

COMP 1201 Algorithmics

Minimum Spanning Trees

Hikmat Farhat

Minimum Spanning Trees

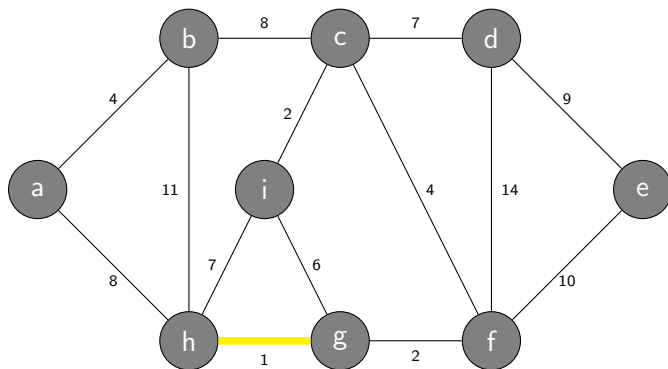
- In many application, when the system is represented by a graph we need to find a **Minimum Spanning Tree** (MST).
- typically we are given a graph $G = (V, E)$ with a weight function $w : E \rightarrow \mathbb{R}$.
- We are looking for a tree $T \subseteq E$ that spans G and has the least weight. (weight of a tree is the sum of the weights of its edges).
- As the name suggest this collection of edges T is
 - 1 **A tree.**
 - 2 **Spanning.** meaning connects all the nodes of the graph.
 - 3 It has the **least** weight of all spanning trees.

Kruskal's Algorithm

- Kruskal's algorithm computes a MST of a given graph.
- Every edge has an associated weight or cost.
- The idea is to build the MST by adding an edge every iteration.
 - ① The edges are considered by increasing order of weight.
 - ② An edge is added if it doesn't create a cycle.
- The algorithm stops when there are no more edges to consider.

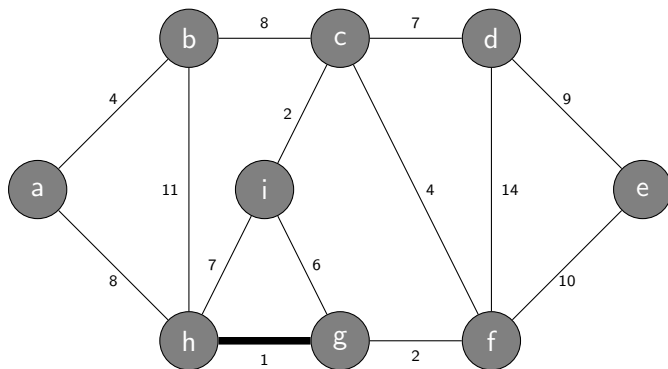
Example (Kruskal's Algorithm)

creates cycle?



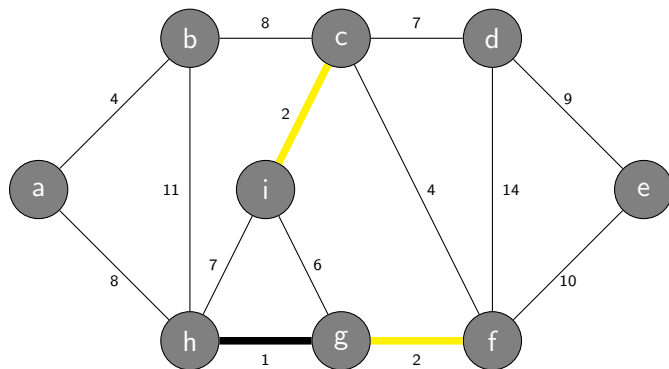
Example (Kruskal's Algorithm)

creates cycle? No, so add it to the MST



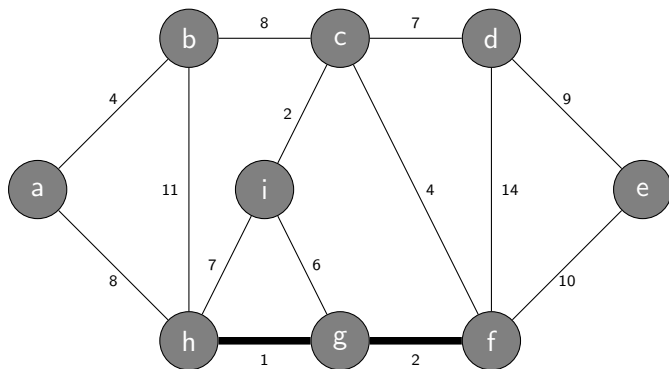
Example (Kruskal's Algorithm)

Two choices



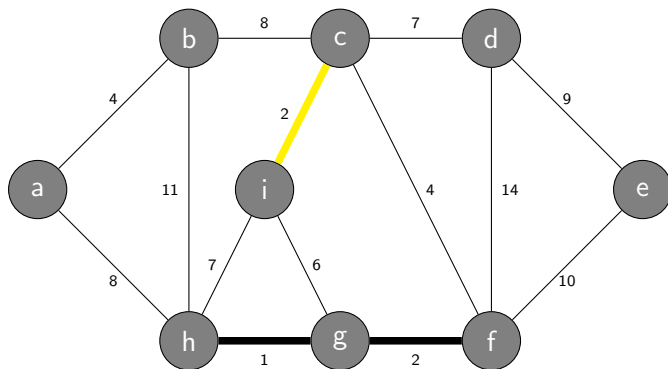
Example (Kruskal's Algorithm)

Two choices. Select one randomly



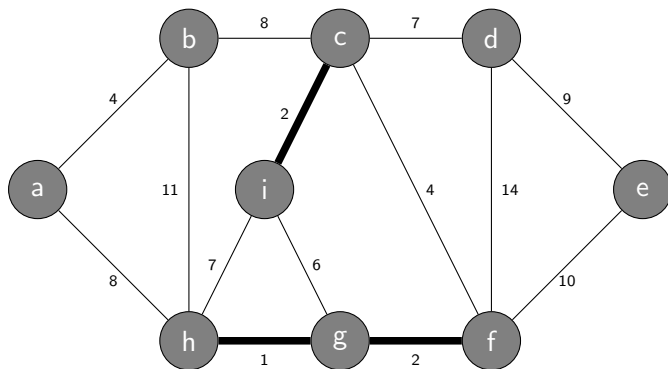
Example (Kruskal's Algorithm)

Creates cycle?



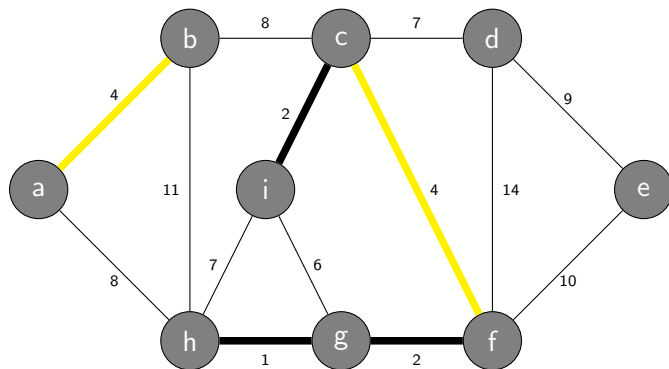
Example (Kruskal's Algorithm)

Creates cycle? No, so add it to the MST



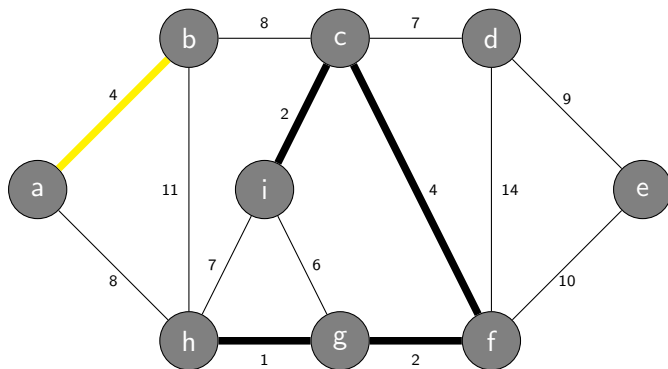
Example (Kruskal's Algorithm)

Two choices.



