L16

4102

3.22.2016

abhi shelat

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

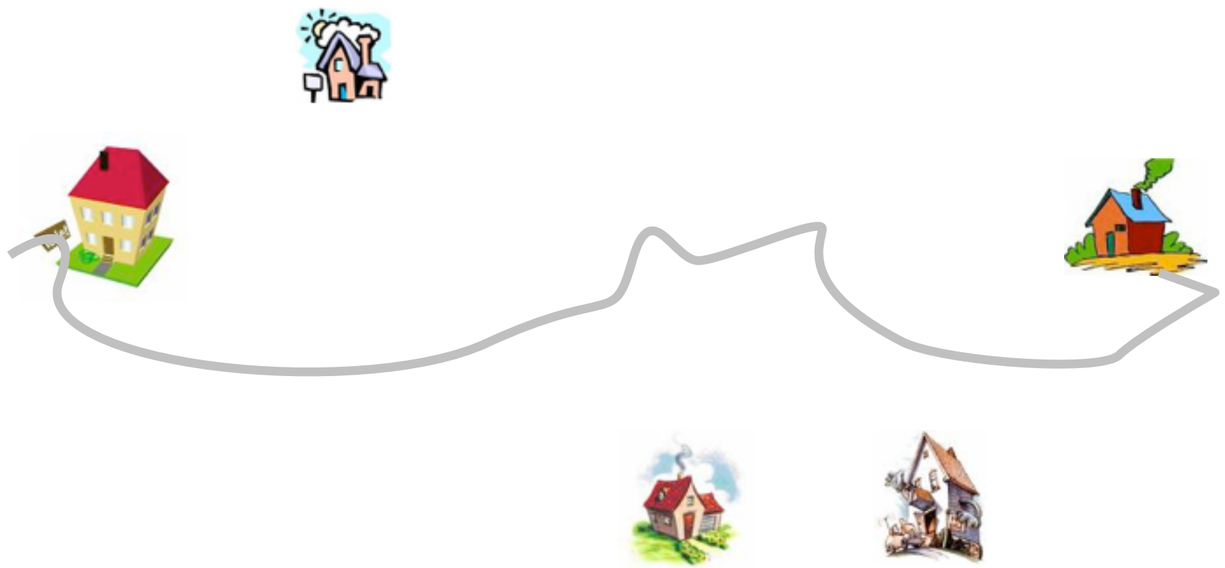


image:www.princegeorgeva.org, thefranciscofamily.org, www.rightdriveacademy.co.uk, www.ccscambridge.org, www.drawingcoach.com, www.pastoral.org.uk, www.daasgallery.com

connecting houses

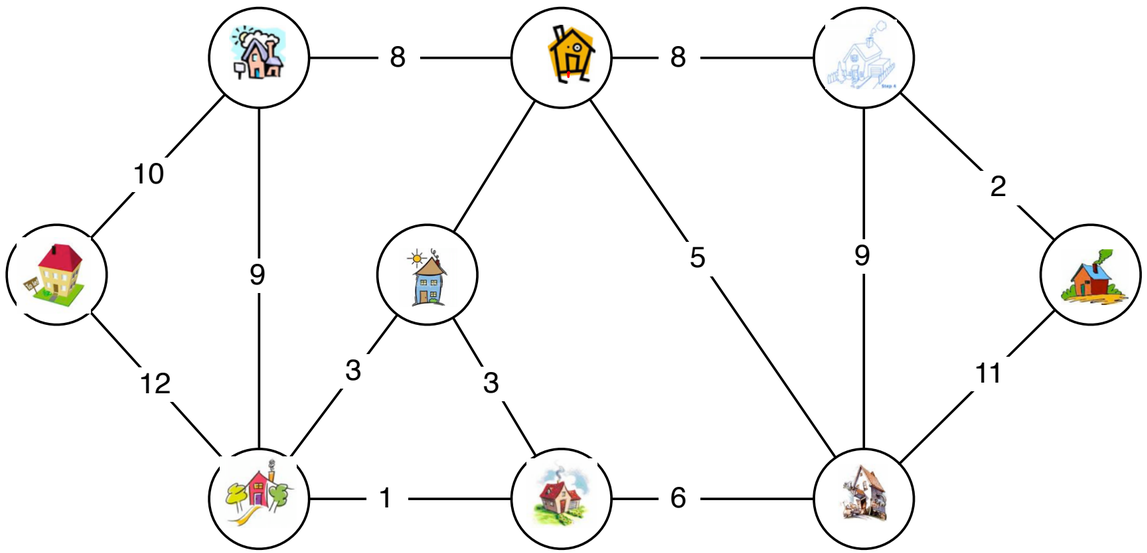
image:www.princegeorgeva.org, thefranciscofamily.org, www.rightdriveacademy.co.uk, www.ccscambridge.org, www.drawingcoach.com, www.pastoral.org.uk, www.daasgallery.com

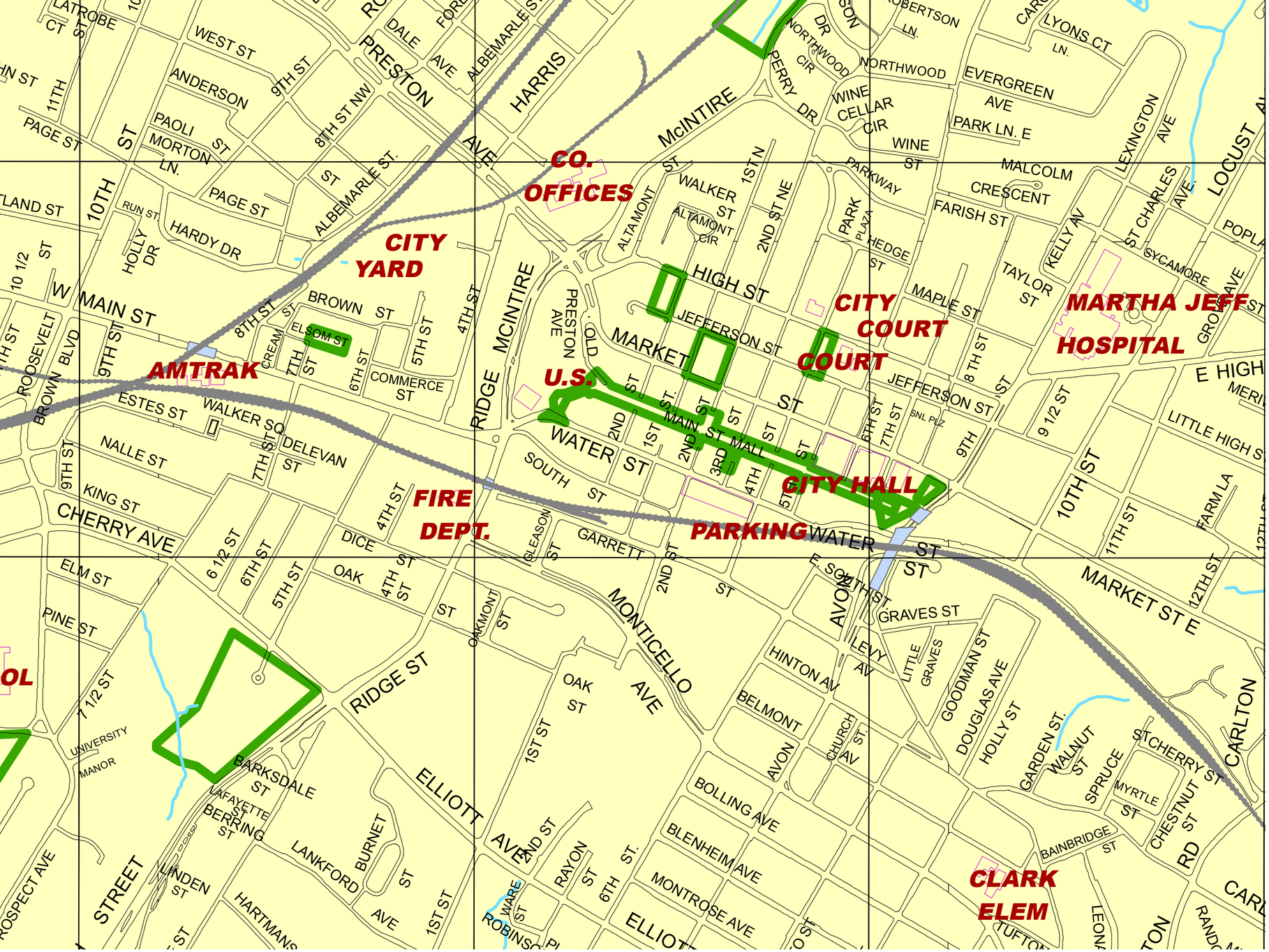
connecting houses

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



MINIMUM SPANNING TREE.

graphs

clrs [ch 22]

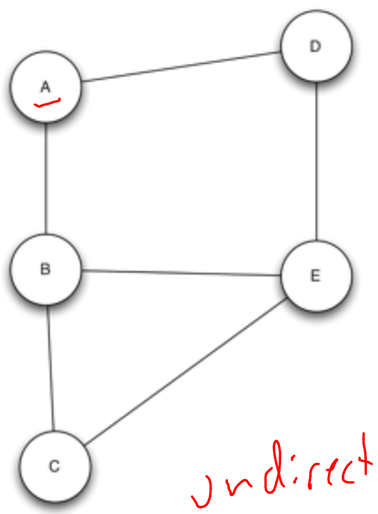
vertices

edges

$$G = (V, E)$$

w(e): $E \longrightarrow \mathbb{N}$

weights for each edge



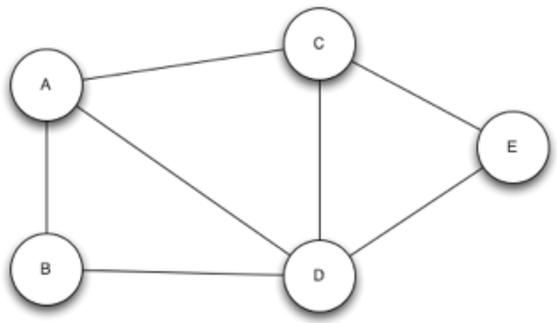
undirected.

$V = \{A, B, C, D, E\}$

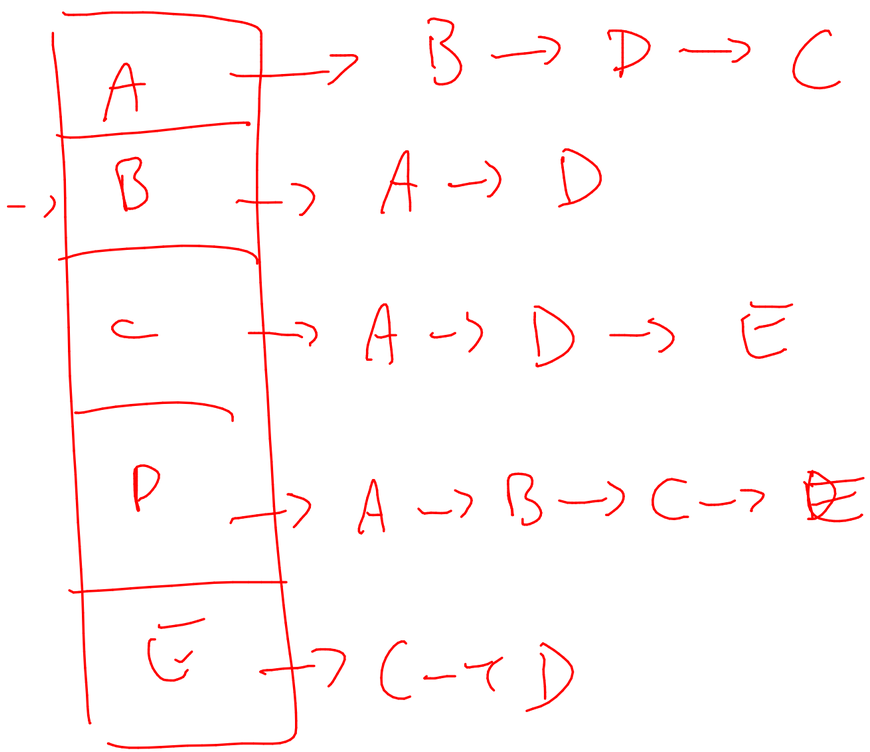
$E = \{(A,B), (A,D), (B,C), (B,E), (C,E), (D,E), (B,A), (D,A), \ldots\}$

representation

$$G = (V, E)$$



adjacency list



space: $|V|+|E|$

time list neighbors: $degree(u)$

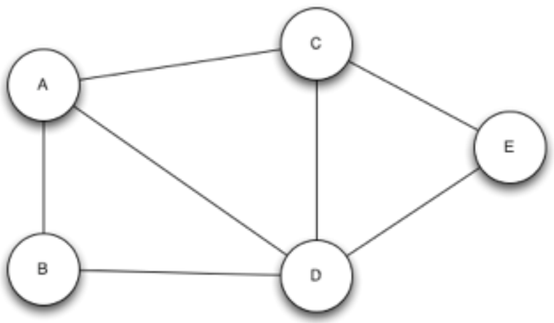
time check an edge:

↳ $degree(A)$

$(A, E) \stackrel{?}{\in} E$??

representation

$$G = (V, E)$$

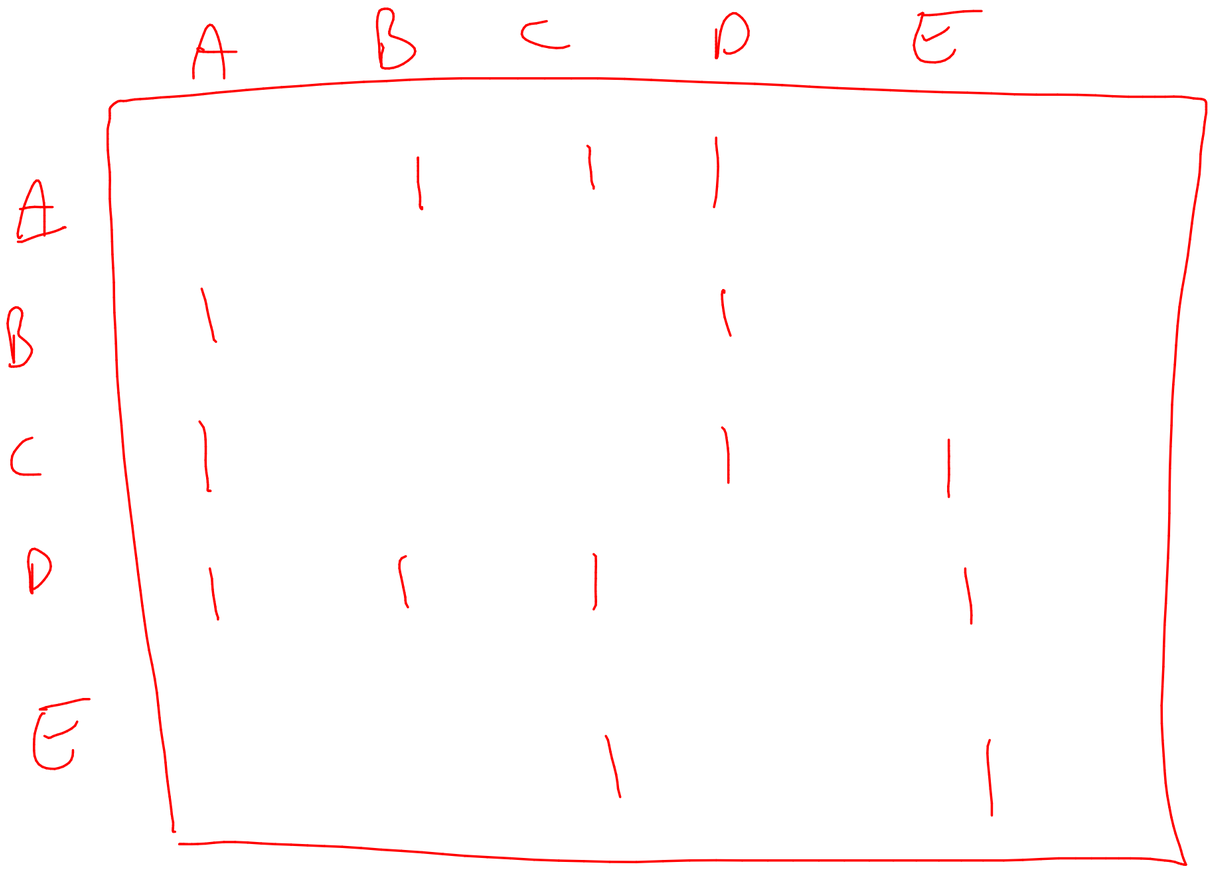


adjacency matrix

space: V^2

time list neighbors: V

time check an edge: $O(1)$



definition: path

- a sequence of nodes $v_1, v_2, \ldots, v_k$
  with the property that $(v_i, v_{i+1}) \in E$ for all i=1,..,k-1

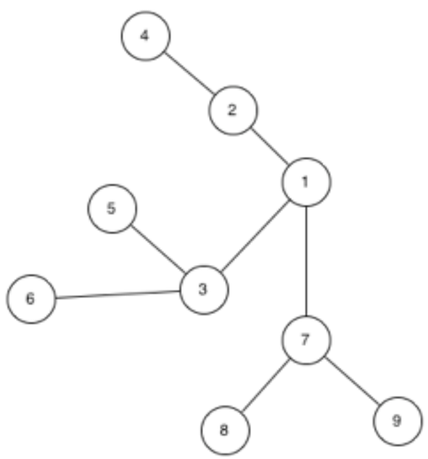
simple path: is a path in which each vertex appears at most once

cycle: is a path of length > 2 such that $v_1 = v_k$.

