Advanced Algorithms Analysis and Design

By

Nazir Ahmad Zafar

Lecture No 32

Minimal Spanning Tree Problem

Today Lecture Covers

- Importance of Minimal Spanning Trees (MST)
- MST Problem
  - Definitions and analysis
  - Generic Solution
  - Proofs of correctness
- Kruskal's Algorithm
  - Algorithm
  - Analysis
- Prim's Algorithm
  - Algorithm
  - Analysis
- Conclusion

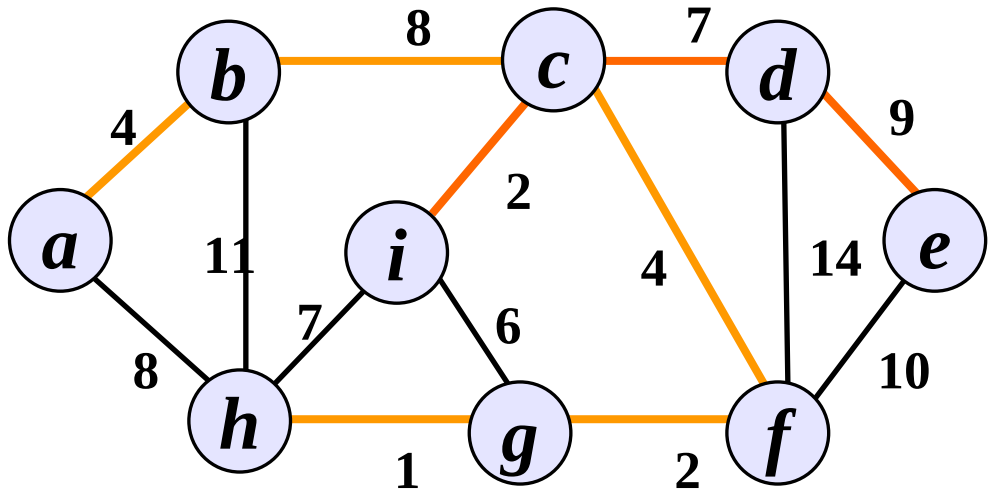
Minimum Spanning Tree

- Given a graph *G* = (V, E) such that
  - *G* is connected and undirected
  - *w(u, v)* weight of edge (u, v)

- T is a **Minimum Spanning Tree (MST)** of *G* if
  - T is acyclic subset of E ($T \subseteq E$)
  - It connects all the vertices of G and
  - Total weight, *w*(T) = $\sum_{(u,v) \in T} w(u,v)$ is minimized.
  - MST not unique and no Cycle

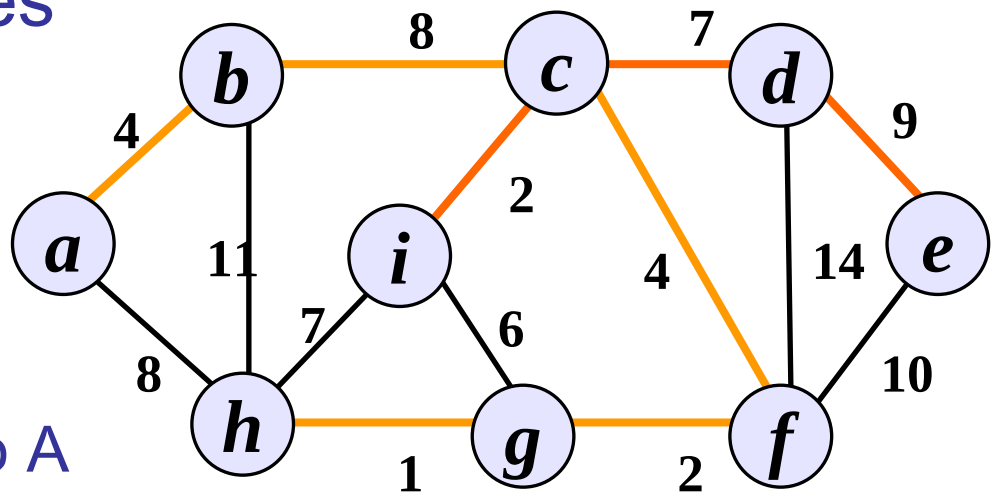
Example of MST

- 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
  - Generic solution



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

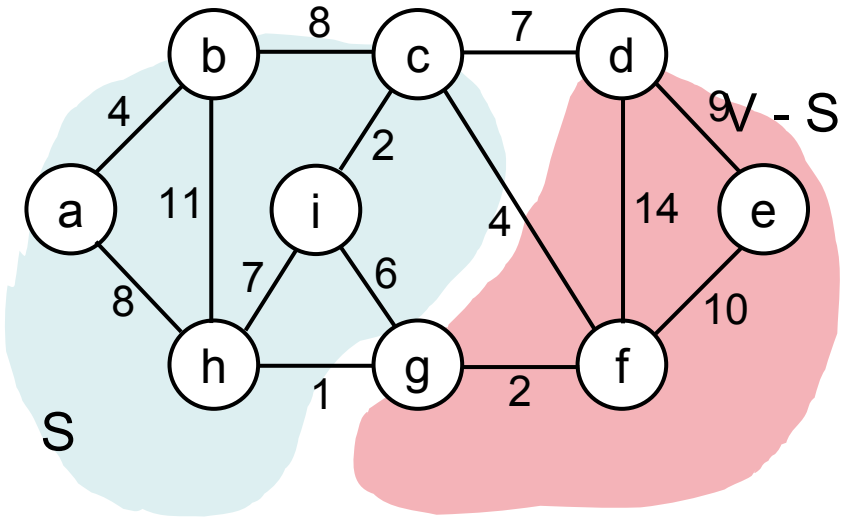


A generic solution:

- Build a set A of edges (initially empty)
- Incrementally add edges to A such that they would belong to a MST
- An edge (u, v) is **safe** for A $\Leftrightarrow$ A $\cup$ {(u, v)} is also a subset of some MST
- How to find safe edge?

How to Find Safe Edge?

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



  - Let $S \subset V$ 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)
- Algorithm

Generic Algorithm: Minimum Spanning Tree

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

Initialization

Maintenance

Termination

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 $w : E \rightarrow R$
- 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.

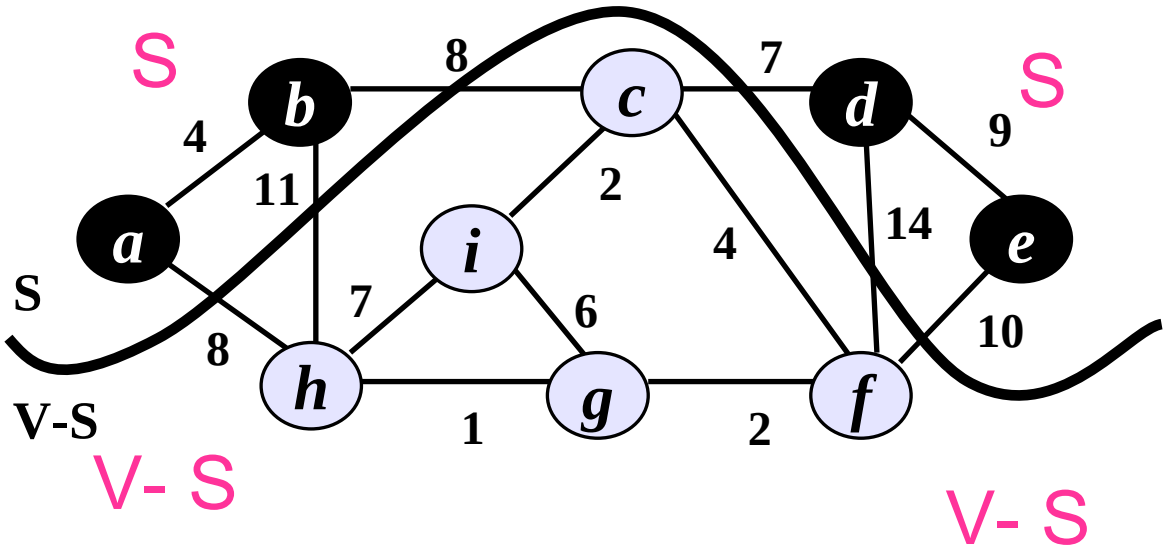
Algorithms, discussed here, to find safe edge are

1. Kruskal's Algorithm
2. Prim's Algorithm

Definitions

Definitions (Kruskal's Algorithm)

- A **cut** (S, V-S) of an undirected graph is a partition of V
- An edge **crosses** the cut (S, V-S) if one of its endpoints is in S and the other is in V-S.
- A cut **respects** set A of edges if no edge in A crosses cut.
- An edge is a **light edge** crossing a cut if its weight is the minimum of any edge crossing the cut.