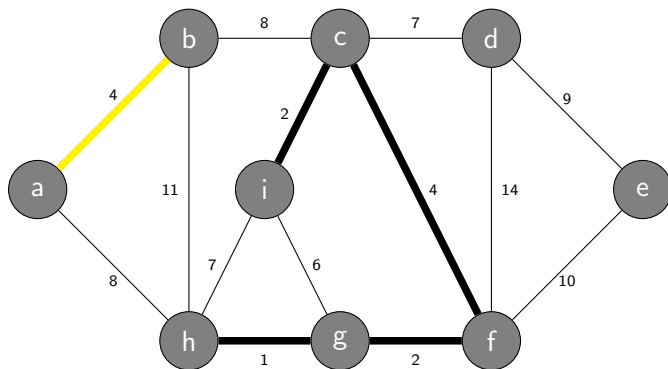
Example (Kruskal's Algorithm)

Two choices. Select one randomly



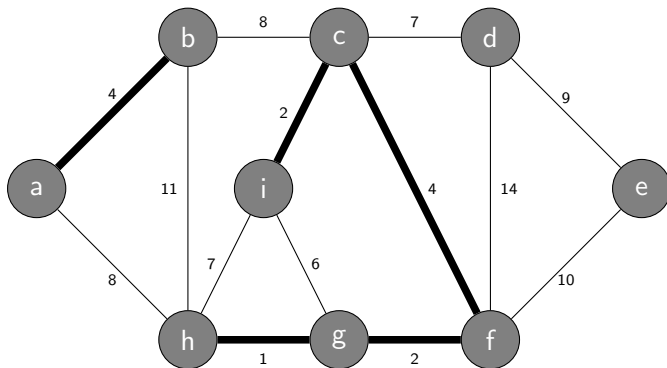
Example (Kruskal's Algorithm)

Creates cycle?



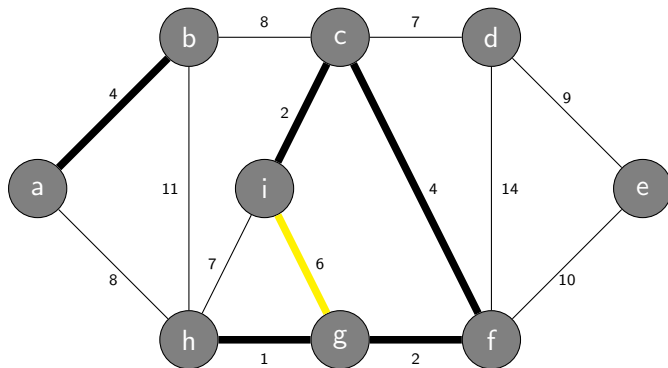
Example (Kruskal's Algorithm)

Creates cycle? No, so add it to the MST



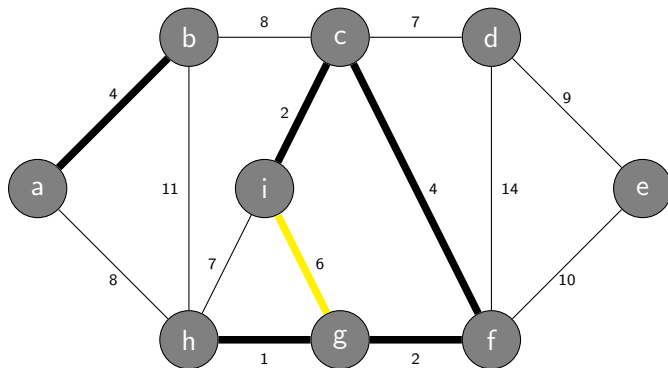
Example (Kruskal's Algorithm)

Creates cycle?



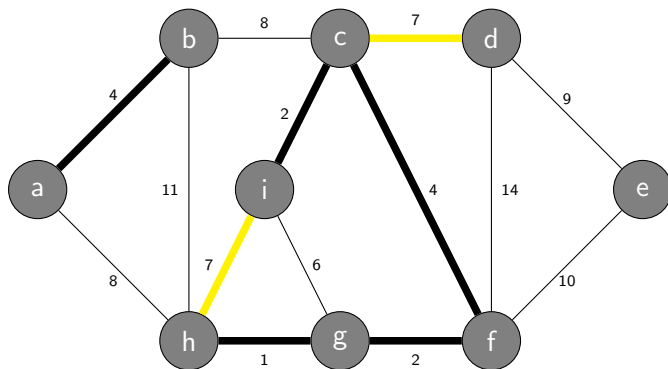
Example (Kruskal's Algorithm)

Creates cycle? Yes, skip this edge



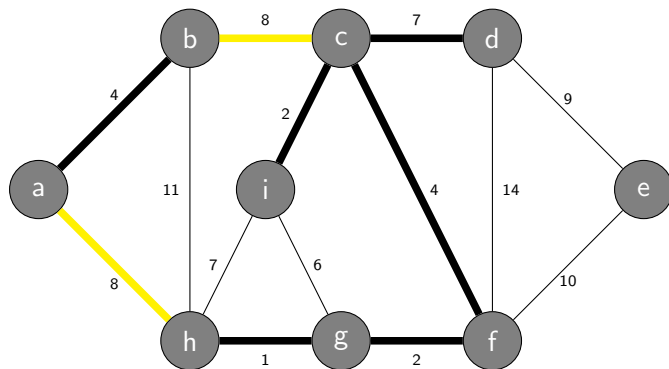
Example (Kruskal's Algorithm)

Two possibilities but one creates a cycle



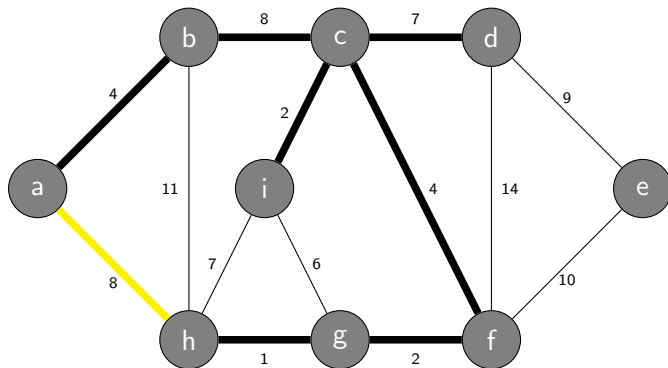
Example (Kruskal's Algorithm)

Two possibilities. Select one randomly.



