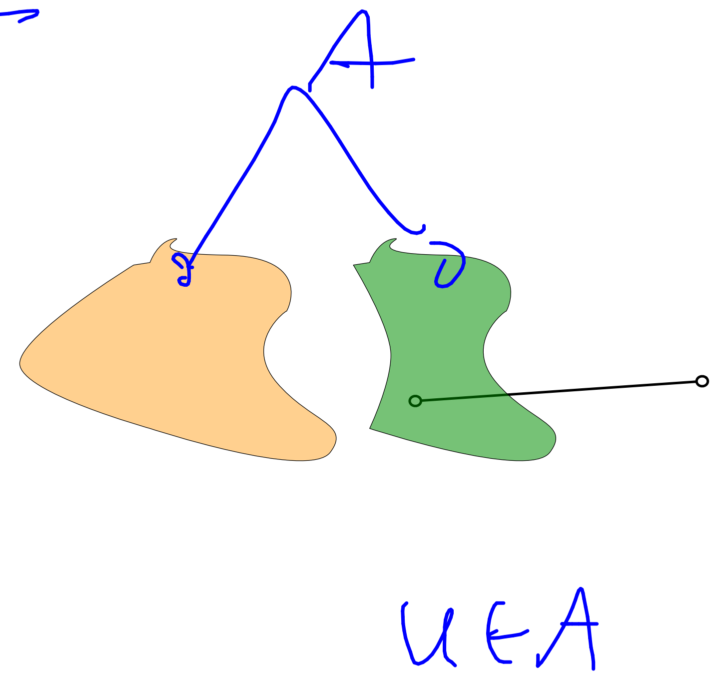
KRUSKAL-PSEUDOCODE(G)

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

correctness

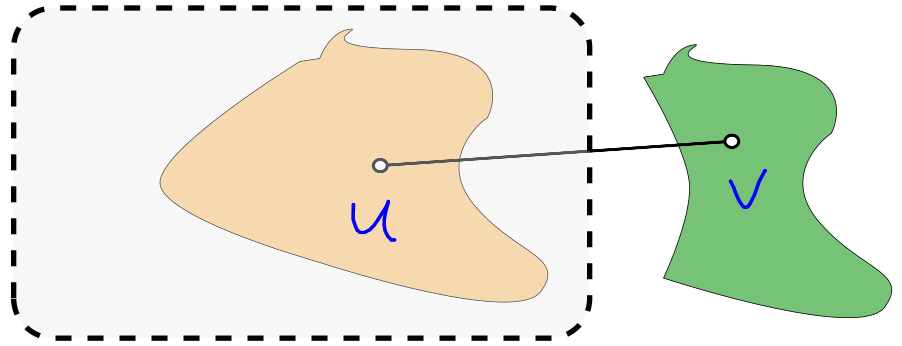
proof: by induction. in step 1, A is part of some MST.
suppose that after k steps, A is part of some MST (line 2).
in line 3, we add an edge e=(u,v) to A.





3 cases for edge e.

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



S to be this component that contains u.

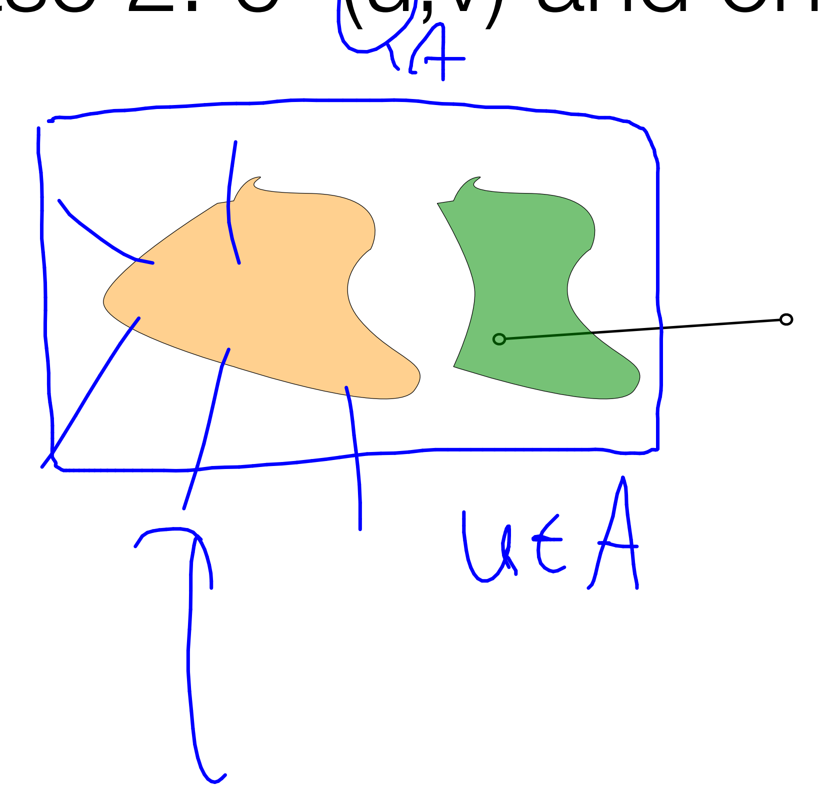
① A respects (S, V-S)

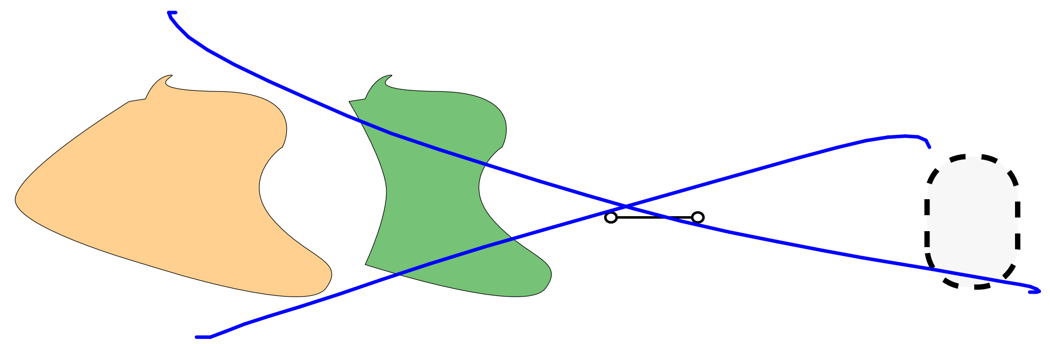
② by the cut thm, e will also be part of some MST.

because e is the lightest edge that crosses this cut

3 cases for edge e.

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





set S=A.

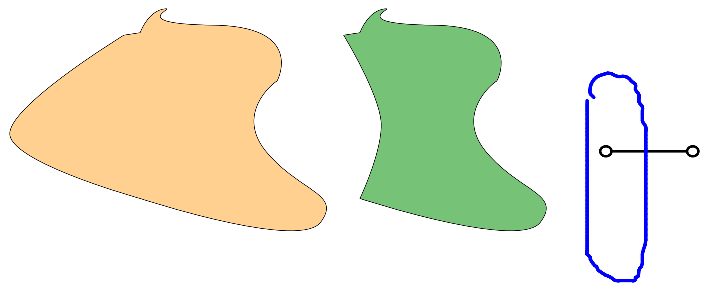
e is the lightest edge which crosses (S, V-S)

b/c it is the lightest edge that does not create a cycle in A. ⟹ by cut thm,

A ∪ {e} is part of an MST.

3 cases for edge e.

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



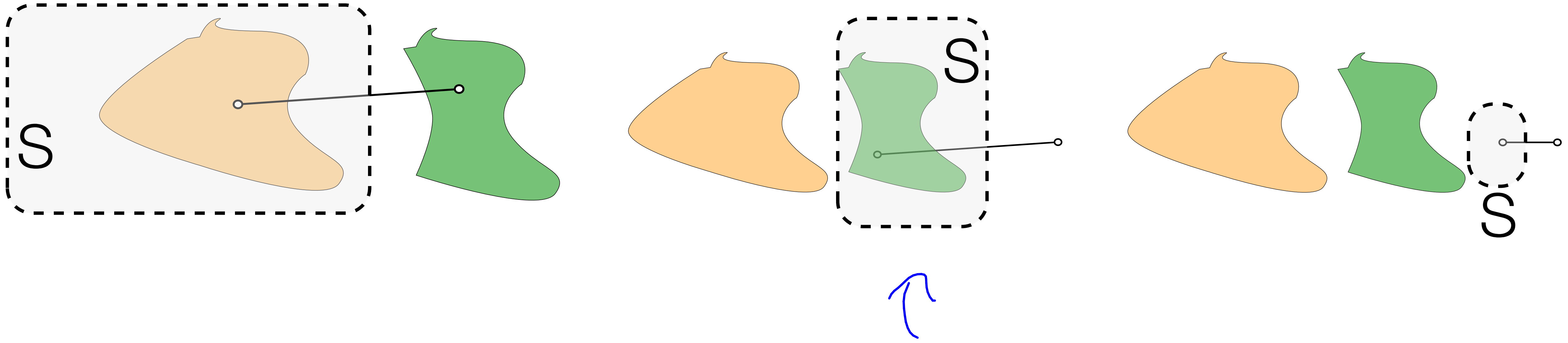
S = {u}.

Again, A respects S

e is the lightest edge to cross (S, V−S)

... etc.

3 cases for edge e



analysis?

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

how to implement.