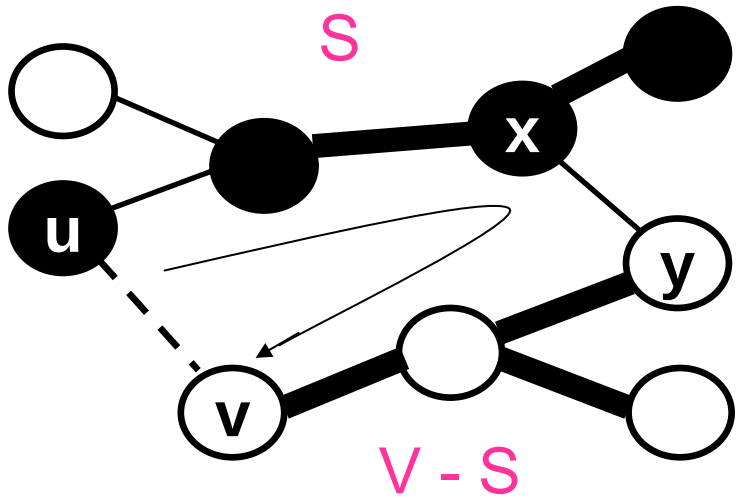


Theorem (Kruskal's Algorithm)

Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function w on E. Let A be a subset of E that is included in some minimum spanning tree for G, let $(S, V - S)$ be any cut of G that respects A, and let (u, v) be a light edge crossing $(S, V - S)$. Then, edge (u, v) is safe for A.

Proof

- Let T be a minimum spanning tree that includes A *(edges of A are shaded)*,
- Assume that T does not contain the light edge (u, v), since if it does, we are done.



Theorem (Contd..)

Construction of another MST,

- For $(u, v) \notin T$
- We construct another MST T' that includes $A \cup \{(u, v)\}$ by cut-and-paste, and showing that (u, v) is a safe A.
- Since (u, v) crosses cut set (S, V-S) and
- $(u, v) \notin T$,
- Hence there must be an edge $(x, y) \in T$ which crosses the cut set
- By removing (x, y) breaks T into two components.
- Adding (u, v) reconnects them to form a new spanning tree $T' = T - \{(x, y)\} \cup \{(u, v)\}$.

Theorem (Contd..)

Show that *T′* is a minimum spanning tree.

- Since (u, v) is a light edge crossing (S, V - S) and (x, y) also crosses this cut, $w(u, v) \leq w(x, y)$.
- Hence, $w(T') = w(T) - w(x, y) + w(u, v) \leq w(T)$.
- But T is a MST, so that $w(T) \leq w(T')$; thus, T′ must be a minimum spanning tree also.

Show that (u, v) is safe for A: (u, v) can be part of MST

- Now (x, y) is not in A, because the cut respects A.
- Since $A \subseteq T$ and $(x, y) \notin A \Rightarrow A \subseteq T - \{(x, y)\}$
- $A \cup \{(u, v)\} \subseteq T - \{(x, y)\} \cup \{(u, v)\} = T'$
- Since T' is an MST $\Rightarrow$ (u, v) is safe for A

Kruskal's Algorithm

MST-KRUSKAL (*G*, *w*)

1 $A \leftarrow \emptyset$

2 **for** each vertex $v \in V[G]$

3 **do** MAKE-SET(*v*)

4 sort edges in non-decreasing order by weight *w*

5 for each (*u*, *v*) in non-decreasing order by weight

6 **do if** FIND-SET(*u*) ≠ FIND-SET(*v*)

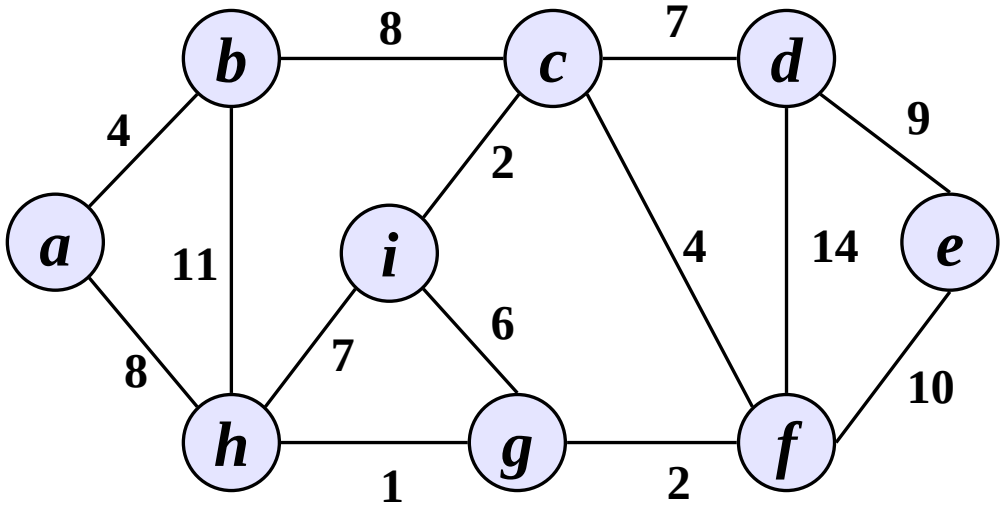
7 **then** $A \leftarrow A \cup \{(u, v)\}$