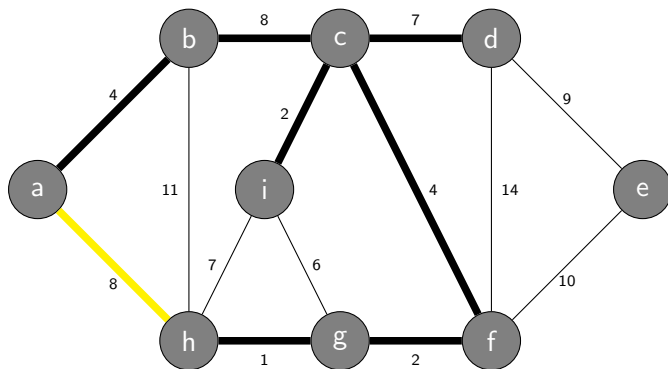
Example (Kruskal's Algorithm)

Creates cycle?



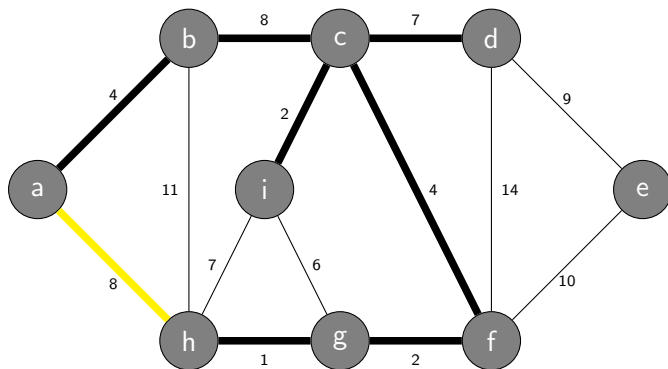
Example (Kruskal's Algorithm)

Creates cycle? Yes, skip this edge.



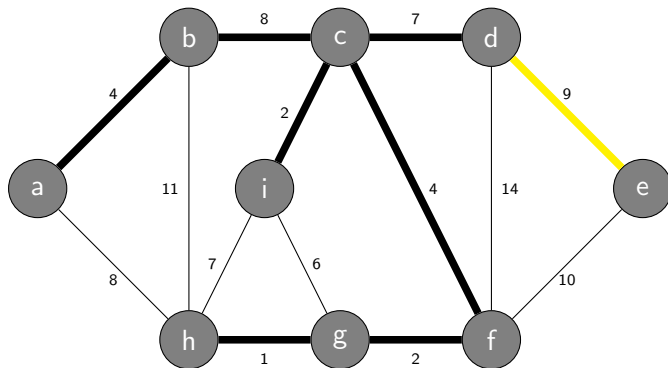
Example (Kruskal's Algorithm)

What if we had selected a-h before b-c?



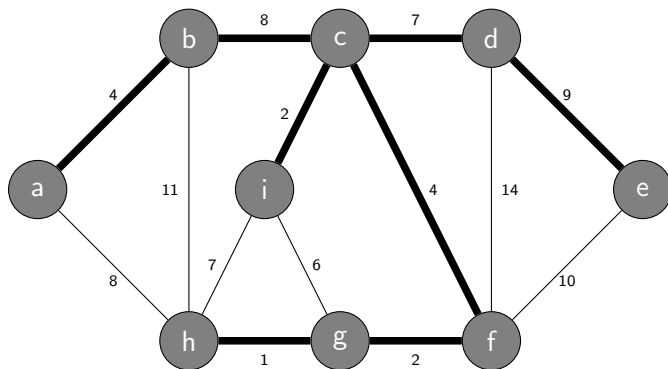
Example (Kruskal's Algorithm)

Creates cycle?



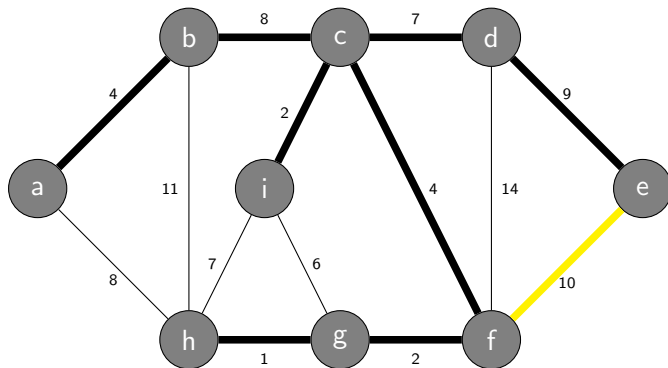
Example (Kruskal's Algorithm)

Creates cycle? No, so add it to the MST.



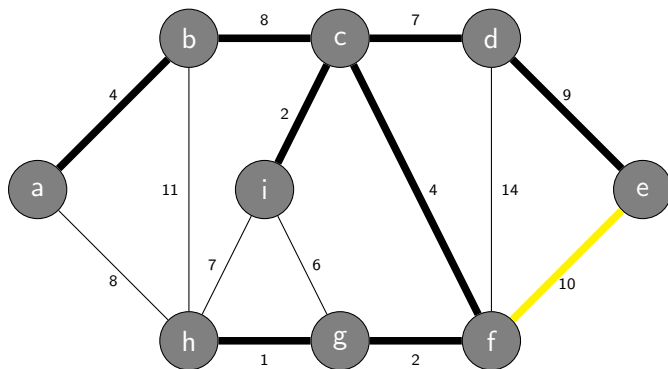
Example (Kruskal's Algorithm)

Creates cycle?



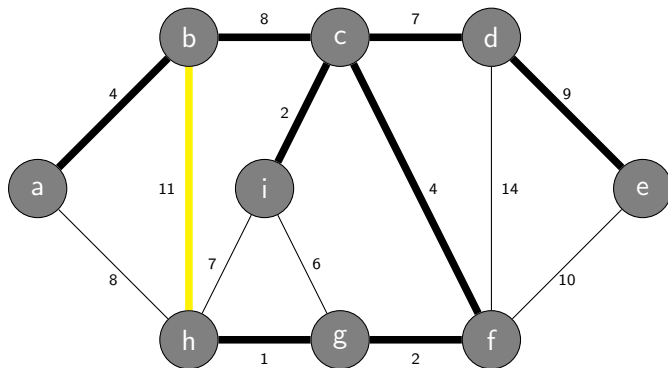
Example (Kruskal's Algorithm)

Creates cycle? Yes so skip this edge.



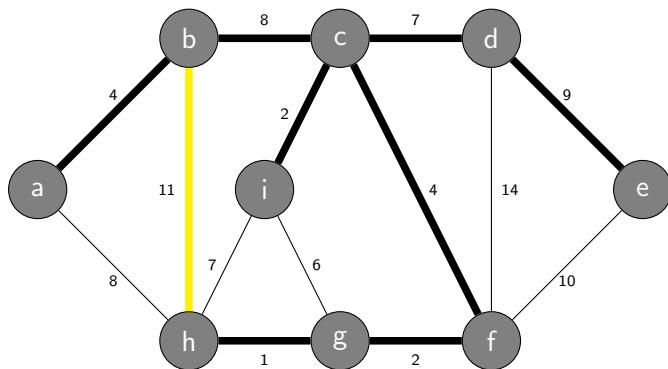
Example (Kruskal's Algorithm)

Creates cycle?



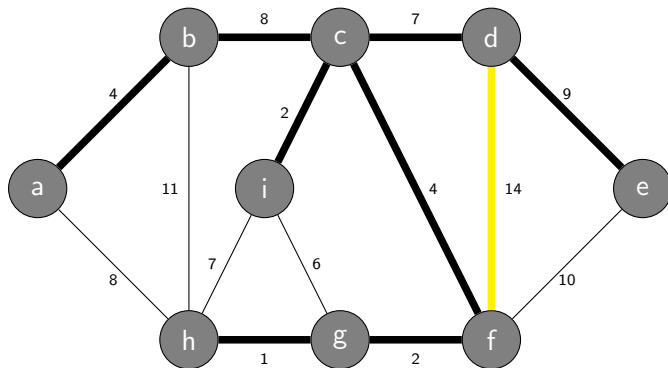
Example (Kruskal's Algorithm)

Creates cycle? Yes so skip this edge.



