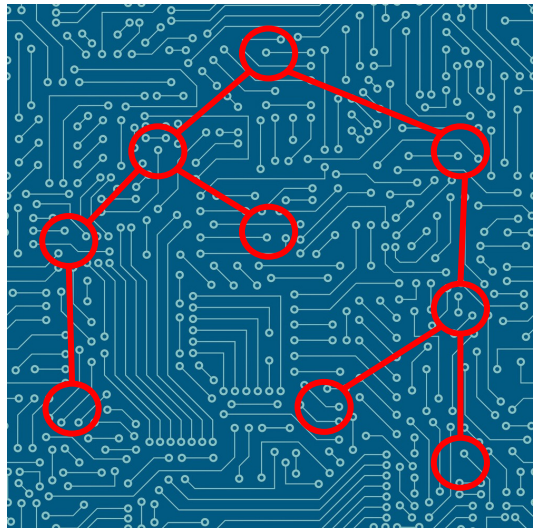
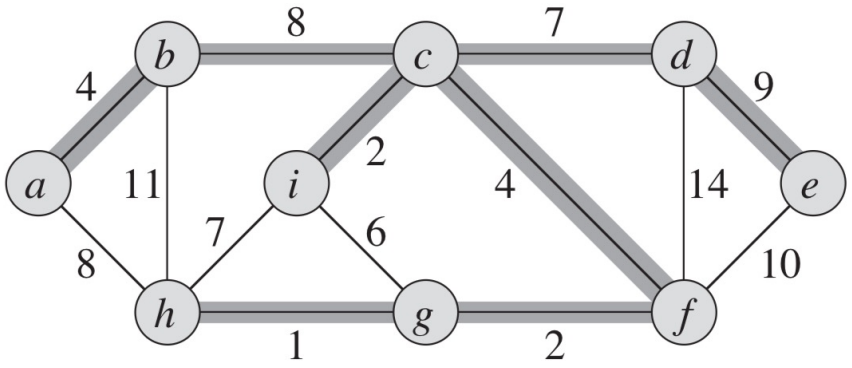
Minimum spanning trees

- Describe the circuit wiring problem with a *connected undirected* graph $G = (V, E)$
  - An undirected graph is *connected* when there is a path between *every* pair of vertices. (there are no unreachable vertices)
- V := the the set of pins.
- E := the set of connections between pairs of pins.



- For each edge $(u, v) \in E$, use a weight $w(u, v)$ to denote the cost (length of wire) to connect pin u and v.
- **Goal**: find an *acyclic subset* $T \subseteq E$ that connects *all* the vertices (pins) and and whose total weight $w(T) = \sum_{(u,v)\in T} w(u, v)$ is *minimized*.

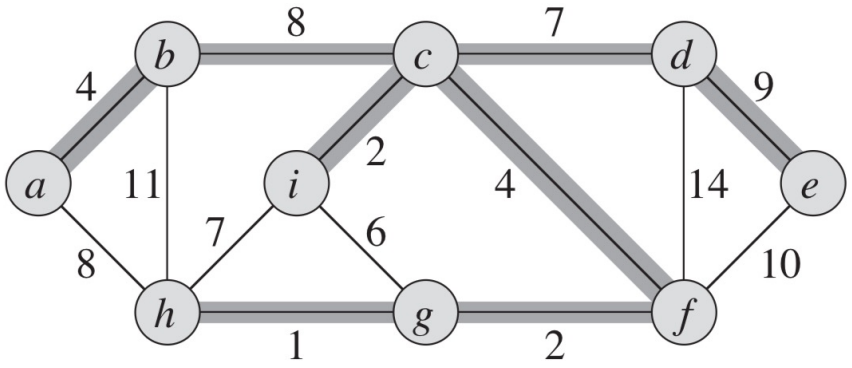
Minimum spanning tree problem



- Since T is *acyclic* and connects all vertices, it must form a ***tre***
- Named as ***spanning tree*** because it "spans" the graph
- The problem of determining the tree T: ***minimum-spanning-tree problem***

NOTE: the term "minimum spanning tree" is short for "minimum-weight spanning tree". We are NOT minimizing the *number*, but the total *weight* of edges, because all spanning trees have exactly $|V| - 1$ edges. (See textbook page 1174, Theorem B.2)

Minimum spanning tree (example)



The total weight = $4 + 8 + 7 + 9 + 2 + 4 + 1 + 2 = 37$

This minimum spanning tree **is not unique**: removing edge (b, c) and replacing it with (a, h) results in another spanning tree with weight 37

Growing a minimum spanning tree (MST)

- A generic method to grow an MST by adding one edge at a time.