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}
```

Kruskal is one way of implementing these 3 steps.

Prim's algorithm

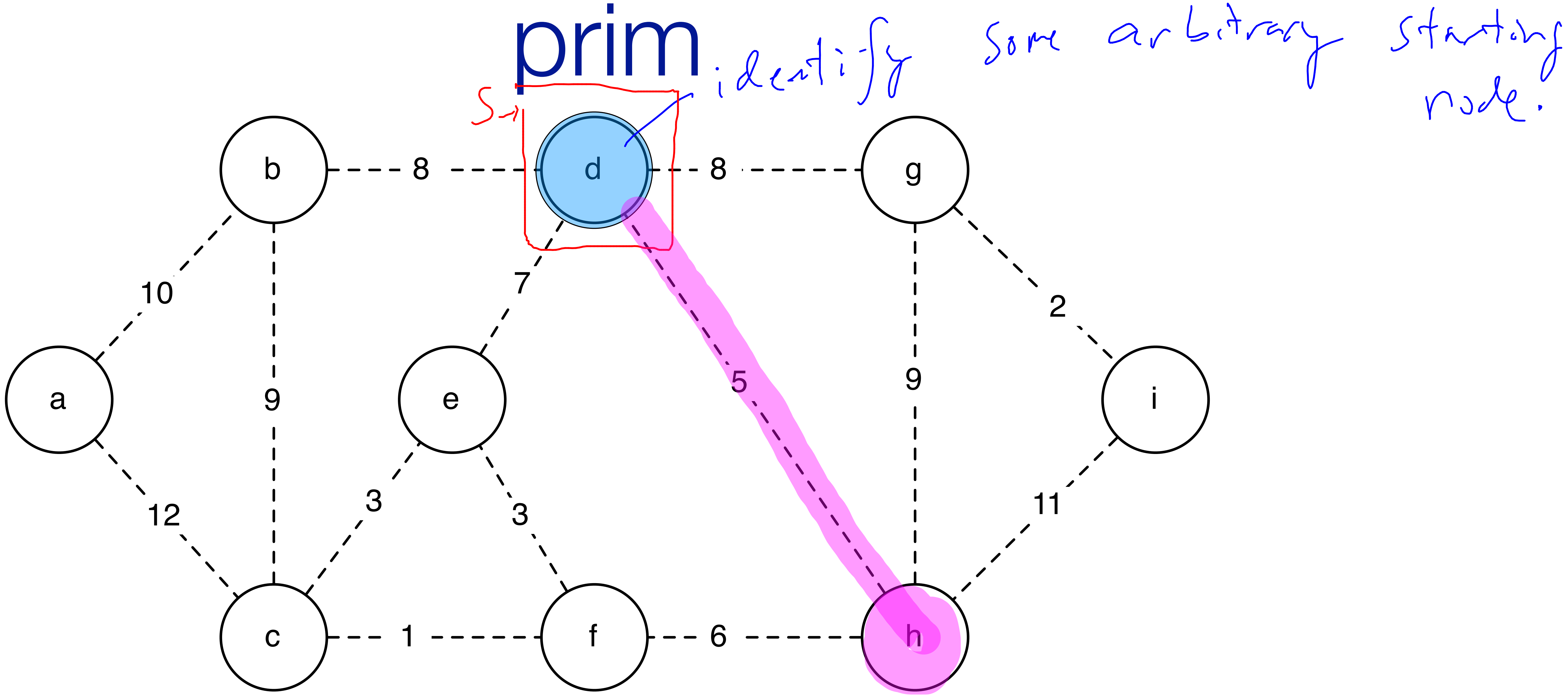
GENERAL-MST-STRATEGY($G = (V, E)$)

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

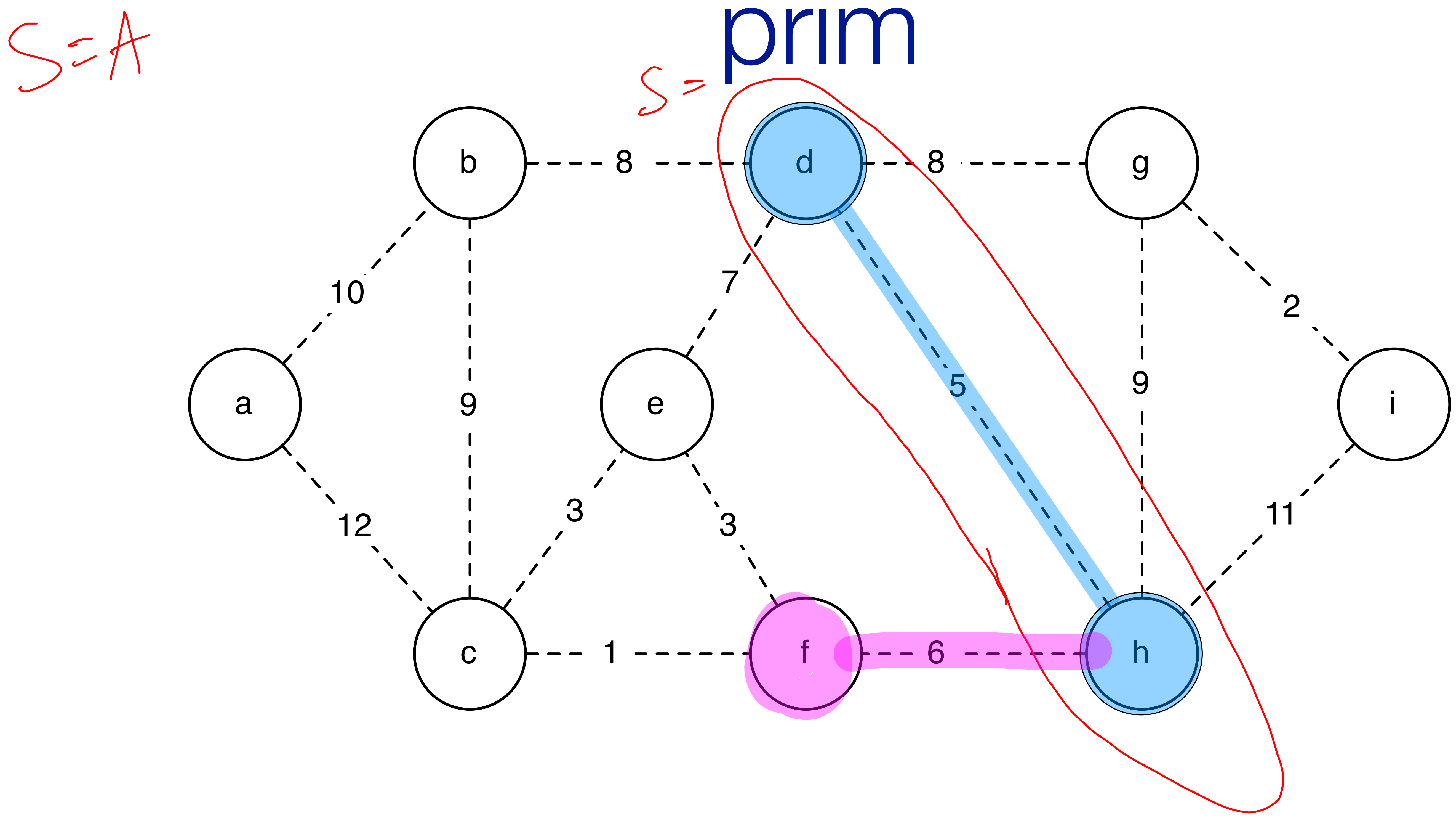
A is a subtree

edge e is lightest edge that grows the subtree

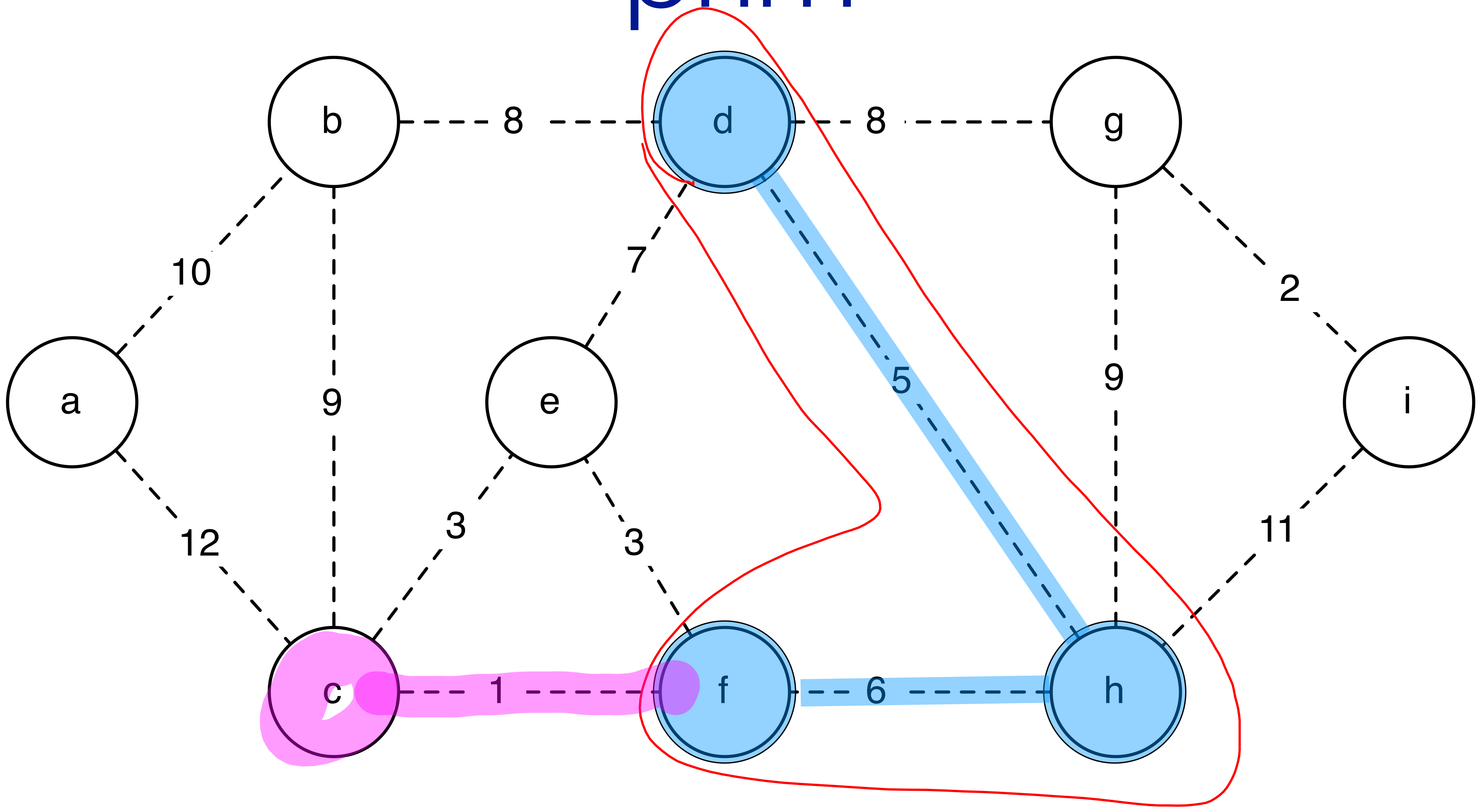
prim


identify some arbitrary starting node.
S
a
b
c
d
e
f
g
h
i
10
12
9
8
7
3
3
1
6
5
9
8
2
11

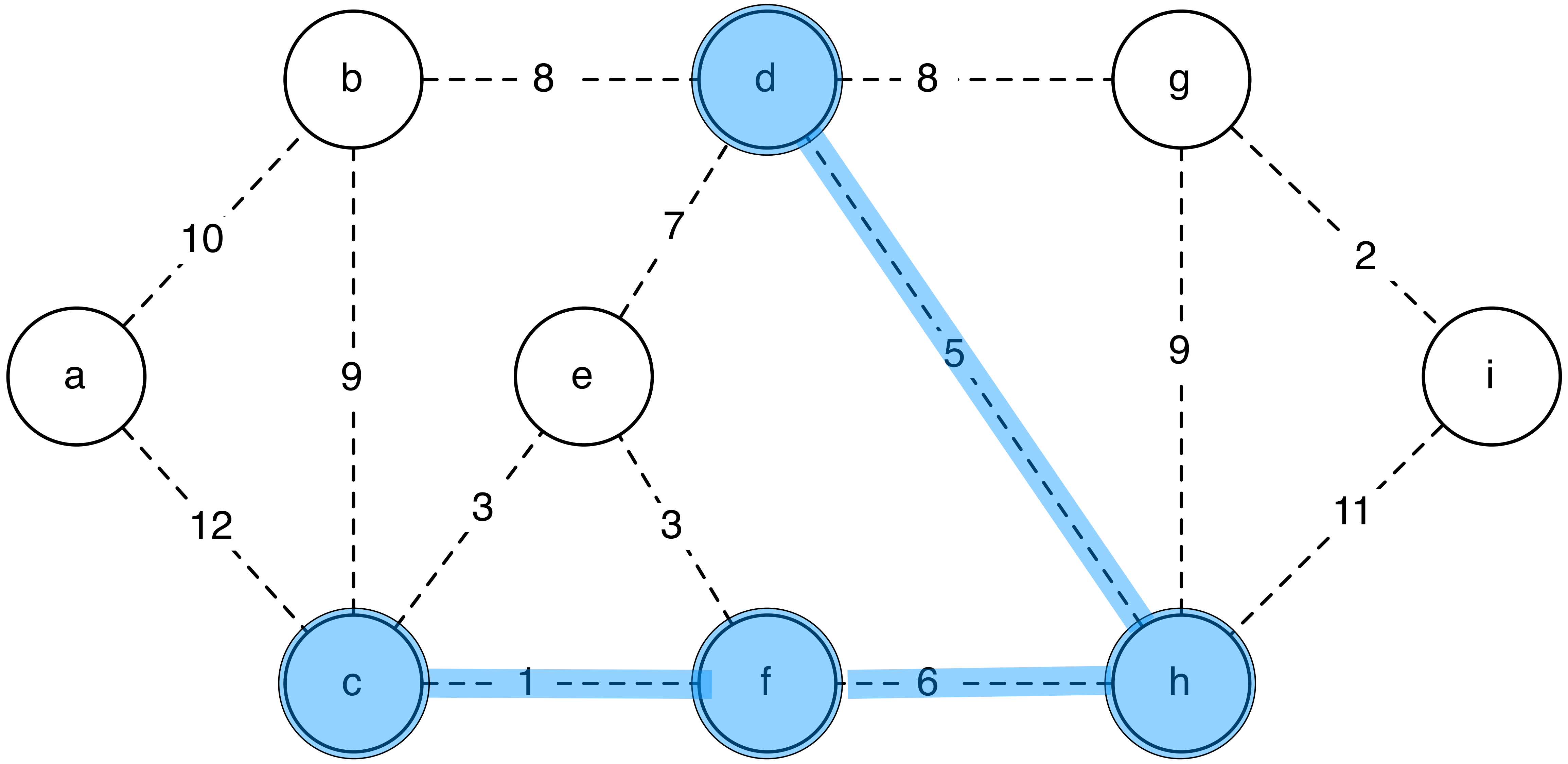