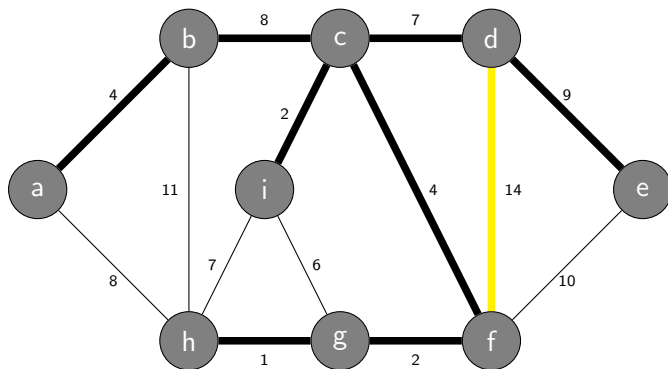
Example (Kruskal's Algorithm)

Creates cycle?



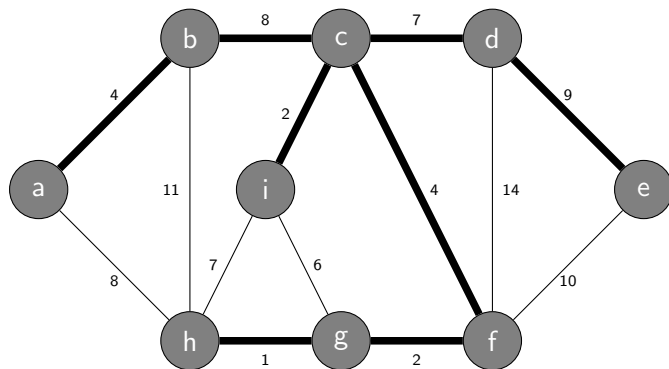
Example (Kruskal's Algorithm)

Creates cycle? Yes so skip this edge.



Example (Kruskal's Algorithm)

No more edges. MST is complete



High level implementation

- Given a graph $G = (V, E)$ with a weight function $w : E \rightarrow R$.
- We build an MST T by adding edges to T one at a time.
- One can iterate over the edges in increasing weight order by sorting E .
- We need a method to check if adding an edge to the current partial solution creates a cycle.
- This can be done by using the union-find data structure.

Disjoint Sets Data Structures

- Given a graph $G = (V, E)$ and arbitrary vertex $v \in V$.
- $\text{MAKE-SET}(v)$: create a new set whose only member is v .
- $\text{FIND-SET}(v)$: returns a pointer to the representative of the set containing v .
- $\text{UNION}(u, v)$: combine the sets containing u and v into a new set.
- An edge $(u, v) \in E$ creates a cycles iff $\text{FIND-SET}(u) = \text{FIND-SET}(v)$.

Algorithm 1: Kruskal's Algorithm

```
 $T \leftarrow \emptyset$   
foreach  $v \in V$  do  
    | MAKE-SET( $v$ )  
 $F \leftarrow \text{SORT-EDGES}(E)$   
foreach  $(u, v) \in F$  do  
    | if FIND-SET( $u$ )  $\neq$  FIND-SET( $v$ ) then  
        |  $T \leftarrow T \cup \{(u, v)\}$   
        | UNION( $u, v$ )
```

Complexity

Let $n = |V|$ and $m = |E|$.

```
1  $T \leftarrow \emptyset$ 
2 foreach  $v \in V$  do
3   | MAKE-SET( $v$ )  $O(1)$ 
4  $F \leftarrow \text{SORT-EDGES}(E)$   $O(m \log m)$ 
5 foreach  $(u, v) \in F$  do  $\Theta(m)$ 
6   | if FIND-SET( $u$ )  $\neq$  FIND-SET( $v$ ) then  $O(\log^* n)$ 
7     |  $T \leftarrow T \cup \{(u, v)\}$   $\Theta(1)$ 
8     | UNION( $u, v$ )  $O(\log^* n)$ 
```

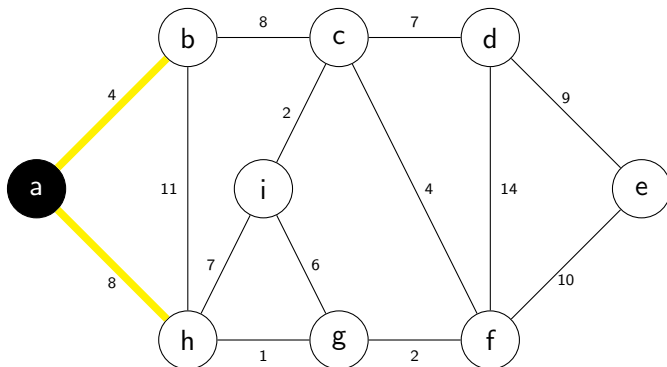
The dominant term is $O(m \log m)$ due to sorting. Often the complexity is written as $O(m \log n)$ since $m \leq n^2$.

Prim's algorithm

- Whereas Kruskal's algorithm is "edge" based, Prim's algorithm is "vertex" based.
- Prim's maintains a set S of vertices, initially containing a single vertex s which could be any vertex in V .
- At each iteration we consider the sets S and $V - S$.
- Find the edge (u, v) with minimum weight such that $u \in S$ and $v \in V - S$.
- Add v to S and (u, v) to T .

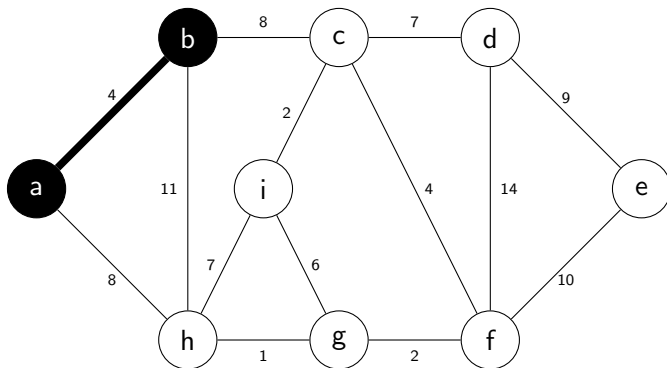
Example (Prim's Algorithm)

$$S = \{a\} \quad V - S = \{b, c, d, e, f, g, h, i\}$$



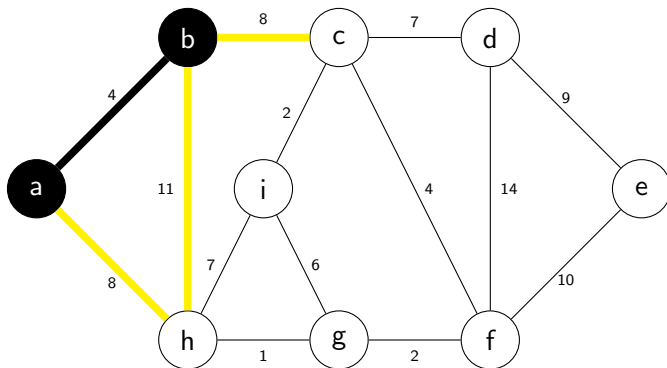
Example (Prim's Algorithm)

