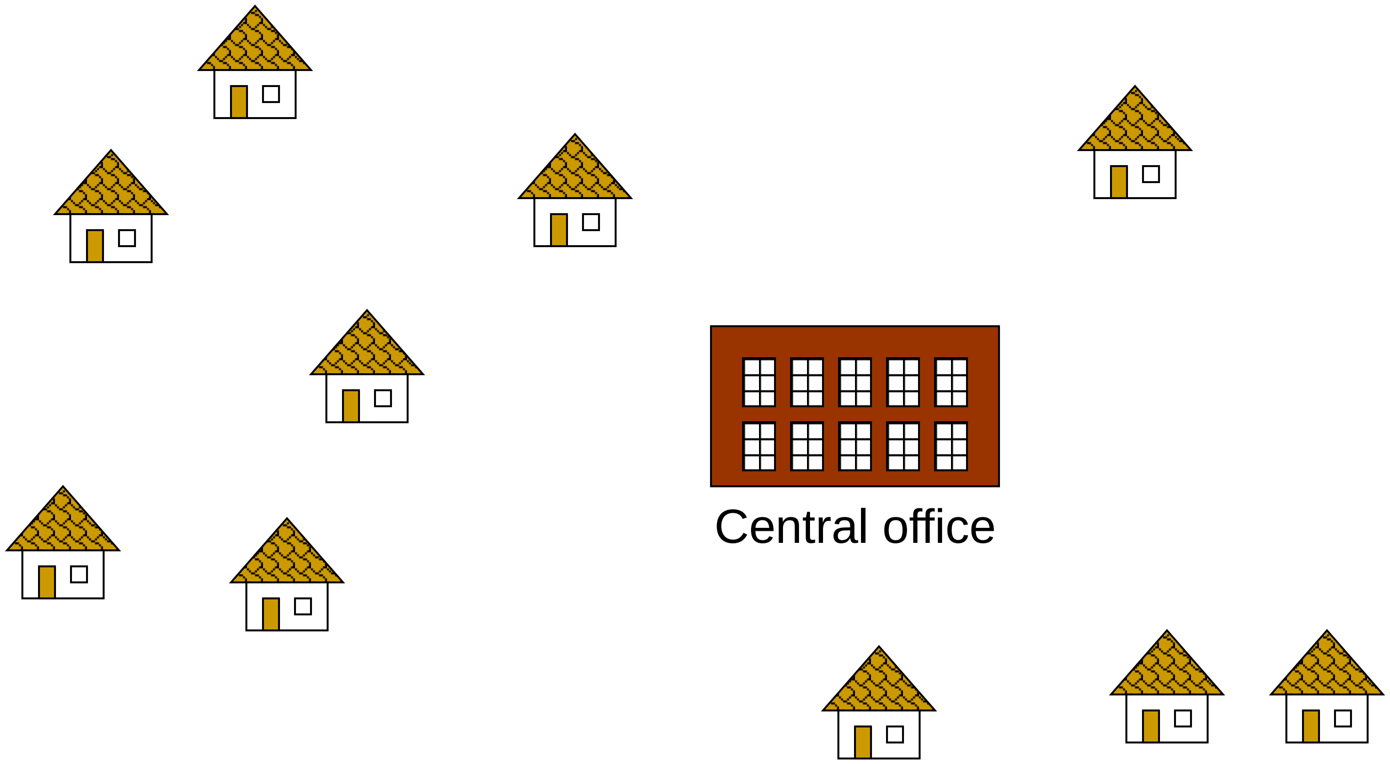
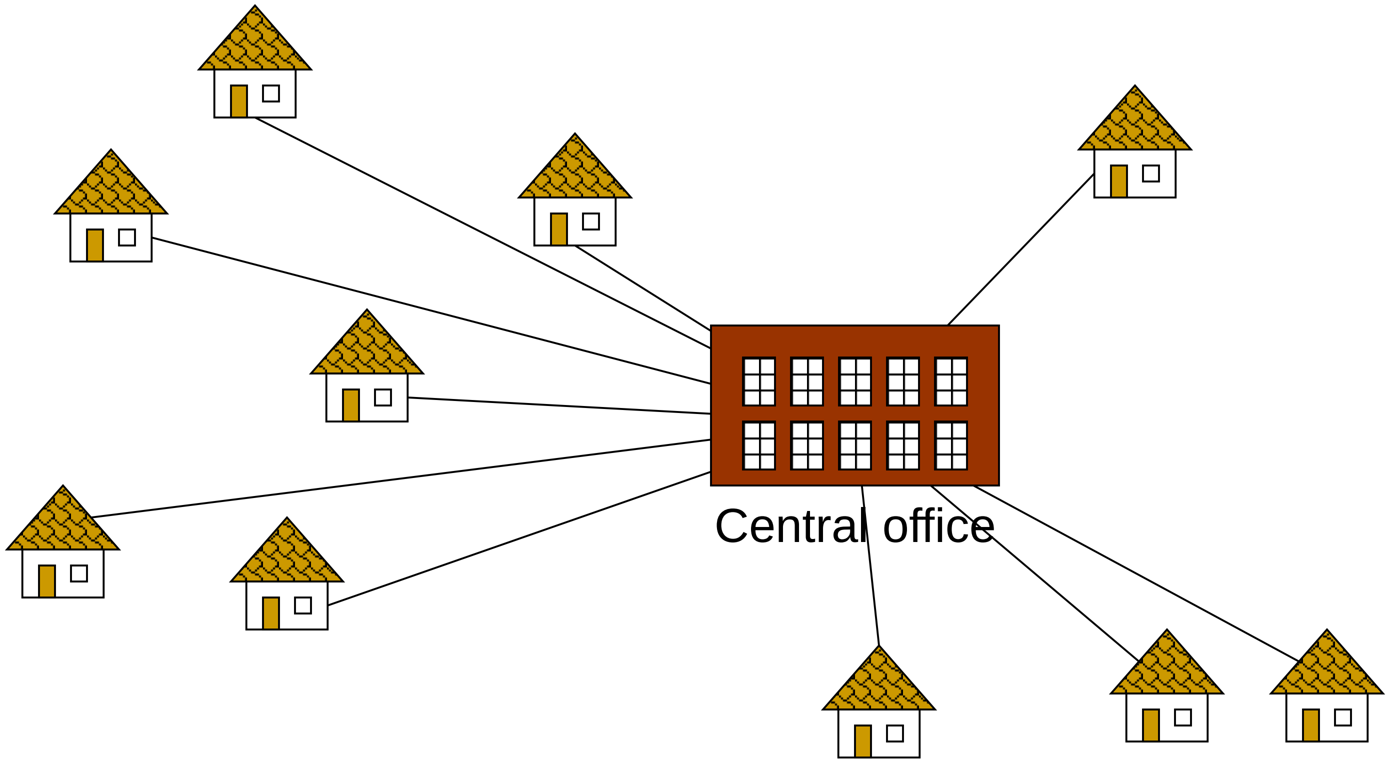
Minimum Spanning Trees

Suggested Readings: CLRS Chap. 23

Problem: Laying Telephone Wire

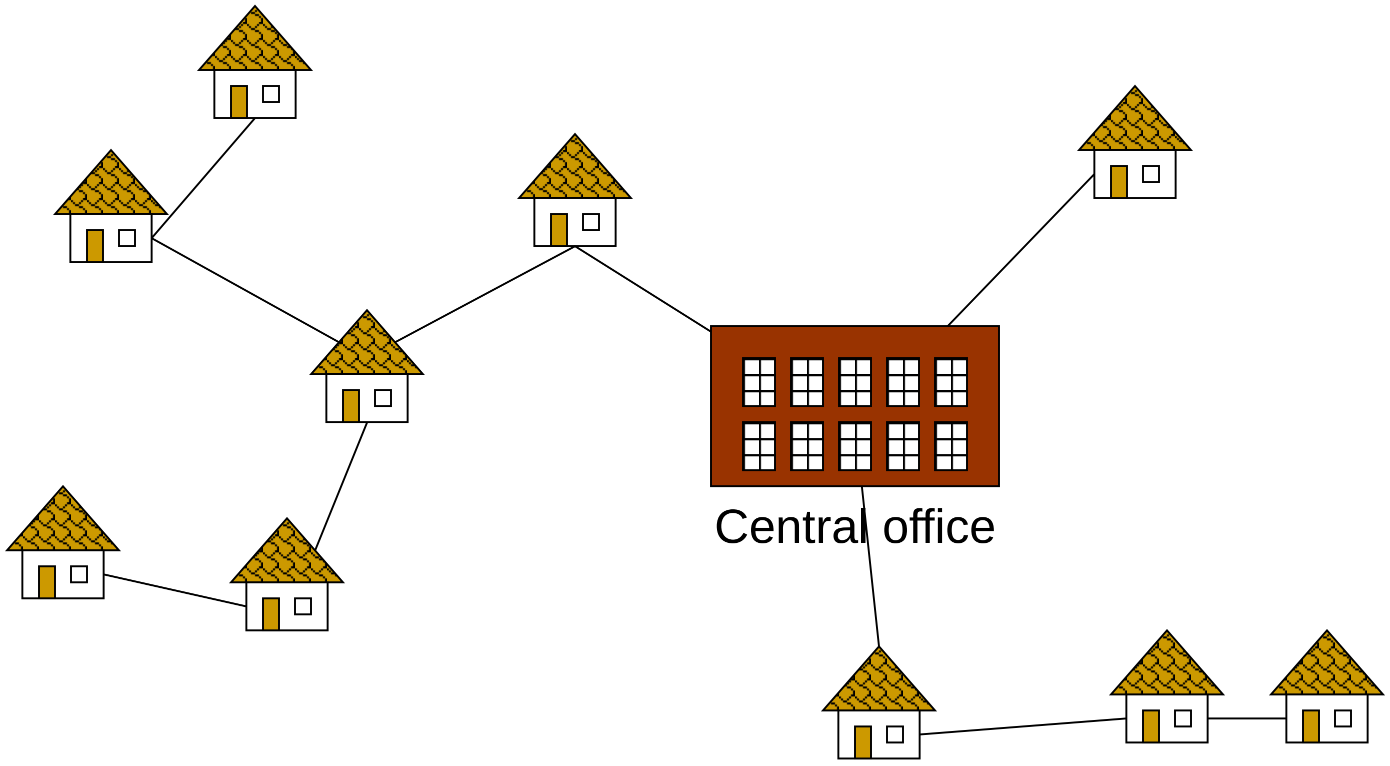


Wiring: Naïve Approach



Expensive!