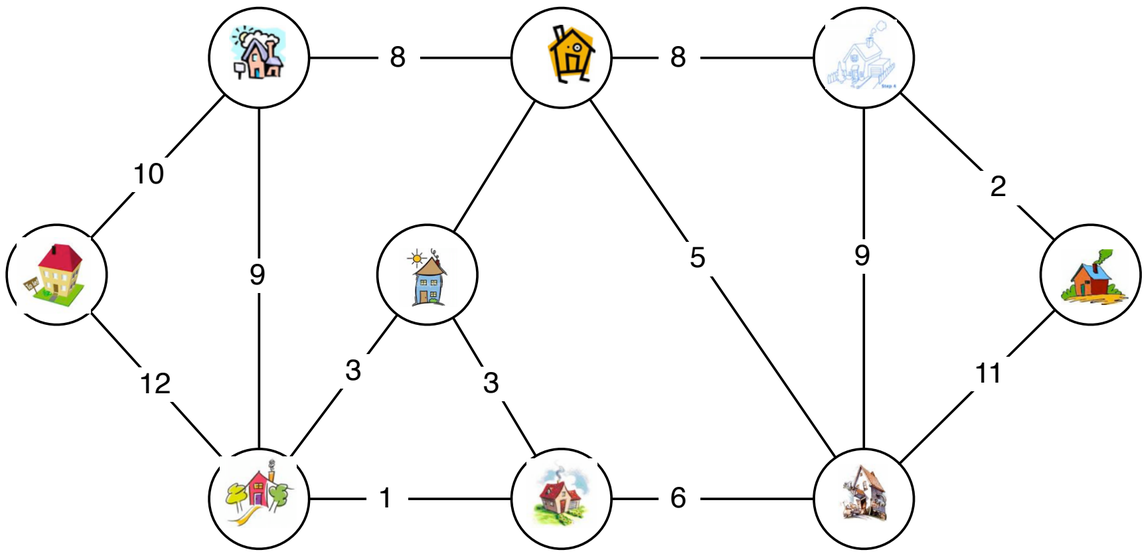
definition: tree

connected graph: is a graph $G=(V,E)$ such that for any pair $u,v \in V$, there exists a path from u to v.

a tree is connected graph with no cycles.



G=(V,E,w)



we want: a tree that spans G, and has minimal cost.

connects all V

minimum spanning tree

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

(a) connects all vertices

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

minimum spanning tree

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

(a) connects all vertices

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

- how many edges does solution have ? $V-1$ edges.
- does solution have a cycle? No

strategy

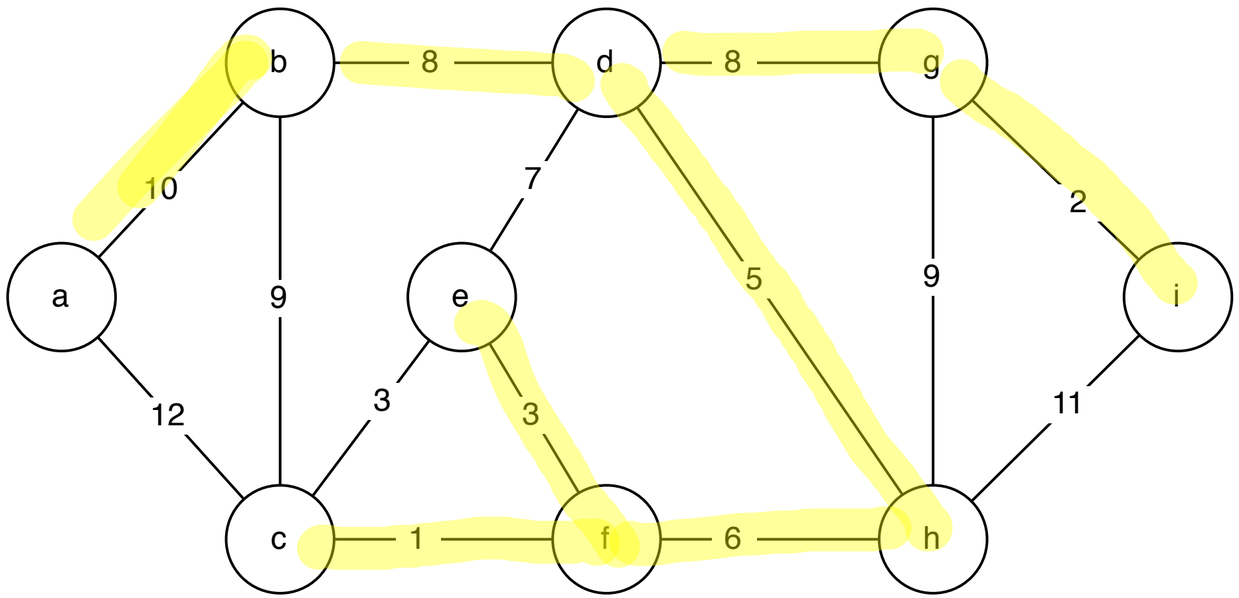
start with an empty set of edges A

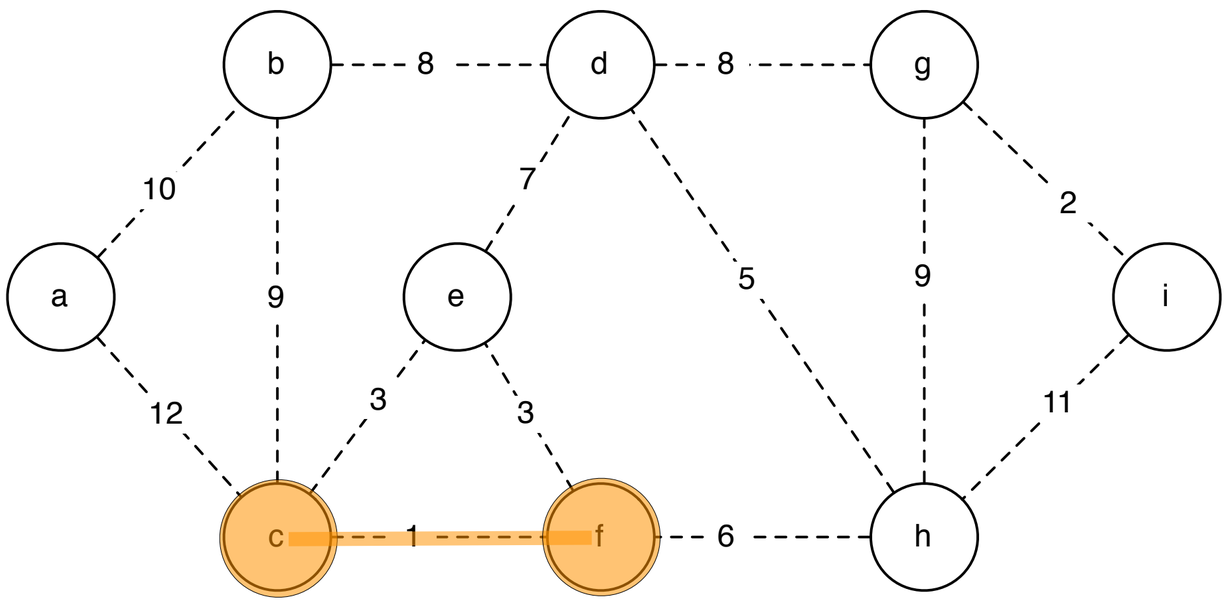
repeat for v-1 times:

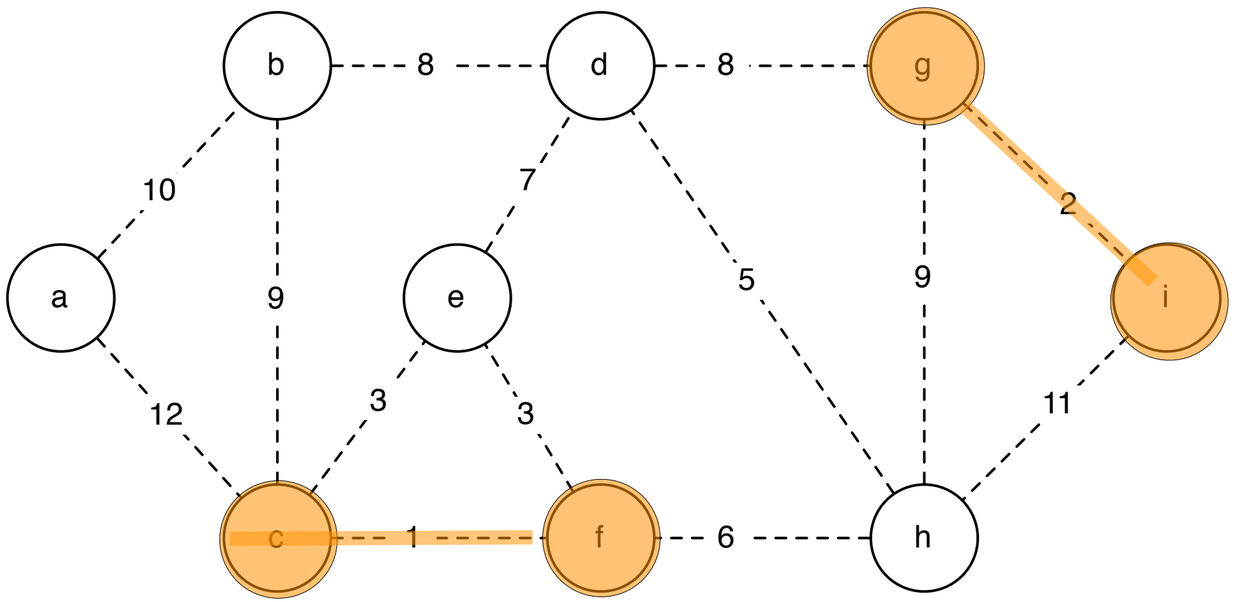
add lightest edge that does not create a cycle