8 UNION (*u*, *v*)

9 return *A*

Total Running time = O (E lg V),

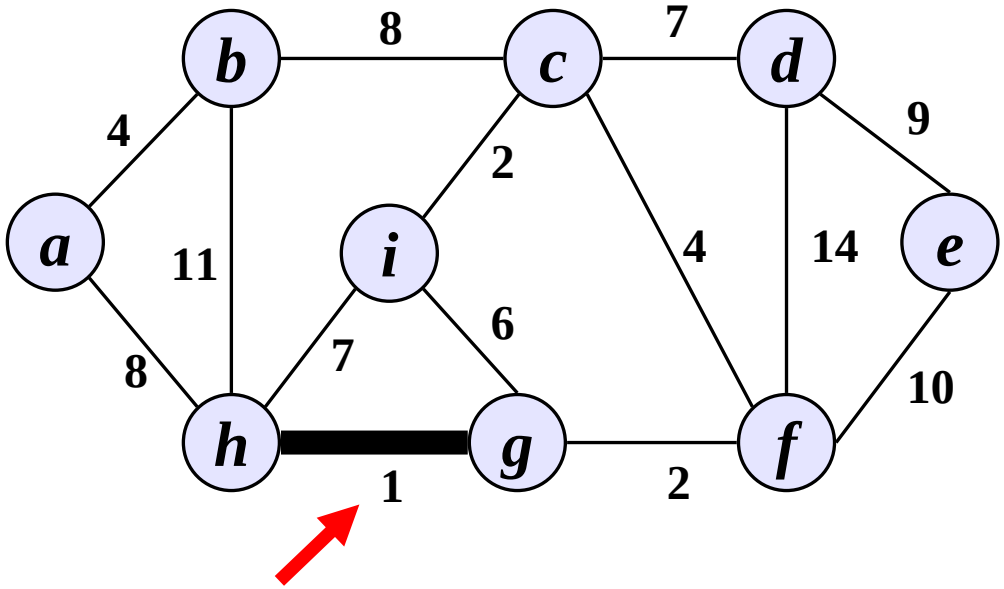
Kruskal's Algorithm



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

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

Kruskal's Algorithm



$A = \varnothing$

(g, h) is the least weight edge

FIND-SET *(g)* ≠ FIND-SET *(h)*

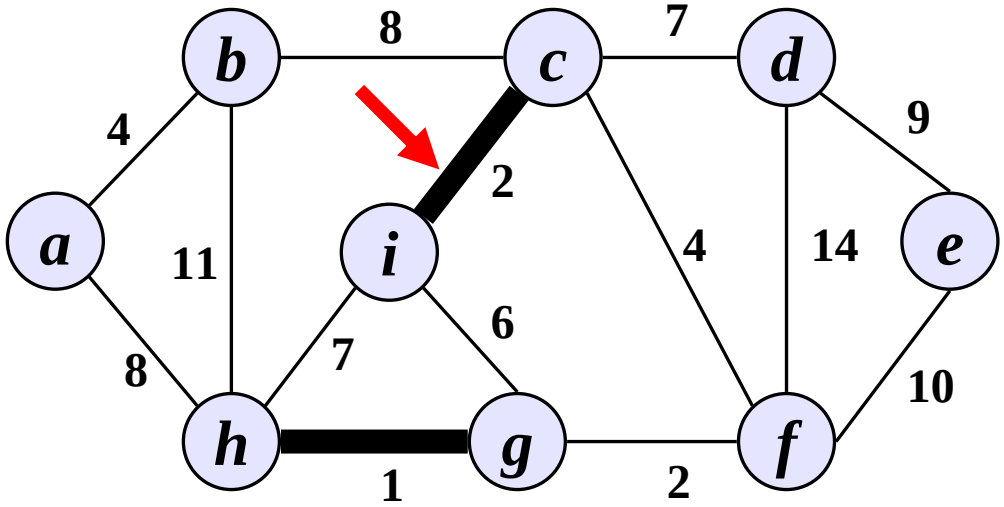
$A \leftarrow A \cup \{(g, h)\}$

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}*

Kruskal's Algorithm



(c, i) is the least weight edge

FIND-SET *(c)* ≠ FIND-SET *(i)*

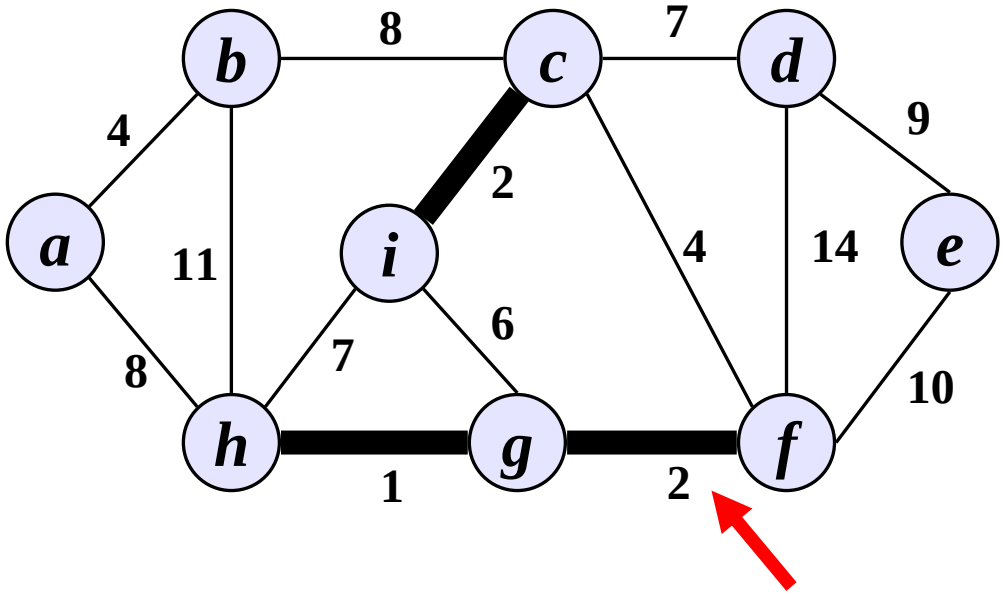
$A \leftarrow A \cup \{(c, i)\}$

UNION *(c, i)*

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

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

Kruskal's Algorithm



(f, g) is the least weight edge

FIND-SET *(f)* ≠ FIND-SET *(g)*

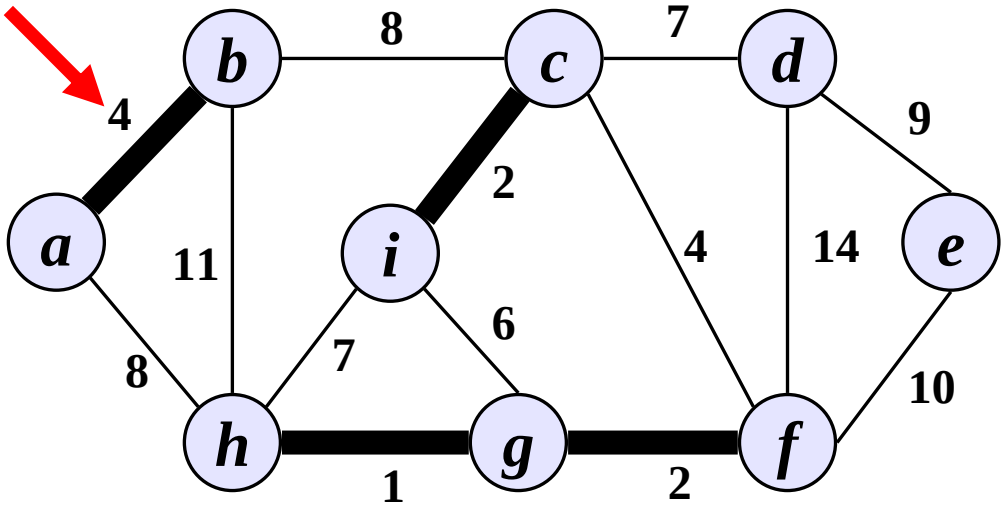
$A \leftarrow A \cup \{(f, g)\}$

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**}*

Kruskal's Algorithm



(a, b) is the least weight edge

FIND-SET *(a)* ≠ FIND-SET *(b)*

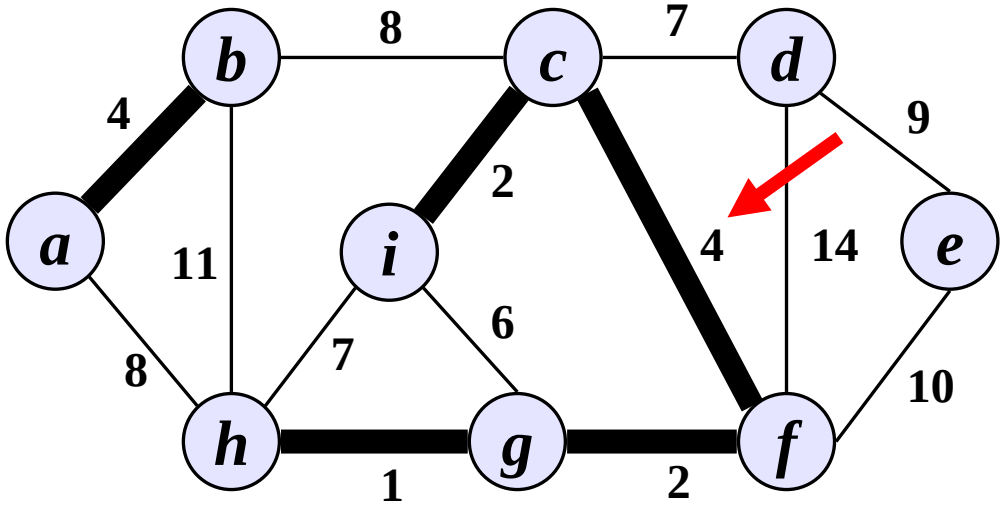
$A \leftarrow A \cup \{(a, b)\}$

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}*

Kruskal's Algorithm



(c, f) is the least weight edge

FIND-SET *(c)* ≠ FIND-SET *(f)*

$A \leftarrow A \cup \{(c, f)\}$

UNION *(c, f)*

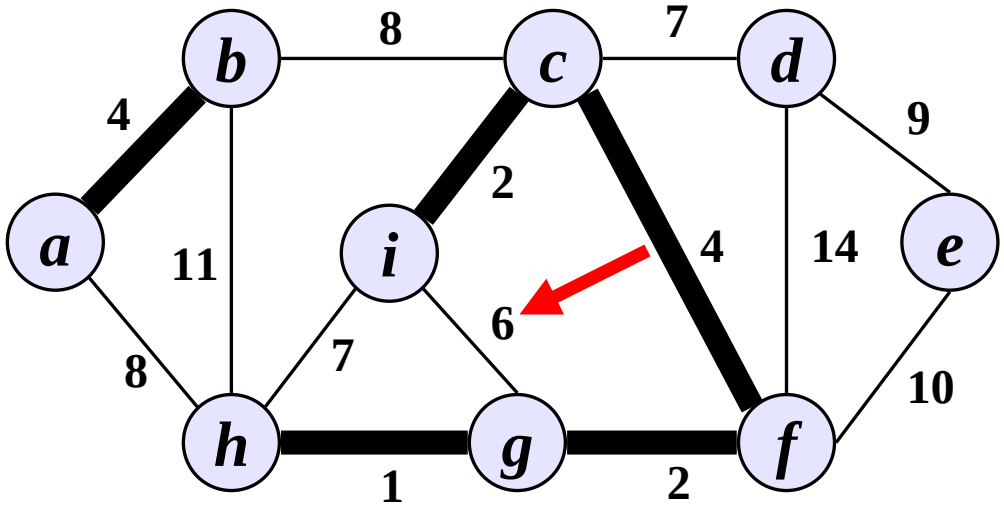
Initial sets = *{a, b}, {c, i}, {d}, {e}, {f, g, h}*