Wiring: Better Approach

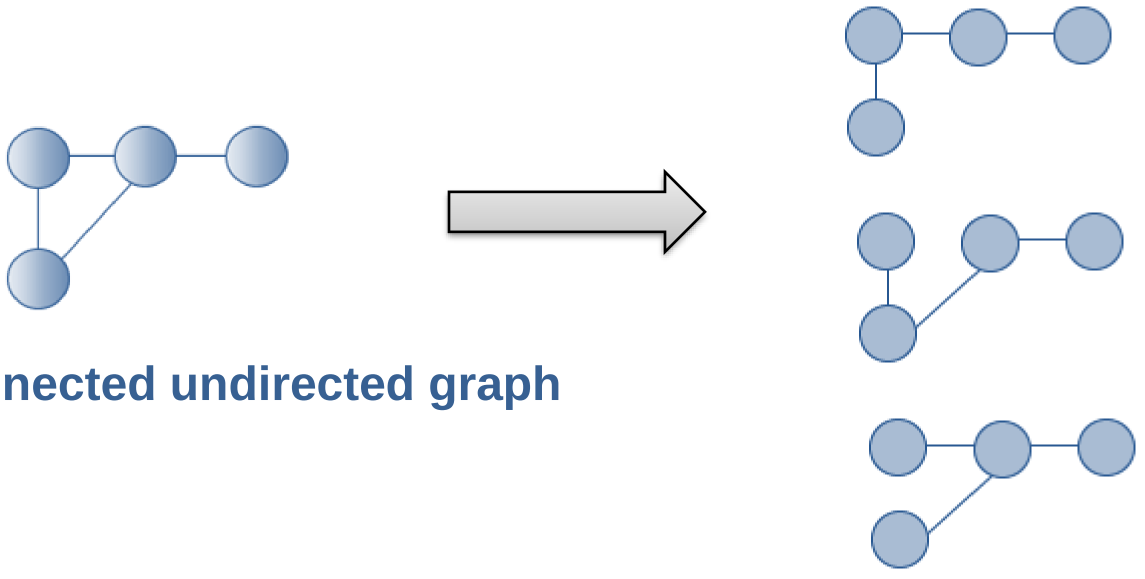


Minimize the total length of wire connecting the customers

Spanning Tree

Given a *connected undirected graph* $\boldsymbol{G = (V, E, w)}$, a spanning tree of ***G*** is a sub-graph that is a tree that covers all the vertices in ***V***.

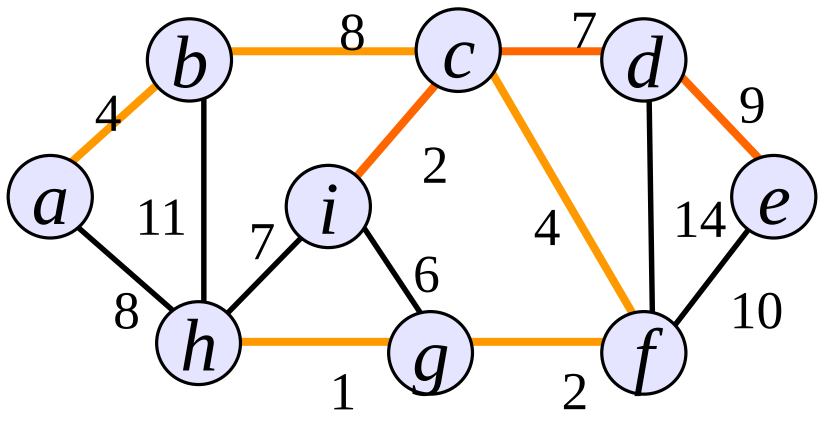
- A single graph can have many spanning trees.



Minimum Spanning Tree (MST)

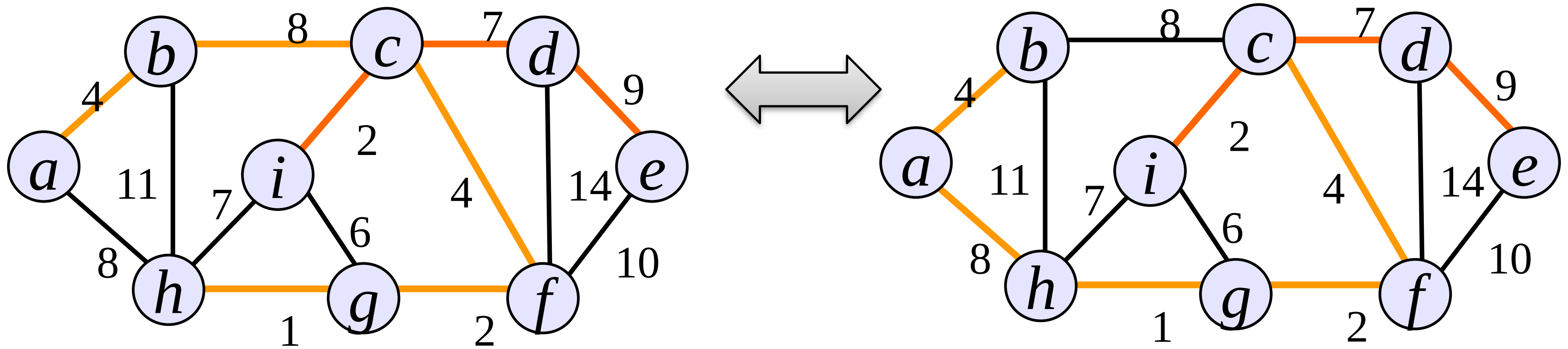
A **minimum spanning tree** is a sub-graph of an undirected weighted graph $\boldsymbol{G = (V, E, w)}$, such that

- ❑ it is a tree (i.e., it is acyclic)
- ❑ it covers all the vertices *V*
  - ❑ contains *|V| - 1* edges
- ❑ the total cost associated with tree edges is the minimum among all possible spanning trees
- ❑ not necessarily unique



Example

- Minimum spanning trees are not unique
  - If we replace (b, c) with (a, h), get a different spanning tree with the same cost
- MST have no cycles
  - We can take out an edge of a cycle, and still have the vertices connected while reducing the cost

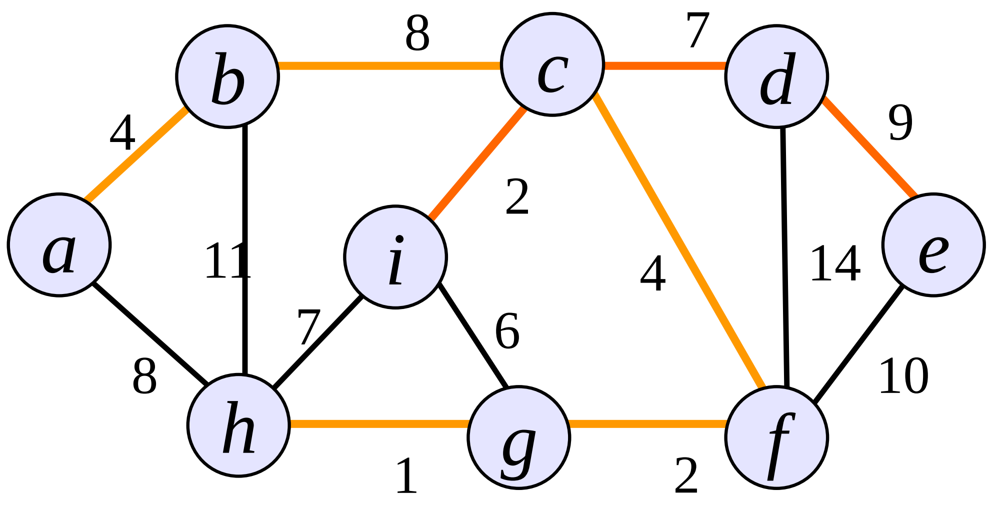


Applications of MST

- Any time you want to visit all vertices in a graph at minimum cost (e.g., wire routing on printed circuit boards, sewer pipe layout, and road planning, etc.)
- **Network Design:** MSTs are used in designing communication networks, electrical grids, and transportation systems. They help in creating cost-effective networks by minimizing the total length or cost of connections between nodes.
- **Power Distribution:** In power systems, MSTs are used for planning the distribution of power lines and optimizing the layout to reduce power losses and operational costs.

Generic Solution : To Compute MST

- Minimum-spanning-tree problem: Find a MST for a connected, undirected graph, with a weight function associated with its edges

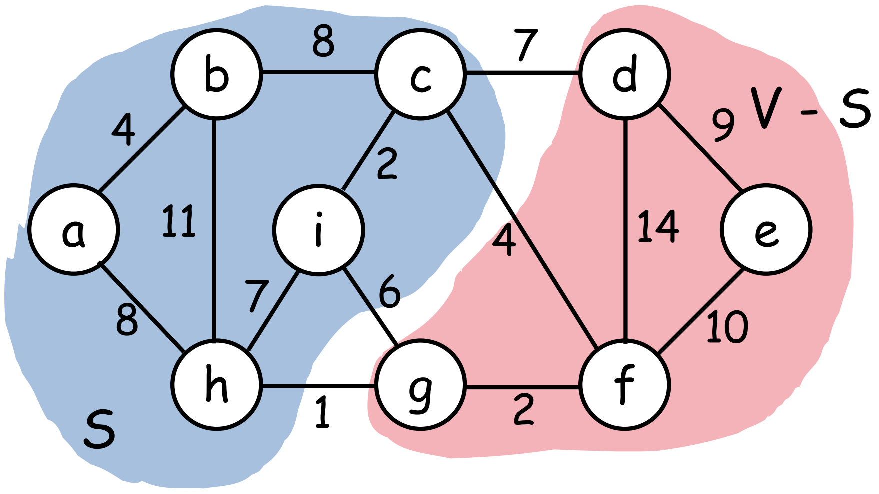


A generic solution:

- Build a set A of edges (initially empty)
- Incrementally add edges to A such that they would belong to a MST
- Safe Edge: An edge (u, v) is safe for A : A ∪ {(u, v)} is also a subset of some MST

How to Find Safe Edge?

- Let us look at edge (h, g)
  - Is it safe for A initially?