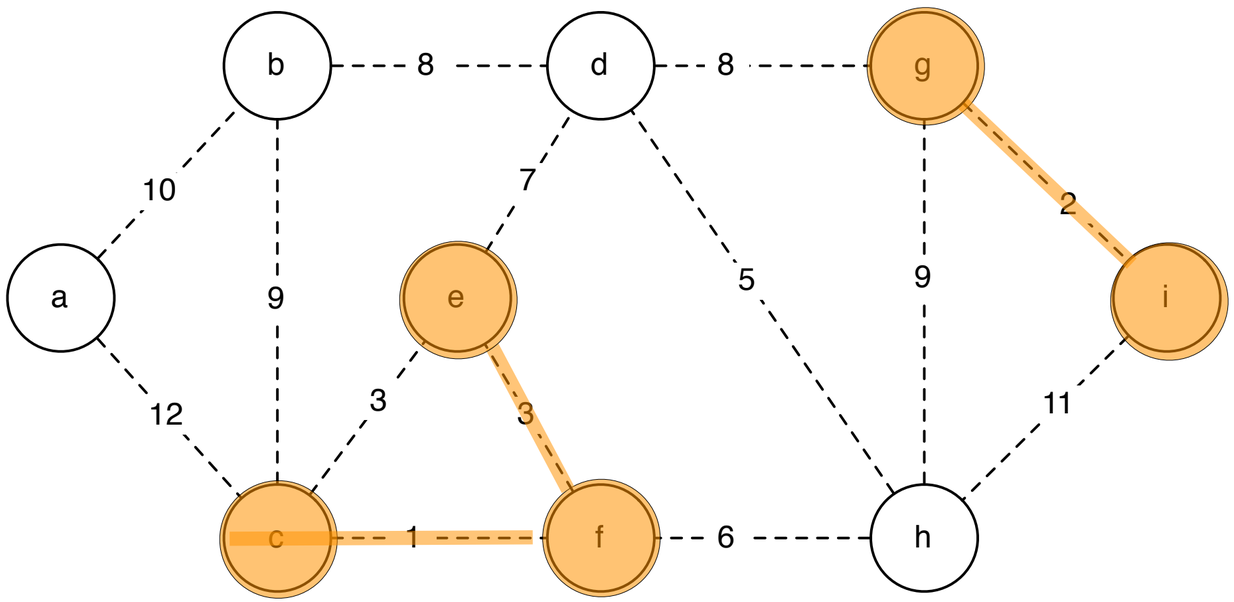
example

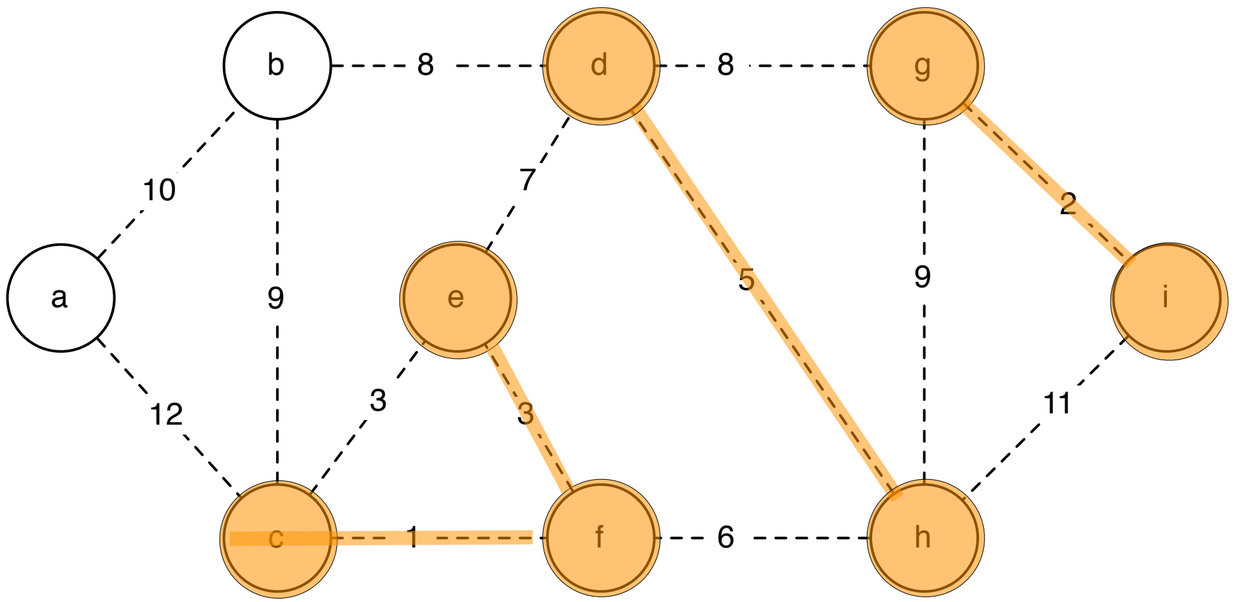
$A = \emptyset$

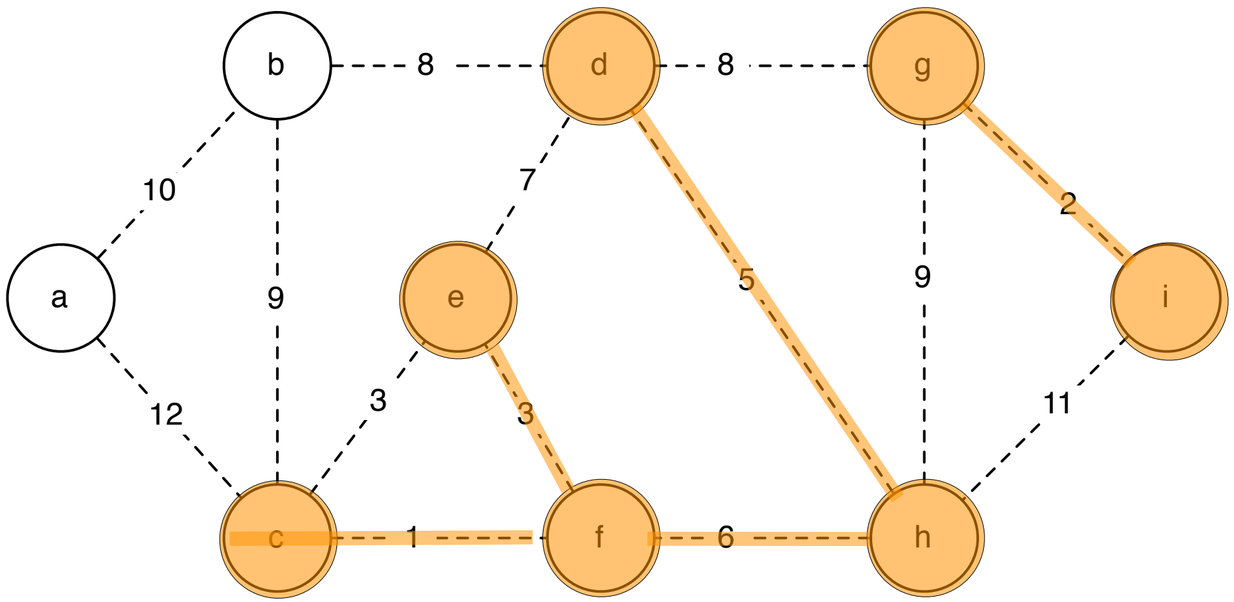


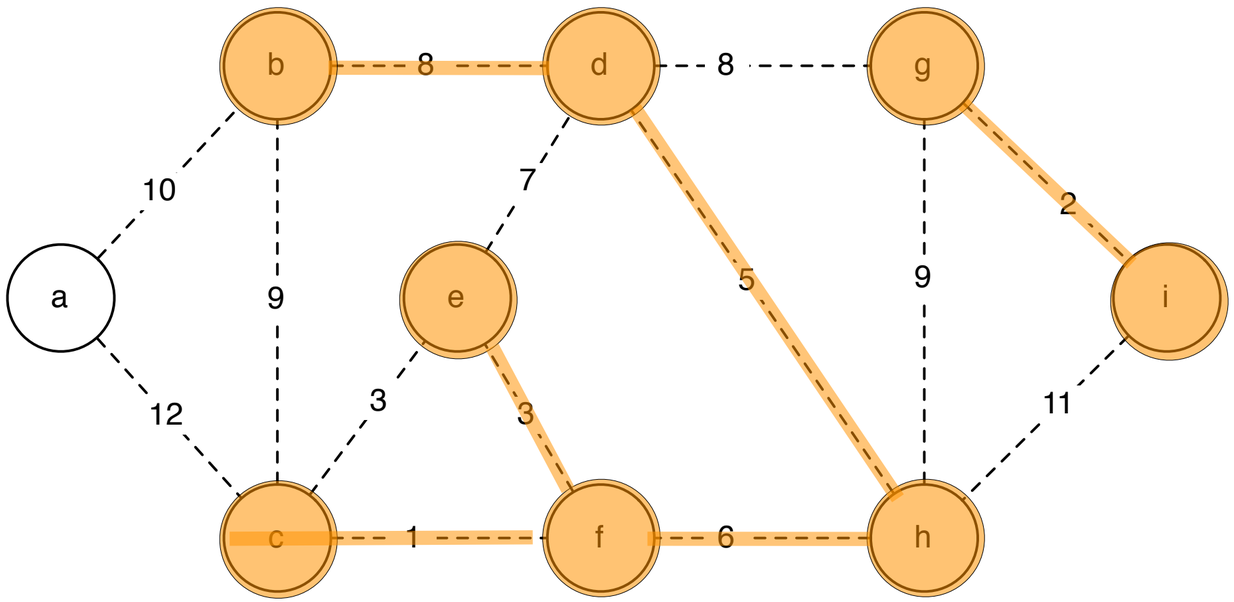

b
8
d
8
g
10
7
2
a
9
e
5
9
i
12
3
3
11
c
1
f
6
h

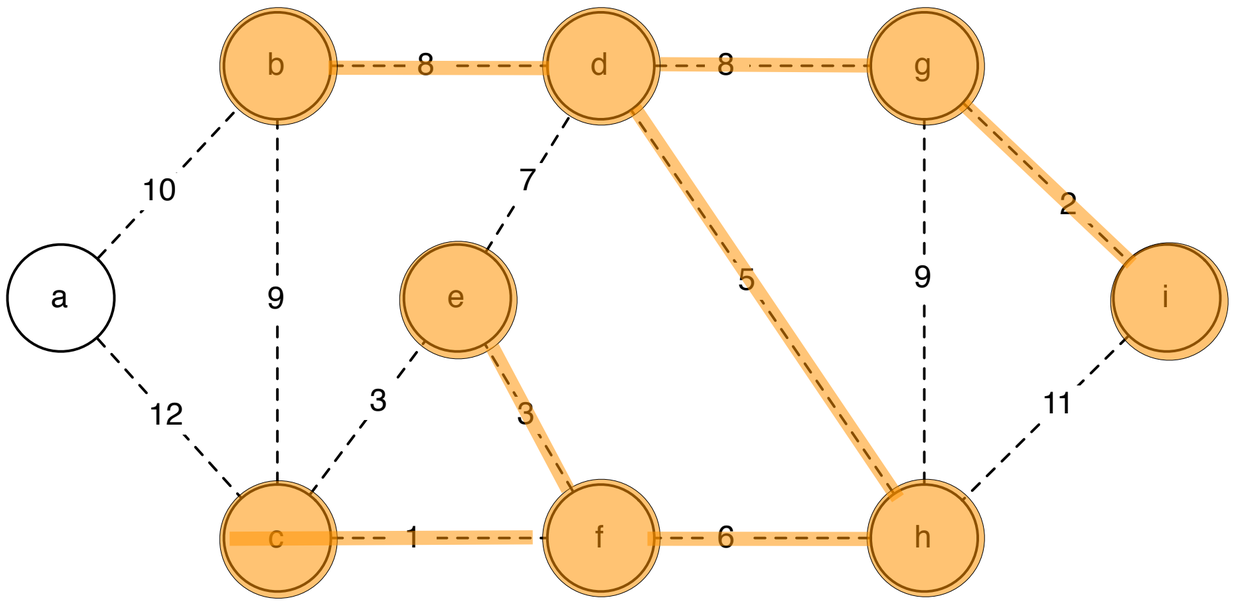

b
8
d
8
g
10
7
2
a
9
e
5
9
i
12
3
3
11
c
1
f
6
h

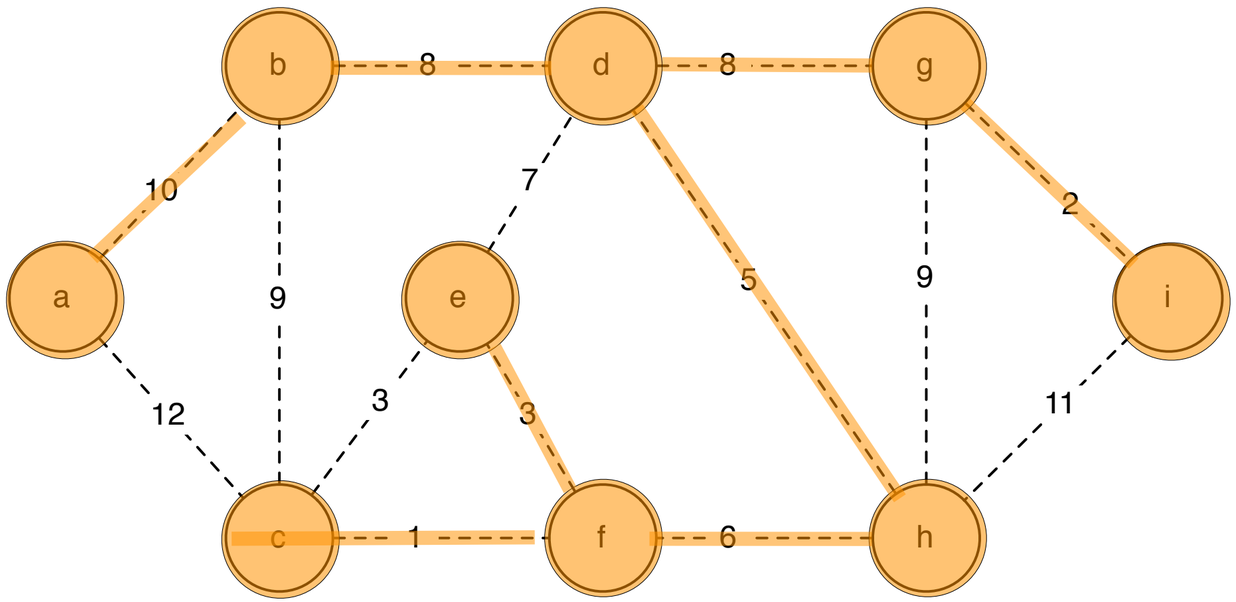

b
8
d
8
g
10
7
2
a
9
e
5
9
i
12
3
3
11
c
1
f
6
h


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

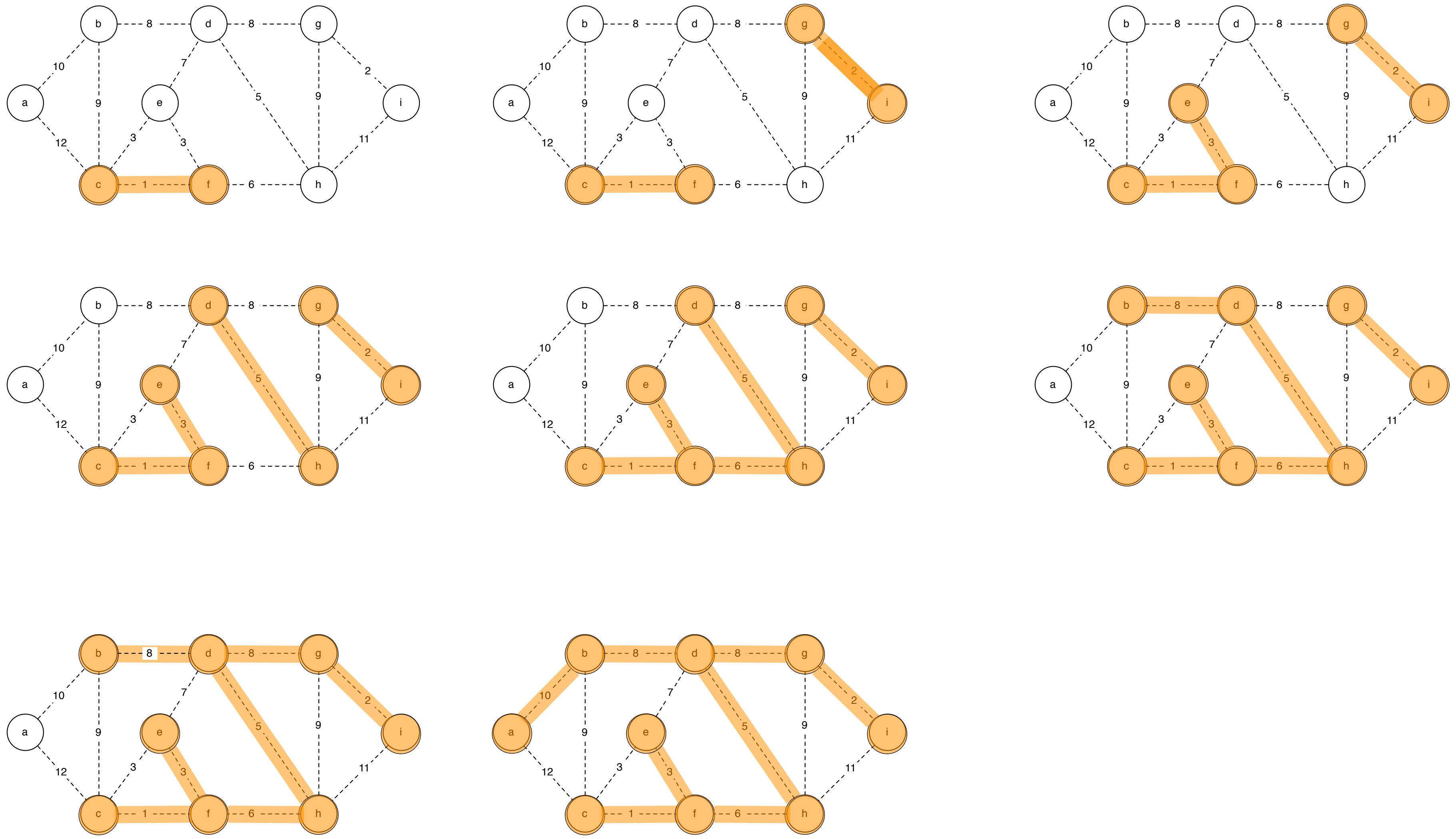

b
8
d
8
g
10
7
2
a
9
e
5
9
i
12
3
3
11
c
1
f
6
h


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


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


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

Kruskal’s algorithm



why does this work?

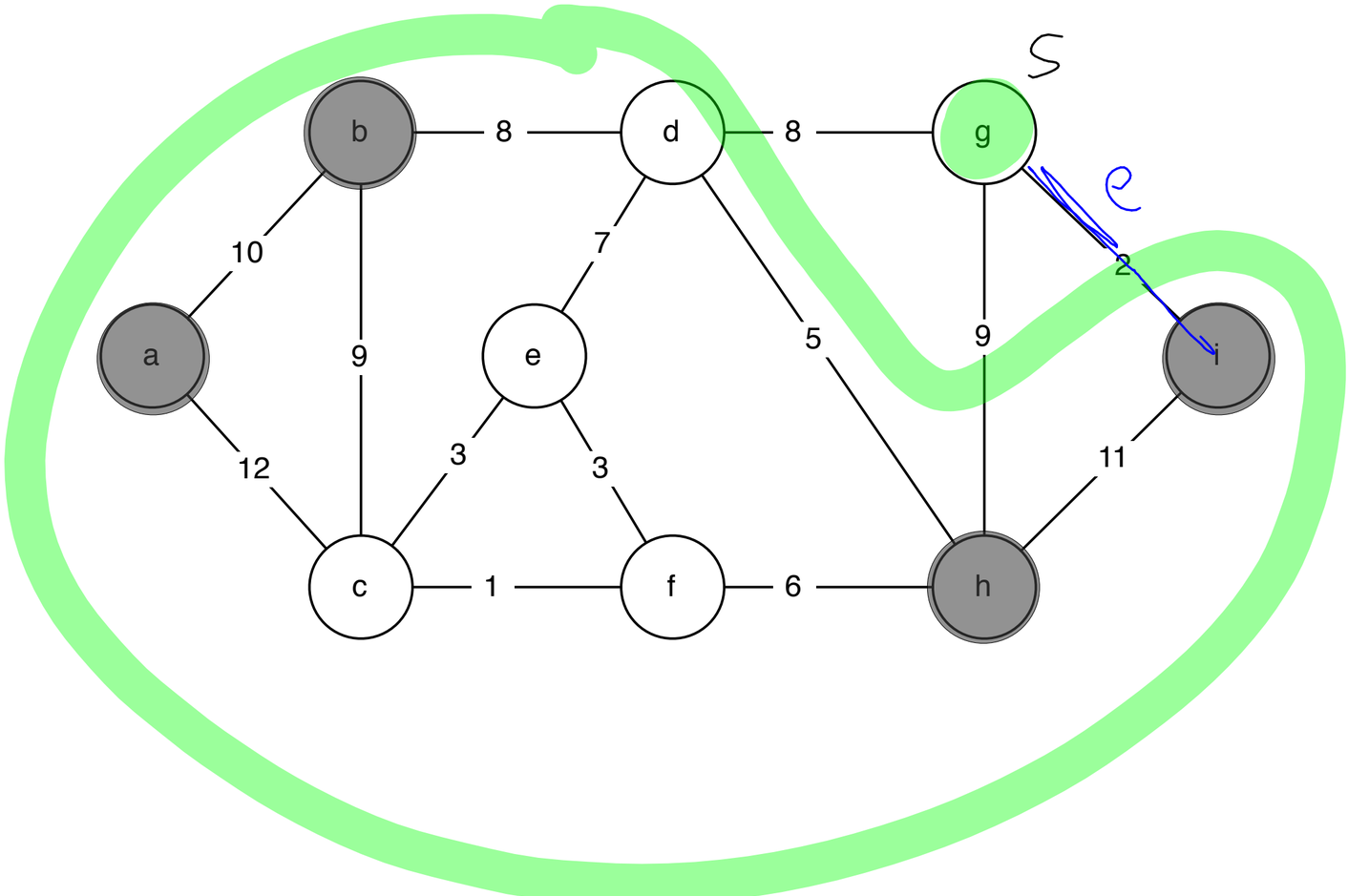
1 $T \leftarrow \emptyset$
2 **repeat** $V - 1$ times:
3 add to T the lightest edge $e \in E$ that does not create a cycle