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

Kruskal's Algorithm

(g, i) is the least weight edge

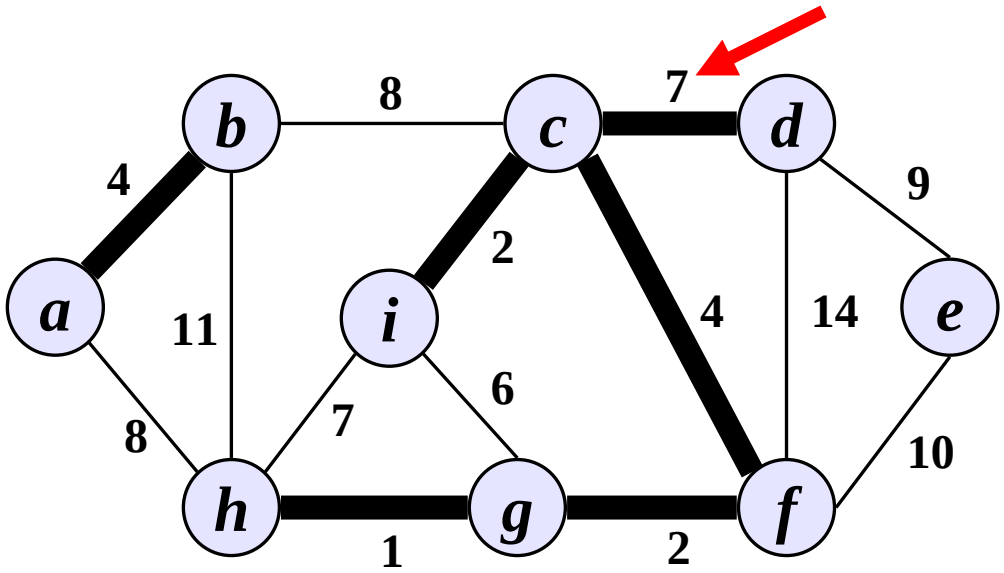
FIND-SET *(g)* = FIND-SET *(i)*



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

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

Kruskal's Algorithm



(c, d) is the least weight edge

FIND-SET *(c)* ≠ FIND-SET *(d)*

$A \leftarrow A \cup \{(c, d)\}$

UNION *(c, d)*

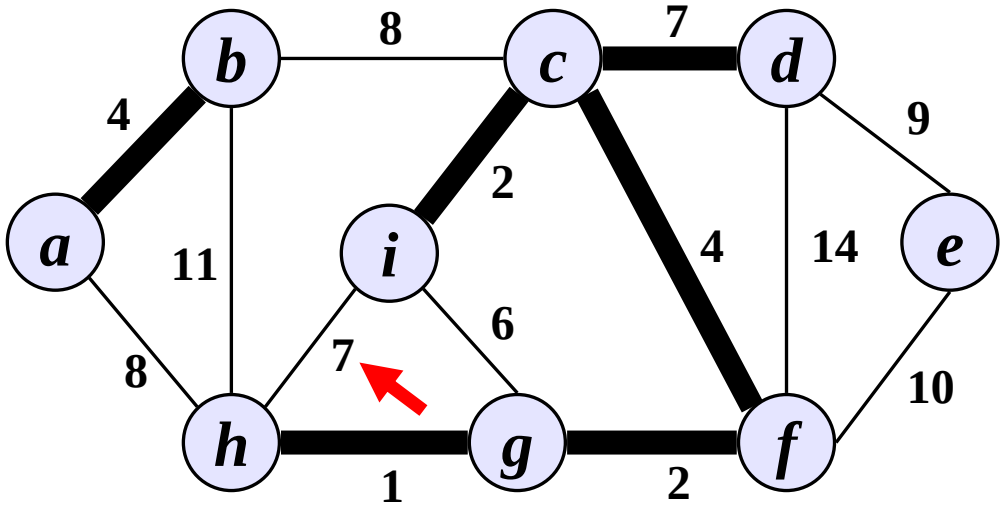
Initial sets = *{a, b}, {**c**, f, g, h, i}, {**d**}, {e}*

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

Kruskal's Algorithm

(h, i) is the least weight edge

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



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

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

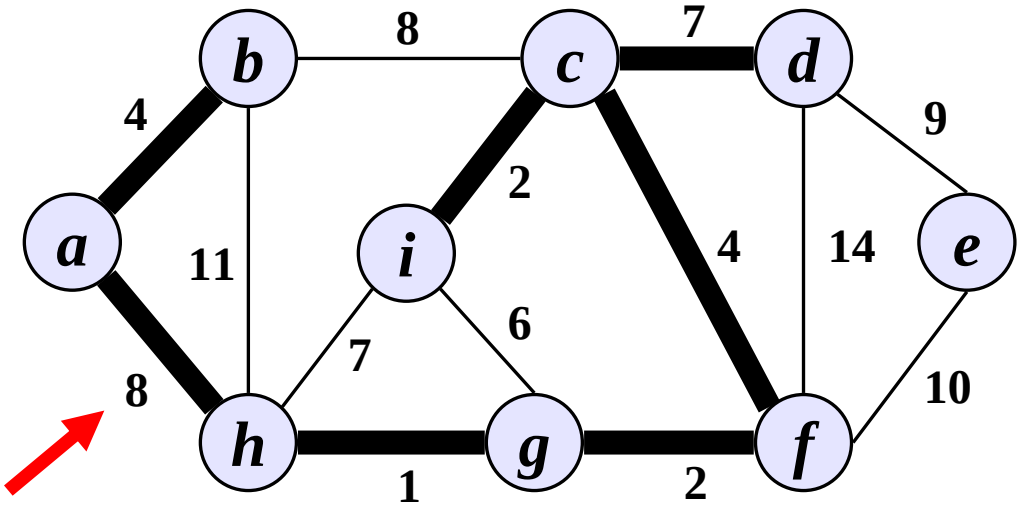
Kruskal's Algorithm

(a, h) is the least weight edge

FIND-SET *(a)* ≠ FIND-SET *(h)*

$A \leftarrow A \cup \{(a, h)\}$

UNION *(a, h)*



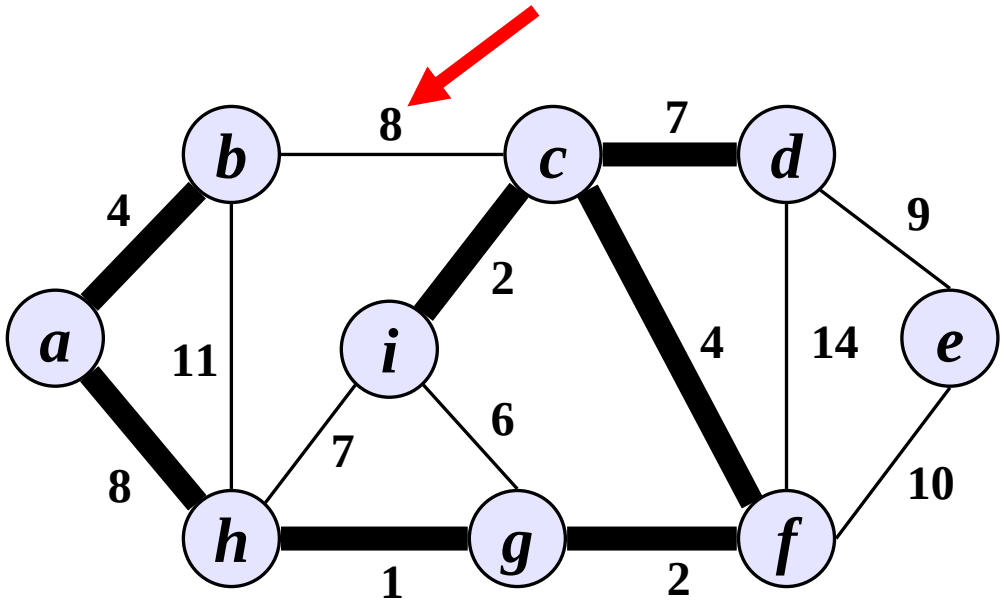
Initial sets = *{**a**, b}, {c, d, f, g, **h**, i}, {e}*