prim



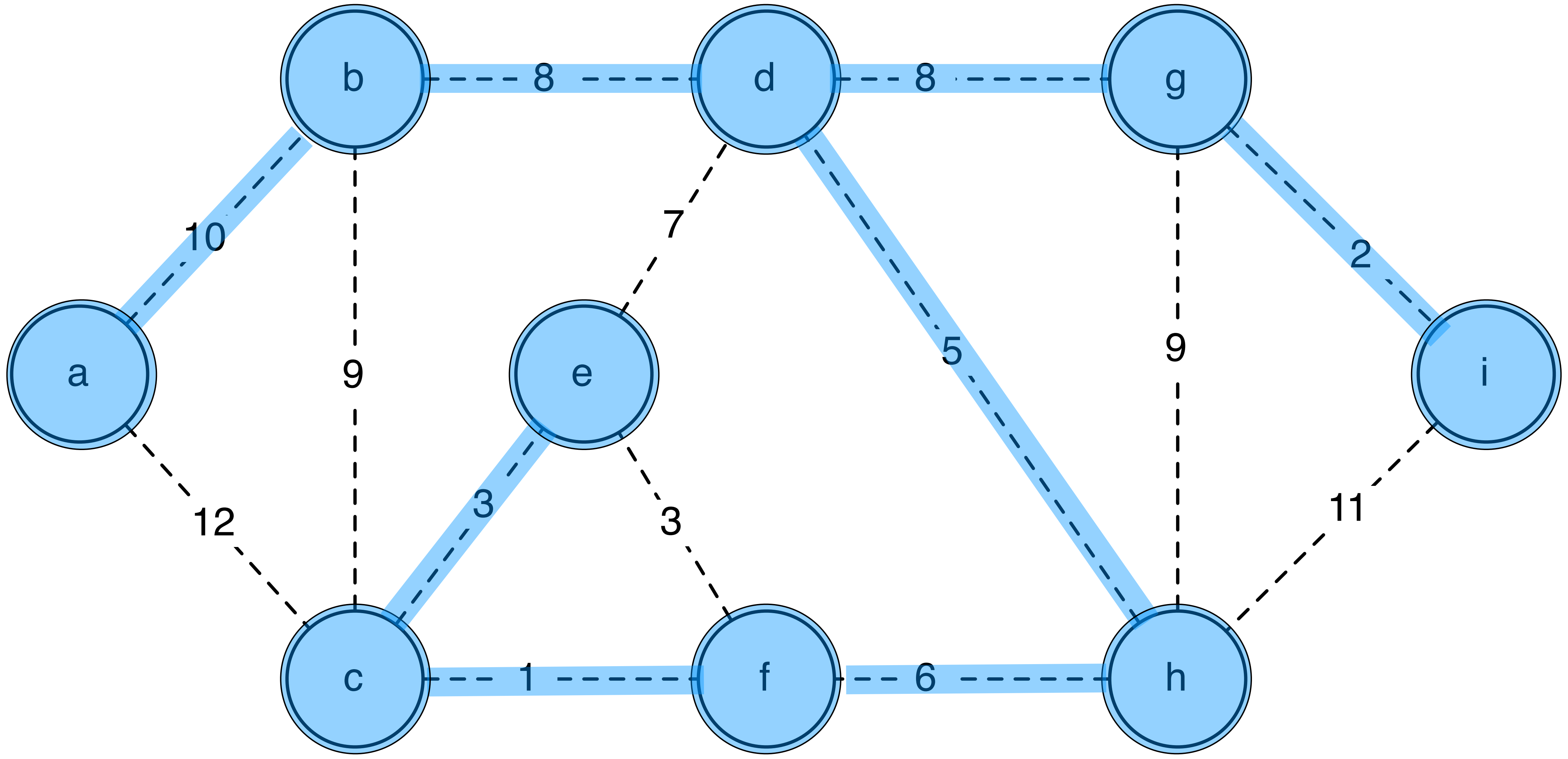
prim



prim



prim



implementation

idea: S will always be the current solution S

→ we can use a Priority Queue to help us determine "the min-weight edge that crosses (S, V−S)"

implementation

new data structure

Priority Queue:

- insert
- extractmin
- decreaseKey(e, key) sets e.key = key

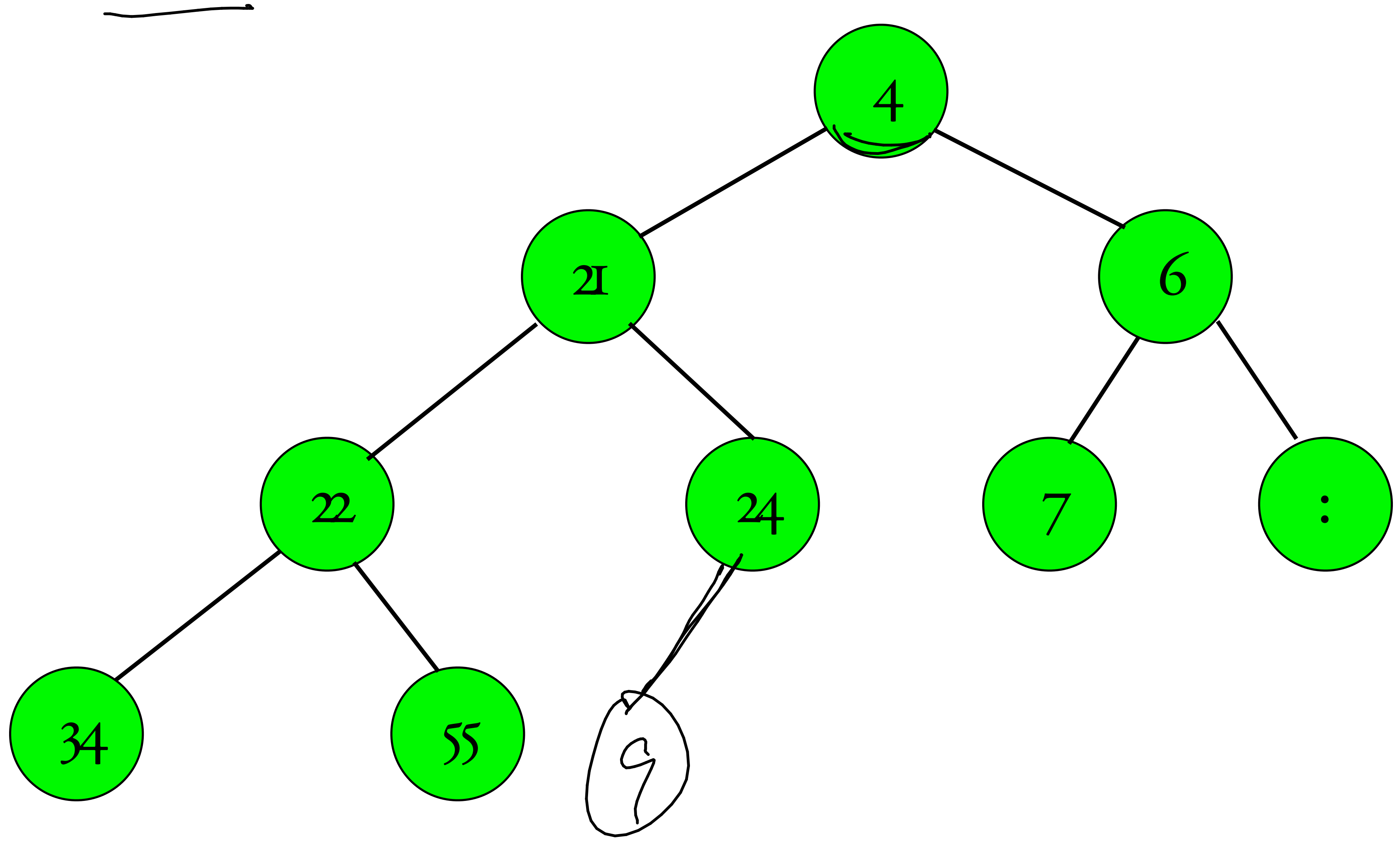
binary heap

full tree, key value <= to key of children

binary heap

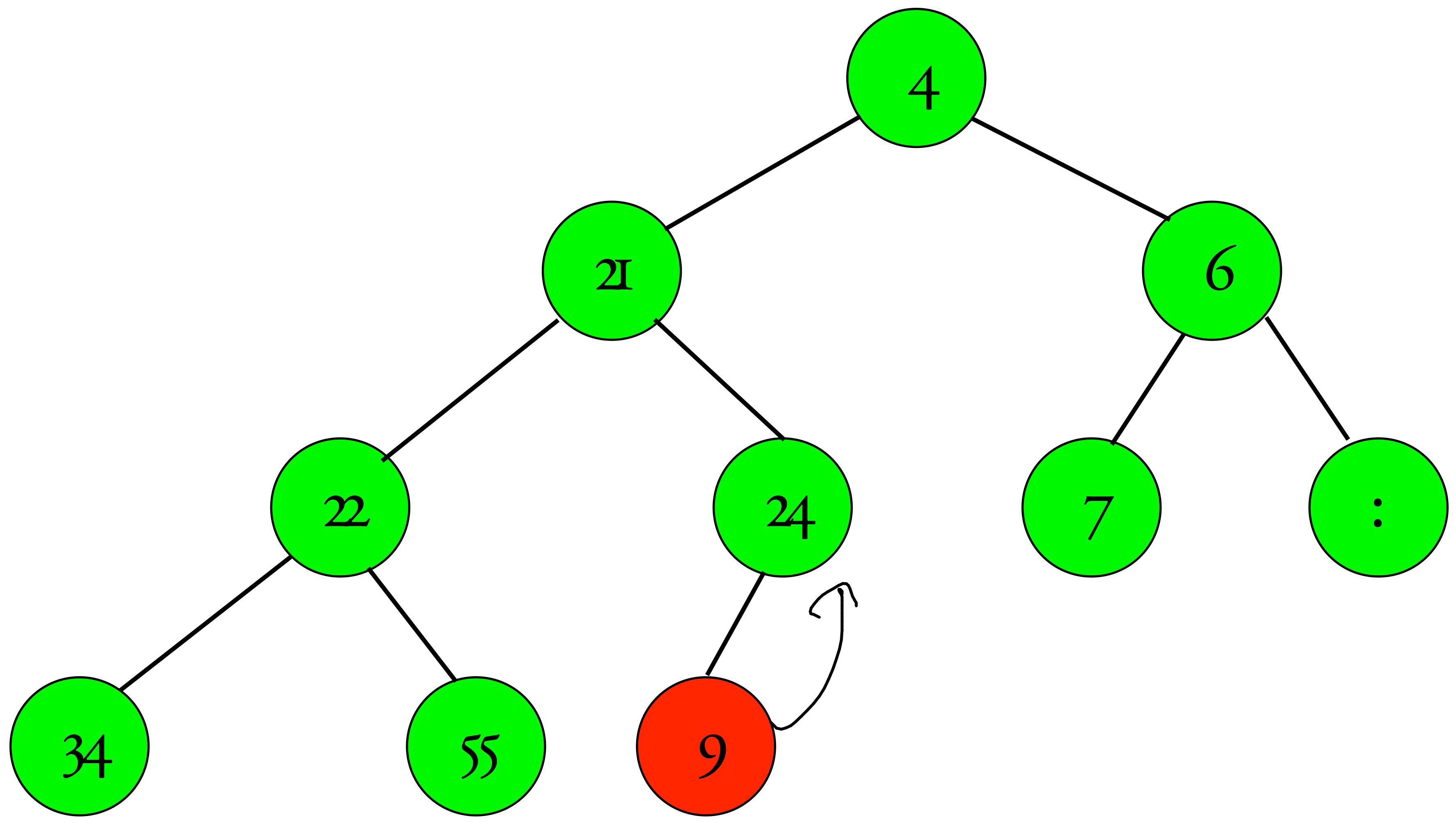
priority queue data structure