definition: cut

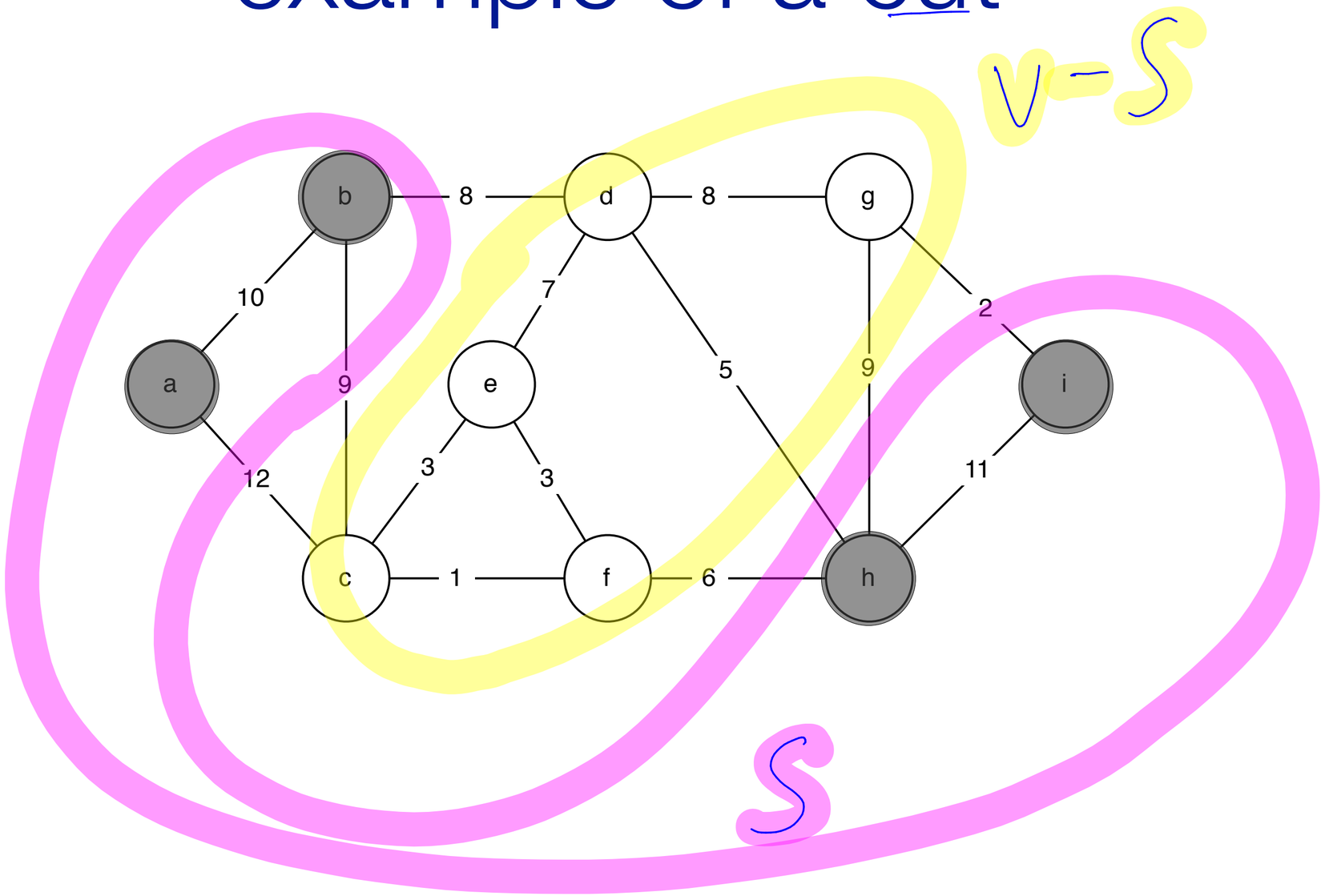
Cut: a cut of $G=(V,E)$ is a partition of V into 2 sets $(S, V-S)$.

example of a cut

$A = \phi$



example of a cut



definition: crossing a cut

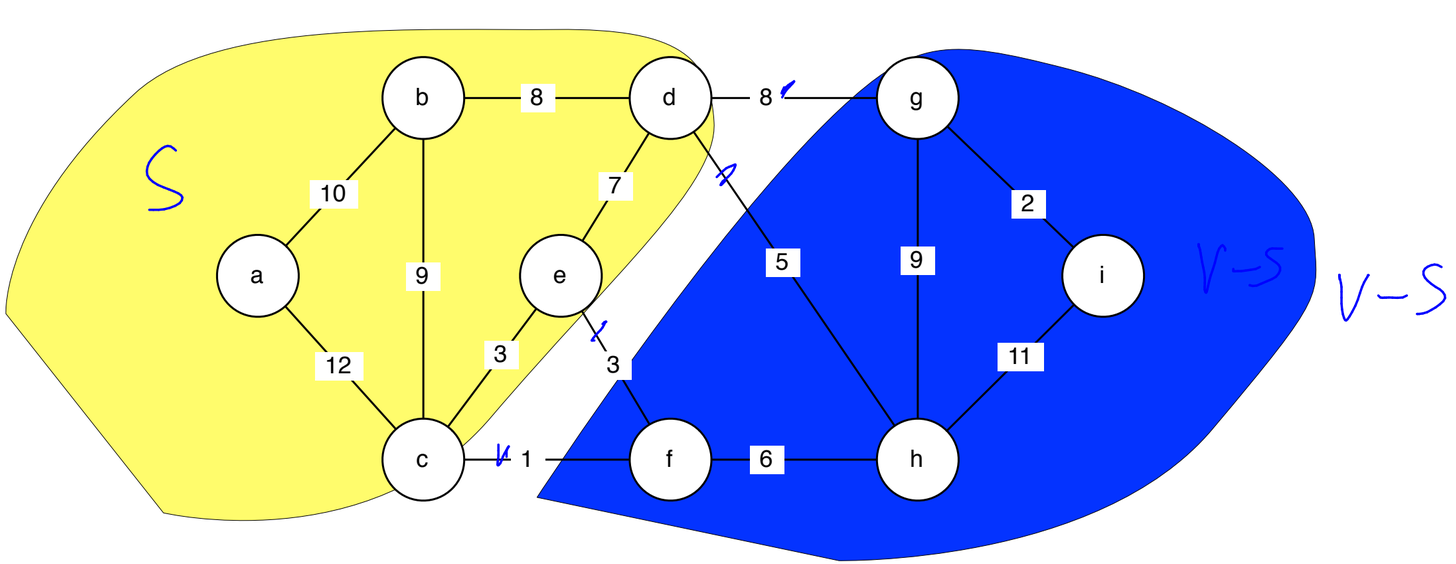
An edge $e=(u,v)$ crosses a cut $(S, V-S)$ if

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

definition: crossing a cut

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

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

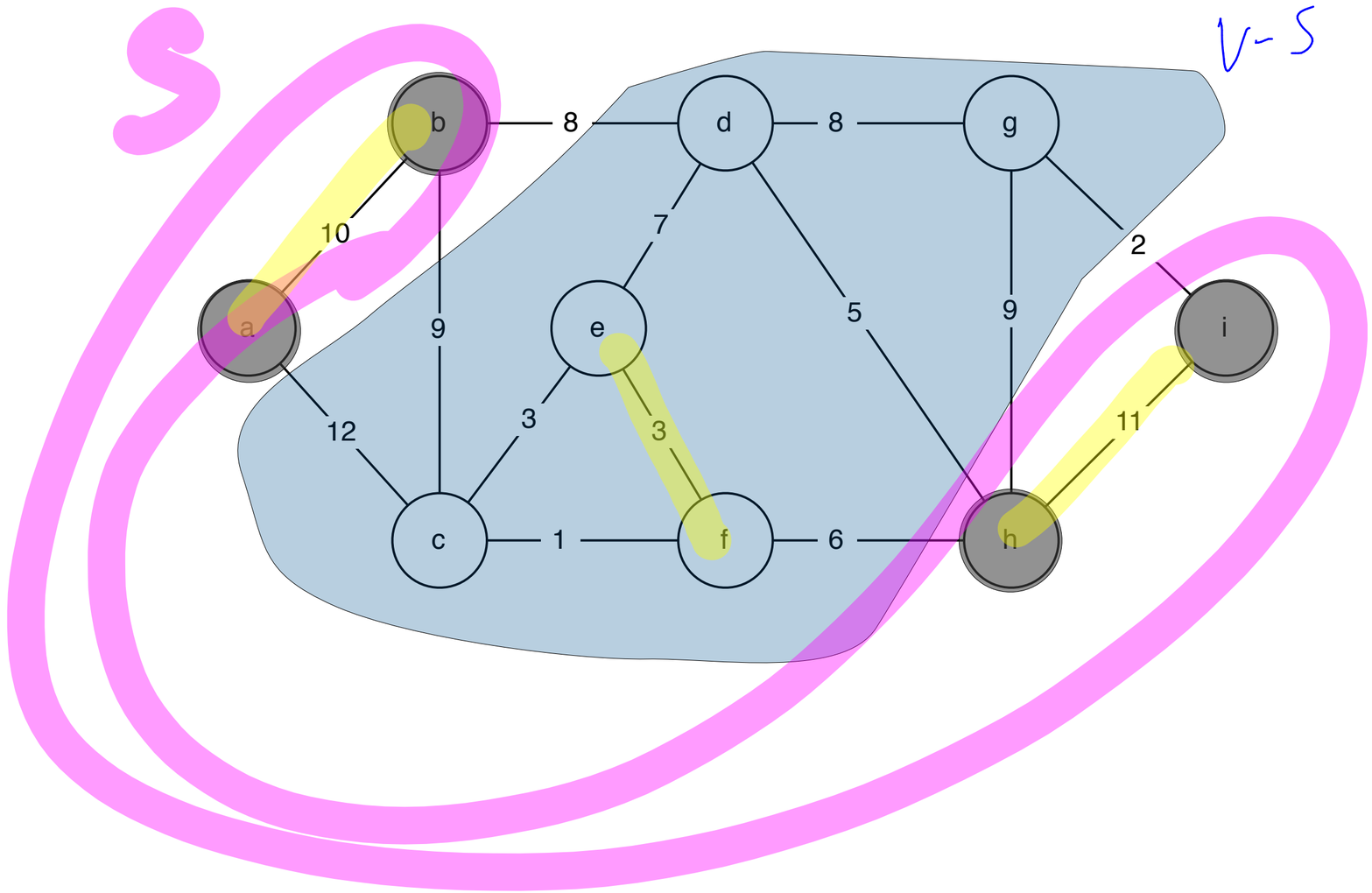


V-S

e=(d, g) crosses

(S, V-S)

example of a crossing



$A = \{$

(e, f)

(a, b)

(h, i)

$\}$

definition: respect

Respect: A set A respects the cut $(S, V-S)$ if no edge $e \in A$ crosses $(S, V-S)$.

Cut theorem

Let A be a subset of edges of some MST T.

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

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

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

Cut theorem

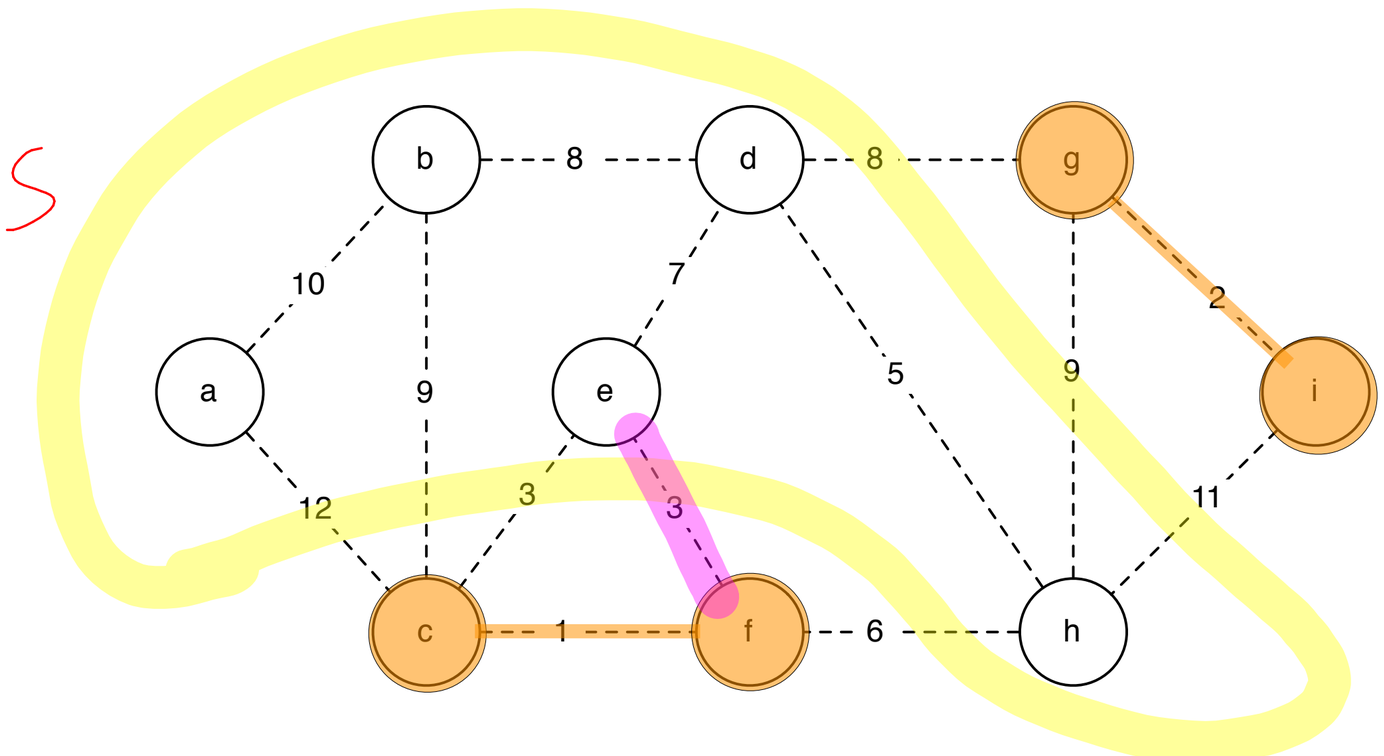
Suppose the set of edges A is part of an m.s.t.

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



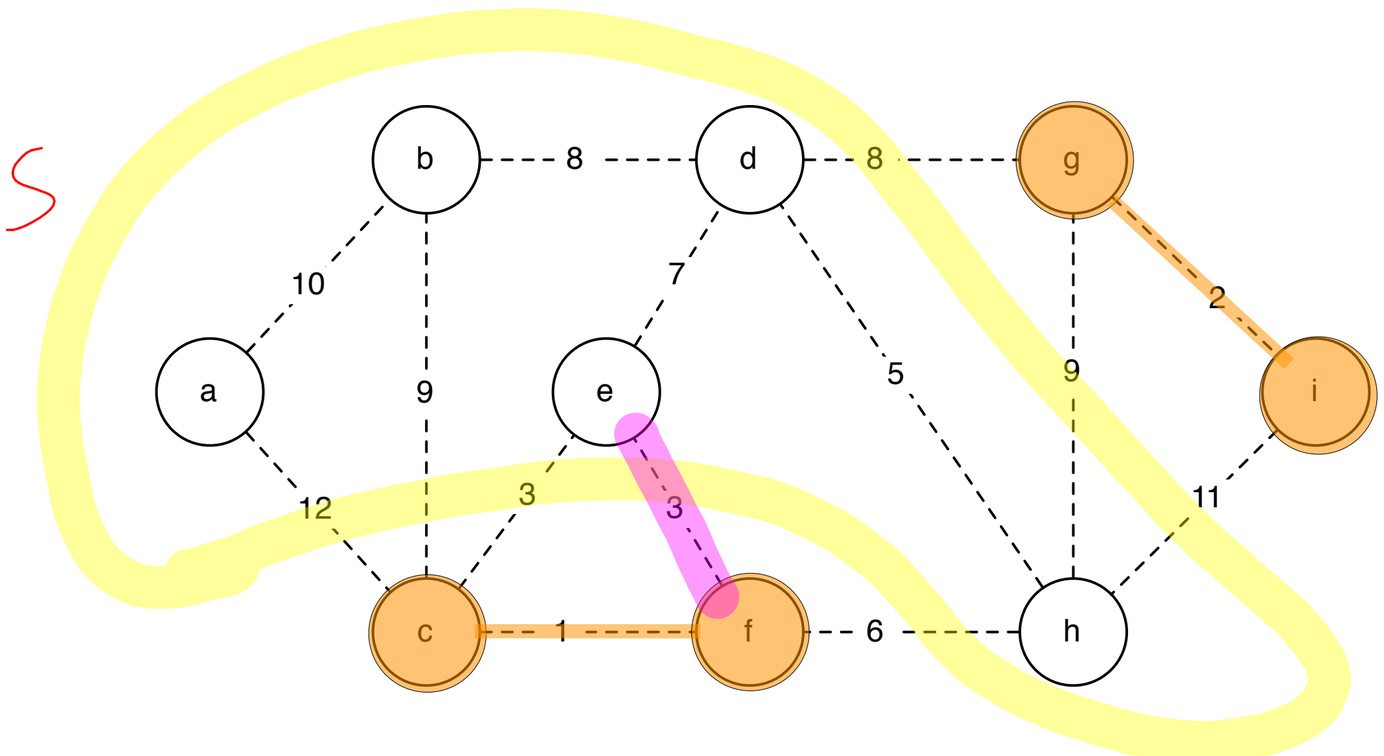
A is the set of orange links (edges)

→ A respects S.