Based on

- Kruskal's algorithm

Based on

- Prim's algorithm

A generic method

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

- Assume that we have a connected, undirected graph $G = (V, E)$ with a weight function $w: E \to \mathbb{R}$
- It manages a set of edge A, and maintains the following rule:
- *Prior to each iteration, A is a subset of some minimum spanning tree (MST).*

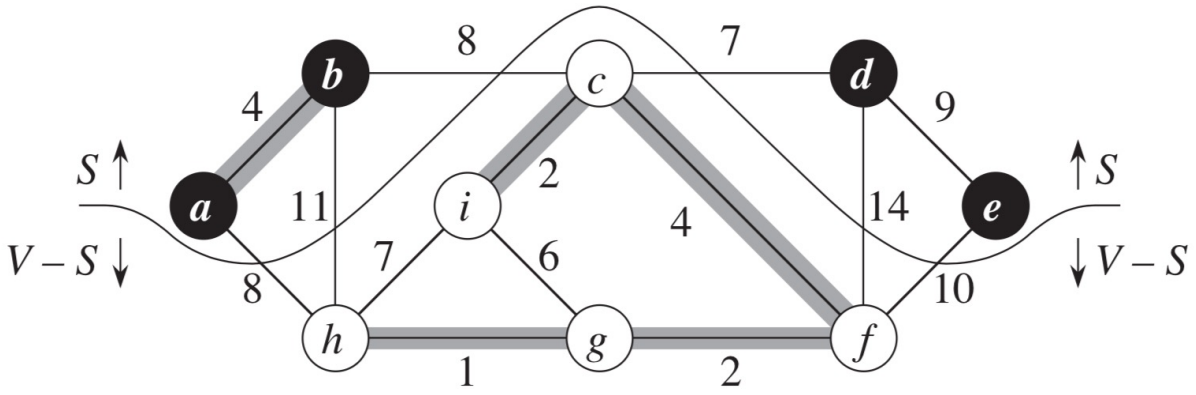
- At each step, find an edge (u, v) to add to A without violating this rule, in the sense that $A \cup \{(u, v)\}$ is also a subset of an MST.
- Such an edge is called a **safe edge** for A.

Defining safe edges

- The tricky part is to find a **safe** edge in line 3.
- Need some rule to recognize the **safe** edges.

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

- Define a "**cut**":
- A ***cut*** $(S, V - S)$ of an undirected graph $G = (V, E)$ is a partition of V.

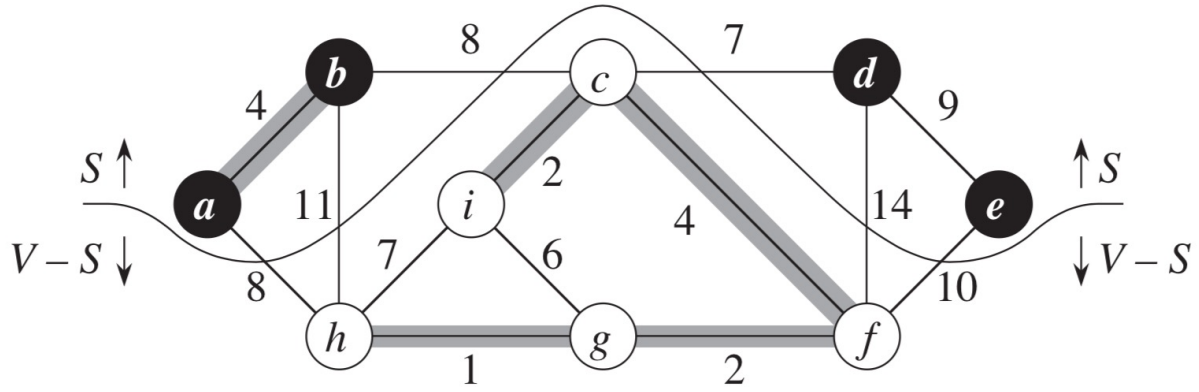


Black vertices are in the set S

White vertices are in the set $V - S$

Define "cross" and "respect"

- We say an edge $(u, v) \in E$ ***crosses*** the cut $(S, V - S)$ if one of its endpoints is in S and the other is in $V - S$.