full tree, key value <= to key of children



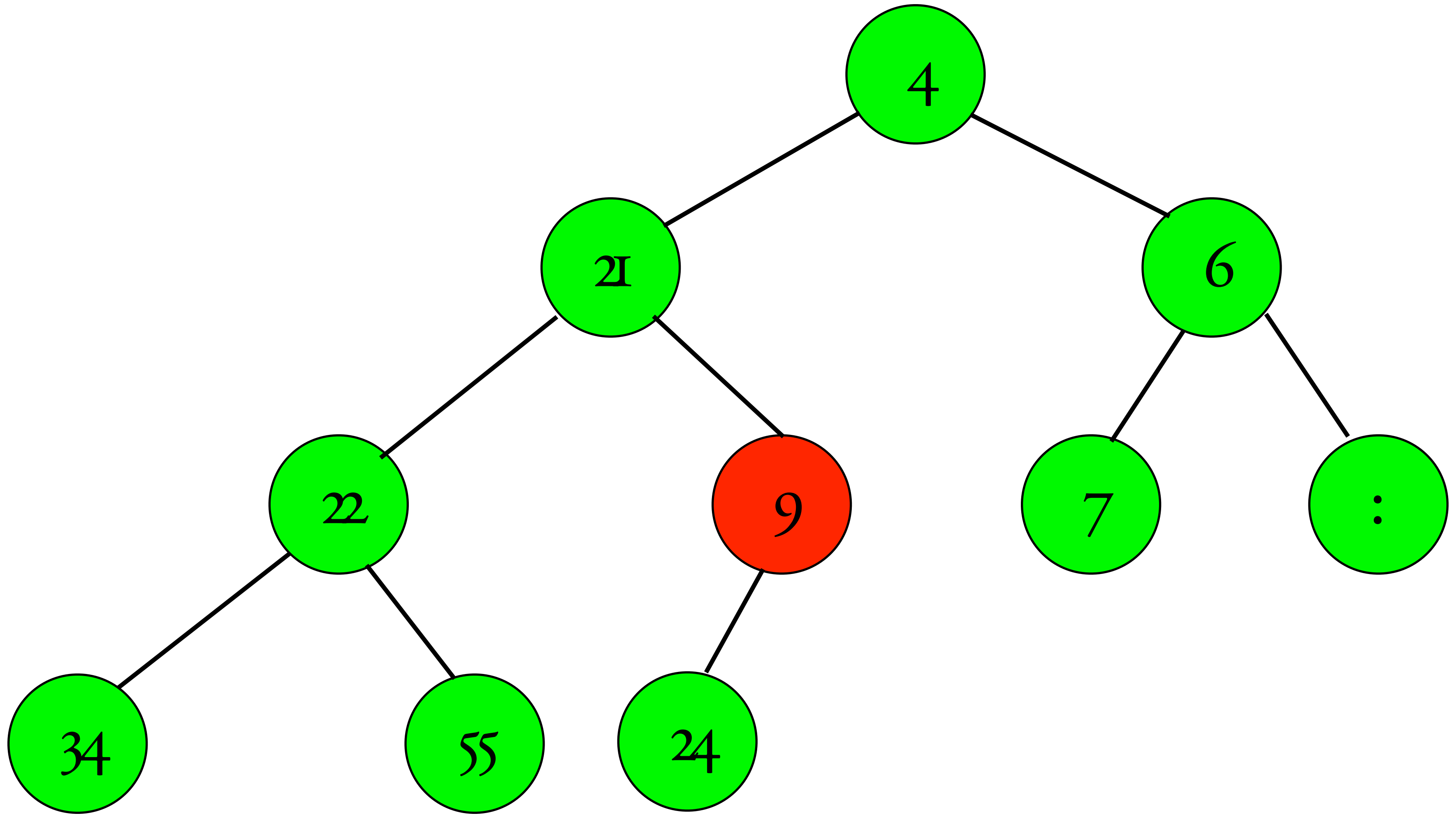
binary heap

full tree, key value <= to key of children



binary heap

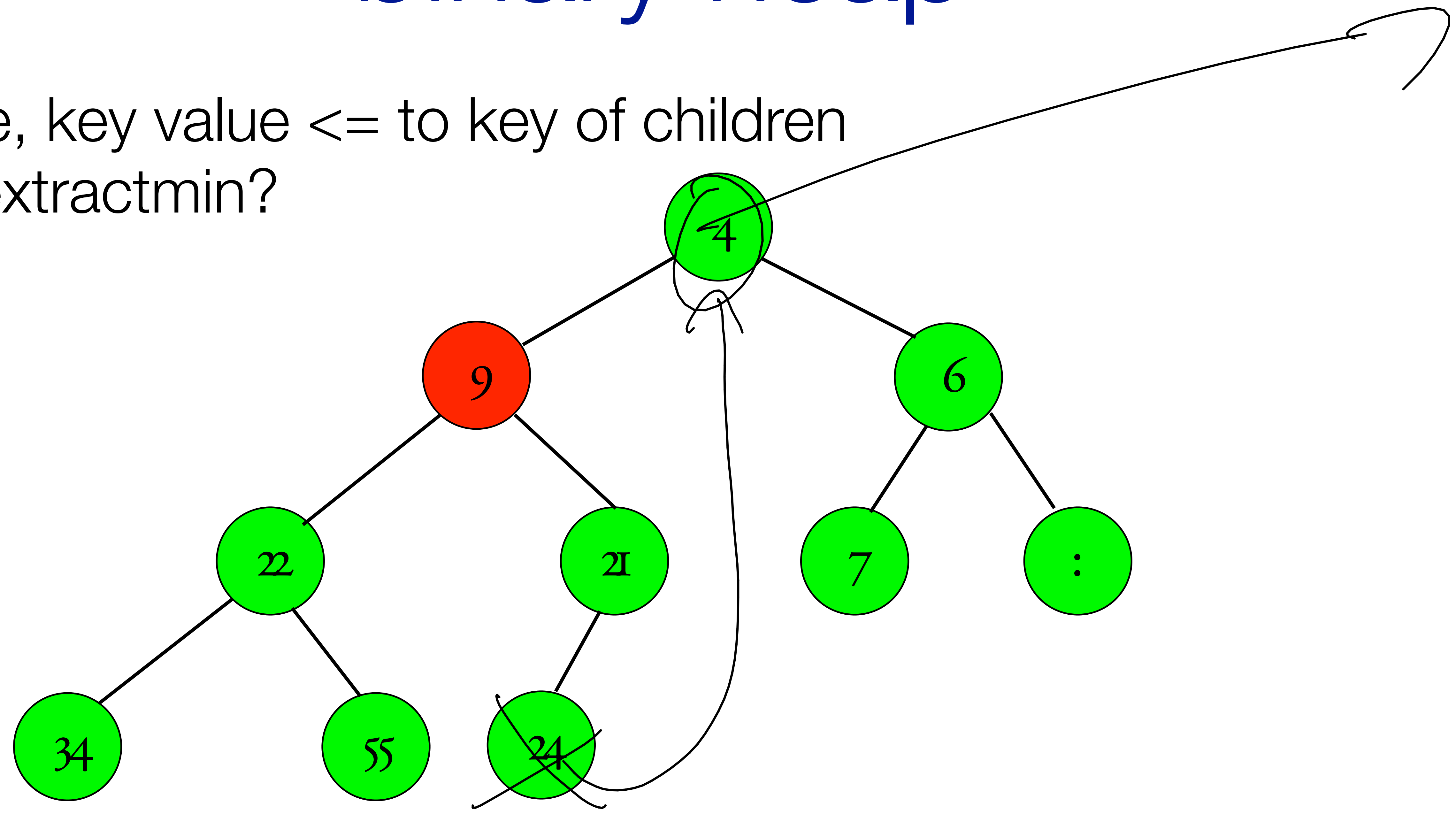
full tree, key value <= to key of children



binary heap

full tree, key value <= to key of children

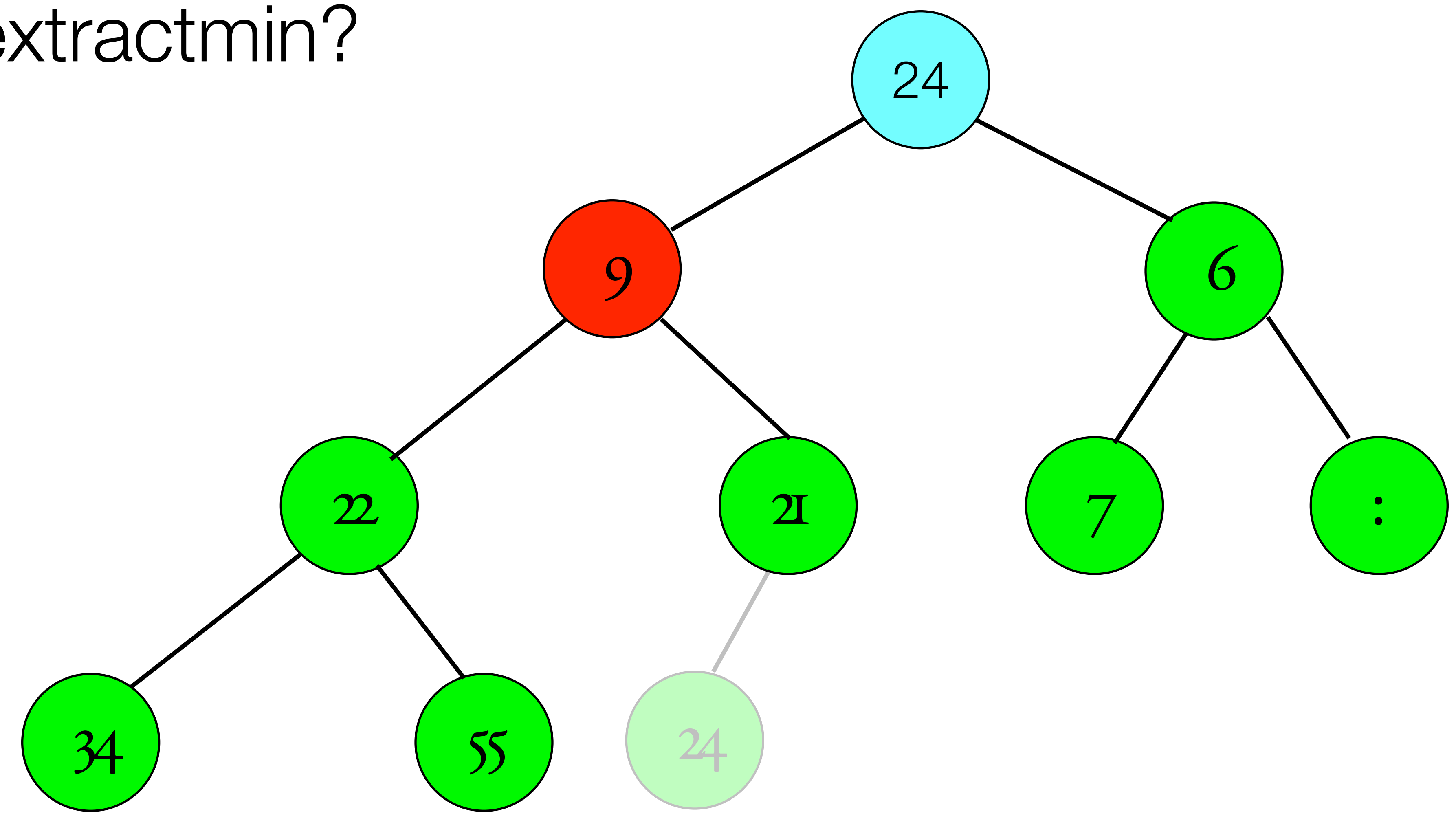
how to extractmin?



binary heap

full tree, key value <= to key of children

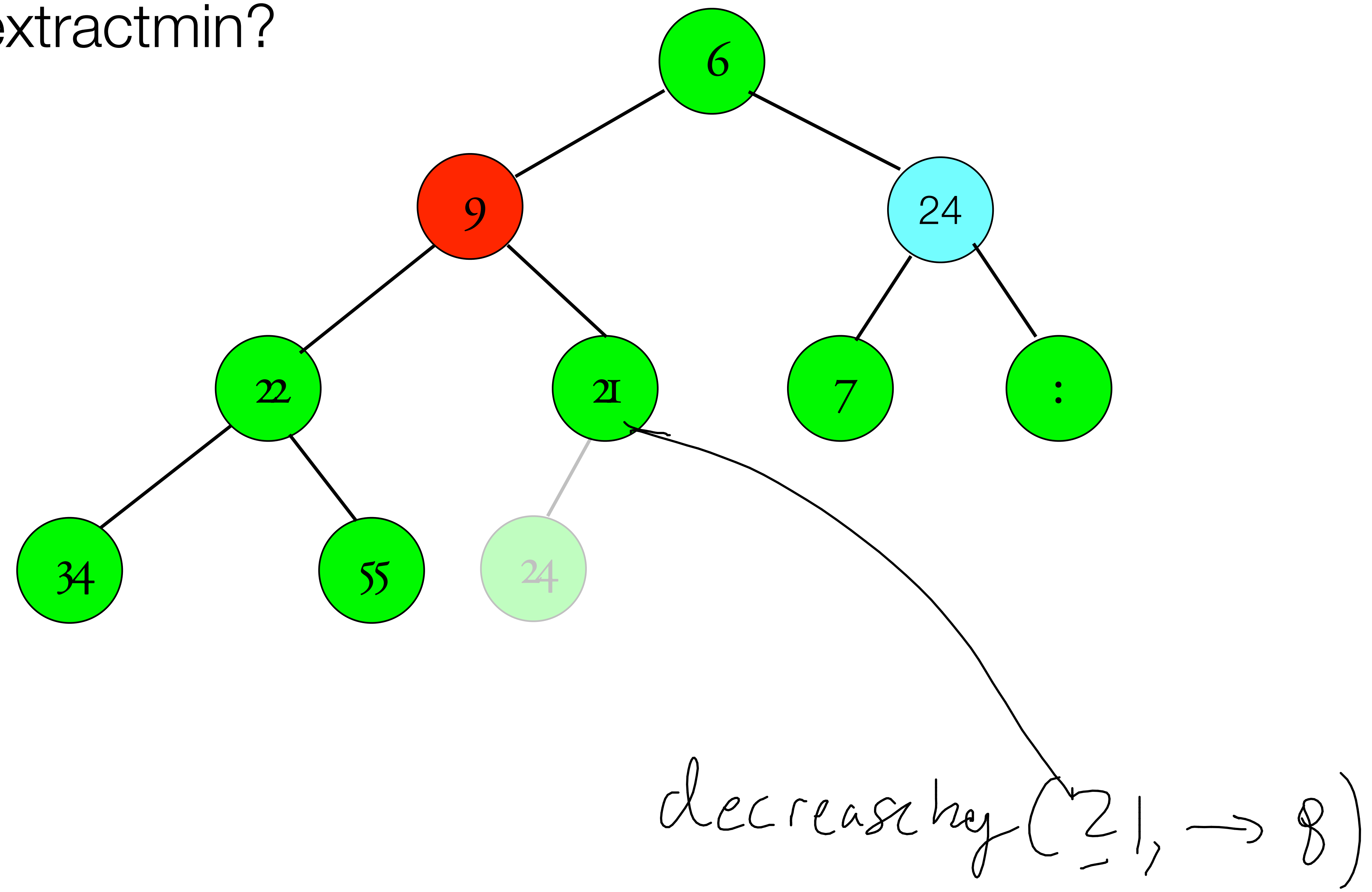
how to extractmin?



binary heap

full tree, key value <= to key of children