→ e is the lightest edge that crosses S.

⇒ $A \cup \{e\}$ is part of some MST.

example of theorem

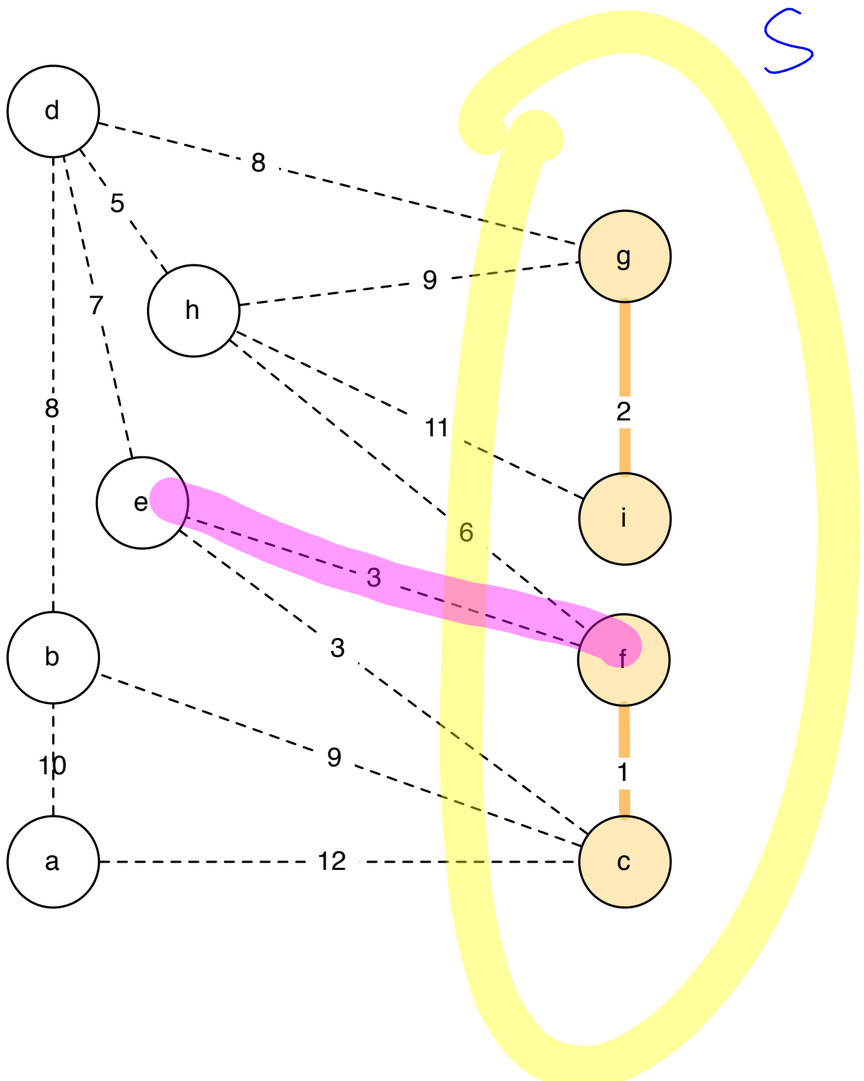


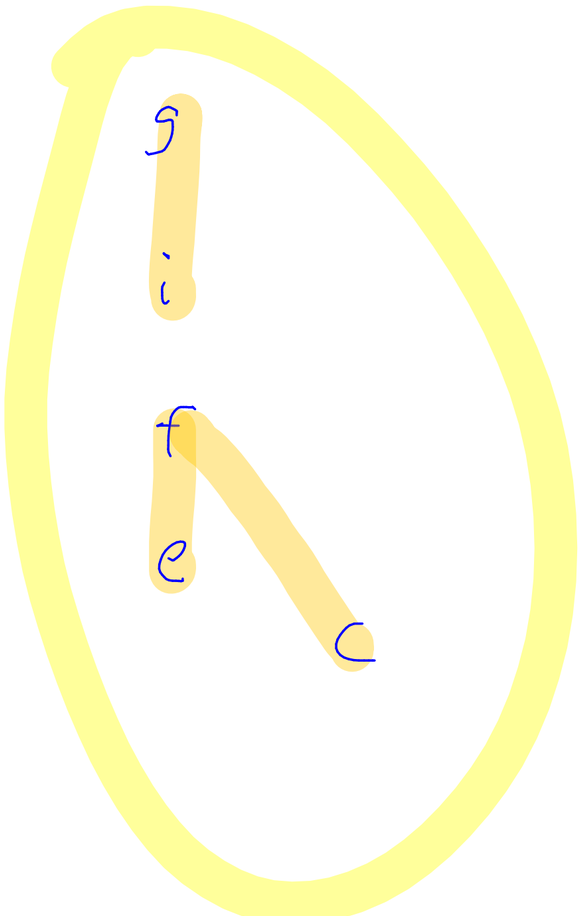
A is the set of orange links (edges)

→ A respects S.

→ e is the lightest edge that crosses S.

⇒ $A \cup \{e\}$ is part of some MST.


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


g
i
f
e
c

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.*

T

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

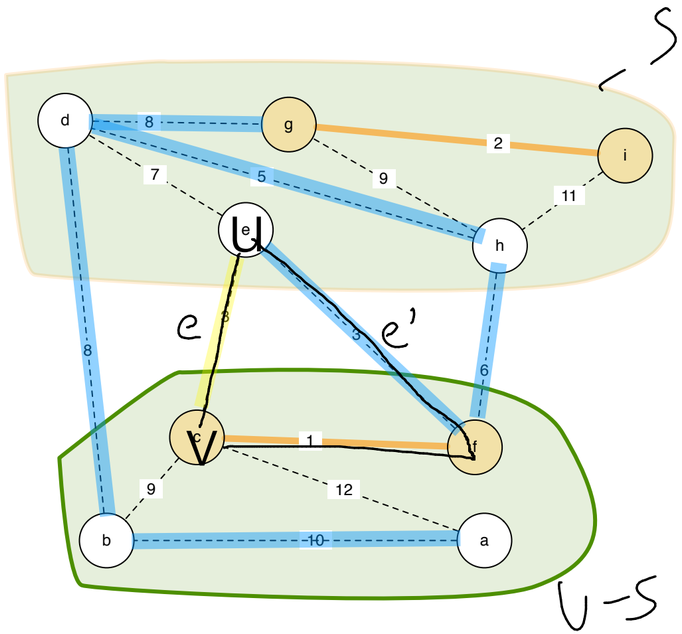
Otherwise, we will construct a T' for which $A \cup \{e\} \subseteq T'$.

Let $e = (u, v)$. Add e to T. Since T was already an MST, T must contain some path from v to u. Let e' be the first edge on this path that crosses $(S, V-S)$.

e' Must exist b/c. Define $T' = T \cup \{e\} - \{e'\}$.

proof of cut thm

Blue edges are T. A = orange.



$wt(T') = wt(T) - wt(e') + wt(e)$

Since e was the lightest edge that crosses $(S, V-S)$,

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

$\Rightarrow wt(T') \leq wt(T)$

$\Rightarrow$ must be equal b/c T was an MST.

$|T'| = V-1$, so T' is also an MST.

correctness

KRUSKAL

```
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. At Step 1, $A \subseteq T$ b/c A is empty.

Suppose at iteration i of loop 2, A is part of some MST.

Since e does not create a cycle in A, there are 3 cases to consider. $e = (u, v)$.

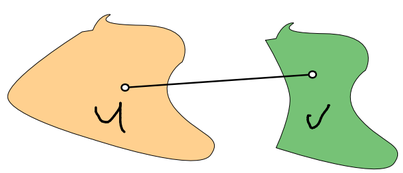
correctness

KRUSKAL-PSEUDOCODE(G)

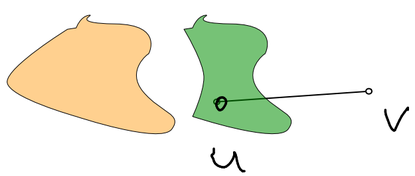
1 $A \leftarrow \emptyset$
2 **repeat** $V - 1$ times:
3 add to A the lightest edge $e \in E$ that does not create a cycle

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).

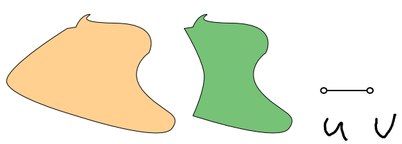
both u, and v are in A.



one end point is covered by A.

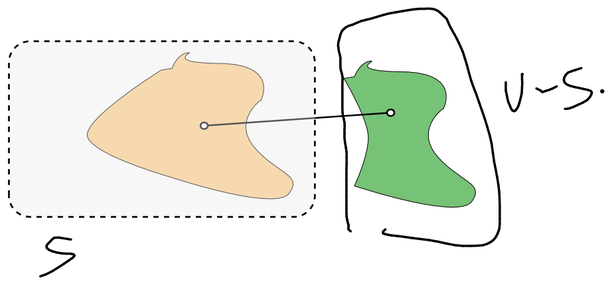


Neither endpoint u, v is covered by A.



3 cases for edge e.

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

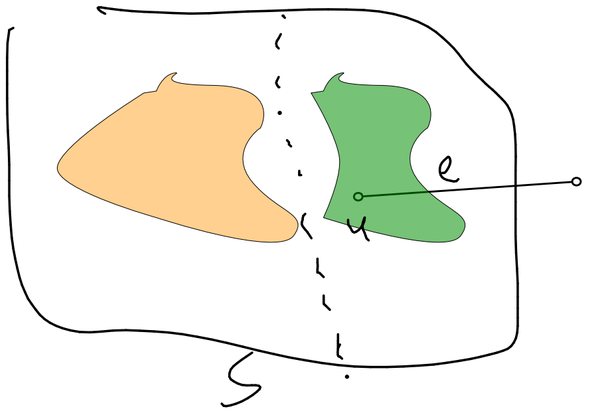


Notice that A respects $(S, V-S)$ and e is the lightest edge to cross this cut.

$\Rightarrow$ By the cut thm, $A \cup \{e\}$ is part of some MST.

3 cases for edge e.

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



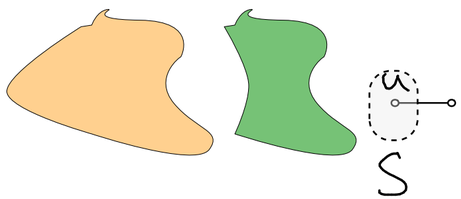
Define S to contain A.

A respects $(S, V-S)$, and the same argument follows.

orange/green: depiction of the set of edges A

3 cases for edge e.

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



as before, all conditions for the cut thm hold.

analysis?

KRUSKAL-PSEUDOCODE(G)

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

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

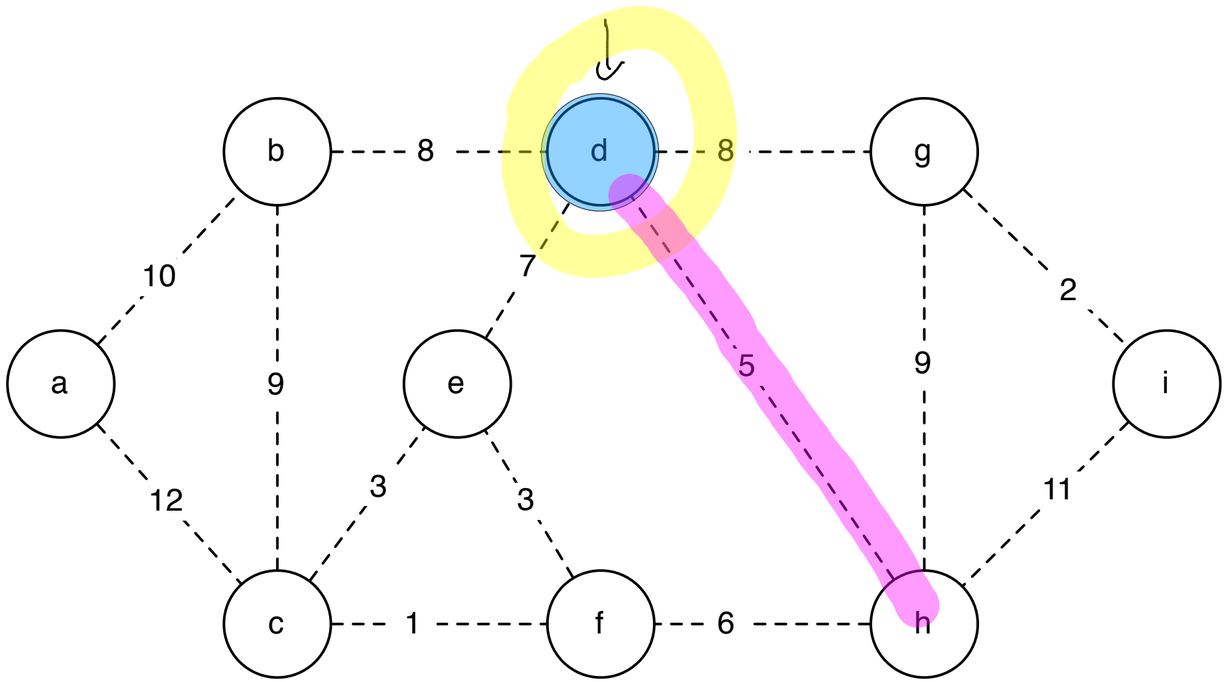
GENERAL-MST-STRATEGY($G = (V, E)$)