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

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}*

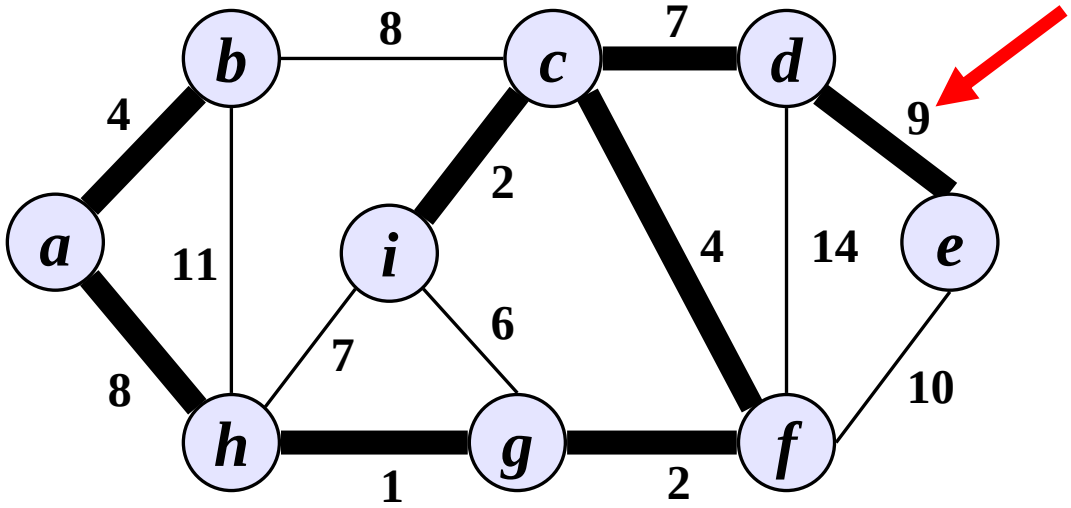
Kruskal's Algorithm

(d, e) is the least weight edge

FIND-SET *(d)* ≠ FIND-SET *(e)*

$A \leftarrow A \cup \{(d, e)\}$

UNION *(d, e)*



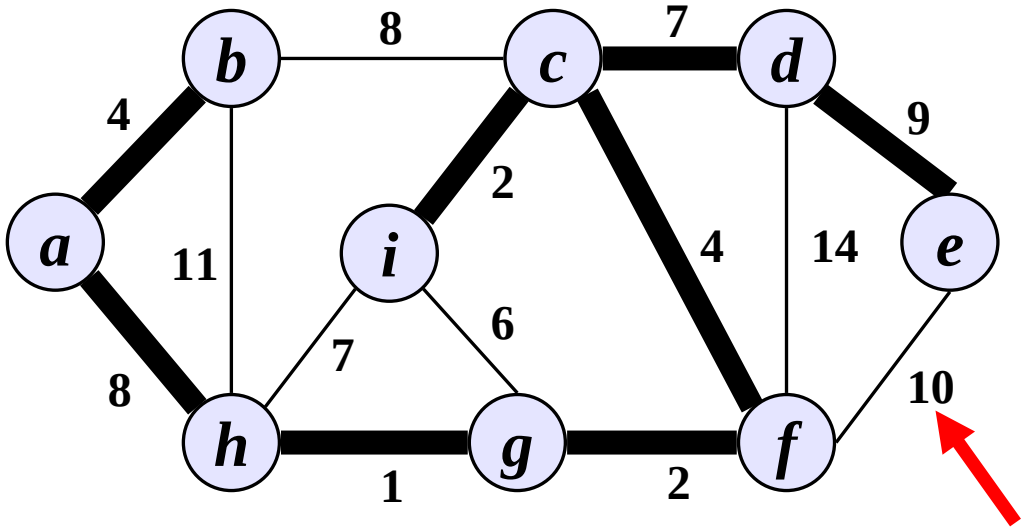
Initial sets = {*a, b, c,* ***d****, f, g, h , i*}, {***e***}

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

Kruskal's Algorithm

(e, f) is the least weight edge

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



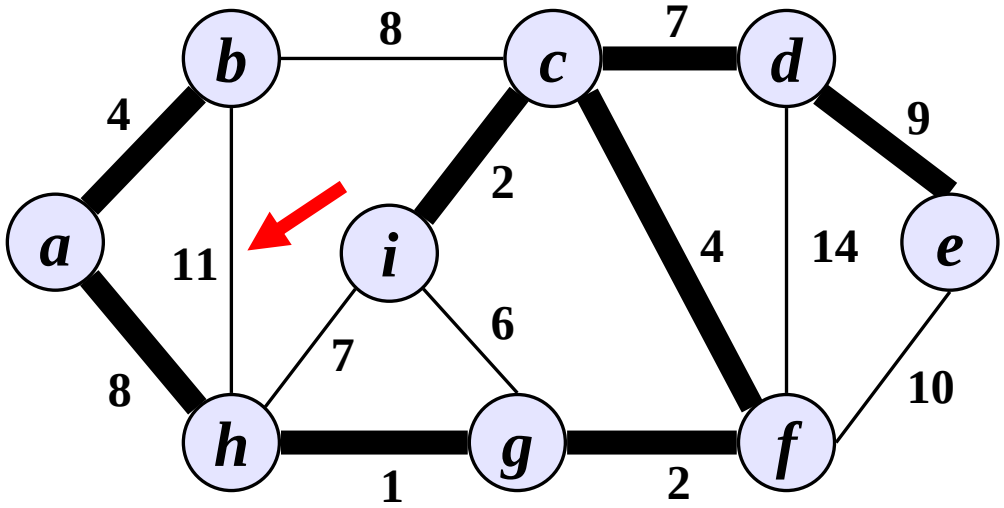
Initial sets = *{a, b, c, d,* ***e, f****, g, h , i}*

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

Kruskal's Algorithm

(b, h) is the least weight edge

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

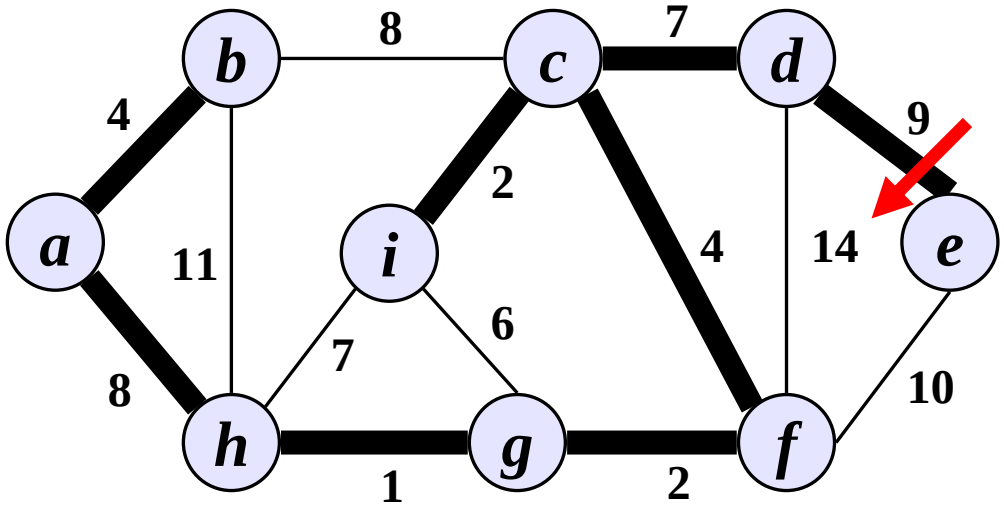


Initial sets = *{a,* ***b****, c, d, e, f, g,* ***h****, i}*
Final sets = *{a, b, c, d, e, f, g, h, i}*

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*}

Correctness of Kruskal's Algorithm

- Used to determine the safe edge of GENERIC-MST
- The algorithm manages set of edges *A* which always form a single tree.
- The tree starts from an arbitrary vertex *r* and grows until tree spans all the vertices in *V*.
- At each step, a light edge is added to the tree *A* that connects *A* to an isolated vertex of $G_A = (V, A)$
- It is a greedy algorithm
  - At each step tree is augmented with an edge that contributes least possible amount to tree's weight
- Since vertices, of each edge considered, are in different sets hence no cycle is created.