  - Let S ⊂ V be any set of vertices that includes h but not g (so that g is in V - S)
  - In any MST, there has to be one edge (at least) that connects S with V - S
  - Why not choose edge with minimum weight (h, g)

Generic Algorithm: Minimum Spanning Tree

```
GENERIC-MST($G$, $w$)
1   $A \leftarrow \emptyset$
2   while $A$ does not form a spanning tree
3       do find an edge $(u, v)$ that is safe for $A$
4           $A \leftarrow A \cup \{(u, v)\}$
5   return $A$
```

Strategy: Growing Minimum Spanning Tree

- The algorithm uses greedy strategy which grows MST one edge at a time.
- Given a connected, undirected graph $G = (V, E)$ with a weight function $w : E \rightarrow \mathbf{R}$
- Algorithm manages a set of edges A, maintaining *loop invariant*
  - ***Prior to each iteration, A is a subset of some MST***
- An edge (u, v) is a **safe edge** for A such that $A \cup \{(u, v)\}$ is also a subset of some MST.
- Algorithms, discussed here, to find safe edge are
  - Kruskal's Algorithm
  - Prim's Algorithm

Prim’s Algorithm

- Prim’s algorithm finds a minimum cost spanning tree by selecting edges from the graph one-by-one as follows:
- It starts with a tree, T, consisting of the starting vertex, $\boldsymbol{r}$.
- Then, it adds the shortest edge emanating from $\boldsymbol{r}$ that connects T to the rest of the graph.
- It then moves to the added vertex and repeats the process.

Prim’s Algorithm

Start with some root node s and greedily grows a tree *T* from *s* outward.
At each step, add the cheapest edge *e* to T that has exactly one endpoint in *T*.

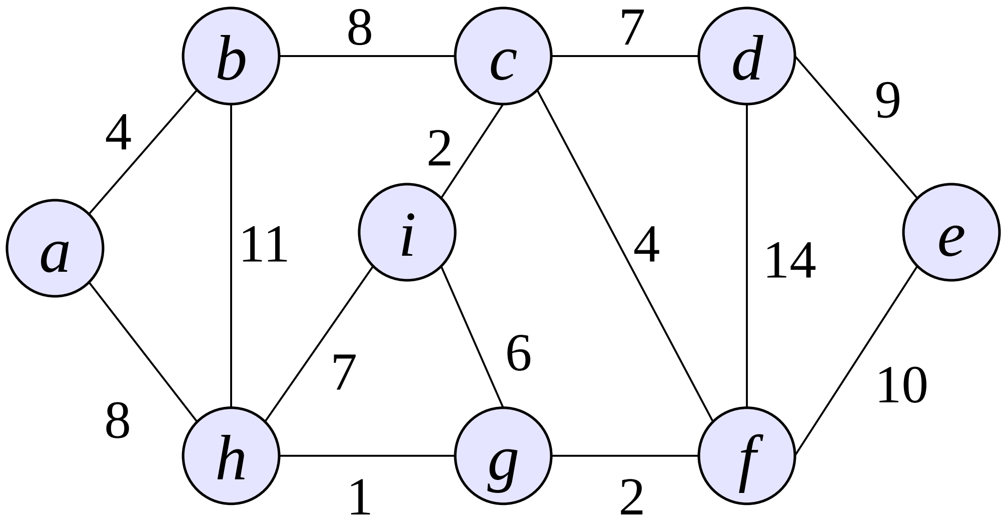
```
MST-PRIM$(G, w, r)$
for each $u \in G.V$
    $u.key = \infty$
    $u.\pi = \text{NIL}$
$r.key = 0$
$Q = G.V$
while $Q \neq \emptyset$
    $u = \text{EXTRACT-MIN}(Q)$
    for each $v \in G.Adj[u]$
        if $v \in Q$ and $w(u, v) < v.key$
            $v.\pi = u$
            $v.key = w(u, v)$
```

Prim’s Algorithm

- The performance depends on the implementation of min-priority queue Q.

- Using binary min-heap
  - Total Running time = O (m log n)

Example: Prim's Algorithm

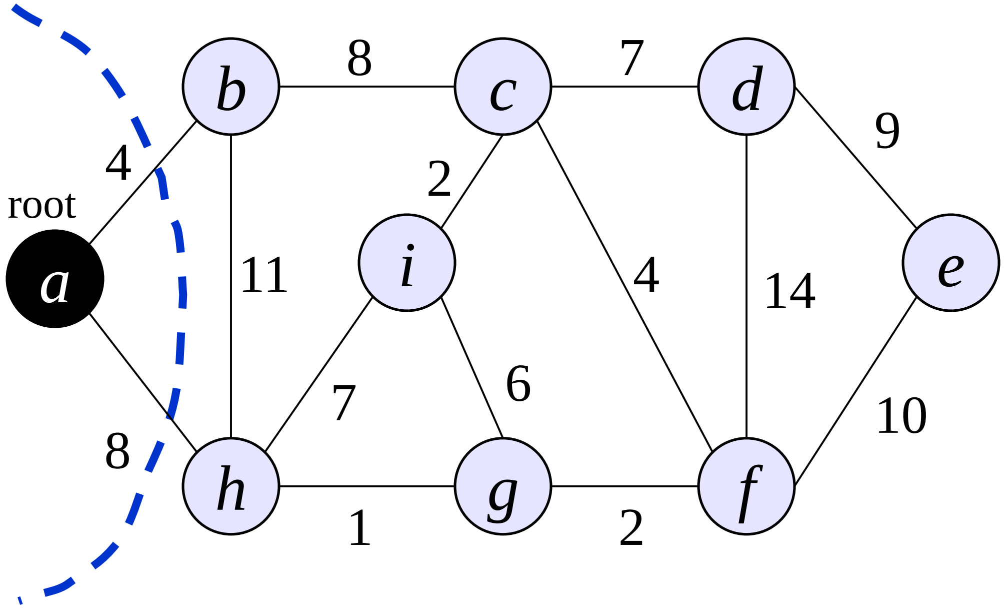


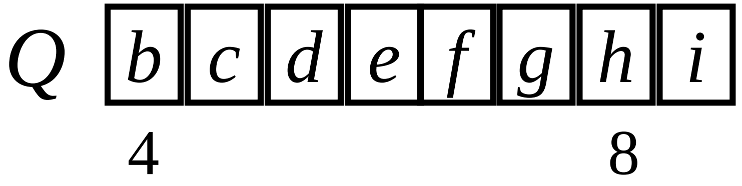
For each vertex u ∈ *V(G)*
key[*u*] = ∞
π [u] = NIL

Considering *a* as root node
key[a] = 0

Q	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*
	0								

Example: Prim's Algorithm





a = EXTRACT-MIN(Q)

$Adj[a] = b, h$

$b \in Q$ and

$w(a, b) < key[b]$ $(4 < \infty)$

$\quad \pi[b] = a$

$\quad key[b] = w(a, b) = 4$

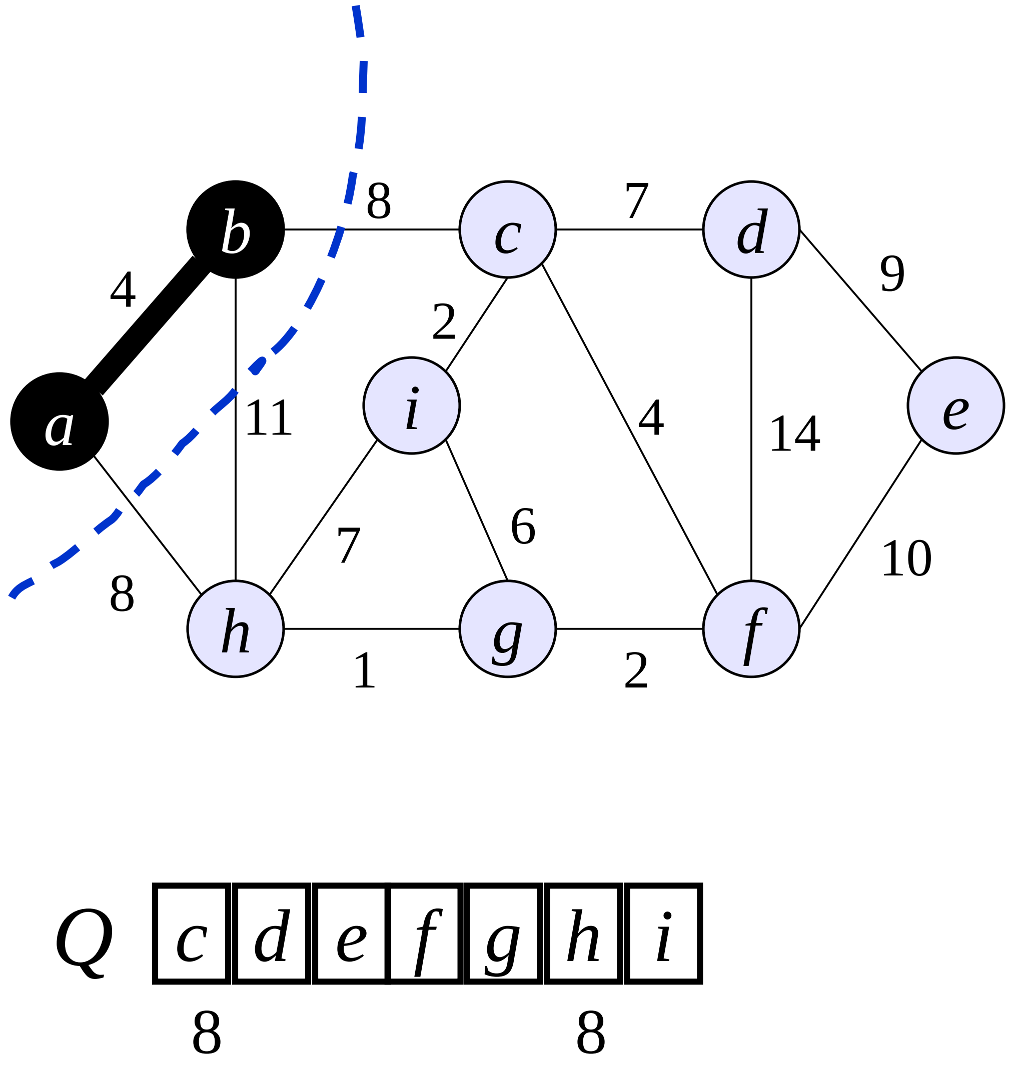
$h \in Q$ and

$w(a, h) < key[h]$ $(8 < \infty)$

$\quad \pi[h] = a$

$\quad key[h] = w(a, h) = 8$

Example: Prim's Algorithm



b = EXTRACT-MIN(*Q*)