```
1  A ← ∅
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 ← A ∪ {e}
```

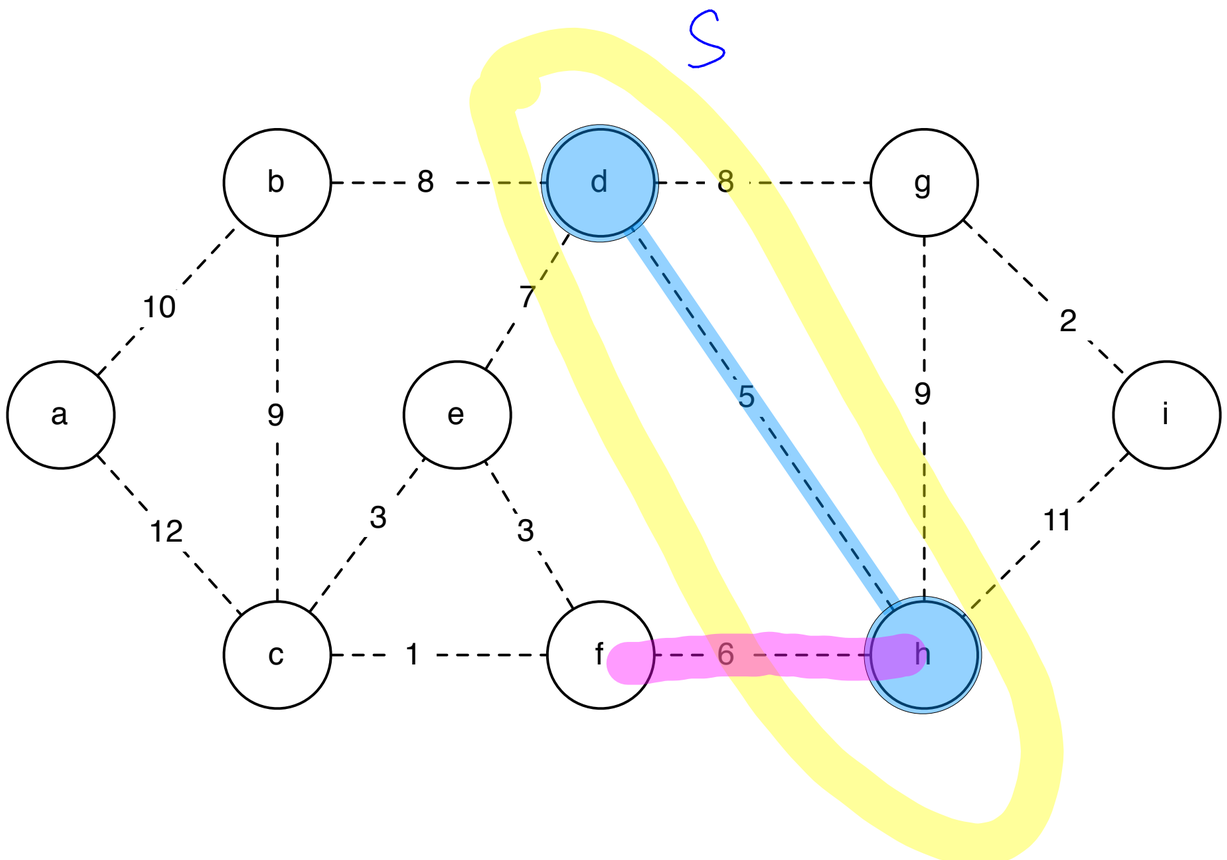
A is a subtree

edge e is lightest edge that grows the subtree

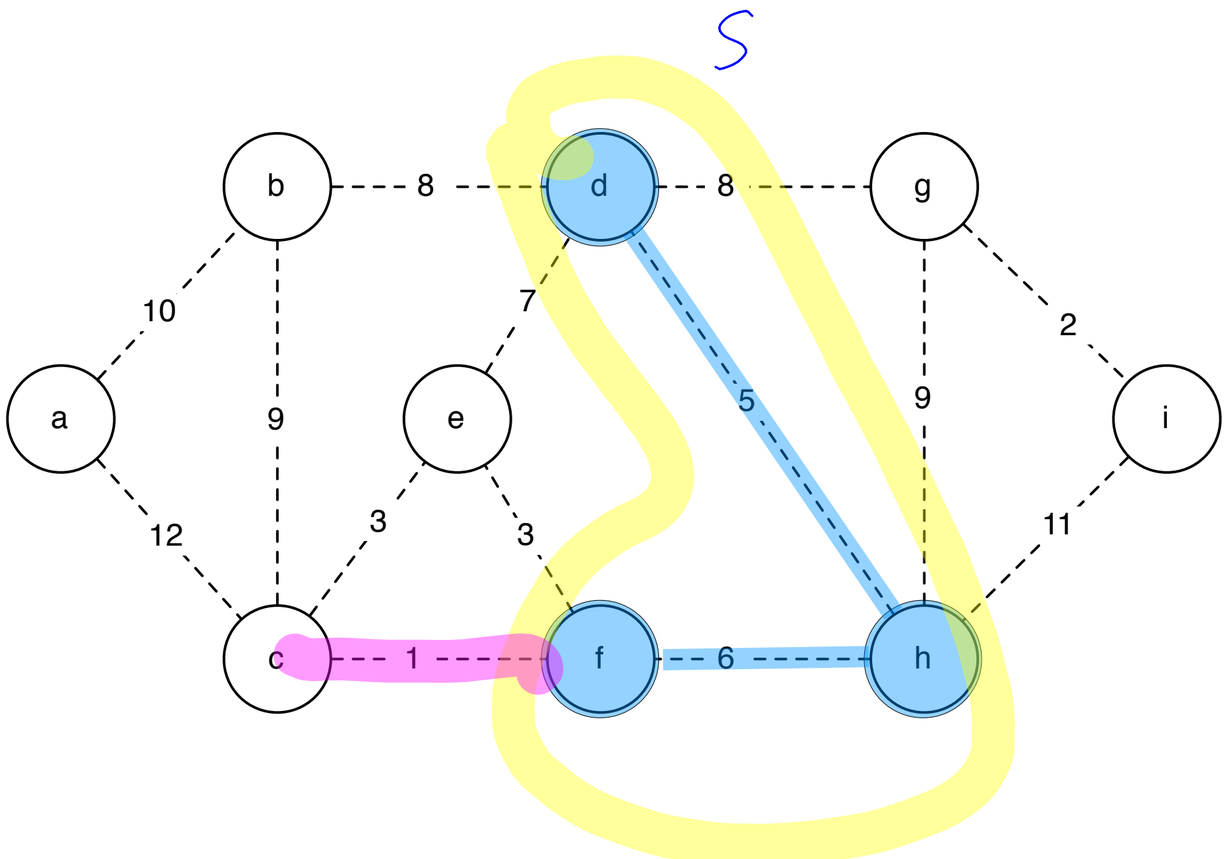
prim



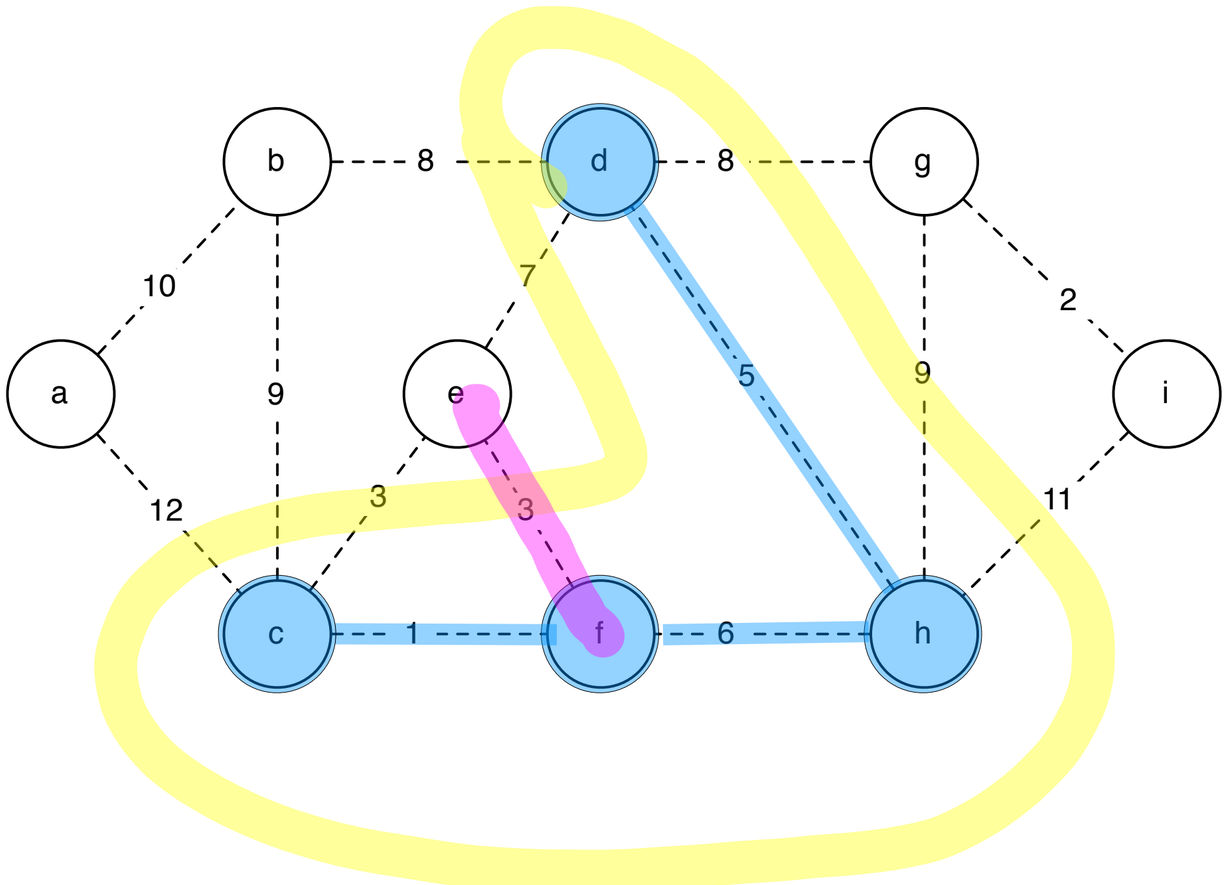
prim



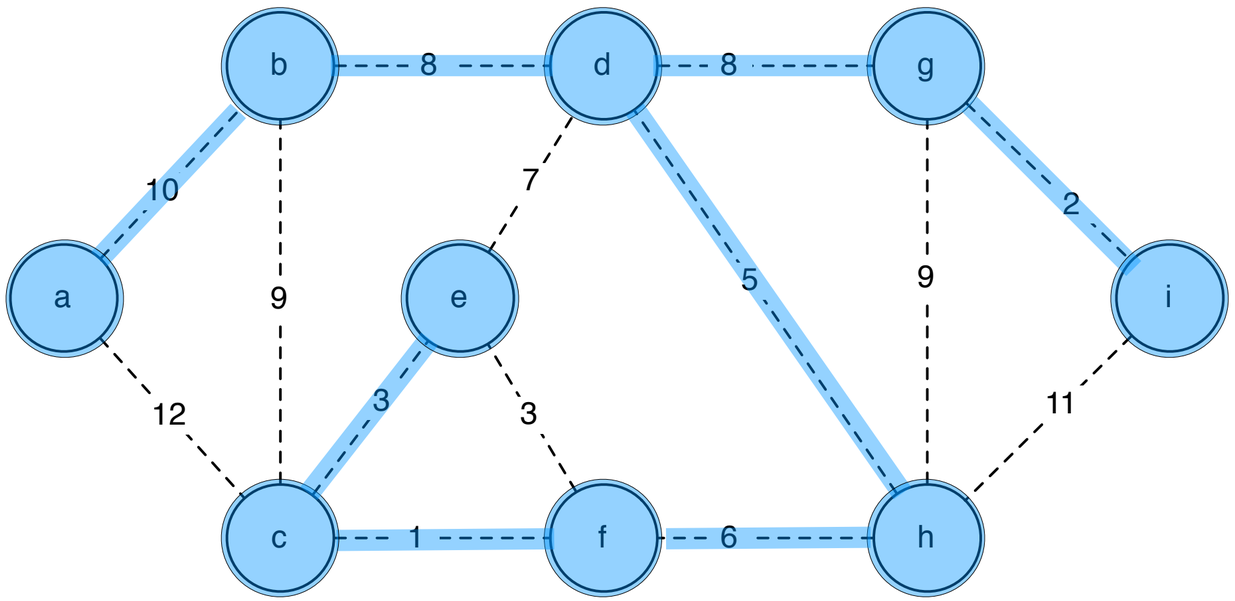
prim



prim



prim



implementation

idea:

implementation

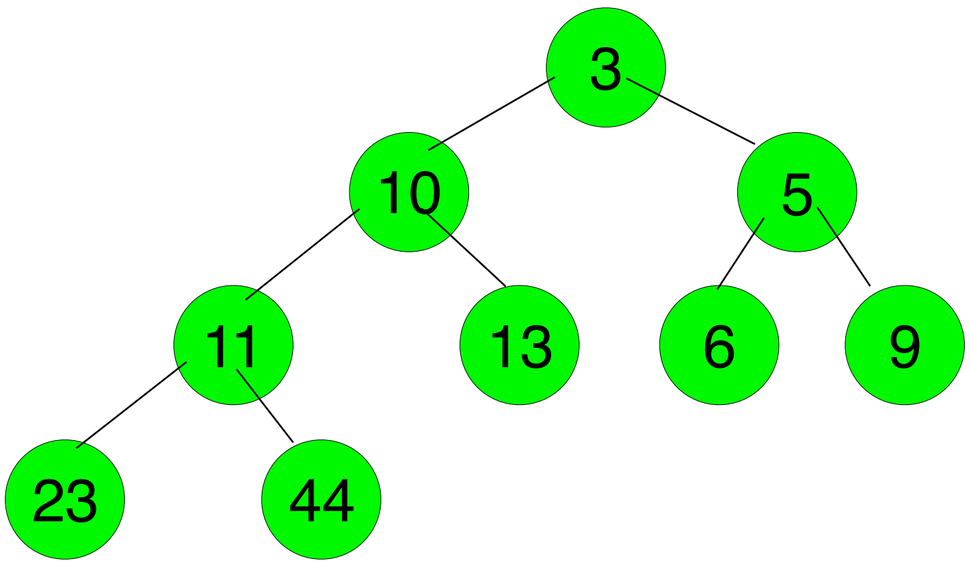
new data structure

binary heap

full tree, key value <= to key of children

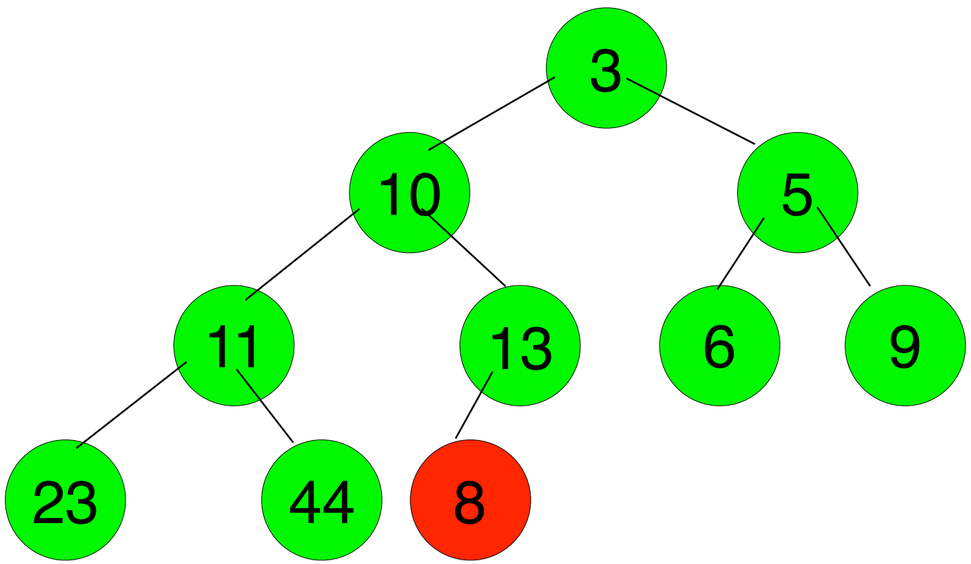
binary heap

full tree, key value <= to key of children



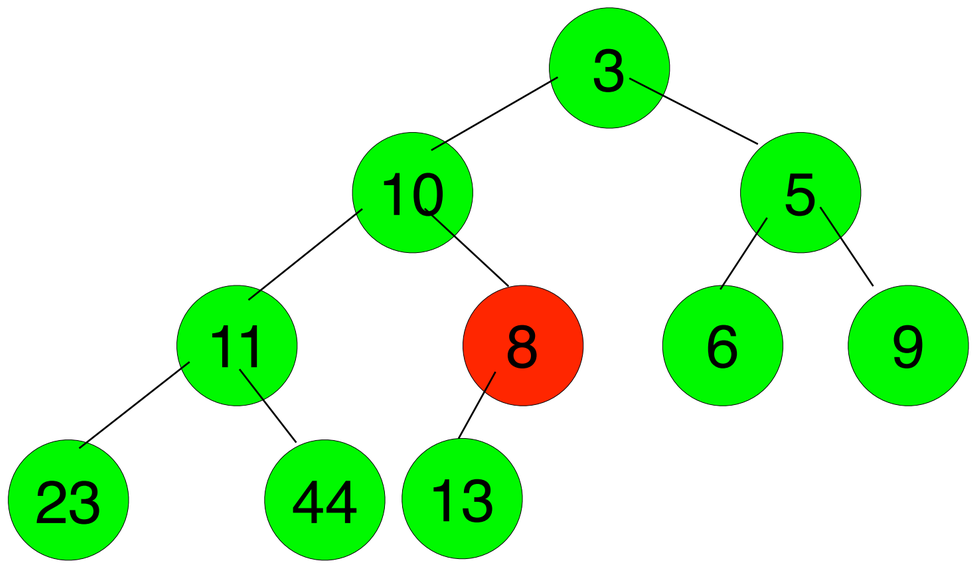
binary heap

full tree, key value <= to key of children



binary heap

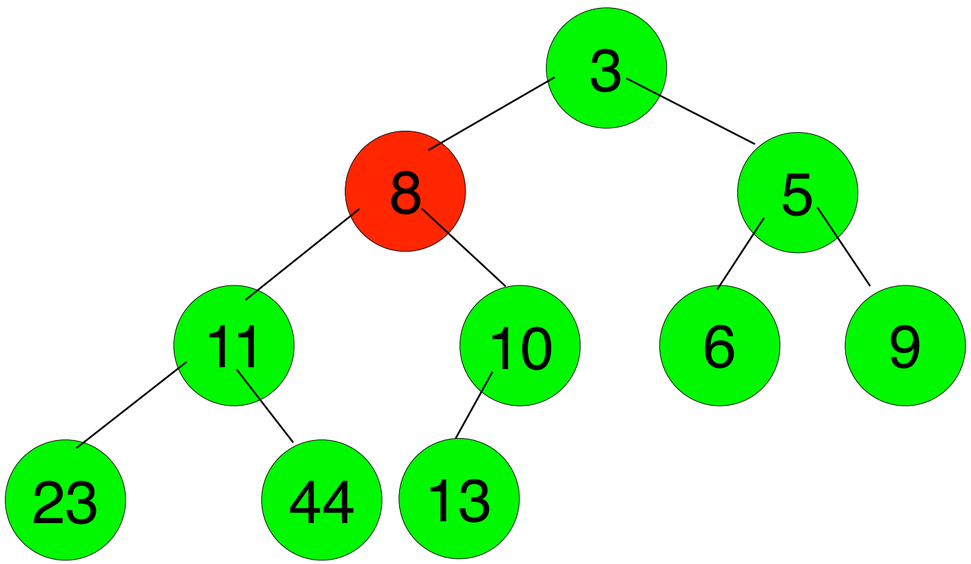
full tree, key value <= to key of children



binary heap

full tree, key value <= to key of children

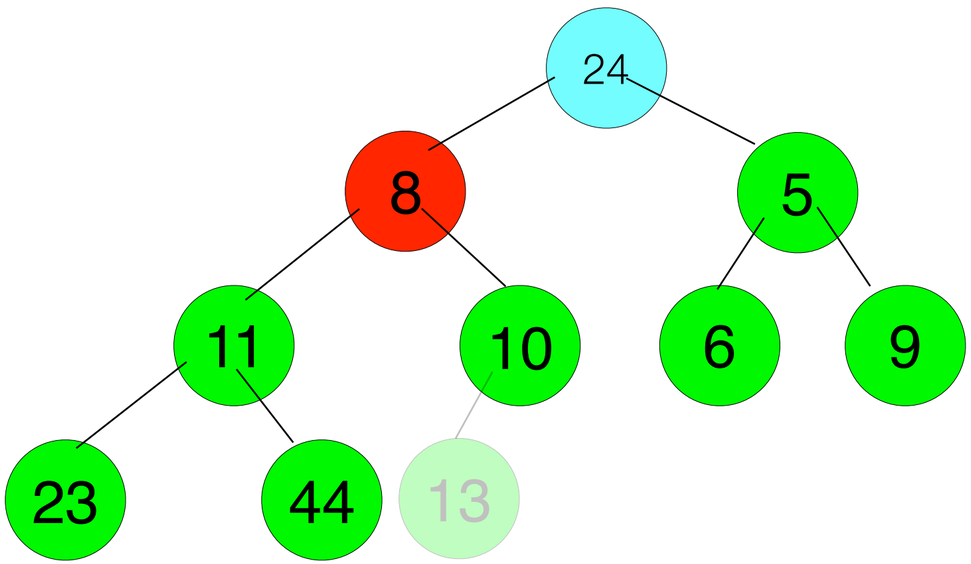
how to extractmin?



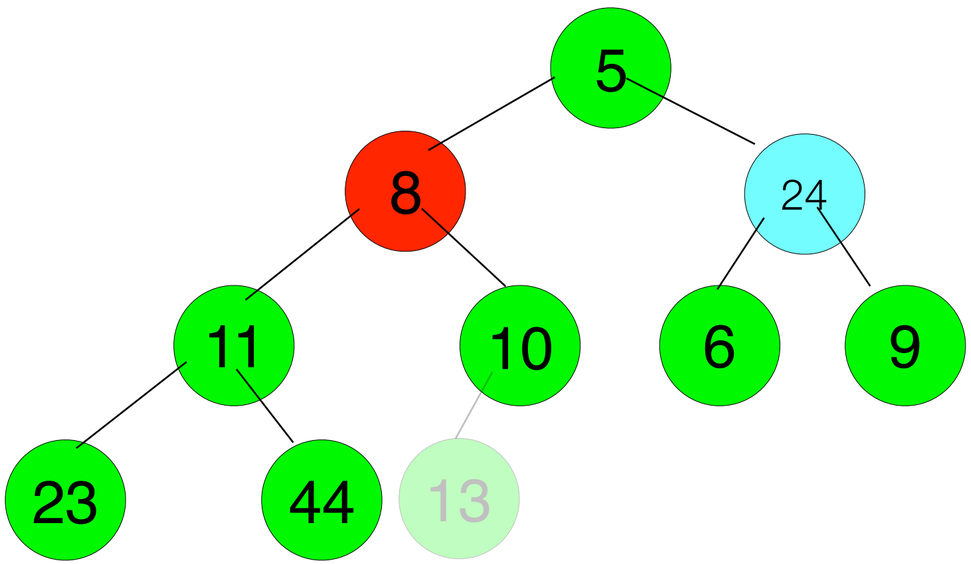
binary heap

full tree, key value <= to key of children

how to extractmin?



binary heap

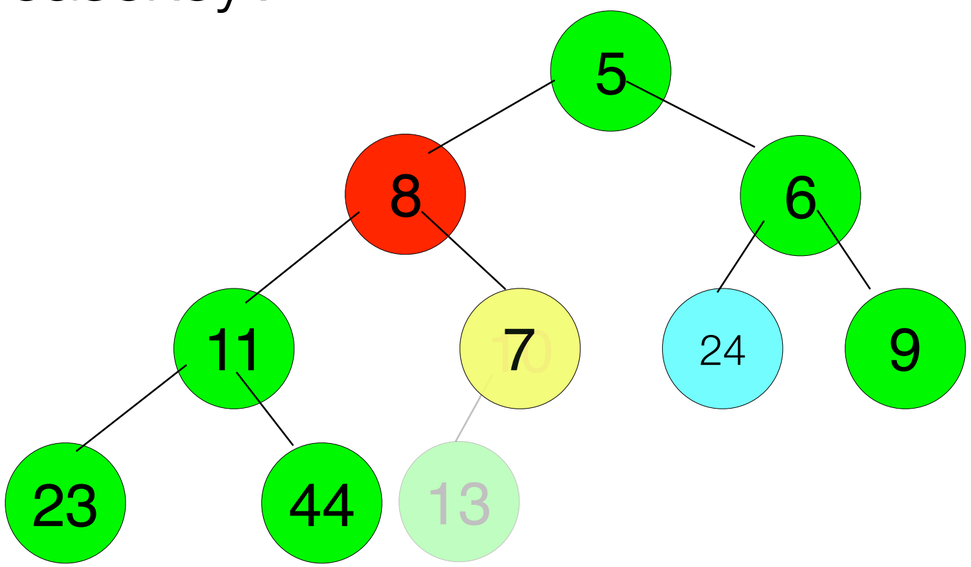


binary heap

full tree, key value <= to key of children

how to extractmin?

how to decreasekey?

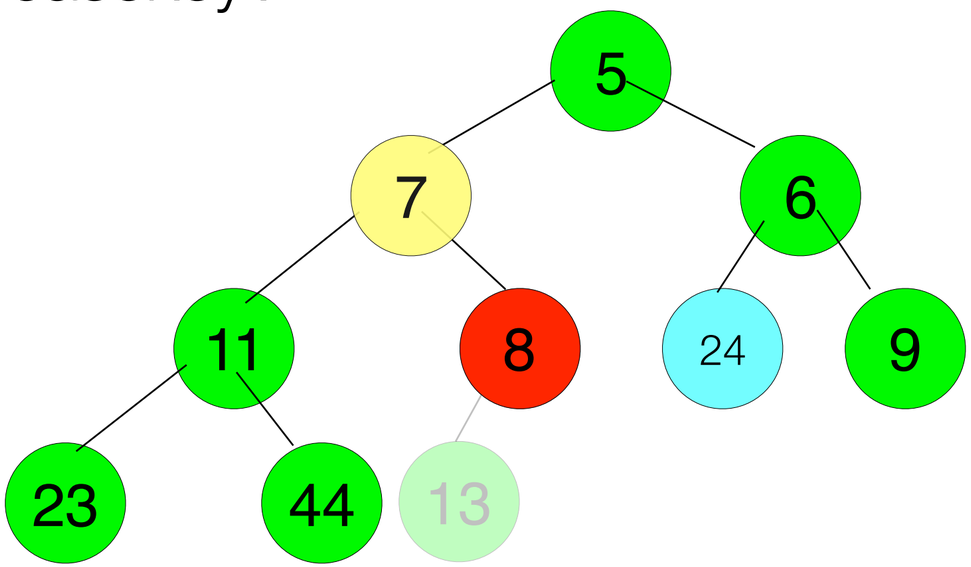


binary heap

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:

makequeue:

extractmin:

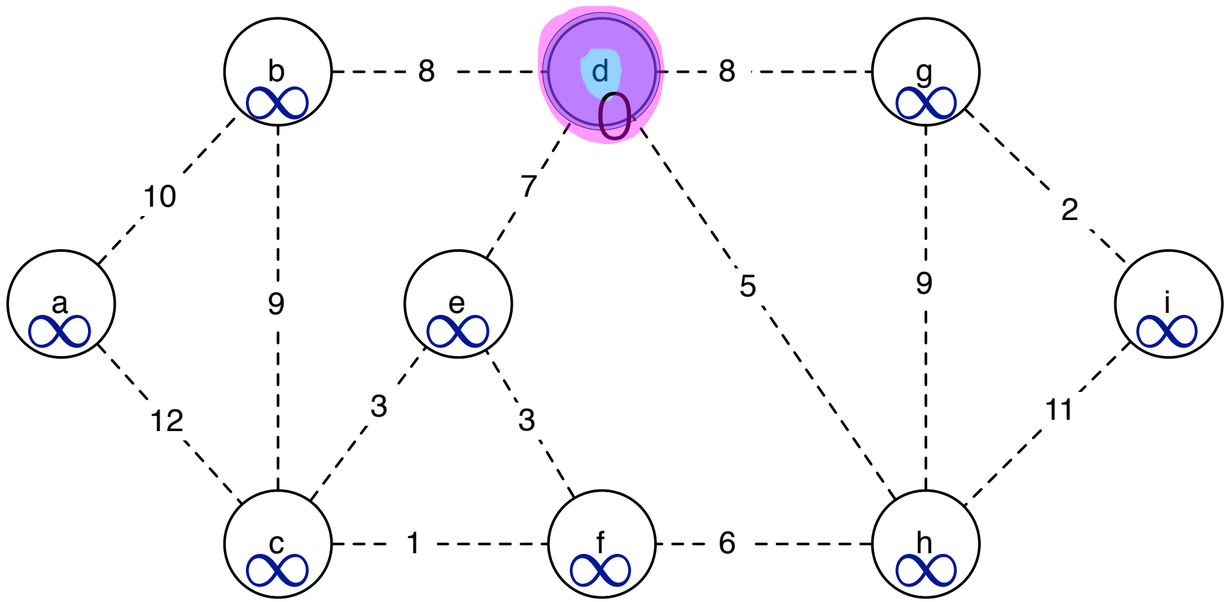
decreasekey:

Prim’s algorithm

implementation

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

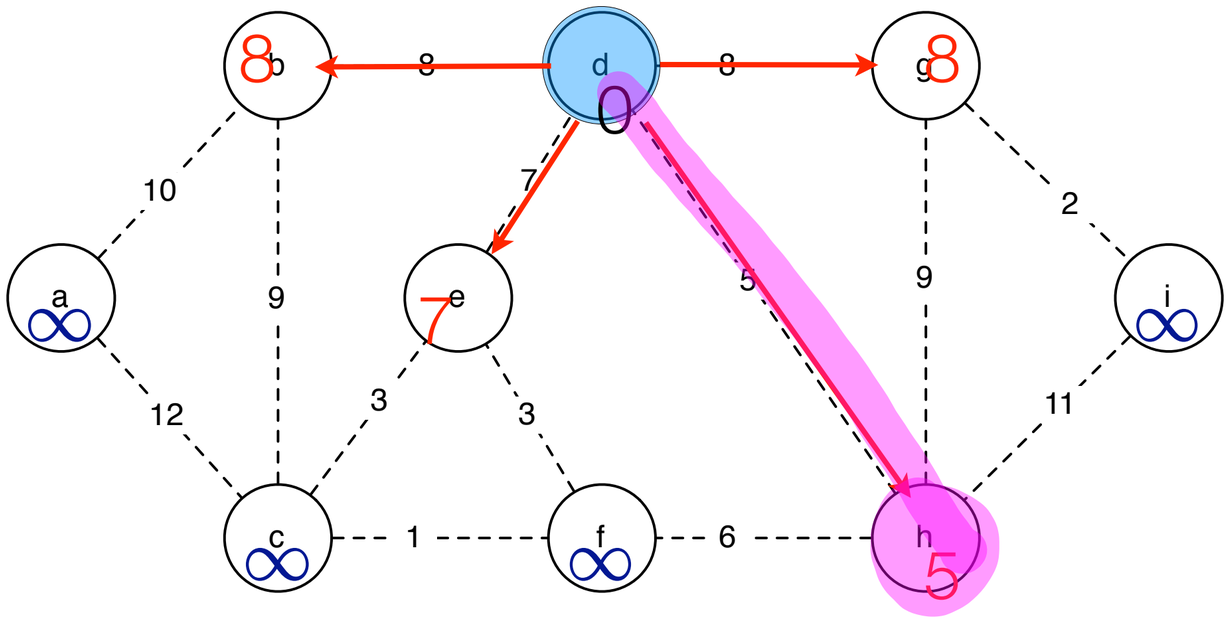
prim



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