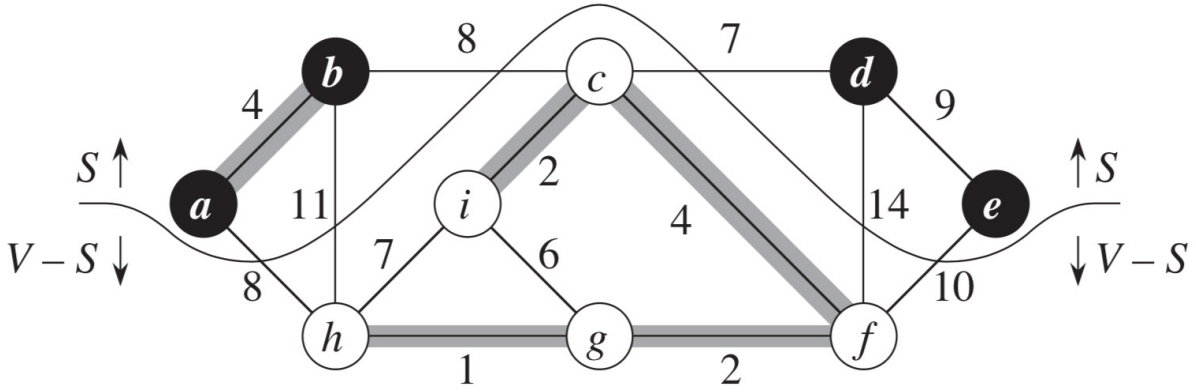


Edges $(a, h), (b, h), (b, c), (c, d), (d, f)$ and (e, f) cross the cut

- Subset A of the edges is shaded: $A = \{(a, b), (c, i), (c, f), (f, g), (g, h)\}$
- We say that a cut **respects** a set A of edges if no edge in A crosses in cut.
  - In the example, the cut respects A because no edge in A crosses the cut.

Define "light edge"

- An edge is a ***light edge*** crossing a cut if its weight is the minimum of any edge crossing the cut.



Edge (c, d) is the unique light edge crossing the cut.

- There can be more than one light edge crossing a cut (i.e., multiple equal weighted edges).

- More generally, an edge is a ***light edge*** (satisfying a given property) if its weight is the minimum of among all the edges (satisfying the property).

Rule for finding safe edges

- The rule for recognizing **safe** edges is given by the following theorem
- ***Theorem***: Let $G = (V, E)$ be a connected, undirected graph with real-valued weight function w defined 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** that **crosses** $(S, V - S)$. Then edge (u, v) is **safe** for A.
- (Theorem 23.1 in textbook on page 626)

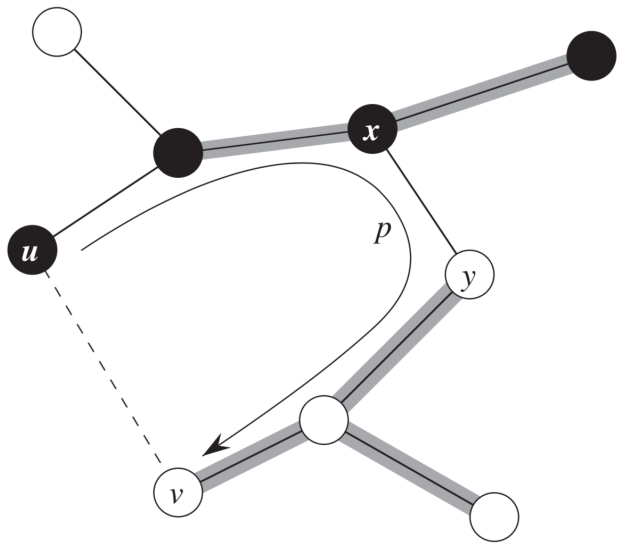
GENERIC-MST(G, w)
1. $A = \emptyset$
2. **while** A does not form a spanning tree
3. find an edge (u, v) that is safe for A
4. $A = A \cup \{(u, v)\}$
5. **return** A

Proof of theorem (optional)

- Let T be a minimum spanning tree that includes A, assuming that T does not contain the light edge (u, v) (if it does, we are done with the proof).

- We shall construct another minimum spanning tree T' that includes $A \cup \{(u, v)\}$, thereby showing that (u, v) is a safe edge for A.

Proof of theorem (cont.) (optional)

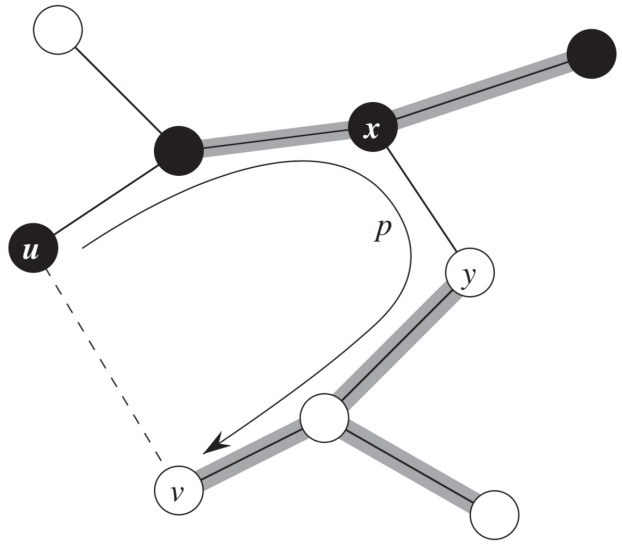


The edges shown are the minimum spanning tree T.
The edges shaded are in A.
I.e., A is a subset of T

- Because we assume T does not contain (u, v), then (u, v) must form a cycle with the edges on the simple path p from u to v in T.
- Since (u, v) are on opposite sides of the cut $(S, V - S)$, at least one edge in T lies on the simple path p and also crosses the cut.
- Let (x, y) be any such edge. (x, y) is not in A, because the cut respects A.
- Removing (x, y) breaks T into two components.
- Adding (u, v) reconnects them to form a new spanning tree $T' = T - \{(x, y)\} \cup \{(u, v)\}$