Adj[b] = a, c, h

a does not belong to Q

$c \in Q$ and

$w(b, c) < key[c]$ $(8 < \infty)$

$\pi[c] = b$

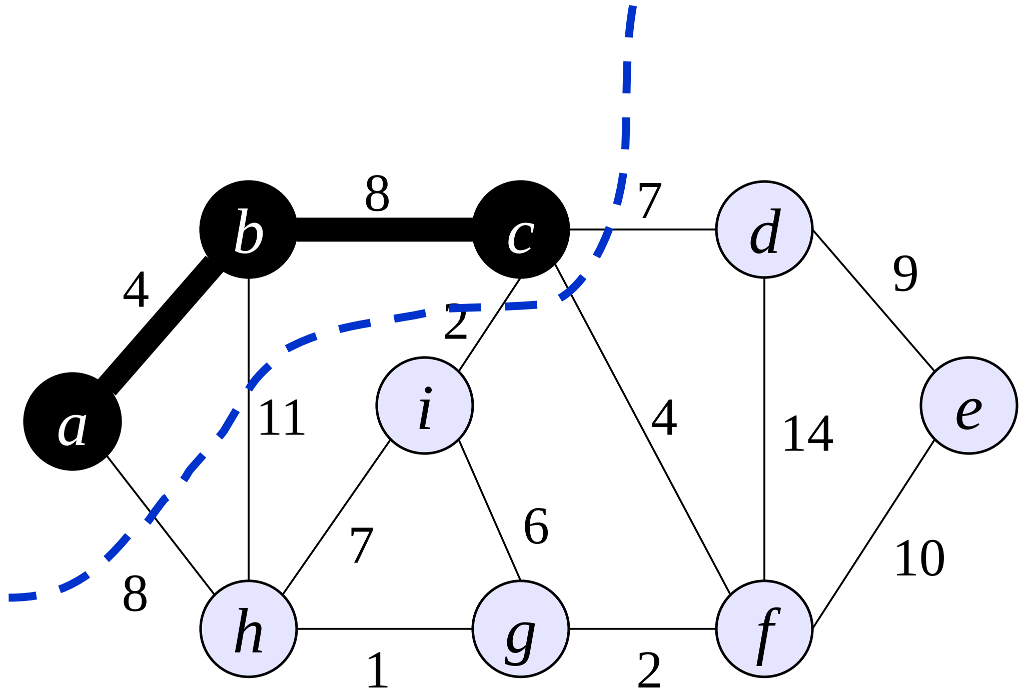
$key[c] = w(b, c) = 8$

$h \in Q$ and

$w(b, h) < key[h]$

(but 11 > 8)

Example: Prim’s Algorithm



Q

d	e	f	g	h	i
7		4		8	2

c = EXTRACT-MIN(Q)

$Adj[c] = b, d, f, i$

b does not belong to Q

$d \in Q$ and

$w(c, d) < key[d]$ $(7 < \infty)$

$\pi[d] = c$

$key[d] = w(c, d) = 7$

$f \in Q$ and

$w(c, f) < key[f]$ $(4 < \infty)$

$\pi[f] = c$

$key[f] = w(c, f) = 4$

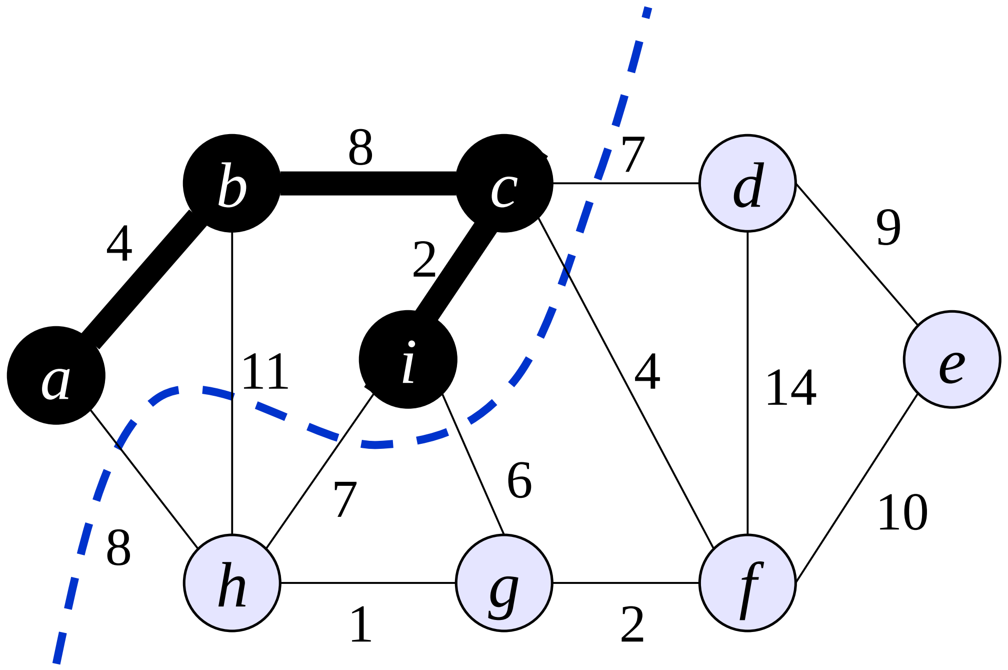
$i \in Q$ and

$w(c, i) < key[i]$ $(2 < \infty)$

$\pi[i] = c$

$key[i] = w(c, i) = 2$

Example: Prim's Algorithm



Q

d	e	f	g	h
7		4	6	7

i = EXTRACT-MIN(Q)

$Adj[i] = c, g, h$

$c \notin Q$

$g \in Q$ and

$w(i, g) < key[g]$ $(6 < \infty)$

$\pi[g] = i$

$key[g] = w(i, g) = 6$

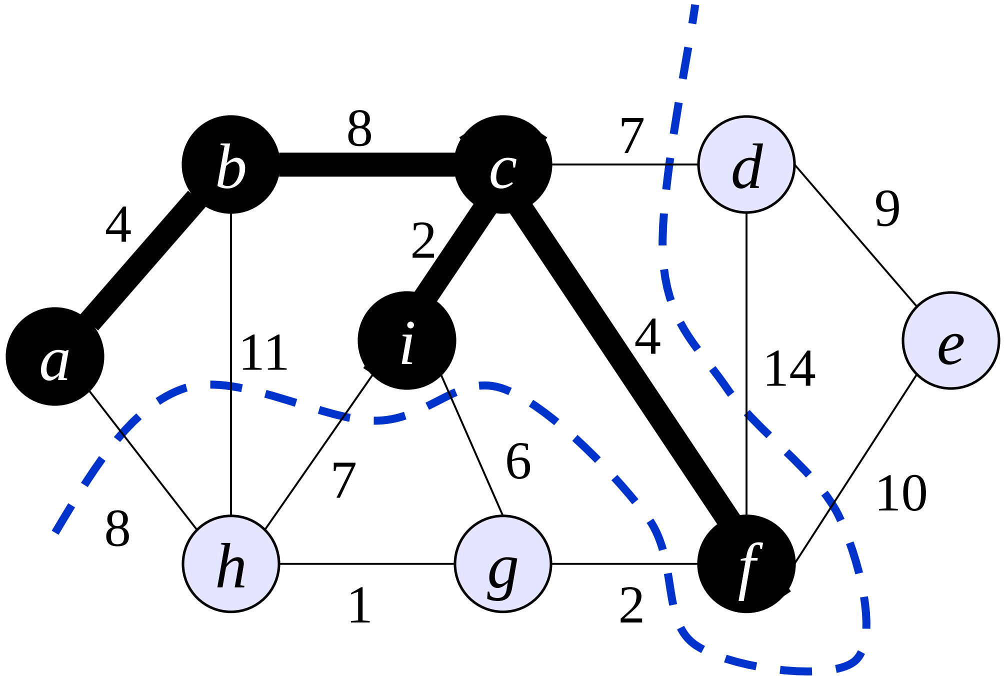
$h \in Q$ and

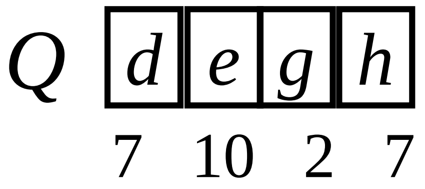
$w(i, h) < key[h]$ $(7 < 8)$

$\pi[h] = i$

$key[h] = w(i, h) = 7$

Example: Prim’s Algorithm





f = EXTRACT-MIN(*Q*)
Adj[f] = *c*, *d*, *e*, *g*

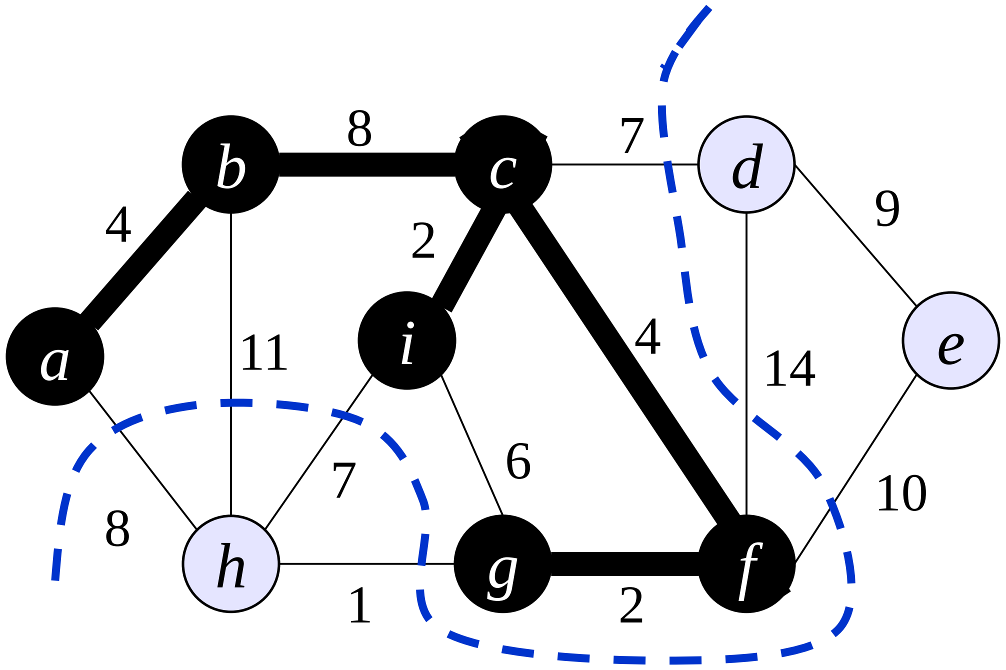
$c \notin Q$

$d \in Q$ and
$w(f, d) < key[d]$
But (14 < 7)