```
1   Q ← ∅        ▷ Q is a Priority Queue
2   Initialize each v ∈ V with key k_v ← ∞, π_v ← NIL
3   Pick a starting node r and set k_r ← 0
4   Insert all nodes into Q with key k_v.
5   while Q ≠ ∅
6         do u ← EXTRACT-MIN(Q)
7            for each v ∈ Adj(u)
8                  do if v ∈ Q and w(u, v) < k_v
9                        then π_v ← u
10                            DECREASE-KEY(Q, v, w(u, v))    ▷ Sets k_v ← w(u, v)
```

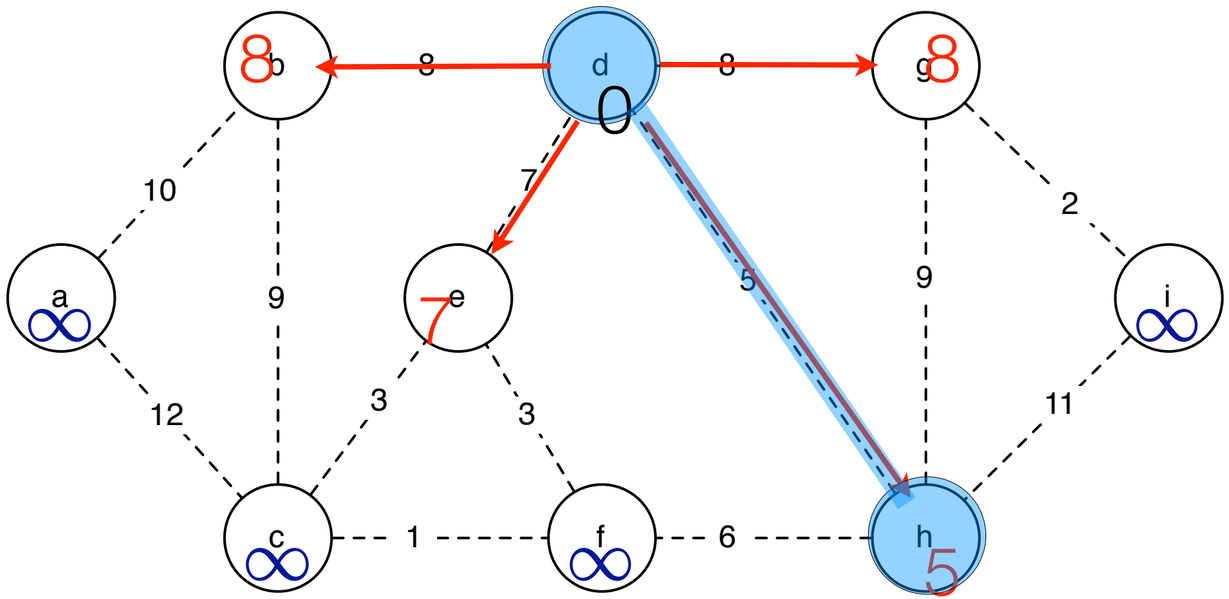
prim



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

1 $Q \leftarrow \emptyset$ ▷ Q is a Priority Queue
2 Initialize each $v \in V$ with key $k_v \leftarrow \infty$, $\pi_v \leftarrow$ NIL
3 Pick a starting node r and set $k_r \leftarrow 0$
4 Insert all nodes into Q with key k_v.
5 **while** $Q \neq \emptyset$
6 **do** $u \leftarrow$ EXTRACT-MIN(Q)
7 **for** each $v \in Adj(u)$
8 **do if** $v \in Q$ and $w(u, v) < k_v$
9 **then** $\pi_v \leftarrow u$
10 DECREASE-KEY$(Q, v, w(u, v))$ ▷ Sets $k_v \leftarrow w(u, v)$

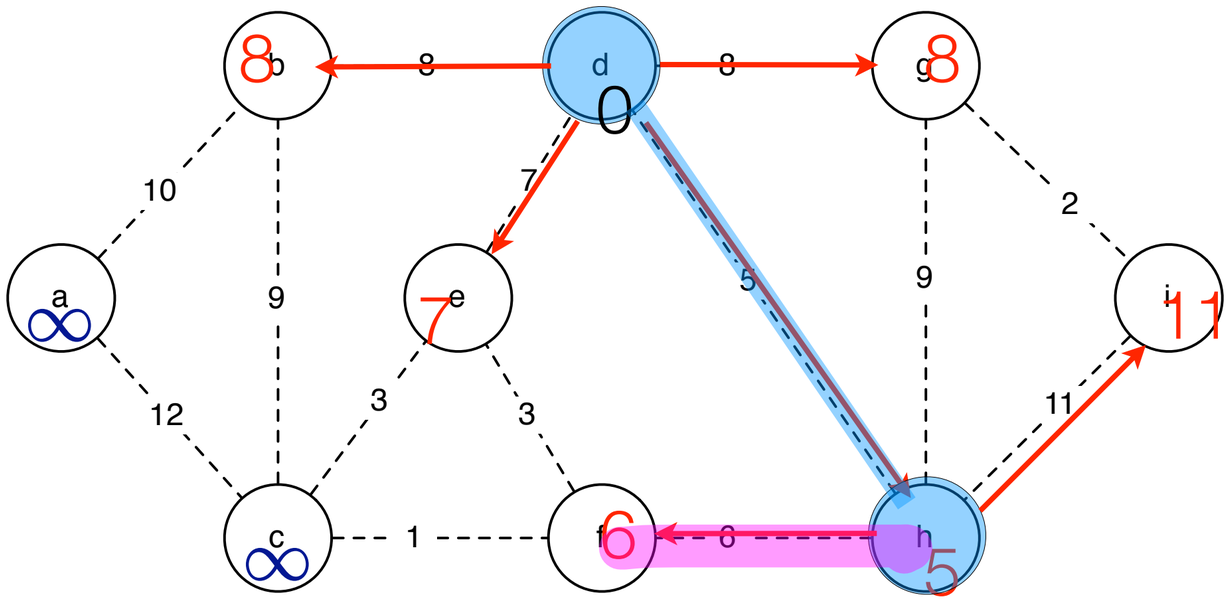
prim



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

1 $Q \leftarrow \emptyset$ ▷ Q is a Priority Queue
2 Initialize each $v \in V$ with key $k_v \leftarrow \infty$, $\pi_v \leftarrow$ NIL
3 Pick a starting node r and set $k_r \leftarrow 0$
4 Insert all nodes into Q with key k_v.
5 **while** $Q \neq \emptyset$
6 **do** $u \leftarrow$ EXTRACT-MIN(Q)
7 **for** each $v \in Adj(u)$
8 **do if** $v \in Q$ and $w(u, v) < k_v$
9 **then** $\pi_v \leftarrow u$
10 DECREASE-KEY$(Q, v, w(u, v))$ ▷ Sets $k_v \leftarrow w(u, v)$

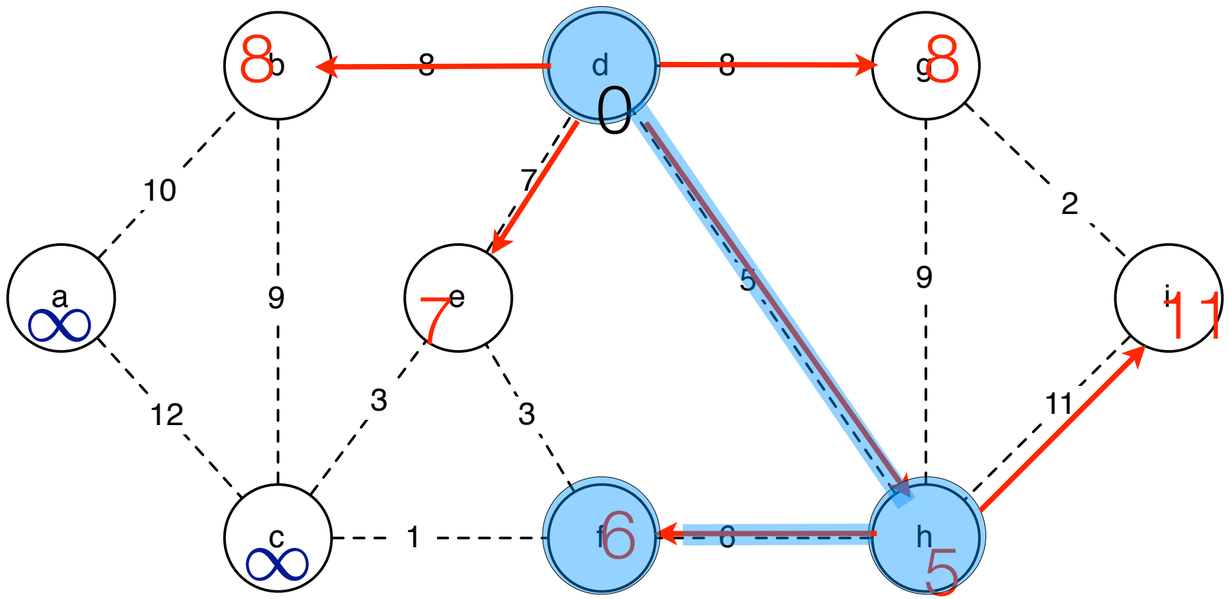
prim



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