how to extractmin?

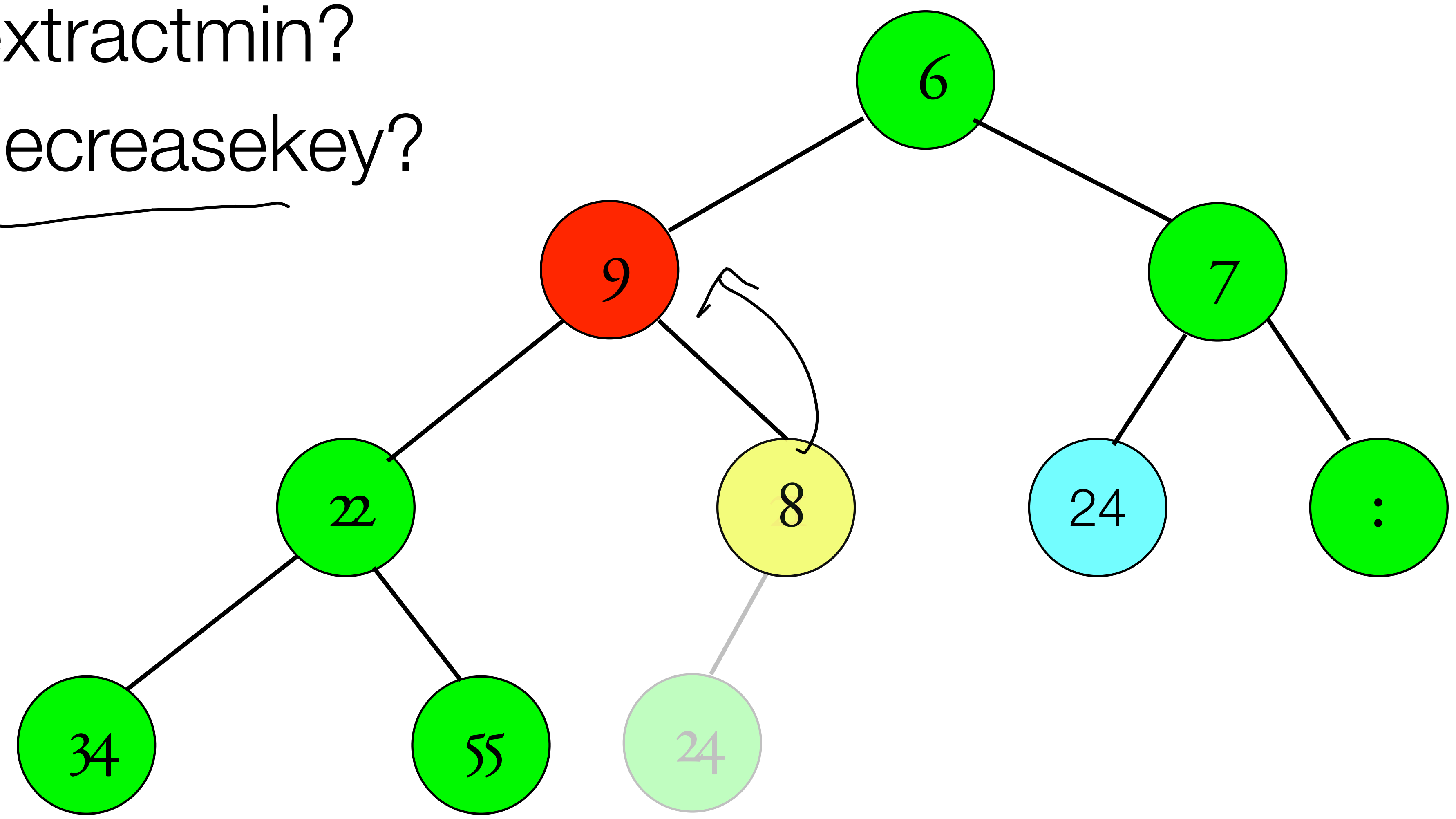


binary heap

full tree, key value <= to key of children

how to extractmin?

how to decreasekey?

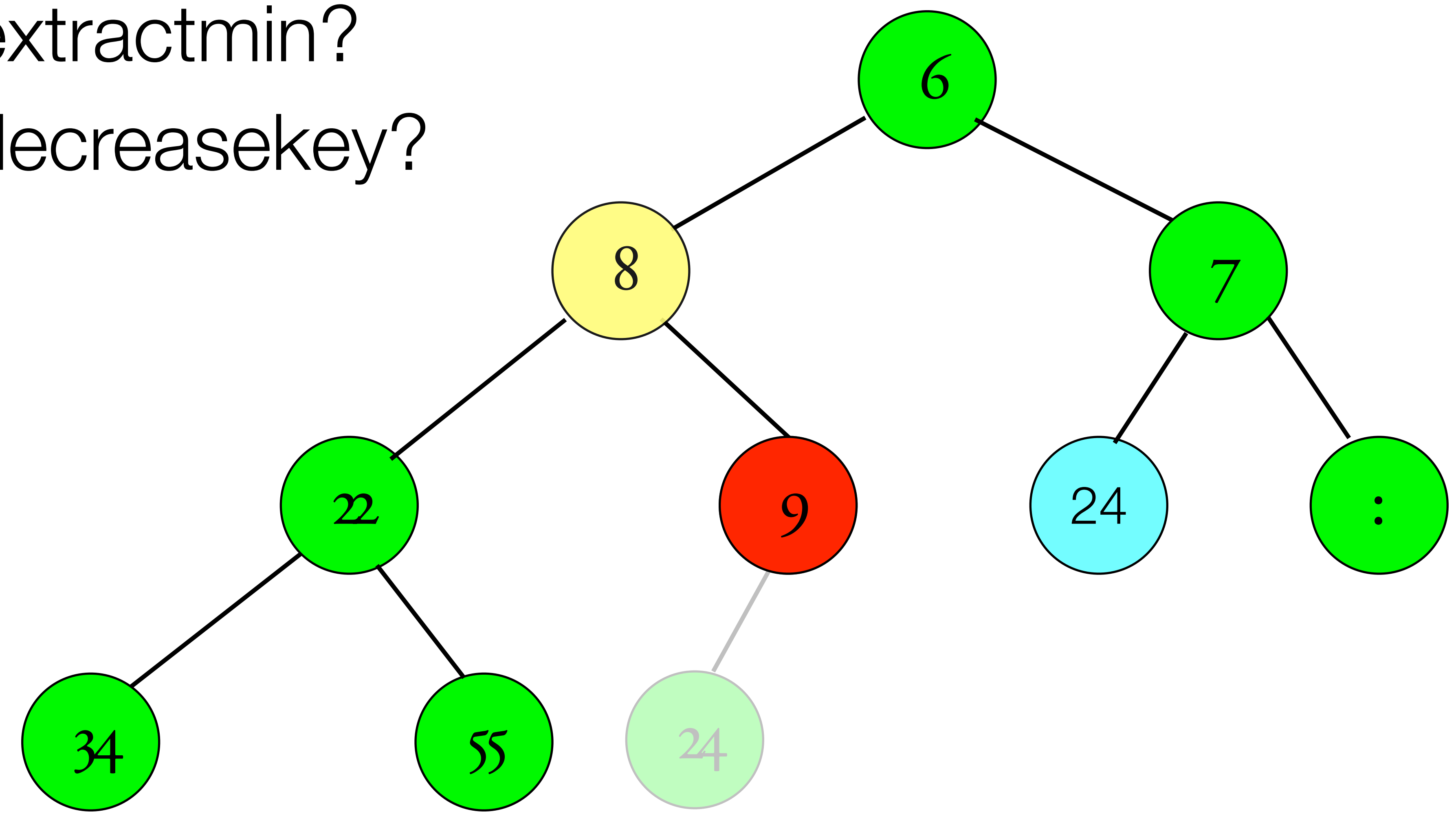


binary heap

full tree, key value <= to key of children

how to extractmin?

how to decreasekey?



implementation

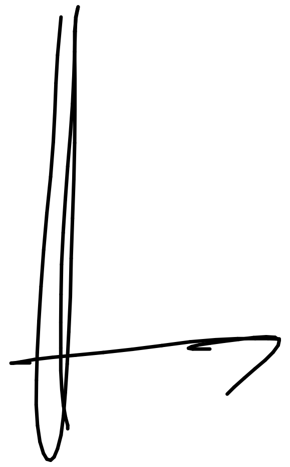
use a priority queue to keep track of light edges

insert: → $\log(n)$

makequeue: →

extractmin: → $\log(n)$

decreasekey: → $\log(n)$



algorithm