$$S = \{a, b\} \quad V - S = \{b, c, d, e, f, g, h, i\}$$



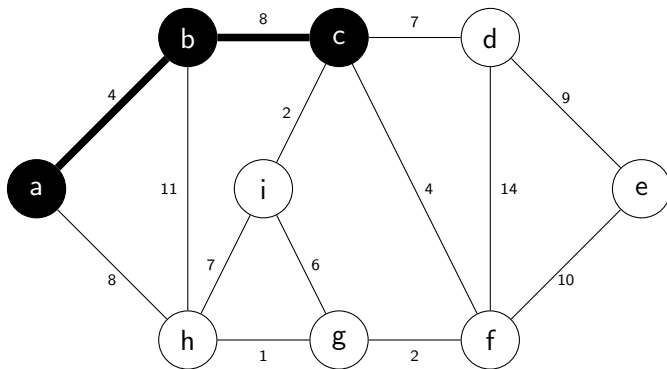
Example (Prim's Algorithm)

$$S = \{a, b\} \quad V - S = \{c, d, e, f, g, h, i\}$$



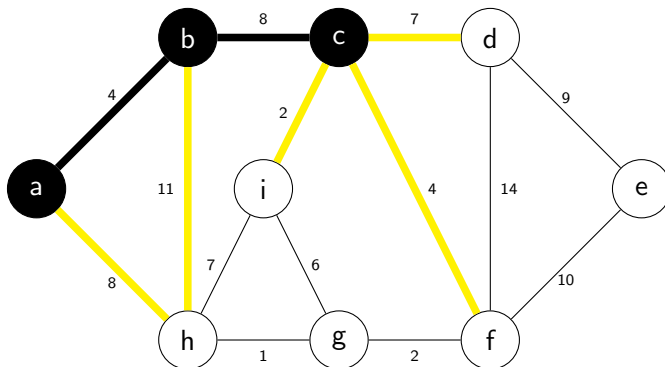
Example (Prim's Algorithm)

$$S = \{a, b, c\} \quad V - S = \{d, e, f, g, h, i\}$$



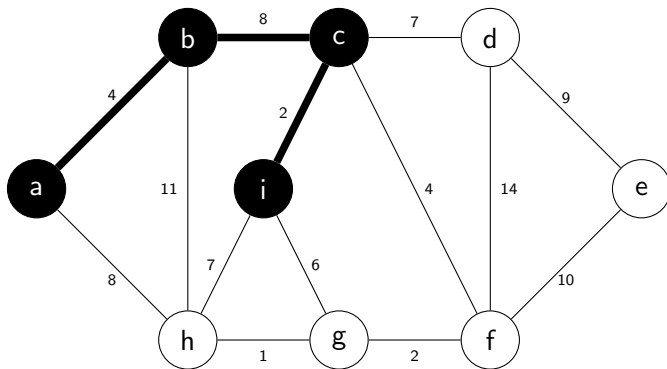
Example (Prim's Algorithm)

$$S = \{a, b, c\} \quad V - S = \{d, e, f, g, h, i\}$$



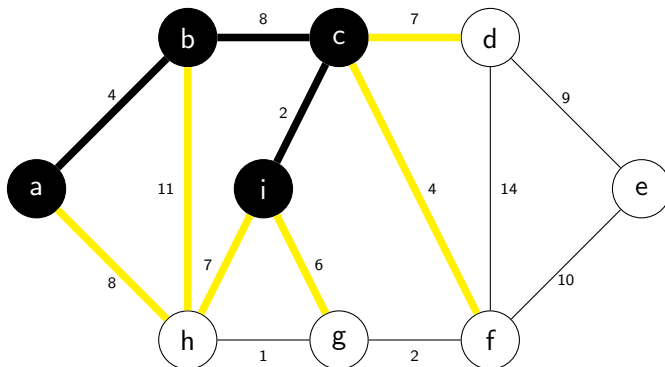
Example (Prim's Algorithm)

$$S = \{a, b, c, i\} \quad V - S = \{d, e, f, g, h\}$$



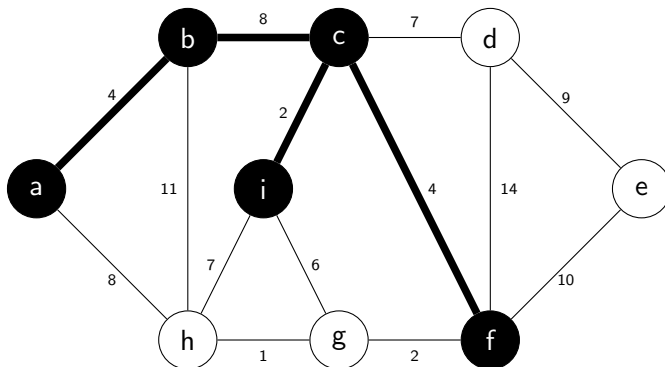
Example (Prim's Algorithm)

$$S = \{a, b, c, i\} \quad V - S = \{d, e, f, g, h\}$$



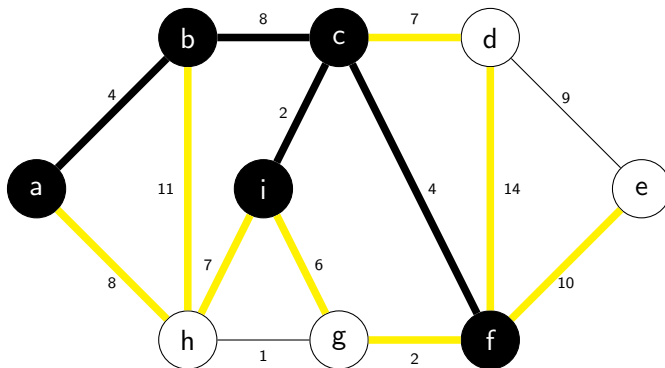
Example (Prim's Algorithm)

$$S = \{a, b, c, i, f\} \quad V - S = \{d, e, g, h\}$$



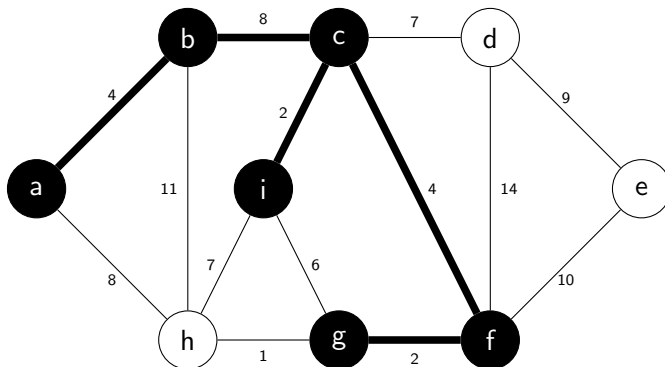
Example (Prim's Algorithm)

$$S = \{a, b, c, i, f\} \quad V - S = \{d, e, g, h\}$$



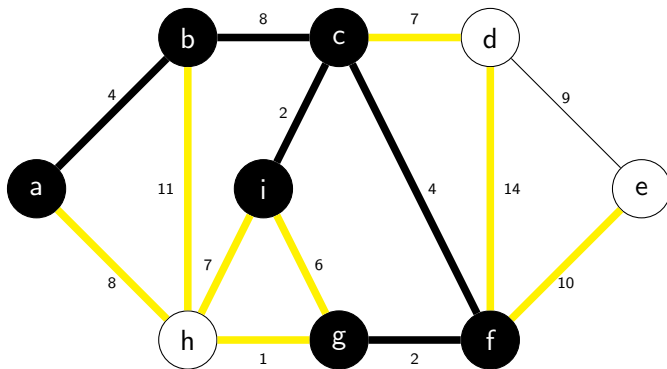
Example (Prim's Algorithm)