Prim's Algorithm

```
MST-PRIM (G, w, r)
 1   for each u ∈ V [G]
 2         do key[u] ← ∞
 3            π[u] ← NIL
 4   key[r] ← 0
 5   Q ← V [G]
 6   while Q ≠ Ø
 7         do u ← EXTRACT-MIN(Q)
 8            for each v ∈ Adj[u]
 9                do if v ∈ Q and w(u, v) < key[v]
10                      then π[v] ← u
11                   key[v] ← w(u, v)
```

Running time

Prim's Algorithm

```
Q ← ∅
for each u ∈ V
    do key[u] ← ∞
       π[u] ← NIL
       INSERT(Q, u)
DECREASE-KEY(Q, r, 0)        ▶ key[r] ← 0
while Q ≠ ∅
    do u ← EXTRACT-MIN(Q)
       for each v ∈ Adj[u]
           do if v ∈ Q and w(u, v) < key[v]
                 then π[v] ← u
                      DECREASE-KEY(Q, v, w(u, v))
```

Total time: O(VlgV + ElgV) = O(ElgV)

O(V) if Q is implemented as a min-heap

Executed |V| times

Takes O(lgV)

Min-heap operations: O(VlgV)

Executed O(E) times

Constant

O(ElgV)

Takes O(lgV)

Prim's Algorithm

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

- Using binary min-heap

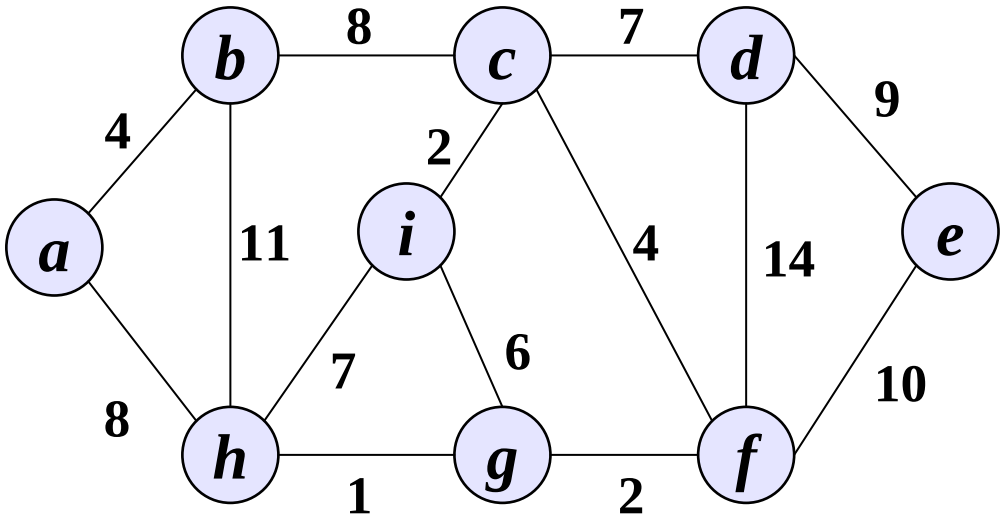
Total Running time = O (E lg V)

- Using fibonacci heaps

Total Running time = O (E + V lg V)

Example

Prim's Algorithm



For each vertex u ∈ $V(G)$

$key[u] \leftarrow \infty$

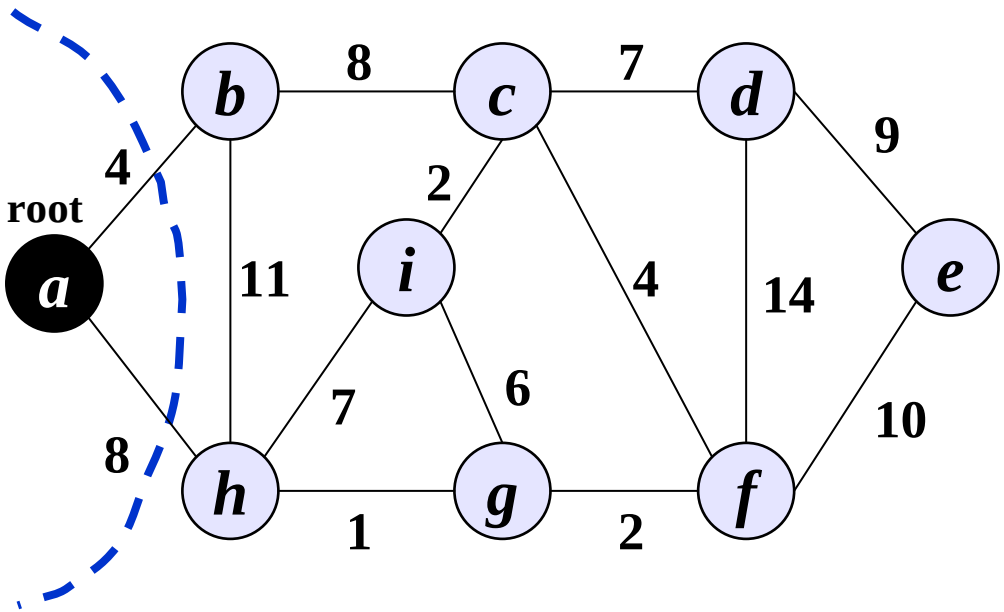
$\pi[u] \leftarrow NIL$

Considering a as root node

$key[a] \leftarrow 0$

Q	a	b	c	d	e	f	g	h	i
	0								

Prim's Algorithm



$a \leftarrow$ EXTRACT-MIN(Q)

$Adj[a] = b, h$

$b \in Q$ and

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

$\pi[b] \leftarrow a$

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

$h \in Q$ and

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

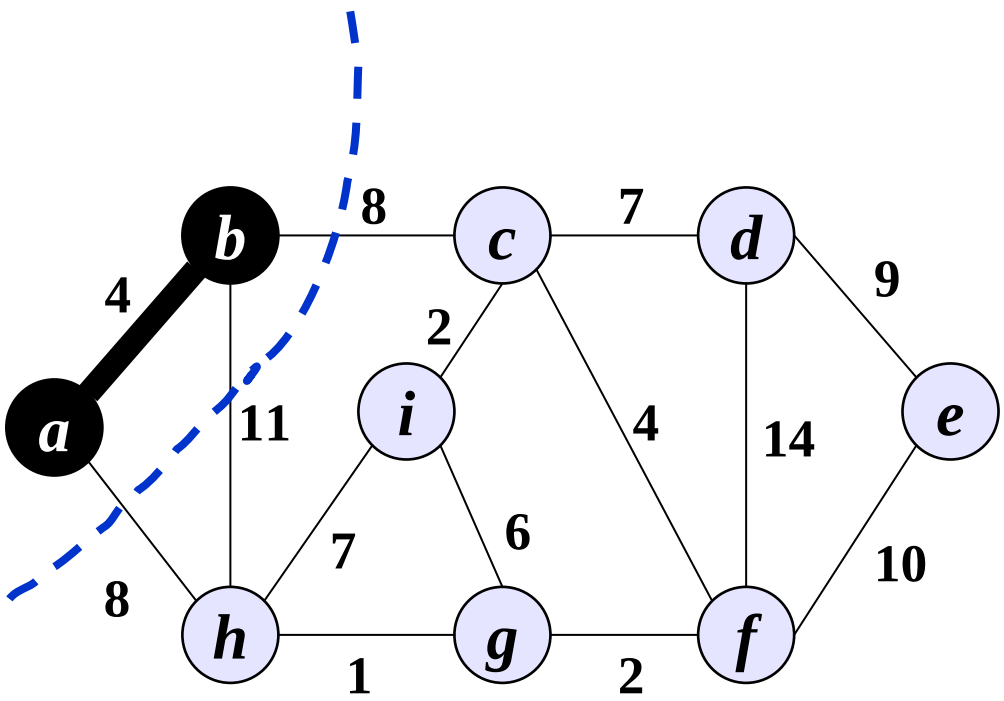
$\pi[h] \leftarrow a$

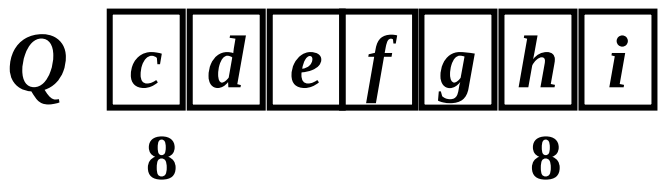
$key[h] \leftarrow w(a, h) = 8$

Q | b | c | d | e | f | g | h | i |

4 8

Prim's Algorithm





$b \leftarrow$ EXTRACT-MIN(Q)