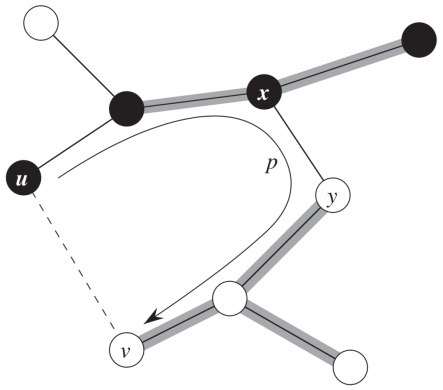
Proof of theorem (cont.) (optional)

- We next show T' is a minimum spanning tree.
- Since (u, v) is a light edge crossing $(S, V - S)$ and (x, y) also crosses the cut, we have
$$w(u, v) \leq w(x, y)$$
- Therefore,
$$w(T') = w(T) - w(x, y) + w(u, v) \leq w(T)$$
- But T is a minimum spanning tree, so that
$$w(T) \leq w(T')$$
- Thus, T' must also be a minimum spanning tree.
- Consequently, (u, v) is **safe** for A.
  - $A \cup \{(u, v)\} = T'$



GENERIC-MST

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



- As the method proceeds, the set A is always acyclic.
- At any point, the graph $G_A = (V, A)$ is a forest, and each of the connected component of G_A is a tree.
- Any safe edge (u, v) for A connects distinct components of G_A.

- The while loops executes $|V| - 1$ times.
- Initially, when $A = \emptyset$, there are $|V|$ trees in G_A, and each iteration reduces that number by 1.
- When the forest contains only a single tree, the method terminates.

Kruskal's algorithm

- The set A is a **forest**, represented by a disjoint-set data structure
- The safe edge added to A is always a ***least-weight*** edge that connects two distinct components.

```
MST-KRUSKAL(G, w)
1.  A = ∅
2.  for each vertex v ∈ G.V
3.      MAKE-SET(v)
4.  sort the edges of G.E into nondecreasing order by weight w
5.  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6.      if FIND-SET(u) ≠ FIND-SET(v)
7.          A = A ∪ {(u, v)}
8.          UNION(u, v)
9.  return A
```

Create $|V|$ trees (sets), one containing each vertex

Examines edges in order of weight, from lowest to highest.
For each edge (u, v) whether the endpoints u and v belong to the same tree (set).
If they do, (u, v) cannot be added to A (they would create a cycle)

Otherwise, (u, v) is added to A (line 7), and the two trees (sets) are merged (line 8).

Review about disjoint-set

- A disjoin-set data structure maintains a collection $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ of disjoint sets
- (two sets are *disjoint* if they have *no element* in common)

- We identify each set by a *representative*, which is some member of the set (randomly chosen or by some rules).

- Chapter 21 in textbook.

Review: Operations on disjoint-set

- MAKE-SET(x)
  - It creates a new set whose only member (and thus representative) is x.

- UNION(x, y)
  - It unites the sets that contain x and y, say S_x and S_y, into a new set that is the union of them.
  - E.g., if $S_x = \{x, z, w\}$, $S_y = \{y, p, q\}$, then $\text{UNION}(x, y) = \{x, z, w, y, p, q\}$
  - Assuming that S_x and S_y are disjoint prior to the operation.

- FIND-SET(x)
  - It returns the representative of the (unique) set containing x.

Review:
Use disjoint-set for connected component problem

- To determine the ***connected component*** of an undirected graph

```
CONNECTED-COMPONENT(G)
1.  for each vertex v ∈ G.V
2.      MAKE-SET(v)
3.  for each edge (u, v) ∈ G.E
4.      if FIND-SET(u) ≠ FIND-SET(v)
5.          UNION(u, v)
```

The procedure initially places each vertex in its own set.

For each edge (u, v), it unites the sets containing u and v

Kruskal's algorithm

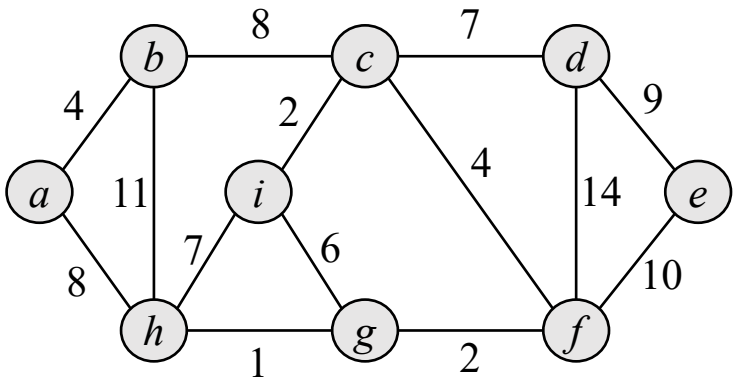
← Similar →

MST-KRUSKAL(G, w)
1. $A = \emptyset$
2. **for** each vertex $v \in G.V$
3. MAKE-SET(v)
4. sort the edges of $G.E$ into nondecreasing order by weight w
5. **for** each edge $(u, v) \in G.E$, taken in nondecreasing order by weight
6. **if** FIND-SET(u) $\neq$ FIND-SET(v)
7. $A = A \cup \{(u, v)\}$
8. UNION(u, v)
9. **return** A