$$S = \{a, b, c, i, f, g\} \quad V - S = \{d, e, h\}$$



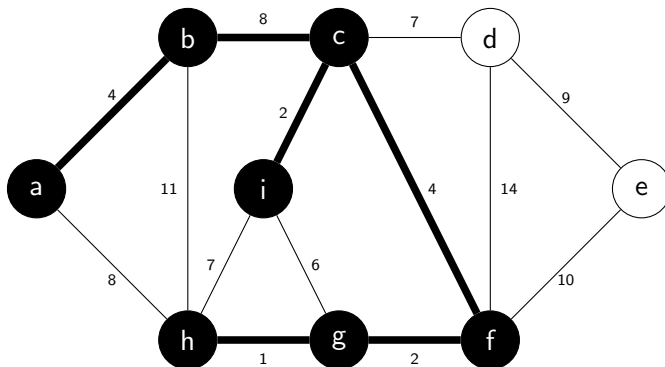
Example (Prim's Algorithm)

$$S = \{a, b, c, i, f, g\} \quad V - S = \{d, e, h\}$$



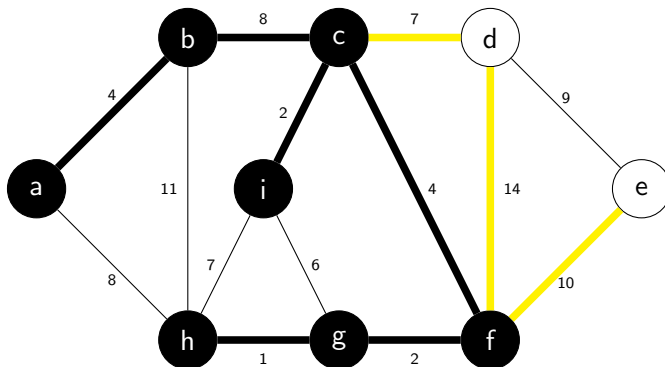
Example (Prim's Algorithm)

$$S = \{a, b, c, i, f, g, h\} \quad V - S = \{d, e\}$$



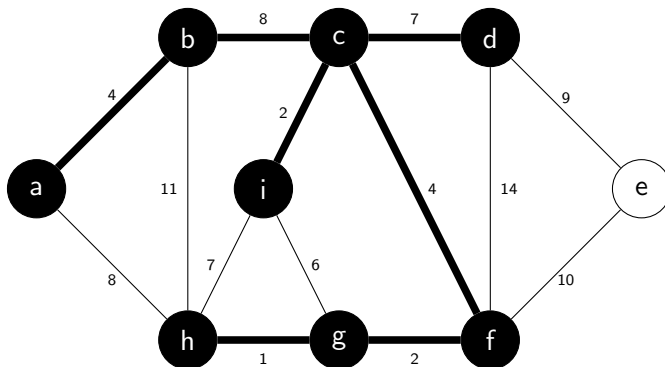
Example (Prim's Algorithm)

$$S = \{a, b, c, i, f, g, h\} \quad V - S = \{d, e\}$$



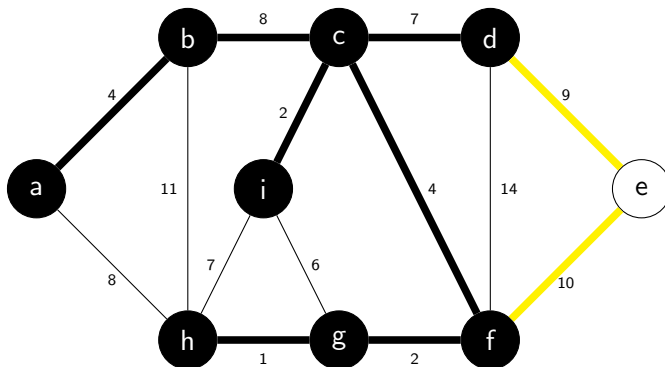
Example (Prim's Algorithm)

$$S = \{a, b, c, i, f, g, h, d\} \quad V - S = \{e\}$$



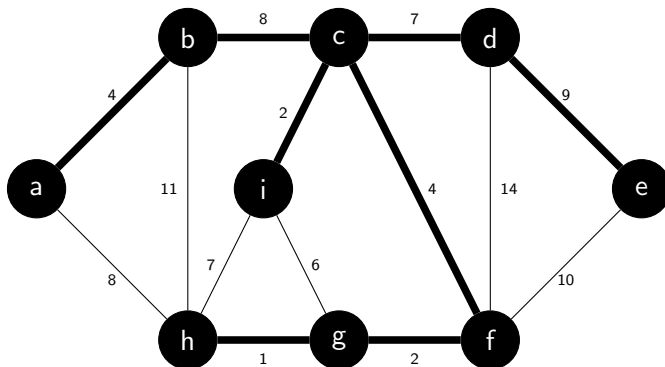
Example (Prim's Algorithm)

$$S = \{a, b, c, i, f, g, h, d\} \quad V - S = \{e\}$$



Example (Prim's Algorithm)

$$S = \{a, b, c, i, f, g, h, d, e\} \quad V - S = \{\}$$



High level implementation

Maintain sets S and $V - S$. Initially $S = \emptyset, T = \emptyset$. Then iterate the following:

- 1 Find $x \in S, y \in V - S$ such that $w(x, y) \leq w(u, v)$ for all $v \in S, u \in V - S$
- 2 Remove y from $V - S$ and add it to S . Also add edge (x, y) to T
- 3 Repeat until $V - S = \emptyset$

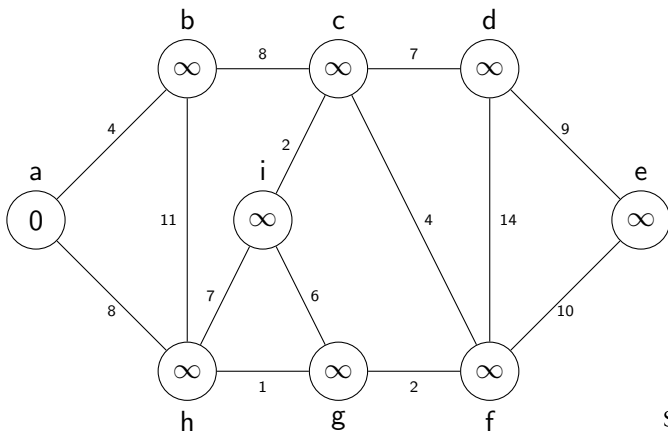
The challenge is to **efficiently** implement step 1 because brute force is $O(n^2)$ for each iteration.

Efficient implementation of step 1

- The trick is to not recompute the minimum edge each time but reuse previous results.
- Suppose that on a given iteration we have computed all the weights of edges (u, v) with $u \in S$ and $v \in V - S$.
- Further suppose for some $x \in S, y \in V - S$, $w(x, y)$ has the smallest weight.
- Then y is removed from $V - S$ and added to S .
- All we have to do is update the weights of edges (y, v) for all $v \in V - S$.