$Adj[b] = a, c, h$

$a \notin Q$

$c \in Q$ and

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

$\pi[c] \leftarrow b$

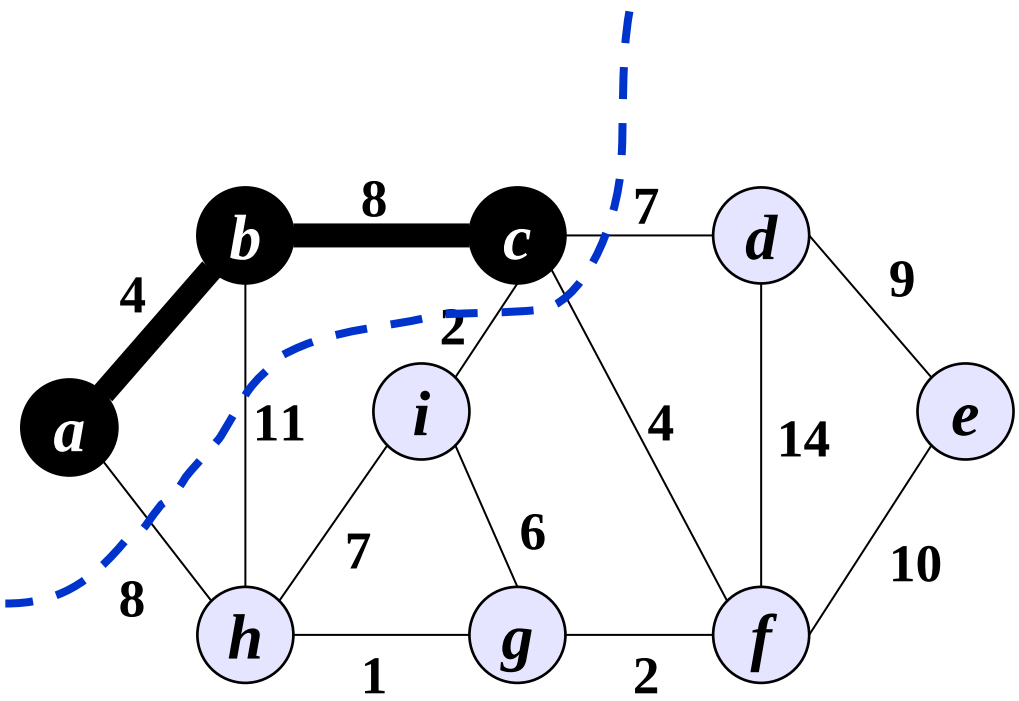
$key[c] \leftarrow w(b, c) = 8$

$h \in Q$ and

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

(but $11 > 8$)

Prim's Algorithm



$c \leftarrow$ EXTRACT-MIN(Q)

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

$b \notin Q$

$d \in Q$ and

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

$\pi[d] \leftarrow c$

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

$f \in Q$ and

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

$\pi[f] \leftarrow c$

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

$i \in Q$ and

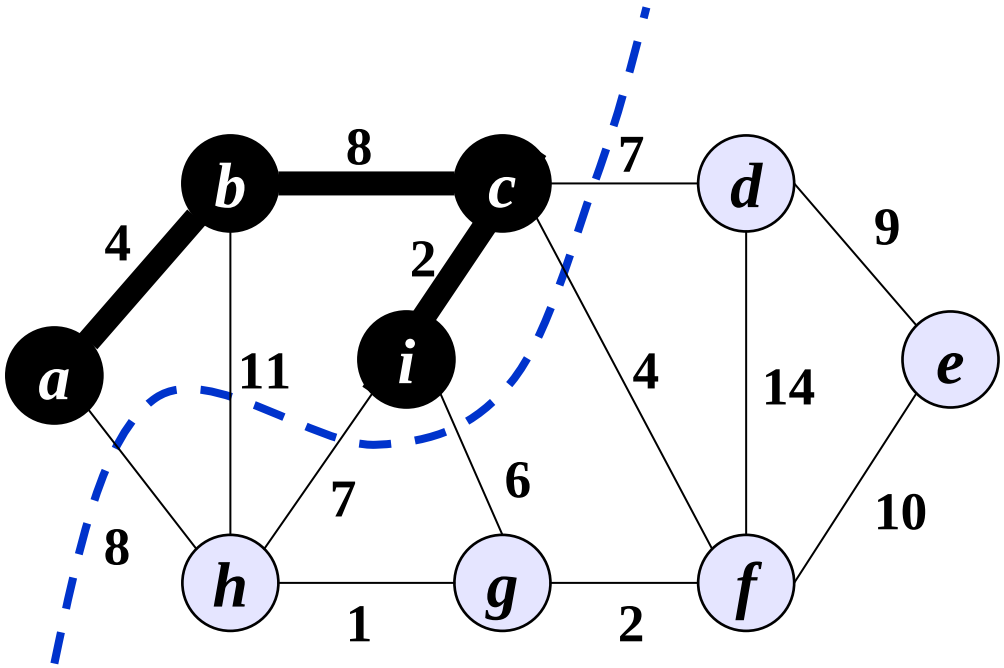
$w(c, i) < key[i]$ $(2 < \infty)$

$\pi[i] \leftarrow c$

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

Q	d	e	f	g	h	i
	7		4		8	2

Prim's Algorithm



Q	d	e	f	g	h
7		4	6	7

$i \leftarrow$ EXTRACT-MIN(Q)