CONNECTED-COMPONENT(G)
1. **for** each vertex $v \in G.V$
2. MAKE-SET(v)
3. **for** each edge $(u, v) \in G.E$
4. **if** FIND-SET(u) $\neq$ FIND-SET(v)
5. UNION(u, v)

Difference: MST assumes that the graph G is ***connected***, that is, there is only one connected component

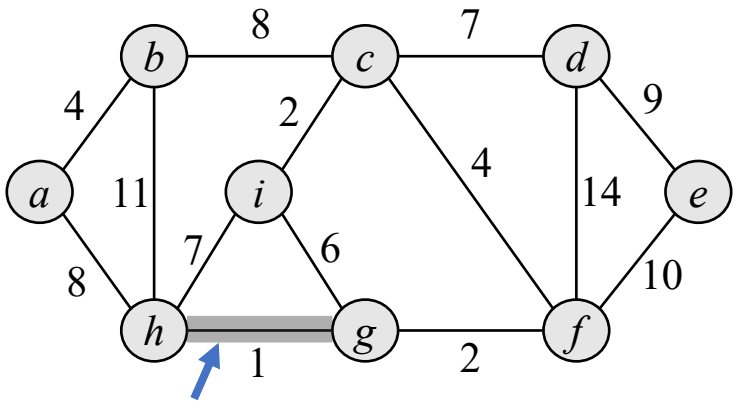
Example for MST-KRUSKAL



$A = \emptyset$

All vertices are initialized to trees:
$\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g\}, \{h\}, \{i\}$

Iteration 1: Edge (g, h) is considered. They belong to two sets

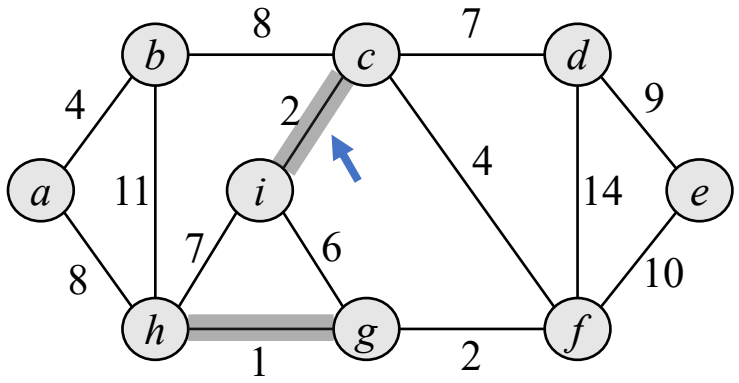


$A = \{(g, h)\}$

Resulting forest:
$\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g, h\}, \{i\}$