$e \in Q$ and
$w(f, e) < key[e]$ ($10 < \infty$)
    $\pi[e] = f$
    $key[e] = w(f, e) = 10$

$g \in Q$ and
$w(f, g) < key[g]$ (2 < 6)
    $\pi[g] = f$
    $key[g] = w(f, g) = 2$

Example: Prim's Algorithm



g = EXTRACT-MIN(Q)
$Adj[g] = f, h, i$

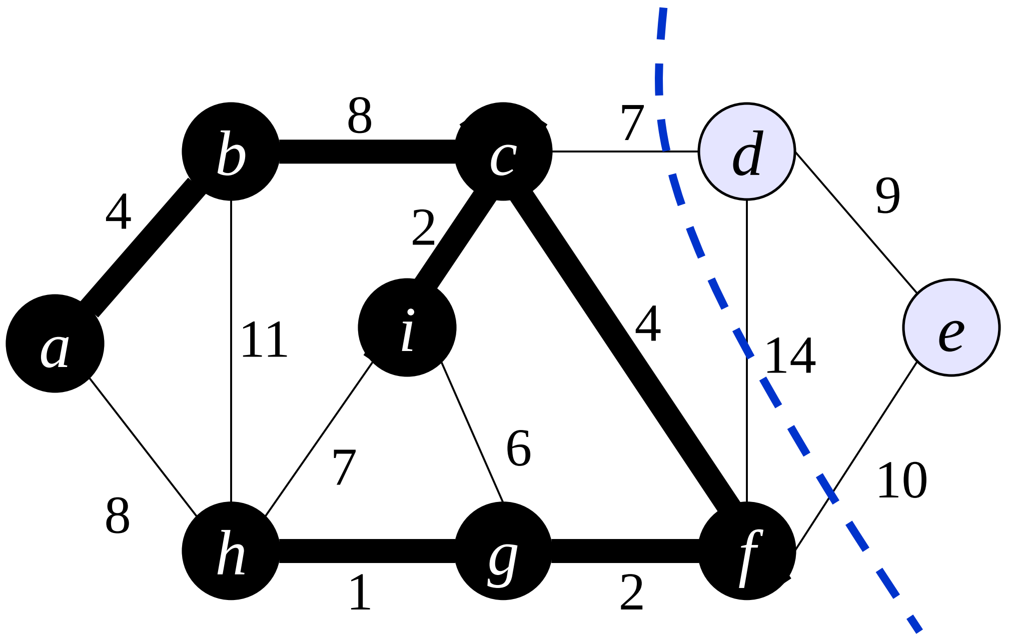
$f \notin Q$

$h \in Q$ and
$w(g, h) < key[h]$ (1 < 7)
    $\pi[h] = g$
    $key[h] = w(g, h) = 1$

$i \notin Q$

Q	d	e	h
	7	10	1

Example: Prim's Algorithm



h = EXTRACT-MIN(Q)

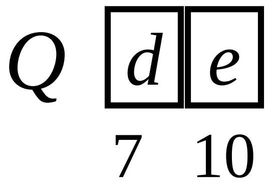
$Adj[h] = a, b, g, i$

$a \notin Q$

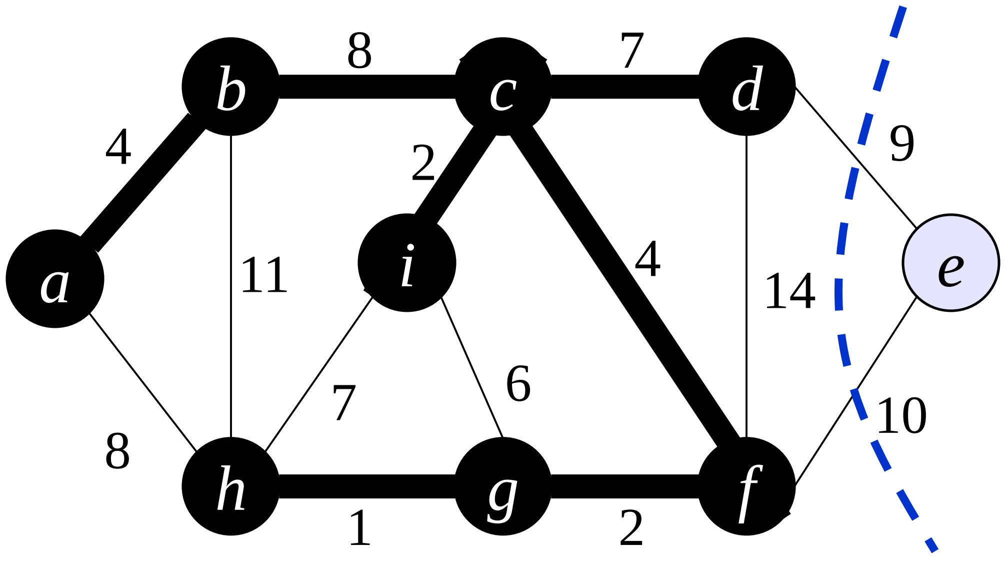
$b \notin Q$

$g \notin Q$

$i \notin Q$



Example: Prim's Algorithm



d = EXTRACT-MIN(Q)

$Adj[d] = c, e, f$

$c \notin Q$

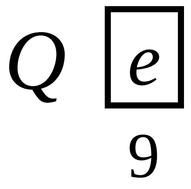
$e \in Q$ and

$w(d, e) < key[e]$ (9 < 10)