makequeue(V), set all keys to ∞

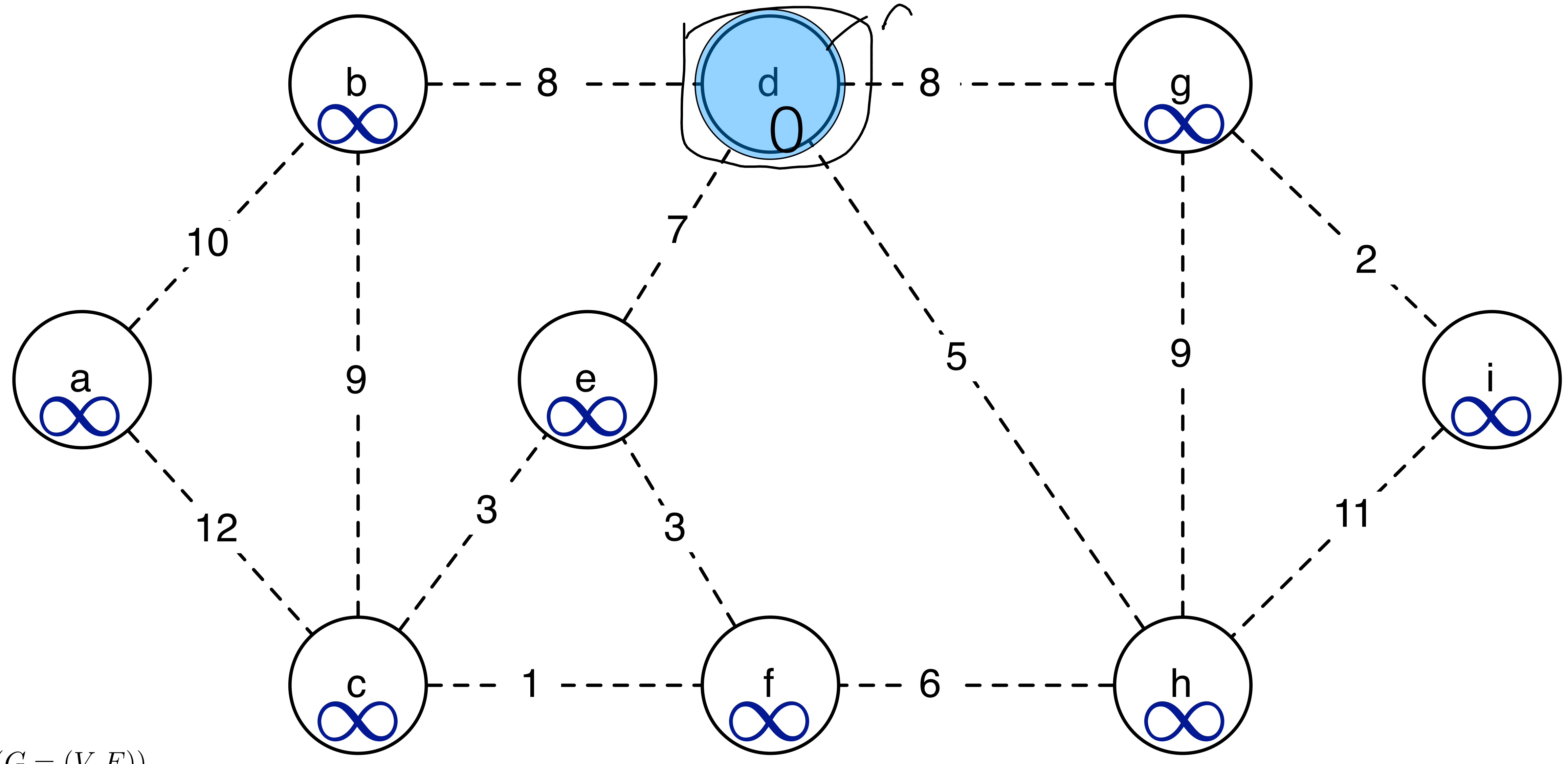
pick some arbitrary node to start, v:

$k_v = 0$

implementation

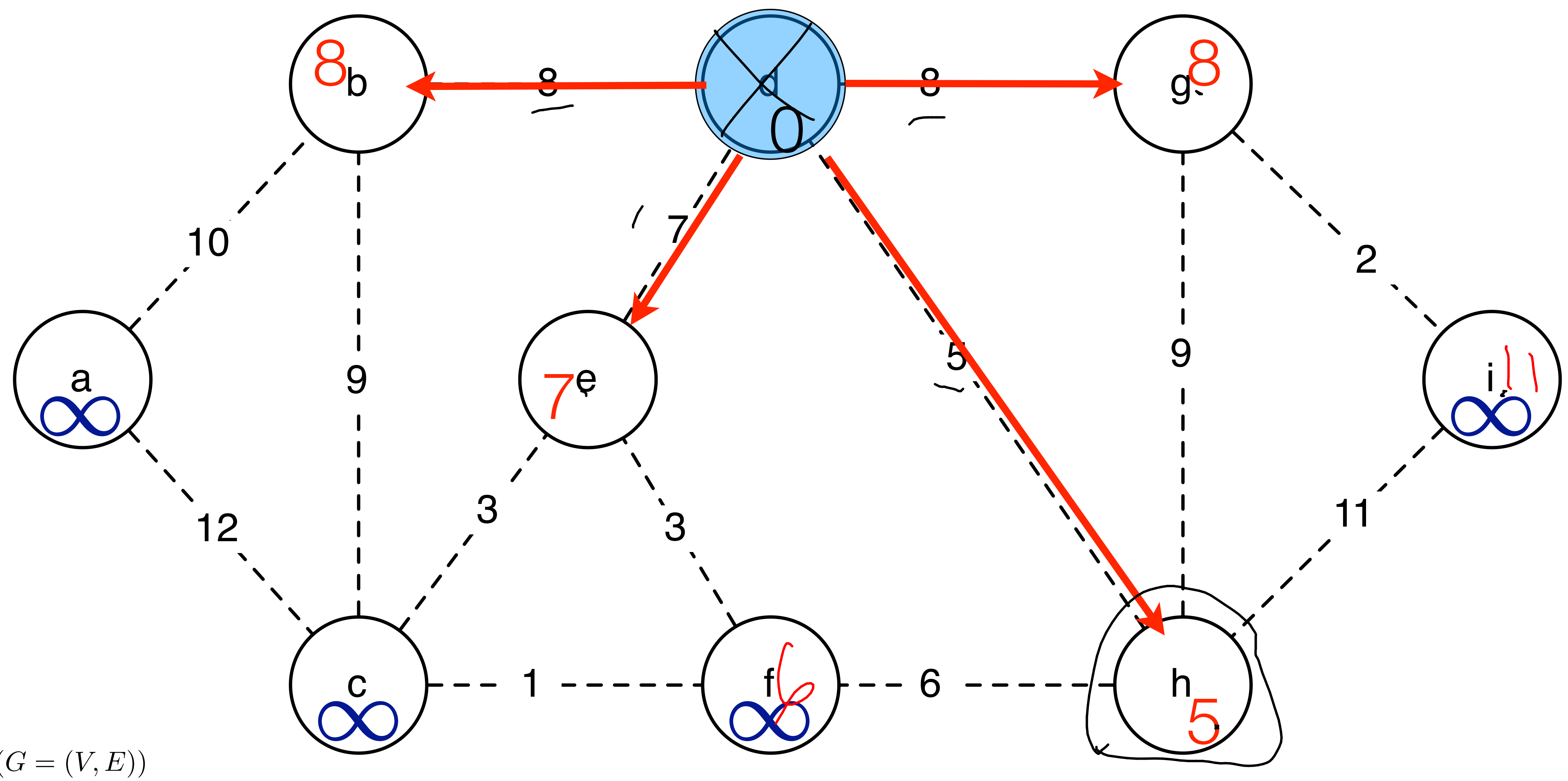
```
PRIM(G = (V, E))
Q ← ∅        ▷ Q is a Priority Queue
Initialize each v ∈ V with key k_v ← ∞, π_v ← NIL
Pick a starting node r and set k_r ← 0
Insert all nodes into Q with key k_v.
while Q ≠ ∅
      do u ← EXTRACT-MIN(Q)
         for each v ∈ Adj(u)
             do if v ∈ Q and w(u, v) < k_v
   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