```
1   Q ← ∅        ▷ Q is a Priority Queue
2   Initialize each v ∈ V with key k_v ← ∞, π_v ← NIL
3   Pick a starting node r and set k_r ← 0
4   Insert all nodes into Q with key k_v.
5   while Q ≠ ∅
6         do u ← EXTRACT-MIN(Q)
7            for each v ∈ Adj(u)
8                  do if v ∈ Q and w(u, v) < k_v
9                        then π_v ← u
10                            DECREASE-KEY(Q, v, w(u, v))    ▷ Sets k_v ← w(u, v)
```

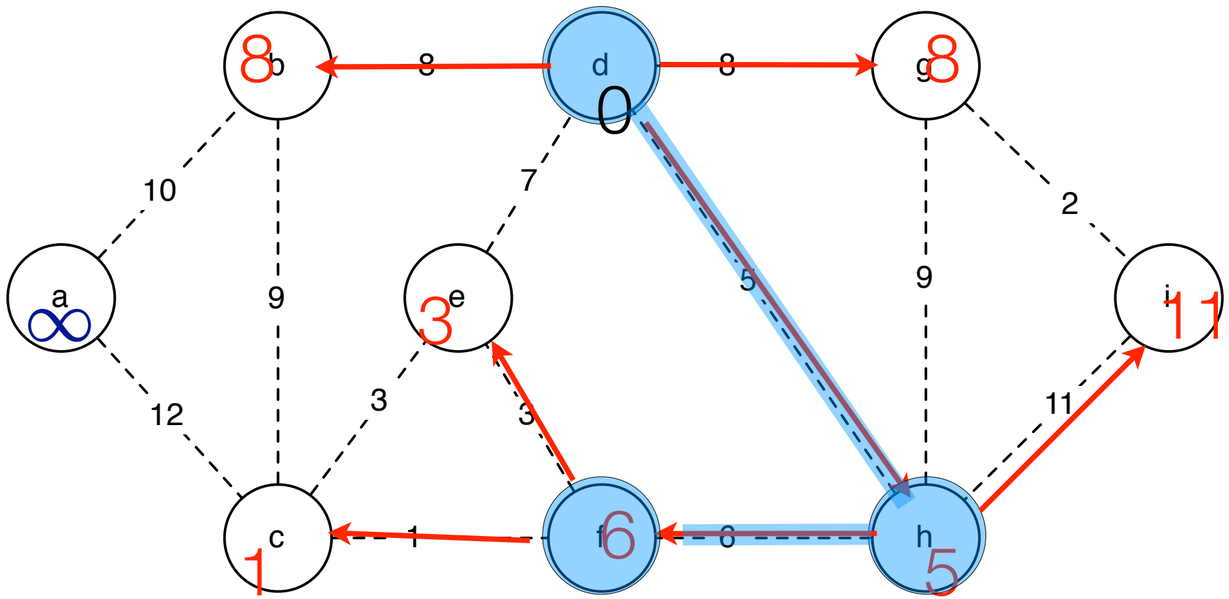
prim



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

```
1   Q ← ∅        ▷ Q is a Priority Queue
2   Initialize each v ∈ V with key k_v ← ∞, π_v ← NIL
3   Pick a starting node r and set k_r ← 0
4   Insert all nodes into Q with key k_v.
5   while Q ≠ ∅
6         do u ← EXTRACT-MIN(Q)
7            for each v ∈ Adj(u)
8                  do if v ∈ Q and w(u, v) < k_v
9                        then π_v ← u
10                            DECREASE-KEY(Q, v, w(u, v))    ▷ Sets k_v ← w(u, v)
```

prim



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

1 $Q \leftarrow \emptyset$ ▷ Q is a Priority Queue
2 Initialize each $v \in V$ with key $k_v \leftarrow \infty$, $\pi_v \leftarrow$ NIL
3 Pick a starting node r and set $k_r \leftarrow 0$
4 Insert all nodes into Q with key k_v.
5 **while** $Q \neq \emptyset$
6 **do** $u \leftarrow$ EXTRACT-MIN(Q)
7 **for** each $v \in Adj(u)$
8 **do if** $v \in Q$ and $w(u, v) < k_v$
9 **then** $\pi_v \leftarrow u$
10 DECREASE-KEY$(Q, v, w(u, v))$ ▷ Sets $k_v \leftarrow w(u, v)$

running time

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

V(

log(V)

E

log V

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

implementation

```
PRIM(G = (V, E))
 1  Q ← ∅          ▷ Q is a Priority Queue
 2  Initialize each v ∈ V with key k_v ← ∞, π_v ← NIL
 3  Pick a starting node r and set k_r ← 0
 4  Insert all nodes into Q with key k_v.
 5  while Q ≠ ∅
 6        do u ← EXTRACT-MIN(Q)
 7           for each v ∈ Adj(u)
 8                 do if v ∈ Q and w(u, v) < k_v
 9                       then π_v ← u
10                            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

faster implementation

```
PRIM(G = (V, E))
 1  Q ← ∅          ▷ Q is a Priority Queue
 2  Initialize each v ∈ V with key k_v ← ∞, π_v ← NIL
 3  Pick a starting node r and set k_r ← 0
 4  Insert all nodes into Q with key k_v.
 5  while Q ≠ ∅
 6        do u ← EXTRACT-MIN(Q)
 7           for each v ∈ Adj(u)
 8                 do if v ∈ Q and w(u, v) < k_v
 9                       then π_v ← u
10                            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: (randomized)	E
Euclidean mst:	$V \log V$

Ackerman function

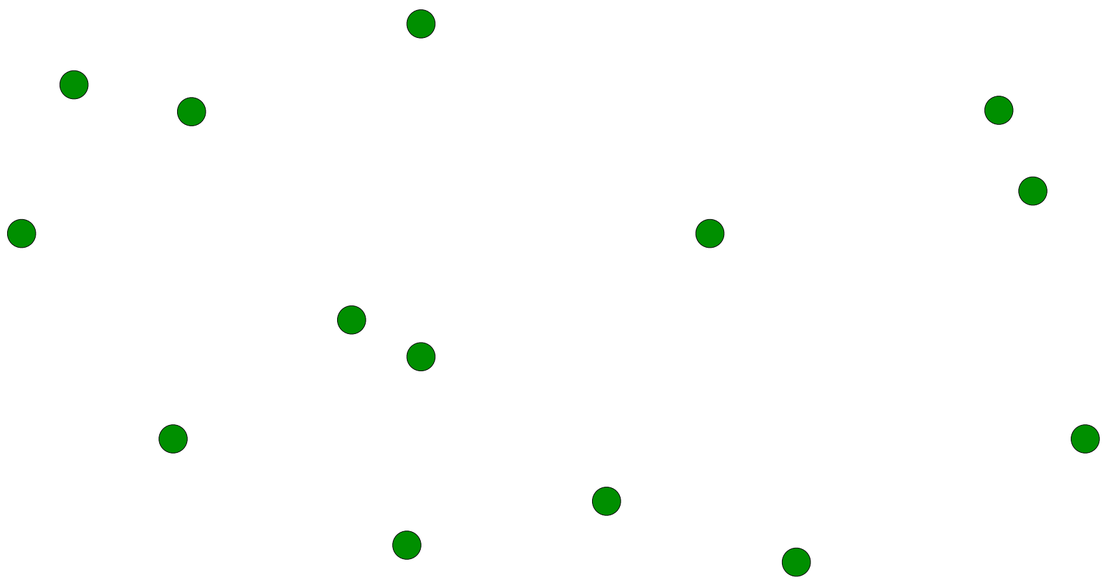
$$A(m,n) = \begin{cases} n+1 & m = 0 \\ A(m-1,1) & m > 0,\ n = 0 \\ A(m-1,A(m,n-1)) & m,n > 0 \end{cases}$$

$A(4,2) =$

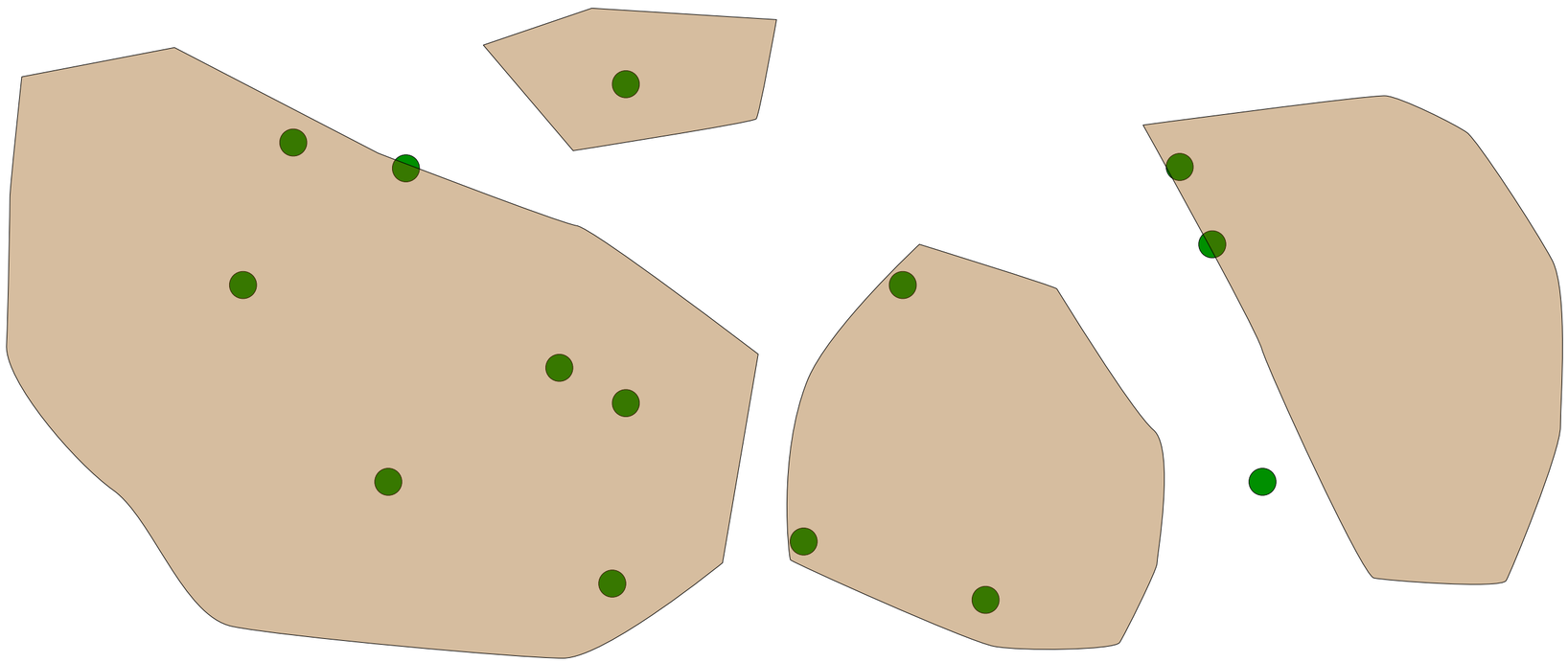
inverse ackerman

$\alpha(n) =$

application of mst



application of mst



application of mst