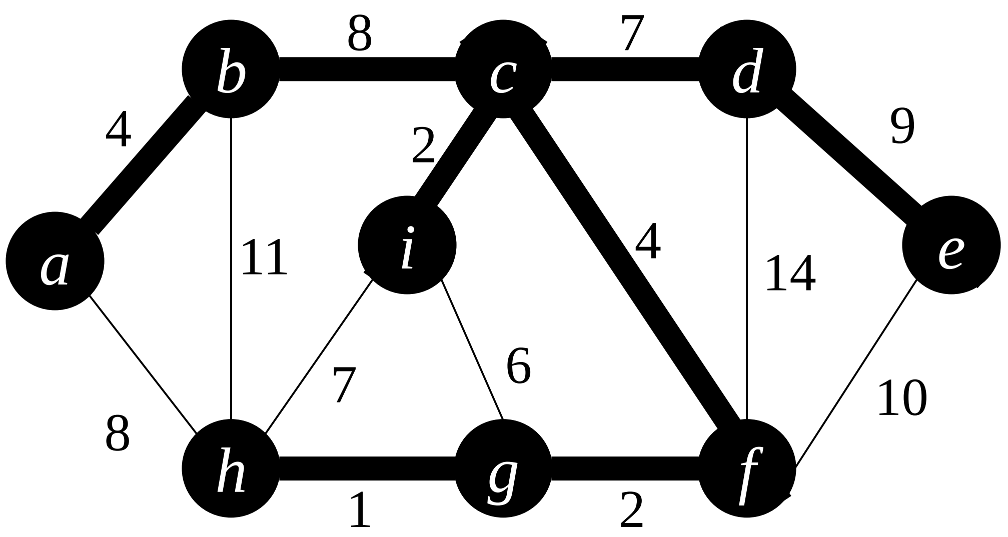
  $\pi[e] = d$

  $key[e] = w(d, e) = 9$

$f \notin Q$



Example: Prim's Algorithm



e = EXTRACT-MIN(*Q*)
Adj[e] = *d*, *f*

$d \notin Q$

$f \notin Q$

Q □

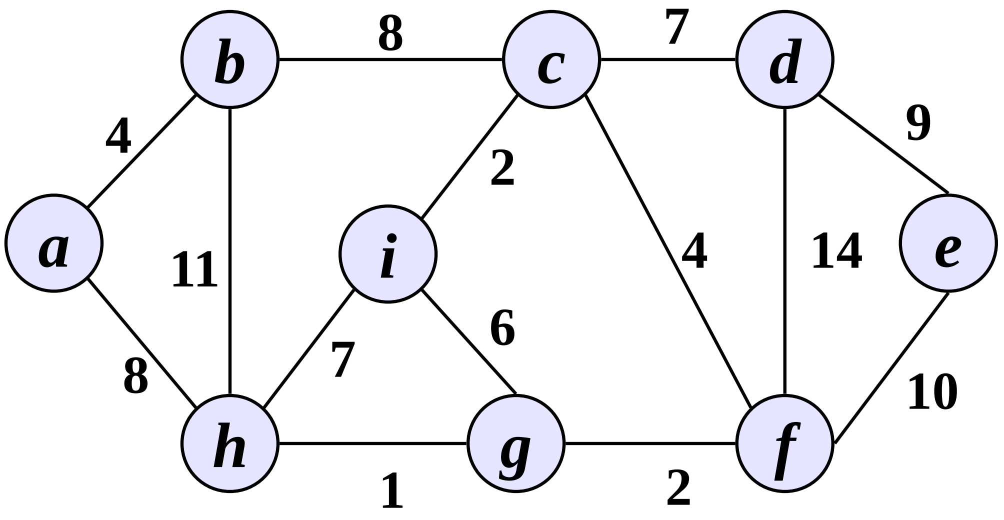
Kruskal’s Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex $v \in G.V$
    MAKE-SET($v$)
sort the edges of $G.E$ into nondecreasing order by weight $w$
for each edge $(u, v) \in G.E$, taken in nondecreasing order by weight
    if FIND-SET($u$) $\neq$ FIND-SET($v$)
        $A = A \cup \{(u, v)\}$
        UNION($u, v$)
return $A$
```

Total Running time

= O (m log n)

Example: Kruskal's Algorithm



Initial sets = *{a}*, *{b}*, *{c}*, *{d}*, *{e}*, *{f}*, *{g}*, *{h}*, *{i}*

Edges	**Weight**
(g, h)	1
(c, i)	2
(f, g)	2
(a, b)	4
(c, f)	4
(g, i)	6
(c, d)	7
(h, i)	7
(a, h)	8
(b, c)	8
(d, e)	9
(e, f)	10
(b h)	11
(d, f)	14

Example: Kruskal's Algorithm

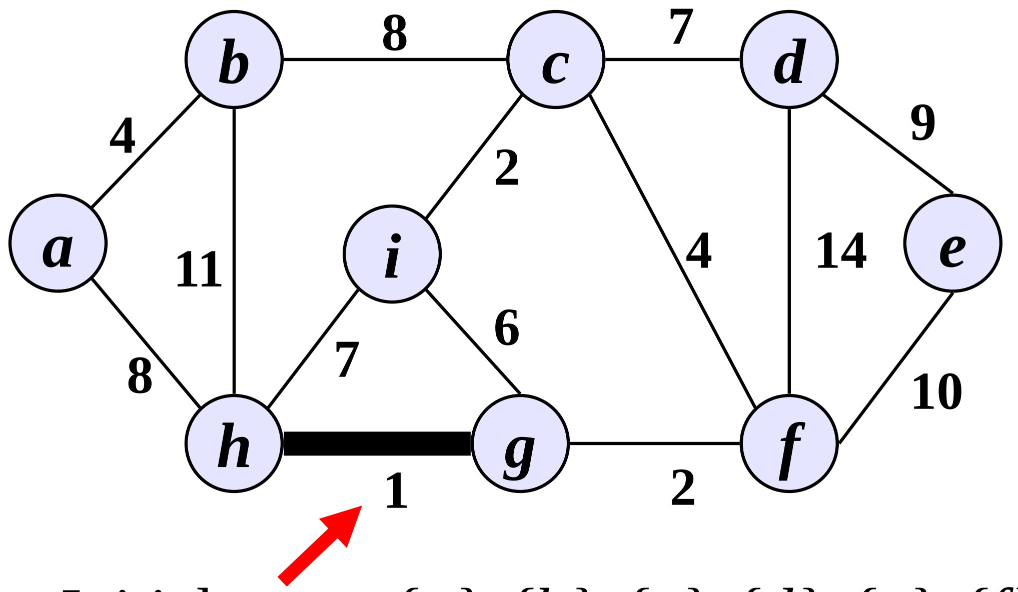
$A = \emptyset$

(g, h) is the least weight edge

FIND-SET *(g)* ≠ FIND-SET *(h)*

A = A U {(g, h)}

UNION *(g, h)*



Initial sets = *{a}*, *{b}*, *{c}*, *{d}*, *{e}*, *{f}*, *{**g**}*, *{**h**}*, *{i}*

Final sets = *{a}*, *{b}*, *{c}*, *{d}*, *{e}*, *{f}*, *{**g, h**}*, *{i}*

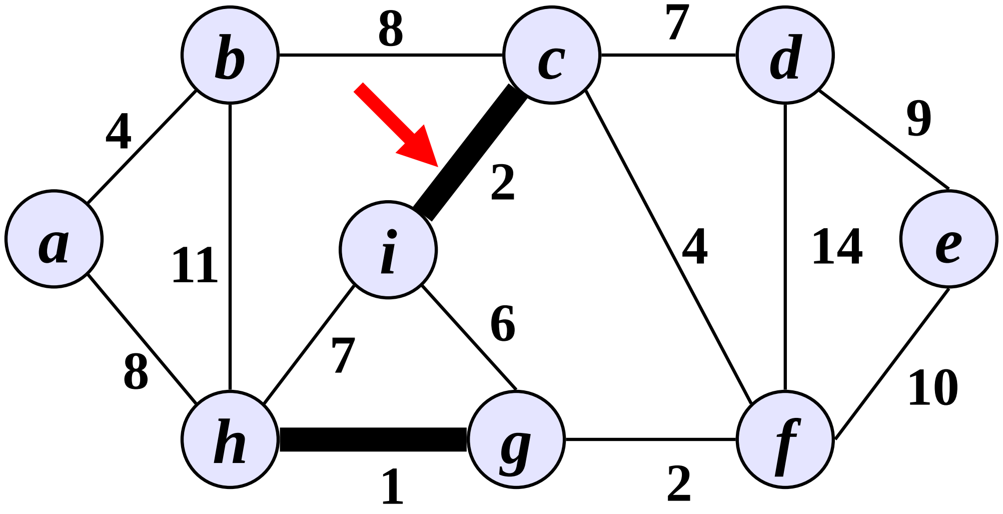
Example: Kruskal's Algorithm

(c, i) is the least weight edge

FIND-SET *(c)* ≠ FIND-SET *(i)*

A = A U {(c, i)}

UNION *(c, i)*



Initial sets = *{a}, {b}, {<u>**c**</u>}, {d}, {e}, {f}, {g, h}, {<u>**i**</u>}*

Final sets = *{a}, {b}, {**c, i**}, {d}, {e}, {f}, {g, h}*

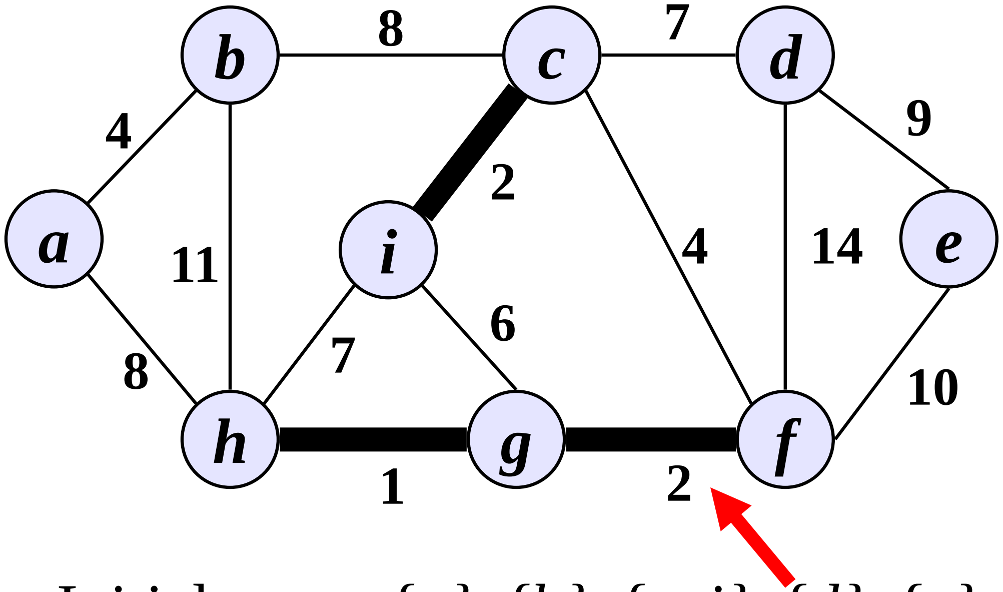
Example: Kruskal's Algorithm

(f, g) is the least weight edge

FIND-SET *(f)* ≠ FIND-SET *(g)*

A = A U {(f, g)}

UNION *(f, g)*



Initial sets = *{a}, {b}, {c, i}, {d}, {e},* {***f***}, {***g***, *h*}

Final sets = *{a}, {b}, {c, i}, {d}, {e},* {***f, g, h***}

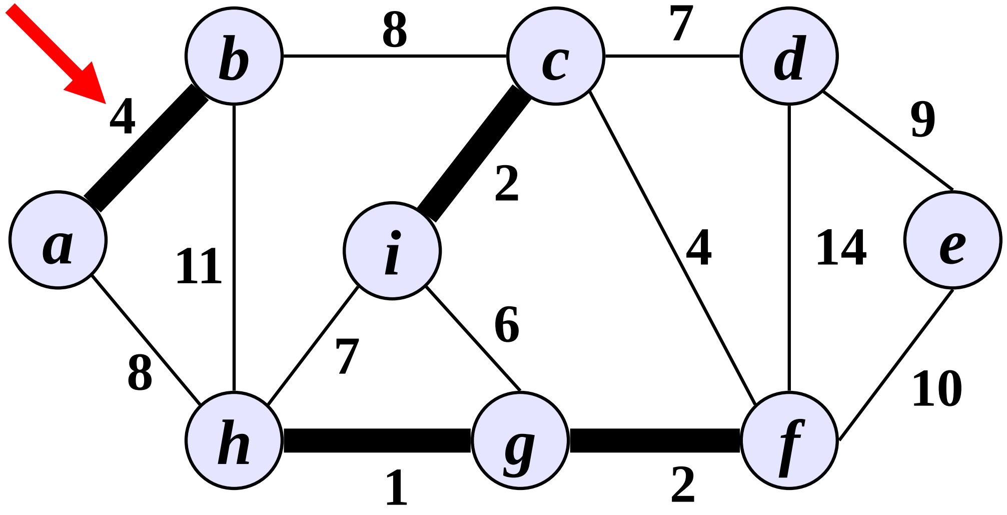