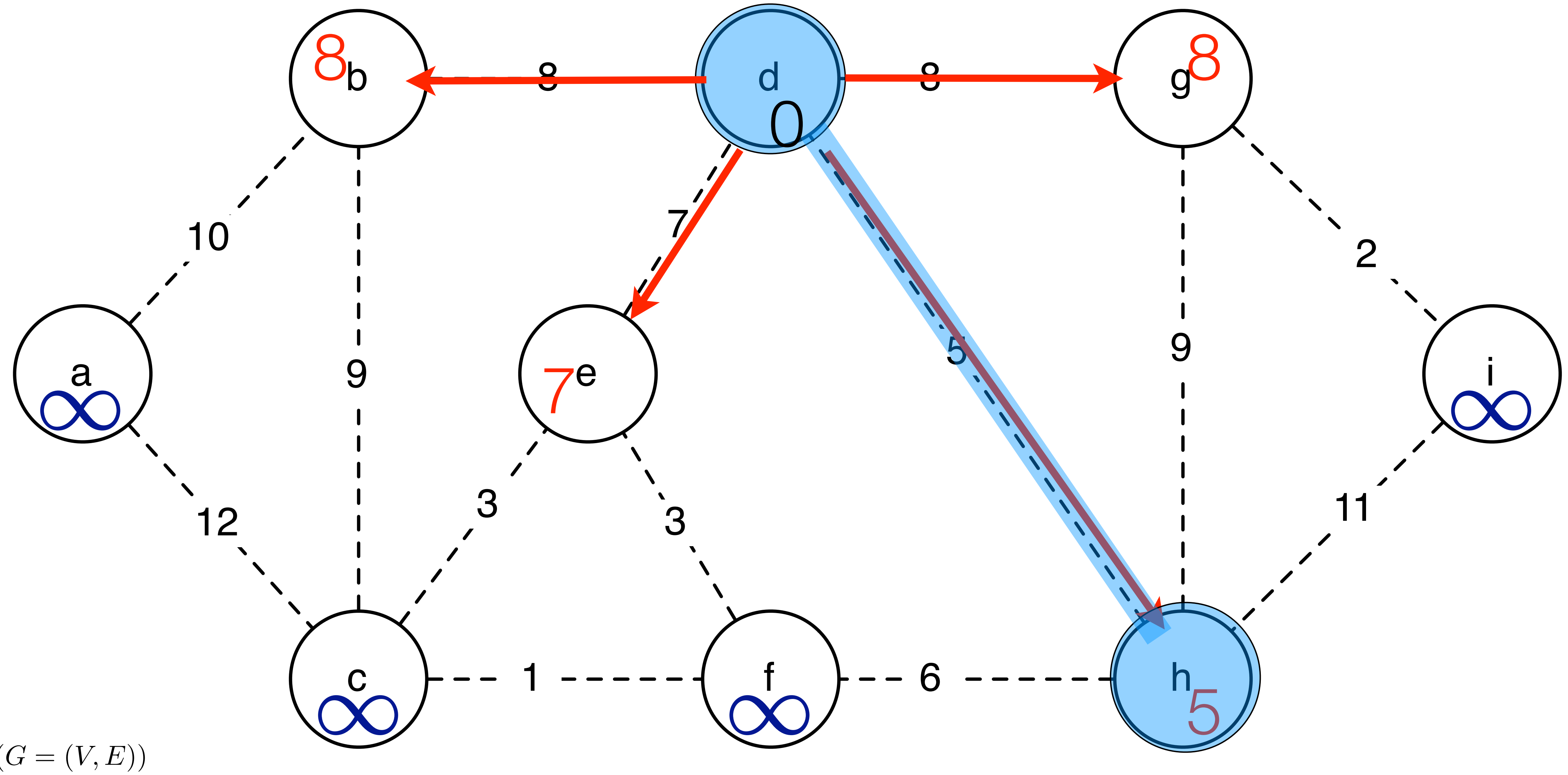
prim



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

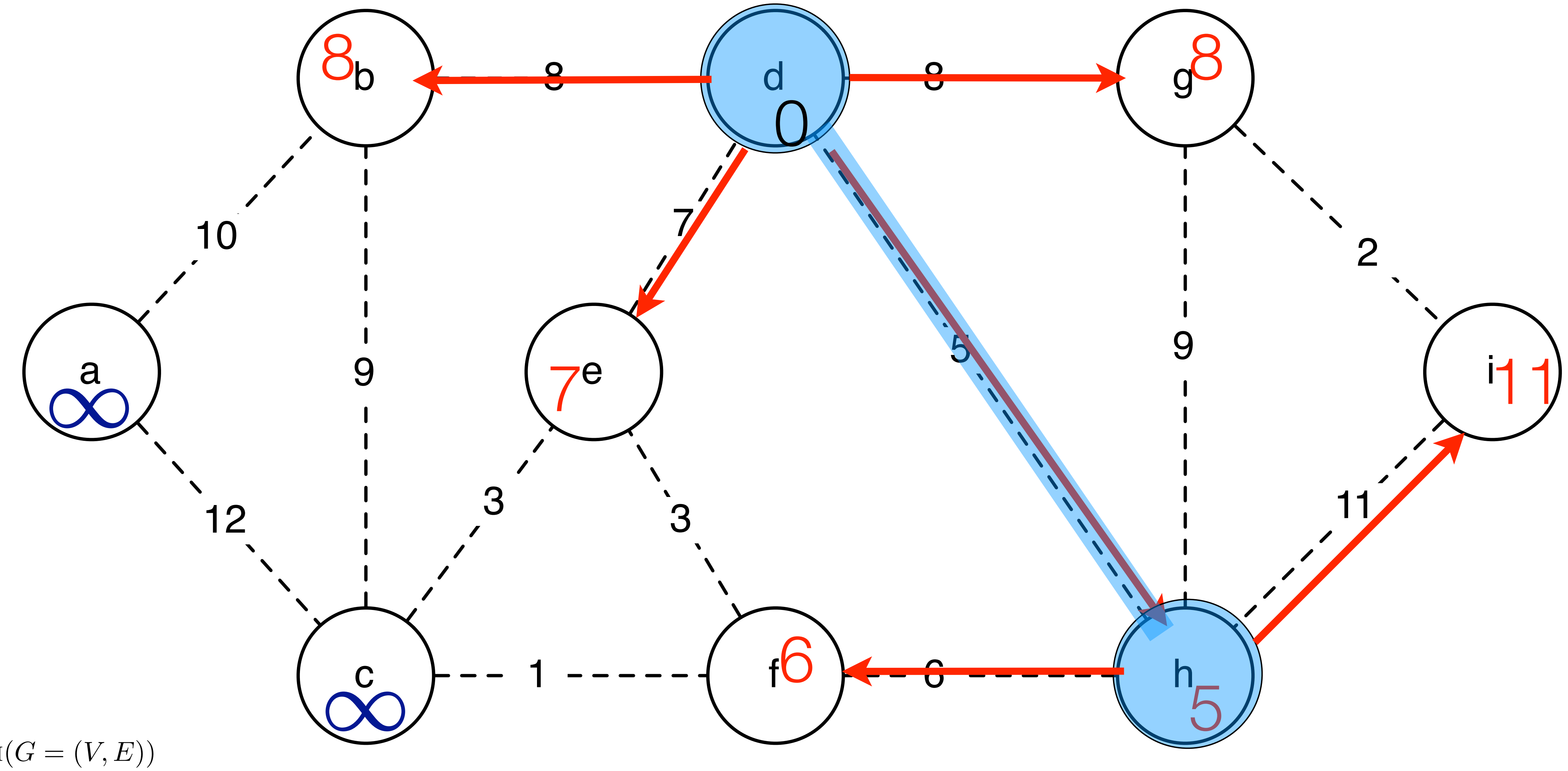
```
Q ← ∅        ▷ Q is a Priority Queue
Initialize each v ∈ V with key k_v ← ∞, π_v ← NIL
Pick a starting node r and set k_r ← 0
Insert all nodes into Q with key k_v.
while Q ≠ ∅
      do u ← EXTRACT-MIN(Q)
         for each v ∈ Adj(u)
             do if v ∈ Q and w(u,v) < k_v
                  then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

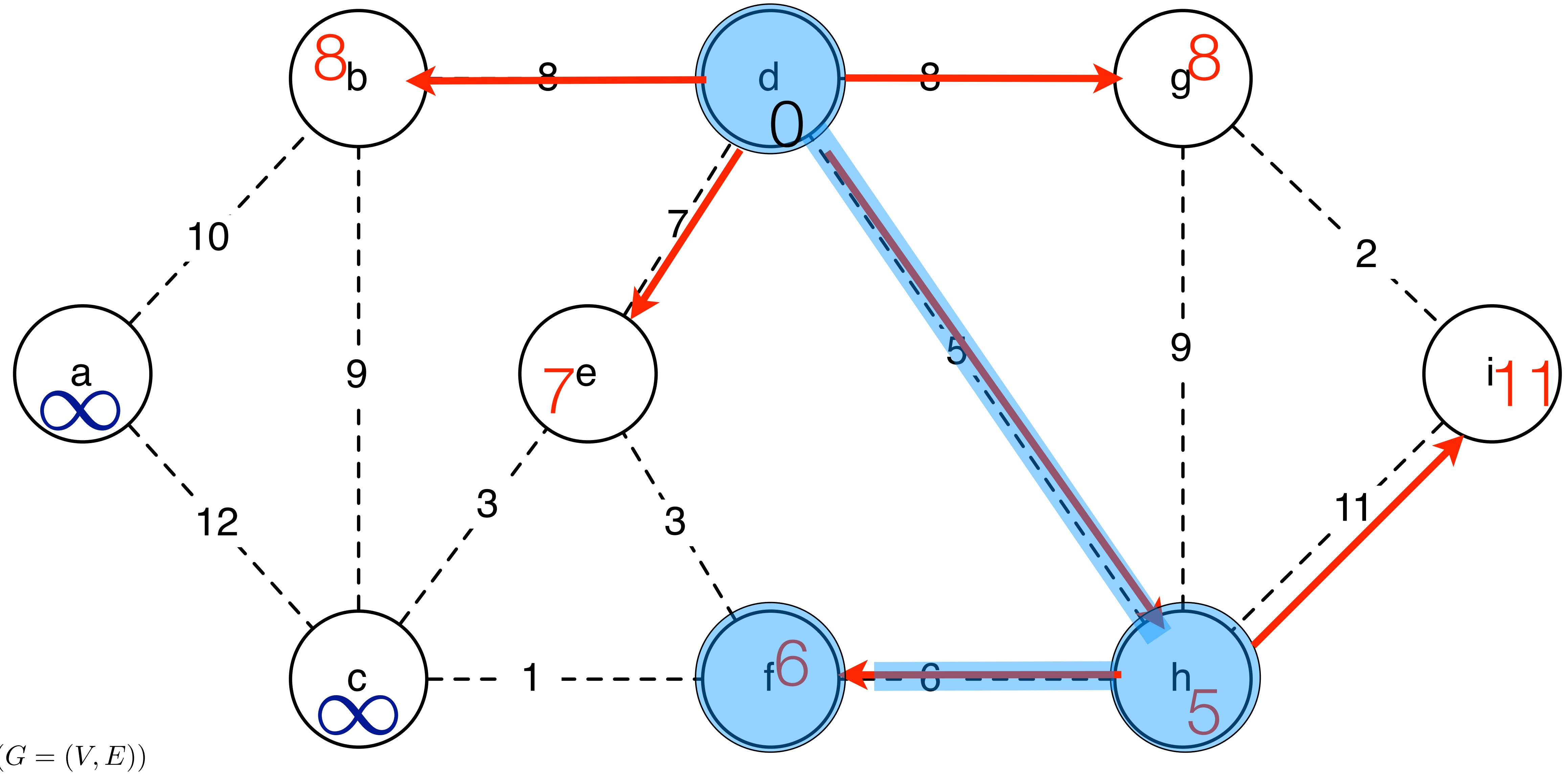
prim



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

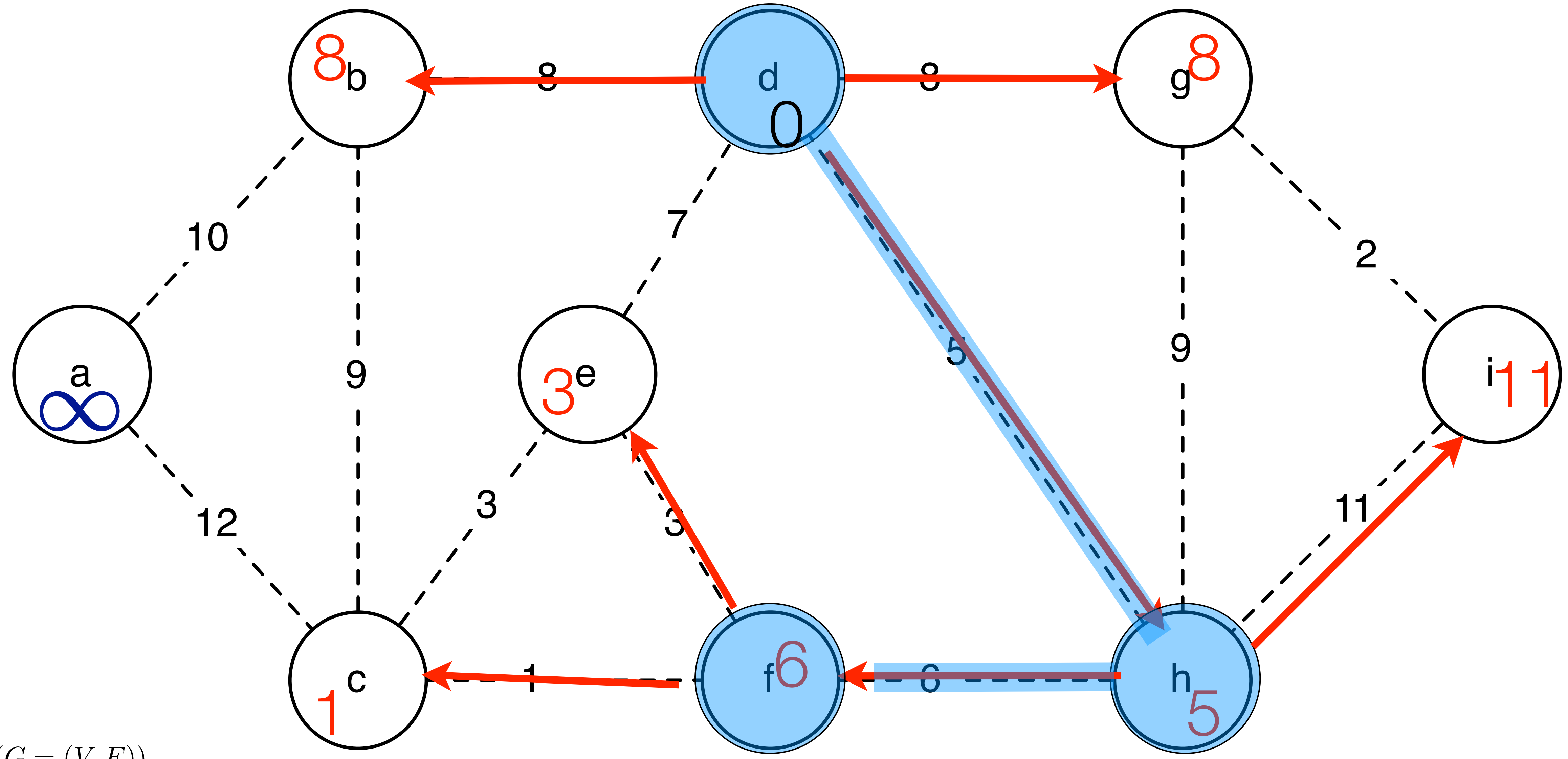
$Q \leftarrow \emptyset$ ▷ Q is a Priority Queue
Initialize each $v \in V$ with key $k_v \leftarrow \infty$, $\pi_v \leftarrow$ NIL
Pick a starting node r and set $k_r \leftarrow 0$
Insert all nodes into Q with key k_v.
while $Q \neq \emptyset$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    for each $v \in Adj(u)$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

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 ← ∅    ▷ 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)
```