Illustrative example

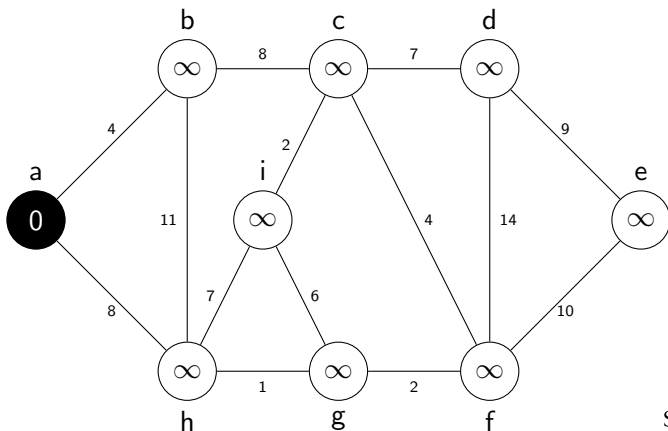
$S = \{\}$ $V - S = \{a, b, c, d, e, f, g, h, i\}$
Initially



Illustrative example

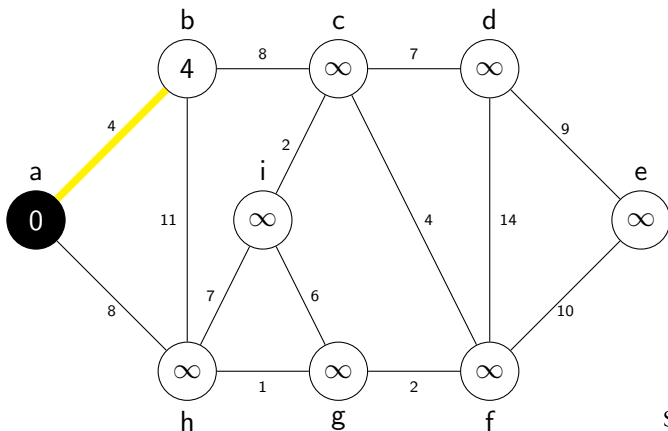
$$S = \{a\} \quad V - S = \{b, c, d, e, f, g, h, i\}$$

remove vertex with smallest key from $V - S$ and add to S



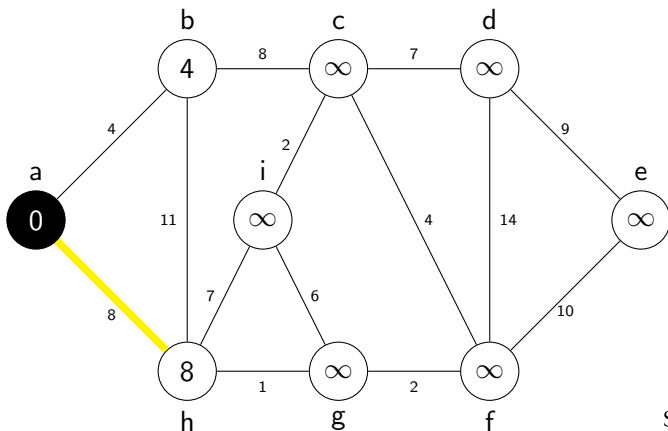
Illustrative example

$S = \{a\}$ $V - S = \{b, c, d, e, f, g, h, i\}$
update the neighbors of a



Illustrative example

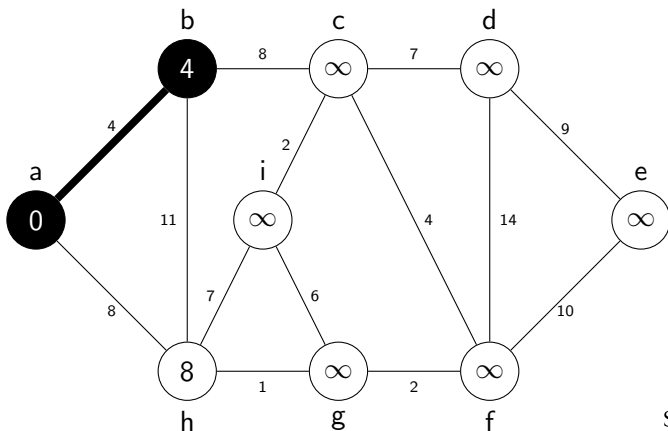
$S = \{a\}$ $V - S = \{b, c, d, e, f, g, h, i\}$
update the neighbors of a



Illustrative example

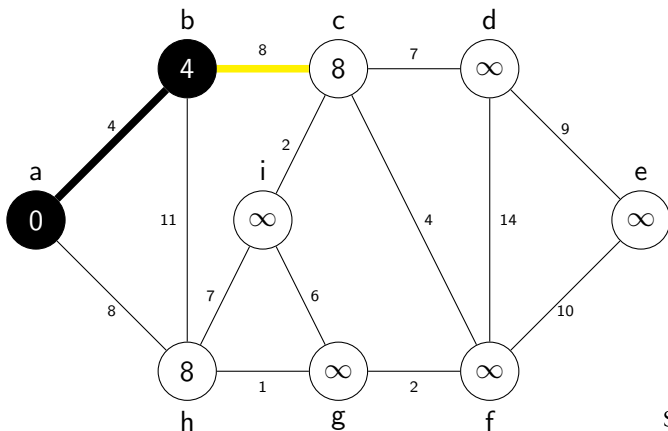
$$S = \{a, b\} \quad V - S = \{c, d, e, f, g, h, i\}$$

remove vertex with smallest key from $V - S$ and add to S



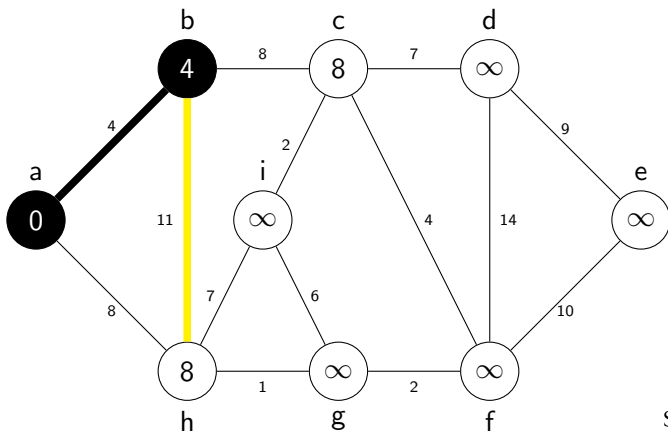
Illustrative example

$S = \{a, b\}$ $V - S = \{c, d, e, f, g, h, i\}$
update the neighbors of b



Illustrative example

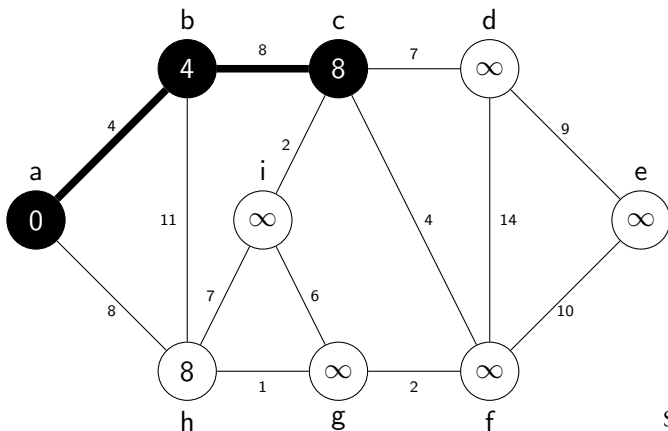
$S = \{a, b\}$ $V - S = \{c, d, e, f, g, h, i\}$
update the neighbors of b