Example: Kruskal's Algorithm

(a, b) is the least weight edge

FIND-SET *(a)* ≠ FIND-SET *(b)*

A = A U {(a, b)}

UNION *(a, b)*



Initial sets = {***a***}, {***b***}, *{c, i}, {d}, {e}, {f, g, h}*

Final sets = {***a, b***}, *{c, i}, {d}, {e}, {f, g, h}*

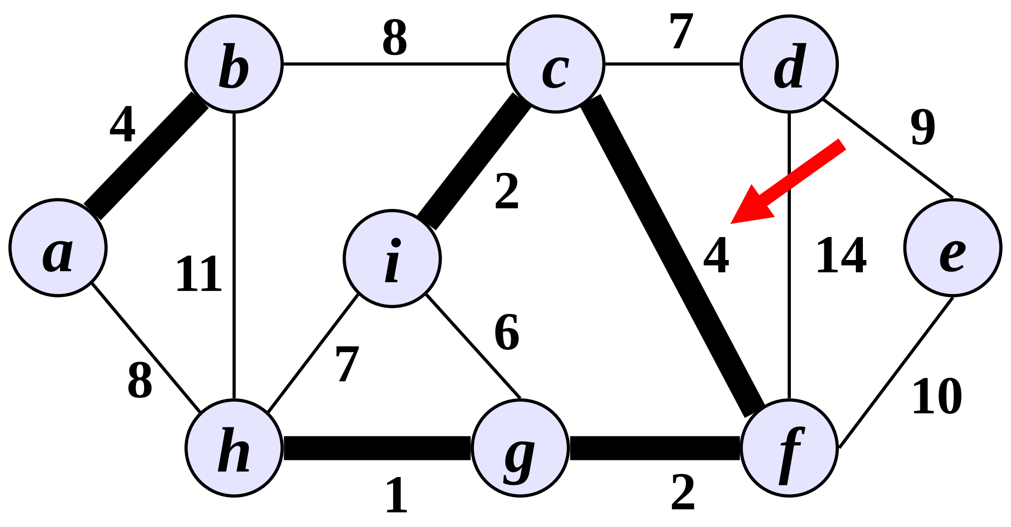
Example: Kruskal's Algorithm

(c, f) is the least weight edge

FIND-SET *(c)* ≠ FIND-SET *(f)*

A = *A* U {(c, *f*)}

UNION *(c, f)*



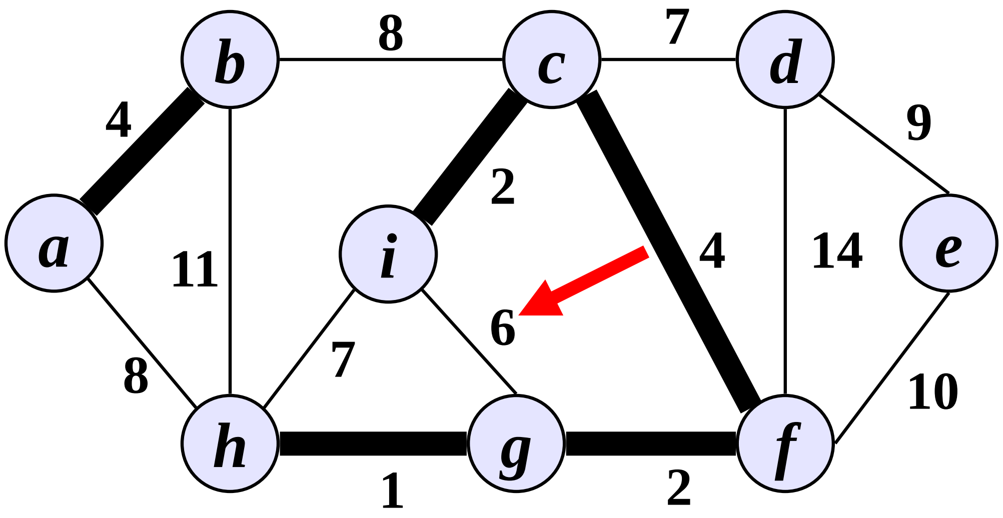
Initial sets = {*a, b*}, {*c, i*}, {*d*}, {*e*}, {***f***, *g, h*}

Final sets = {*a, b*}, {***c, f, g, h , i***}, {*d*}, {*e*}

Example: Kruskal's Algorithm

(g, i) is the least weight edge

FIND-SET *(g)* = FIND-SET *(i)*



Initial sets = {*a*, *b*}, {*c*, *f*, ***g,*** *h*, ***i***}, {*d*}, {*e*}

Final sets = {*a*, *b*}, {*c*, *f*, *g*, *h*, *i*}, {*d*}, {*e*}

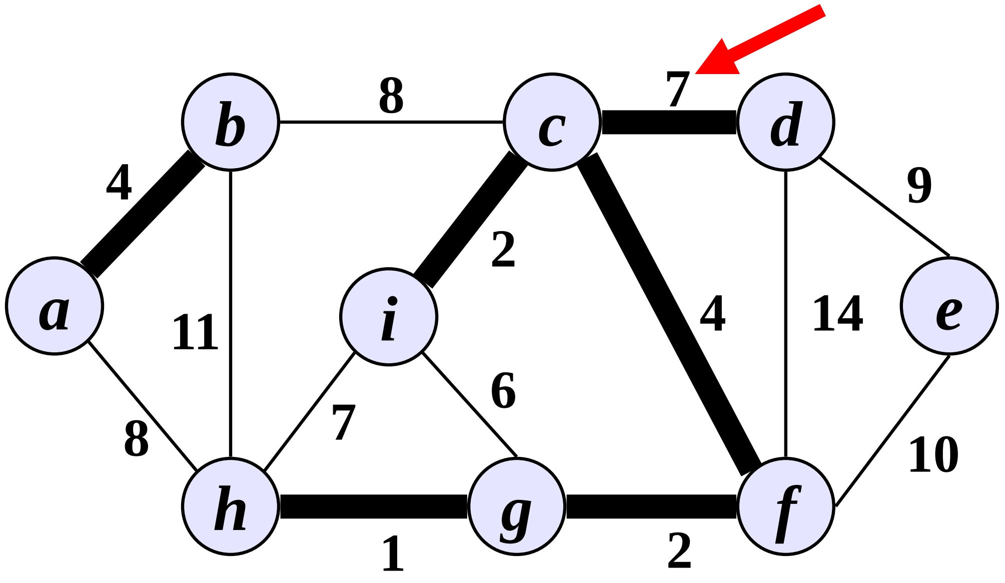
Example: Kruskal's Algorithm

(c, d) is the least weight edge

FIND-SET *(c)* ≠ FIND-SET *(d)*

A = *A* U *{(c, d)}*

UNION *(c, d)*



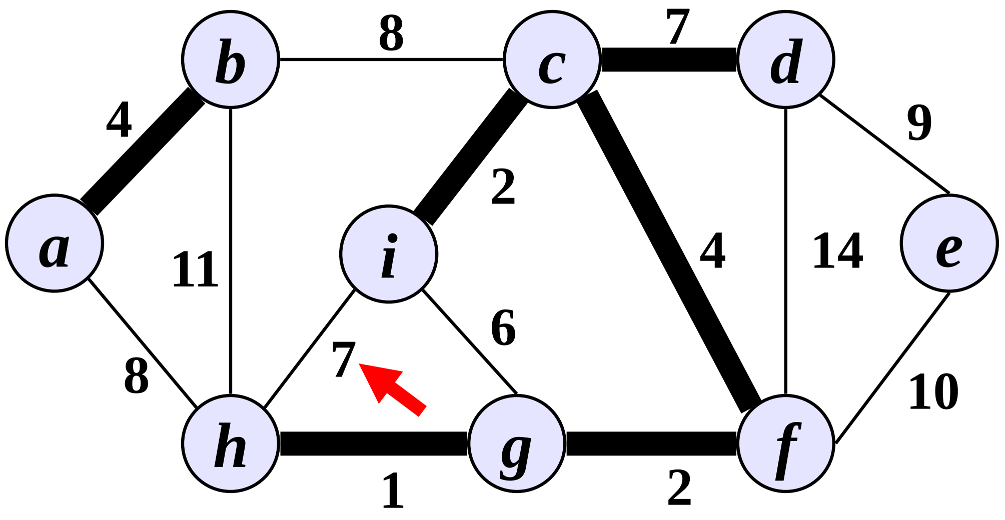
Initial sets = *{a, b}*, *{**c**, f, g, h, i}*, *{**d**}*, *{e}*

Final sets = *{a, b}*, *{**c, d, f, g, h, i**}*, *{e}*

Example: Kruskal's Algorithm

(h, i) is the least weight edge

FIND-SET *(h)* = FIND-SET *(i)*



Initial sets = *{a, b}, {c, d, f, g,* **<u>*h , i*</u>***}, {e}*

Final sets = *{a, b}, {c, f, d, g, h* **,** *i}, {e}*

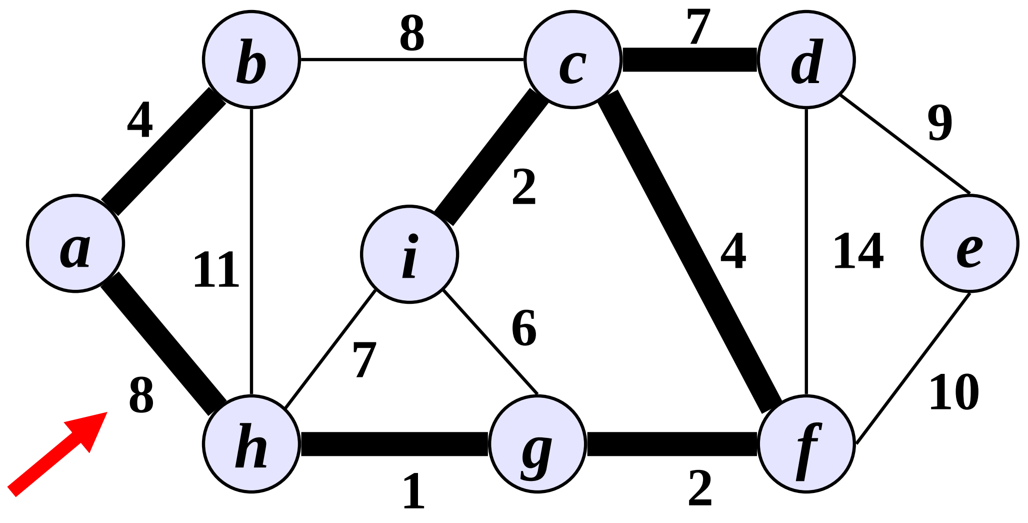
Example: Kruskal's Algorithm

(a, h) is the least weight edge

FIND-SET *(a)* ≠ FIND-SET *(h)*

A = *A* U {*(a, h)*}

UNION *(a, h)*



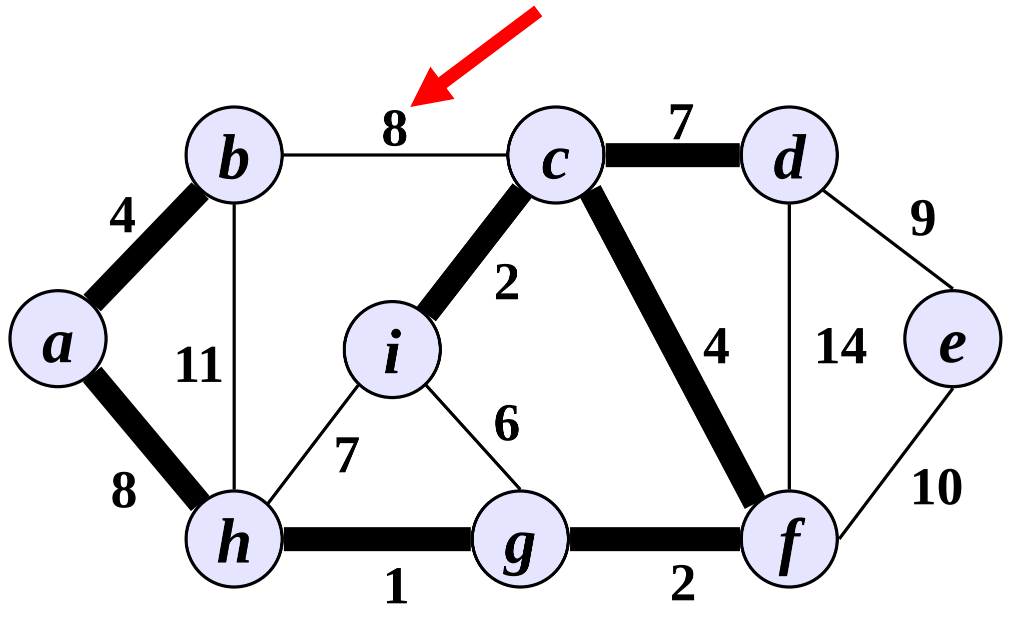
Initial sets = {***a***, *b*}, {*c*, *d*, *f*, *g*, ***h***, *i*}, {*e*}

Final sets = {***a*, *b*, *c*, *d*, *f*, *g*, *h* , *i***}, {*e*}

Example: Kruskal's Algorithm

(b, c) is the least weight edge

FIND-SET *(b)* = FIND-SET *(c*