Example for MST-KRUSKAL

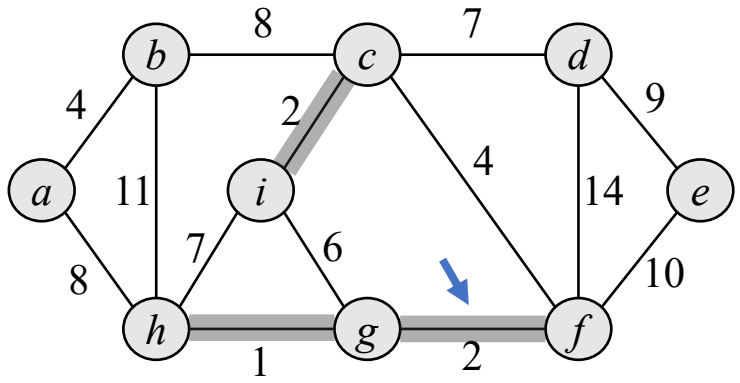
Iteration 2: Edge (c, i) is considered



$A = \{(g, h), (c, i)\}$

Resulting forest:
$\{a\}, \{b\}, \{c, i\}, \{d\}, \{e\}, \{f\}, \{g, h\}$

Iteration 3: Edge (f, g) is considered

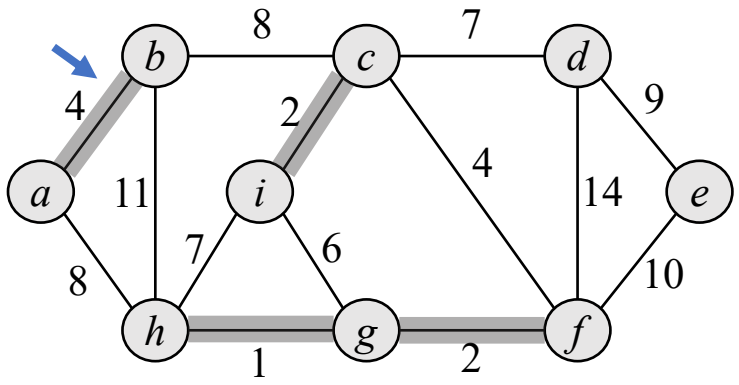


$A = \{(g, h), (c, i), (f, g)\}$

Resulting forest:
$\{a\}, \{b\}, \{c, i\}, \{d\}, \{e\}, \{f, g, h\}$

Example for MST-KRUSKAL

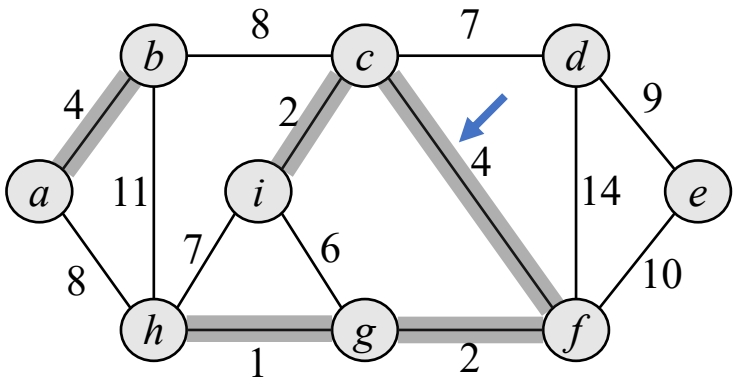
Iteration 4: Edge (a, b) is considered



$A = \{(g, h), (c, i), (f, g), (a, b)\}$

Resulting forest:
$\{a, b\}, \{c, i\}, \{d\}, \{e\}, \{f, g, h\}$

Iteration 5: Edge (c, f) is considered

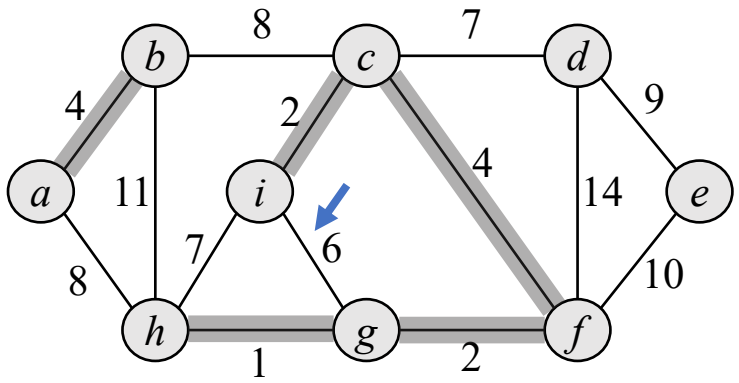


$A = \{(g, h), (c, i), (f, g), (a, b), (c, f)\}$

Resulting forest:
$\{a, b\}, \{d\}, \{e\}, \{c, i, f, g, h\}$

Example for MST-KRUSKAL

Iteration 6: Edge (i, g) is considered. They belong to **the same** set!