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)
```

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

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: E
(randomized)

euclidean mst: $V \log V$

ackerman function

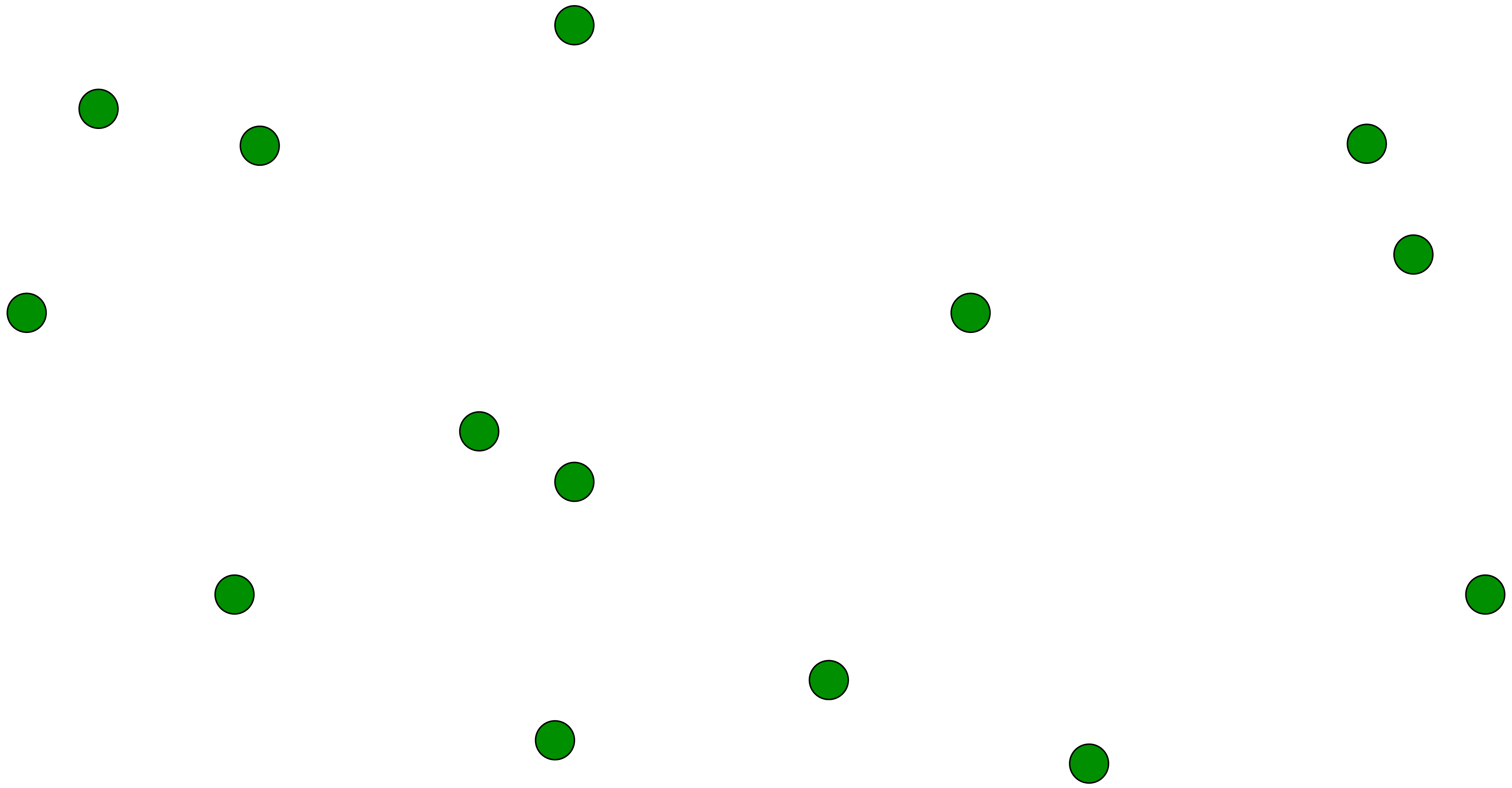
$$A(m,n) = \begin{cases} n+1 & m = 0 \\ A(m-1,1) & m > 0,\ n = 0 \\ A(m-1, A(m,n-1)) & m,n > 0 \end{cases}$$

$$A(4,2) =$$

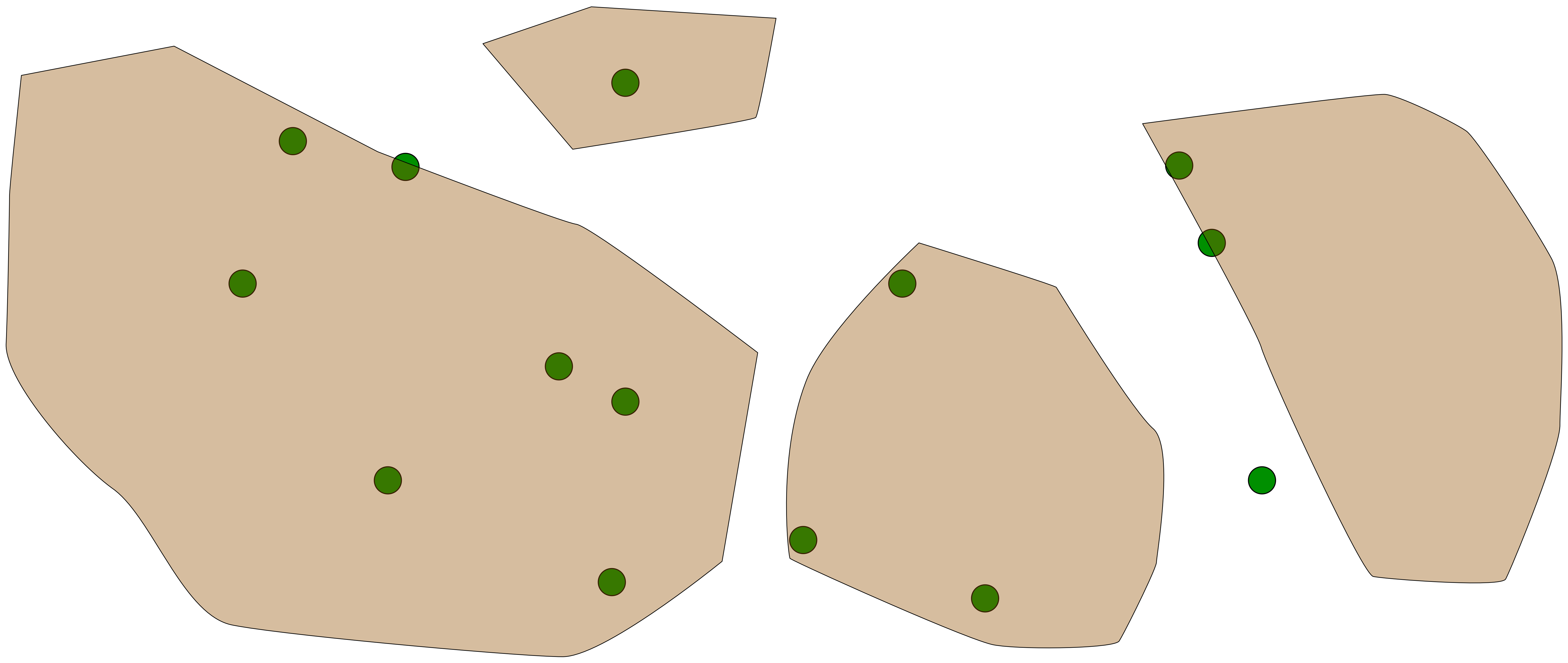
inverse ackerman

$\alpha(n) =$

application of mst



application of mst



application of mst