Illustrative example

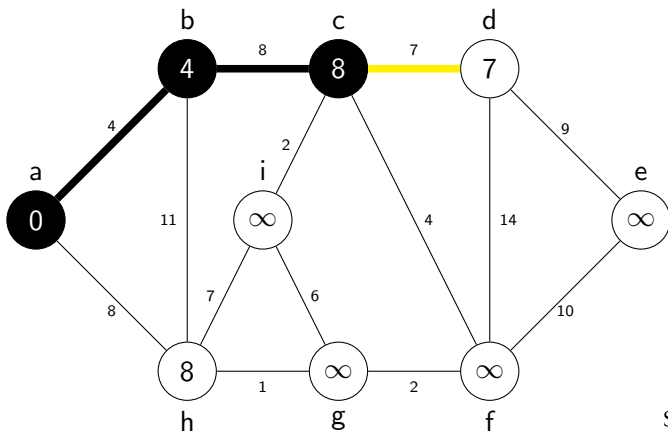
$$S = \{a, b, c\} \quad V - S = \{d, e, f, g, h, i\}$$

remove vertex with smallest key from $V - S$ and add to S



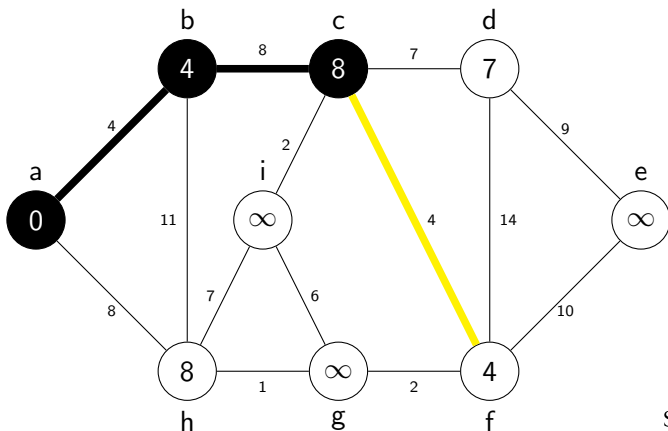
Illustrative example

$S = \{a, b, c\}$ $V - S = \{d, e, f, g, h, i\}$
update the neighbors of c



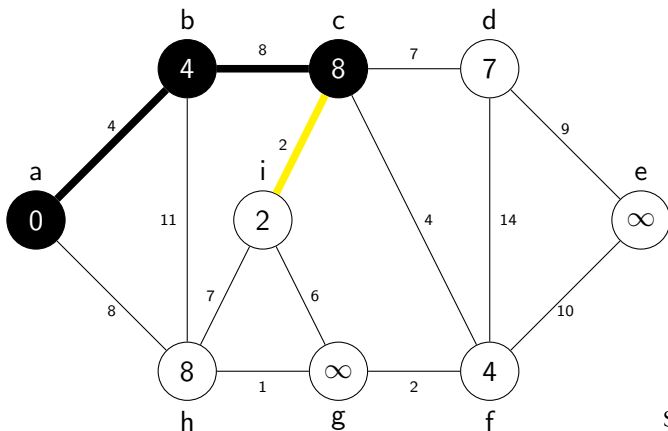
Illustrative example

$S = \{a, b, c\}$ $V - S = \{d, e, f, g, h, i\}$
update the neighbors of c



Illustrative example

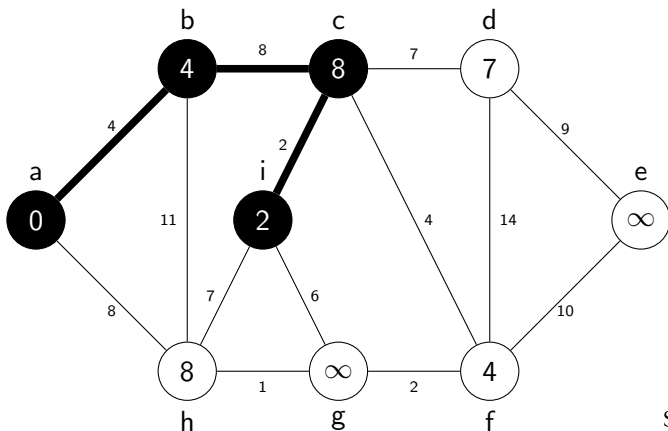
$S = \{a, b, c\}$ $V - S = \{d, e, f, g, h, i\}$
update the neighbors of c



Illustrative example

$$S = \{a, b, c, i\} \quad V - S = \{d, e, f, g, h\}$$

remove vertex with smallest key from $V - S$ and add to S



Pseudo code

Algorithm 2: Prim's algorithm

```
1 foreach  $v \in V$  do
2    $v.key \leftarrow \infty$ 
3    $v.p \leftarrow NULL$ 
4  $s.key \leftarrow 0$ 
5  $Q \leftarrow V$ 
6  $T \leftarrow \emptyset$ 
7 while  $Q \neq \emptyset$  do
8    $u \leftarrow \text{DELETE-MIN}(Q)$ 
9    $T \leftarrow T \cup \{(u, u.p)\}$ 
10  foreach  $v \in \text{adj}[u] \wedge v \notin S$  do
11    if  $w(u, v) < v.key$  then
12       $v.key \leftarrow w(u, v)$ 
13       $v.p \leftarrow u$ 
```

Notes

- The choice of the "source" vertex on line 4 is arbitrary, any vertex can be chosen as source.
- For clarity an if statement on line 9 was omitted.
- It checks if $u.p$ is null in which case no edge is added to T .
- This occurs only for the first iteration because the source s has no predecessor.
- The usual implementation has S as a hash table for $\Theta(1)$ $v \notin S$ operation and Q as a priority queue for $\Theta(\log n)$ DELETE-MIN.

Complexity

1	foreach $v \in V$ do	$\Theta(n)$
2	$v.key \leftarrow \infty, v.p \leftarrow NULL$	both $\Theta(1)$
3	$s.key \leftarrow 0, T \leftarrow \emptyset$	both $\Theta(1)$
4	$Q \leftarrow V$	$\Theta(n)$
5	while $Q \neq \emptyset$ do	$\Theta(n)$
6	$u \leftarrow \text{DELETE-MIN}(Q)$	$\Theta(\log n)$
7	$T \leftarrow T \cup \{(u, u.p)\}$	$\Theta(1)$
8	foreach $v \in \text{adj}[u]$ do	$ \text{adj}[u] $
9	if $v \notin S \wedge w(u, v) < v.key$ then	
10	$v.key \leftarrow w(u, v)$	$\Theta(\log n)$ Why?
11	$v.p \leftarrow u$	$\Theta(1)$

The dominant term comes from the execution of line 10 which is $O(\log n)$ and executed $O(m)$ times for a total of $O(m \log n)$

General MST algorithm

- Both Kruskal's and Prim's algorithms are special cases of a general method to obtain a minimum spanning tree.
- The basic idea is based on the following:
- Maintain a set of edges T .
- Before every iteration T is a subset of some minimum spanning tree.
- At each step we add an edge to T such that T is **still** a subset of some MST.
- An edge having that property is called **safe** for T .

MST (G)

$$T \leftarrow \emptyset$$
while T is not MST **do**

find edge (u, v) safe for T

$$T \leftarrow T \cup \{(u, v)\}$$
return T

- If you are interested in the proof of correctness, see the "extra" notes.