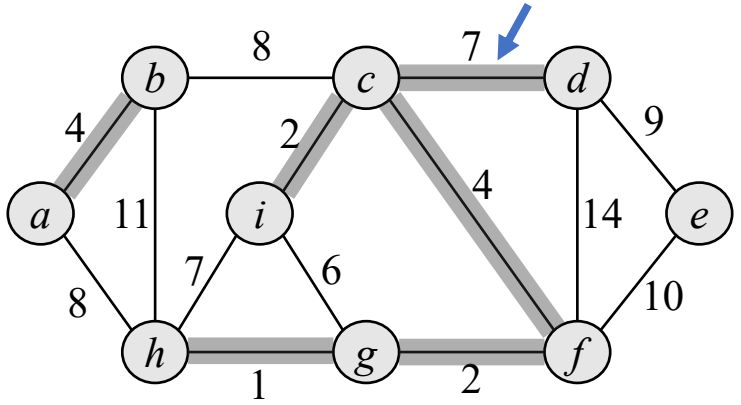


$A = \{(g, h), (c, i), (f, g), (a, b), (c, f)\}$

(i, g) is not added to A

Resulting forest:
$\{a, b\}, \{d\}, \{e\}, \{c, i, f, \mathrm{g}, h\}$

Iteration 7: Edge (c, d) is considered



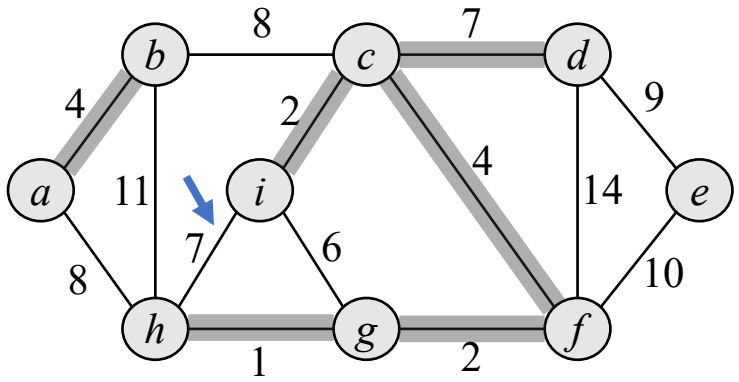
$A = \{(g, h), (c, i), (f, g), (a, b), (c, f), (c, d)\}$

Resulting forest:
$\{a, b\}, \{e\}, \{c, d, i, f, \mathrm{g}, h\}$

Example for MST-KRUSKAL

Iteration 8: Edge (h, i) is considered

They belong to **the same** set!



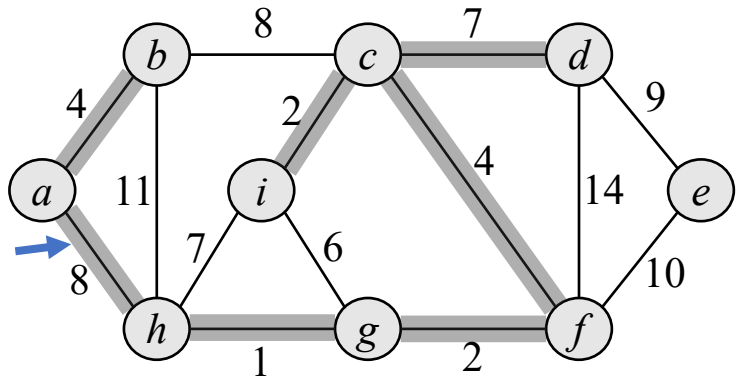
$A = \{(g, h), (c, i), (f, g), (a, b), (c, f), (c, d)\}$

(h, i) is not added to A

Resulting forest:

$\{a, b\}, \{e\}, \{c, d, i, f, \mathrm{g}, h\}$

Iteration 9: Edge (a, h) is considered



$A = \{(g, h), (c, i), (f, g), (a, b), (c, f), (c, d), (a, h)\}$

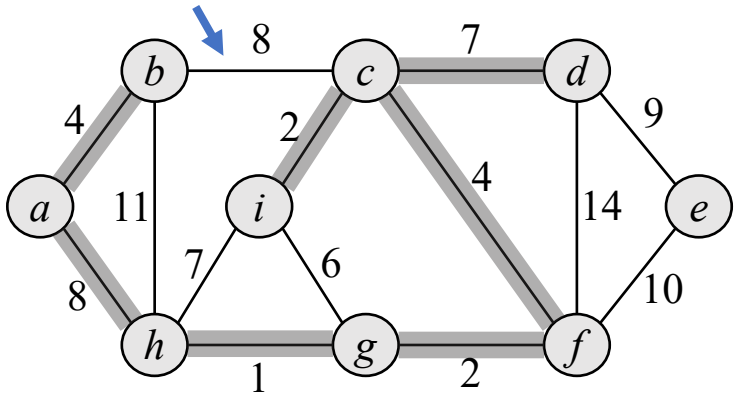
Resulting forest:

$\{e\}, \{a, b, c, d, i, f, \mathrm{g}, h\}$

Example for MST-KRUSKAL

Iteration 10:Edge (b, c) is considered

They belong to **the same** set!



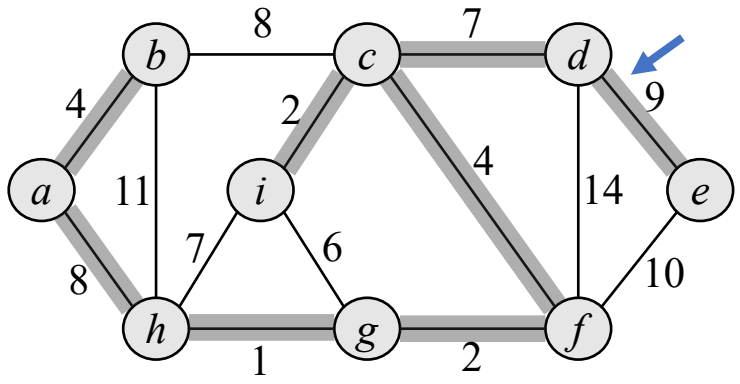
$A = \{(g, h), (c, i), (f, g), (a, b), (c, f), (c, d), (a, h)\}$

(h, i) is not added to A

Resulting forest:

$\{e\}, \{a, b, c, d, i, f, \mathrm{g}, h\}$

Iteration 11: Edge (d, e) is considered



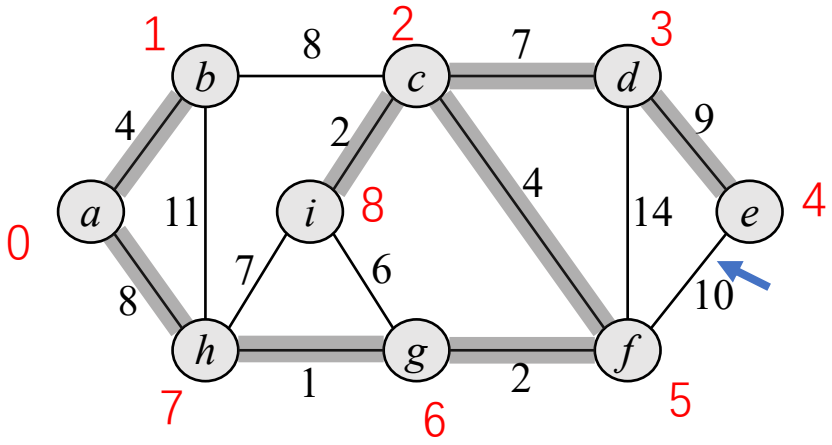
$A = \{(g, h), (c, i), (f, g), (a, b), (c, f), (c, d), (a, h), (d, e)\}$

Resulting forest:

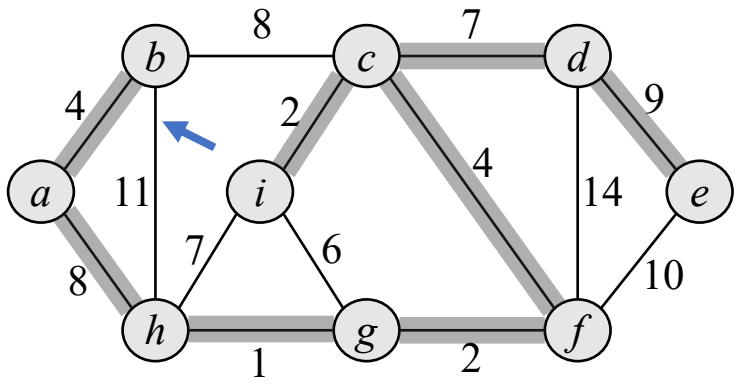
$\{a, b, c, d, e, i, f, \mathrm{g}, h\}$

Example for MST-KRUSKAL

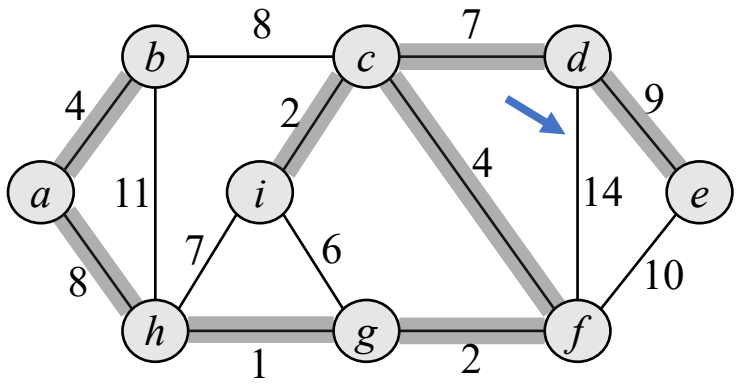
Iteration 12:



Iteration 13:



Iteration 14:



$A = \{(g,h),(c,i),(f,g),(a,b),(c,f),(c,d),(a,h),(d,e)\}$

Resulting forest:
$\{a, b, c, d, e, i, f, \mathrm{g}, h\}$

A contains the edges of a minimum spanning tree

Running time analysis

MST-KRUSKAL(G, w)
1. $A = \emptyset$
2. **for** each vertex $v \in G.V$
3. MAKE-SET(v)
4. sort the edges of $G.E$ into nondecreasing order by weight w
5. **for** each edge $(u, v) \in G.E$, taken in nondecreasing order by weight
6. **if** FIND-SET(u) $\neq$ FIND-SET(v)
7. $A = A \cup \{(u, v)\}$
8. UNION(u, v)
9. **return** A

- It depends on the implementation of the disjoint-set data structure.
- Sorting edges (line 4) takes $O(E \lg E)$ time.
  - $|E| < V^2$, thus, $\lg E = O(\lg V)$
- Line 5 to 8 performs $O(E)$ FIND-SET and UNION operations. Line 2 to 3 has $O(V)$ MAKE-SET operations.
- These take a total of $O((V + E)\alpha(V))$ time.
- With a good implementation, $\alpha(V)$ can be a very slowly growing function, e.g., $\alpha(V) = O(\lg V)$
- Thus, the running time is:
  $$O(E \lg V) + O\big((V + E) \lg V\big) = \boldsymbol{O(E \lg V)}$$