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

$c \notin Q$

$g \in Q$ and

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

$\pi[g] \leftarrow i$

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

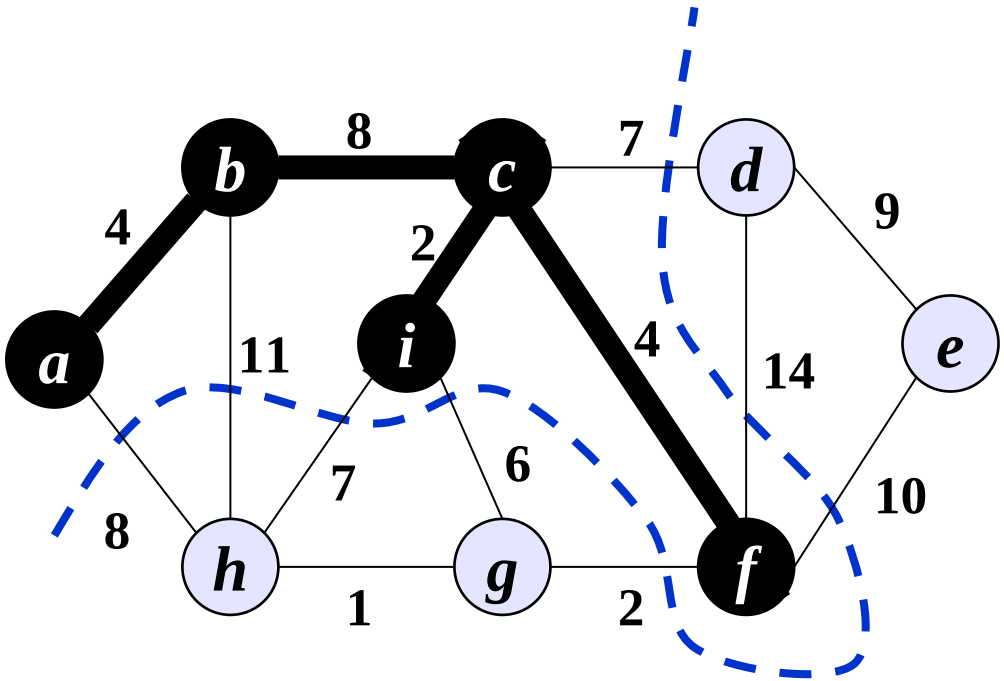
$h \in Q$ and

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

$\pi[h] \leftarrow i$

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

Prim's Algorithm



Q | d | e | g | h

7 10 2 7

$f \leftarrow$ 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

$w(f, e) < key[e]$ $(10 < \infty)$

$\pi[e] \leftarrow f$

$key[e] \leftarrow w(f, e) = 10$

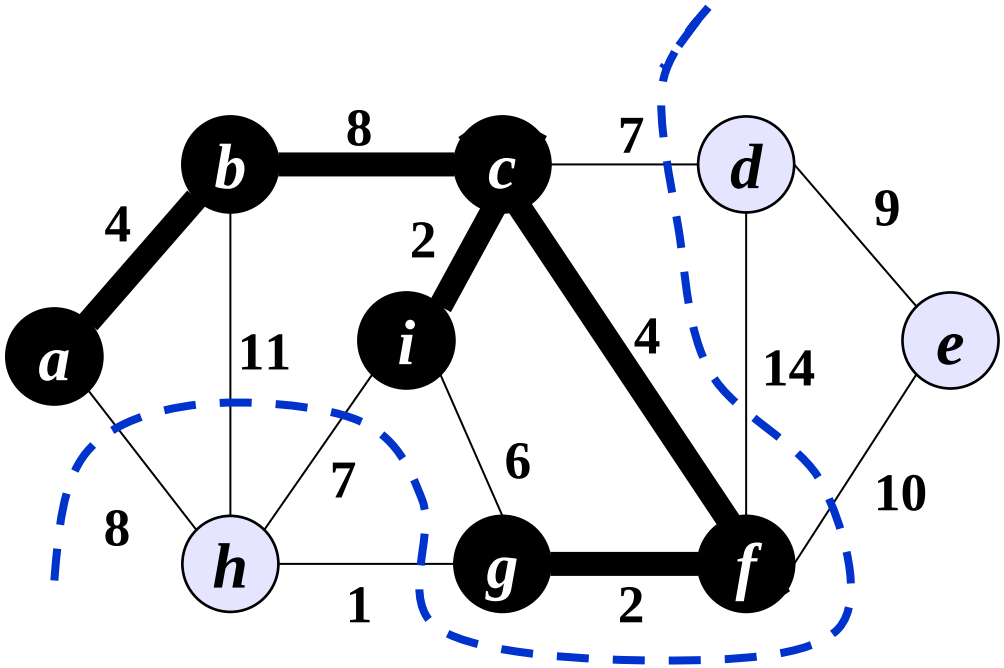
$g \in Q$ and

$w(f, g) < key[g]$ $(2 < 6)$

$\pi[g] \leftarrow f$

$key[g] \leftarrow w(f, g) = 2$

Prim's Algorithm



$g \leftarrow$ EXTRACT-MIN(Q)

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

$f \notin Q$

$h \in Q$ and

$w(g, h) < key[h]$ $(1 < 7)$

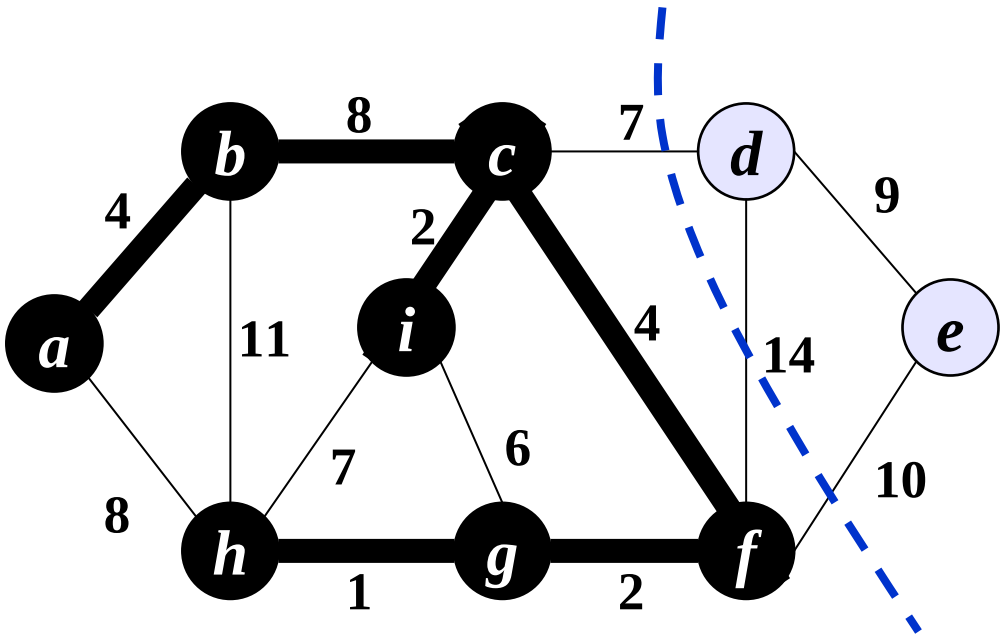
$\pi[h] \leftarrow g$

$key[h] \leftarrow w(g, h) = 1$

$i \notin Q$

Q	d	e	h
	7	10	1

Prim's Algorithm



$h \leftarrow$ EXTRACT-MIN(Q)

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

$a \notin Q$

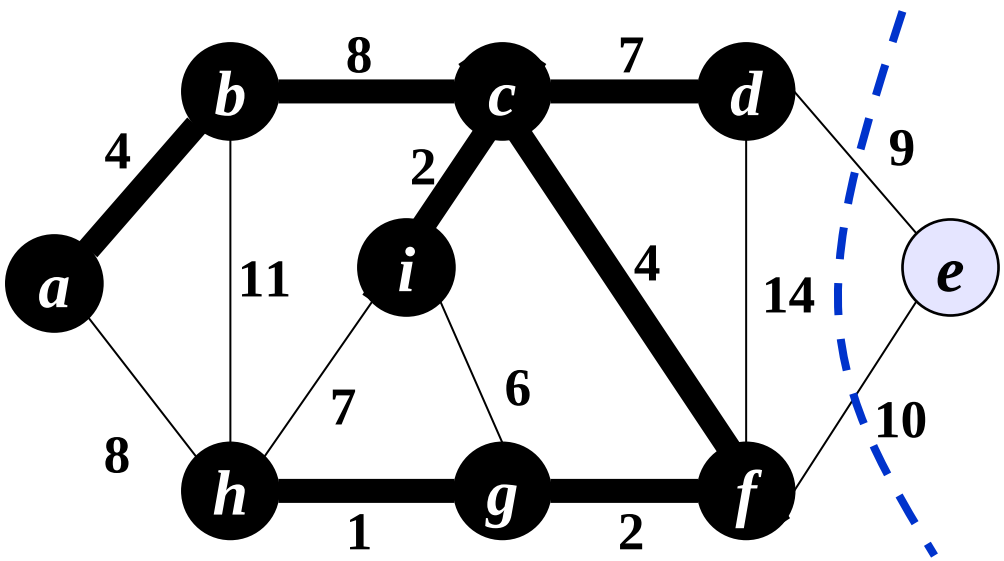
$b \notin Q$

$g \notin Q$

$i \notin Q$

Q	d	e
	7	10

Prim's Algorithm



Q $\boxed{e}$

9

$d \leftarrow$ EXTRACT-MIN(Q)

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

$c \notin Q$

$e \in Q$ and

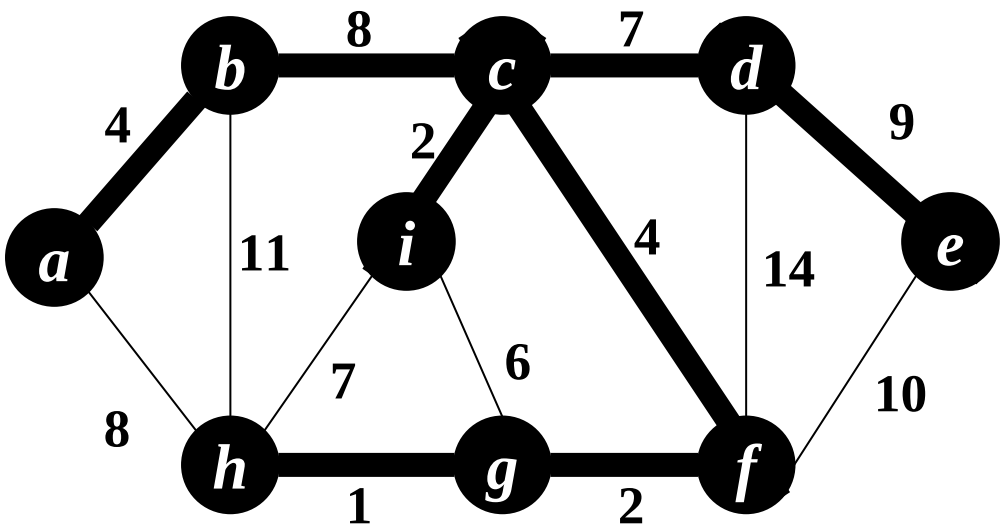
$w(d, e) < key[e]$ $(9 < 10)$

$\pi[e] \leftarrow d$

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

$f \notin Q$

Prim's Algorithm



$e \leftarrow$ EXTRACT-MIN(Q)

$Adj[e] = d, f$

$d \notin Q$

$f \notin Q$

Q $\varnothing$

Importance of Minimal Spanning Trees

There are various applications of Minimal Spanning Trees (MST). Let us consider a couple of real-world examples

1. One practical application would be in designing a network.

- For example, a group of individuals, separated by varying distances, are to be connected in a telephone network.
- Although MST cannot do anything about distance from one connection to another, but it can reduce connecting cost.

2. Another useful application of it is finding airline routes.

- The vertices of the graph would represent cities, and the edges would represent routes between the cities.
- Obviously, more traveling require more cost
- Hence MST can be applied to optimize airline routes by finding the least costly paths with no cycles.