Initial sets = {*a*, **b, c**, *d*, *f*, *g*, *h* , *i*}, {*e*}

Final sets = {*a*, *b*, *c*, *d*, *f*, *g*, *h* **,** *i*}, {*e*}

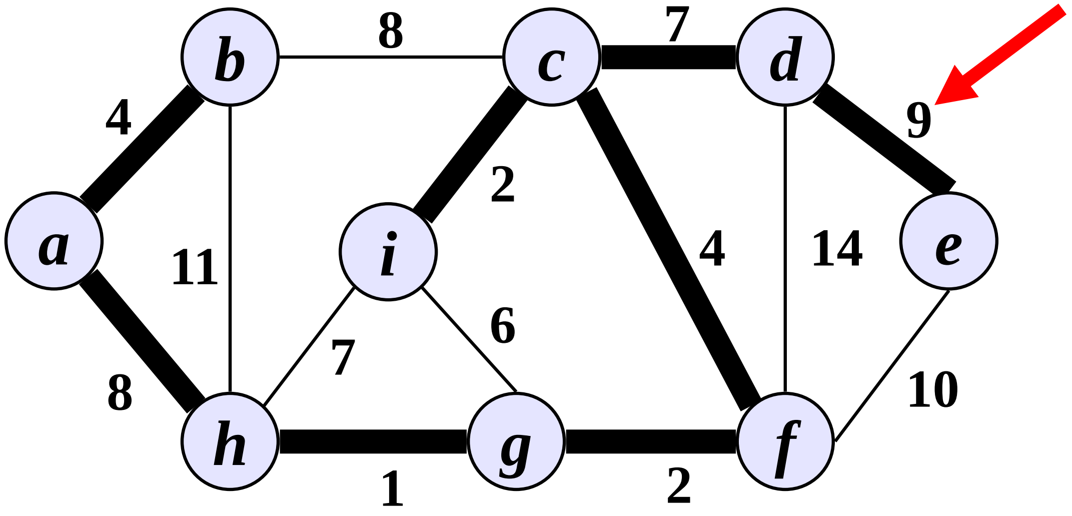
Example: Kruskal's Algorithm

(d, e) is the least weight edge

FIND-SET *(d)* ≠ FIND-SET *(e)*

A = *A* U {(*d*, *e*)}

UNION *(d, e)*



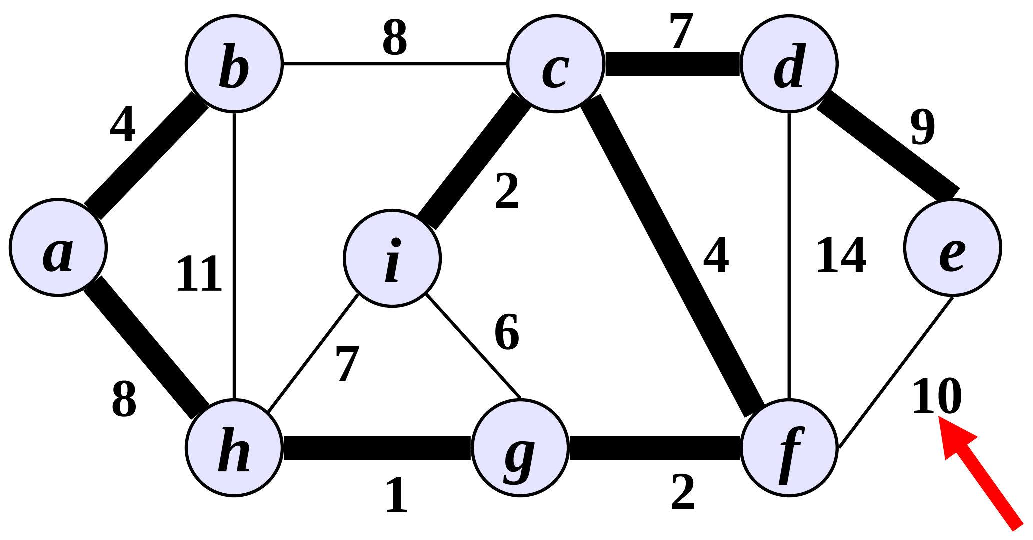
Initial sets = {*a*, *b*, *c*, ***<u>d</u>***, *f*, *g*, *h* , *i*}, {***<u>e</u>***}

Final sets = {***a*****,** ***b*****,** ***c*****,** ***d*****,** ***e*****,** ***f*****,** ***g*****,** ***h*** **,** ***i***}

Example: Kruskal's Algorithm

(e, f) is the least weight edge

FIND-SET *(e)* = FIND-SET *(f)*



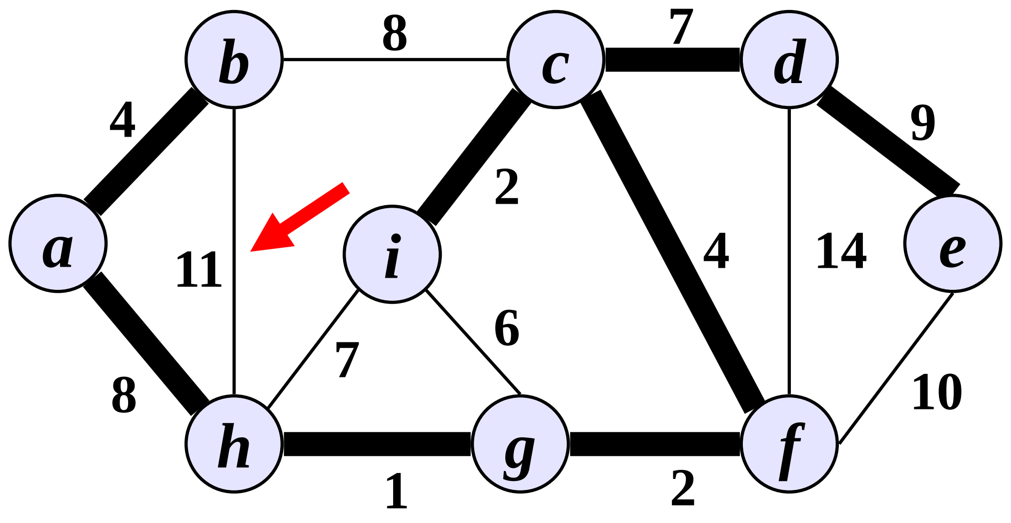
Initial sets = {*a*, *b*, *c*, *d*, ***e, f***, *g*, *h* , *i*}

Final sets = {*a*, *b*, *c*, *d*, *e*, *f*, *g*, *h* , *i*}

Example: Kruskal's Algorithm

(b, h) is the least weight edge

FIND-SET *(b)* = FIND-SET *(h)*



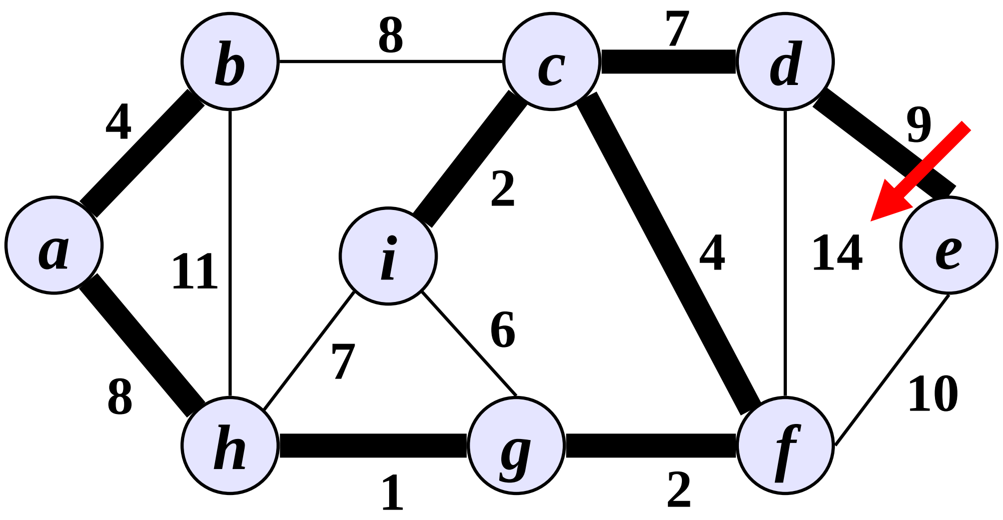
Initial sets = {*a*, ***b***, *c*, *d*, *e*, *f*, *g*, ***h*, *i***}

Final sets = {*a*, *b*, *c*, *d*, *e*, *f*, *g*, *h*, *i*}

Example: Kruskal’s Algorithm

(d, f) is the least weight edge

FIND-SET *(d)* = FIND-SET *(f)*



Initial sets = {*a*, *b*, *c*, ***d***, *e*, ***f***, *g*, *h*, *i*}

Final sets = {*a*, *b*, *c*, *d*, *e*, *f*, *g*, *h* , *i*}

Your Task

- Consider a long road with houses scattered very sparsely along it. (We can picture the road as a long horizontal line segment, with an eastern endpoint and a western endpoint). Further, let’s suppose that the residents of all these houses are avid cell phone users. You want to place cell phone base stations (towers) at certain points along the road, so that every house is within 4 miles of one of the base stations.

- Give an efficient algorithm that achieves this goal, using as few base stations as possible.