CSE 431/531: Algorithm Analysis and Design (Fall 2024)
Graph Algorithms

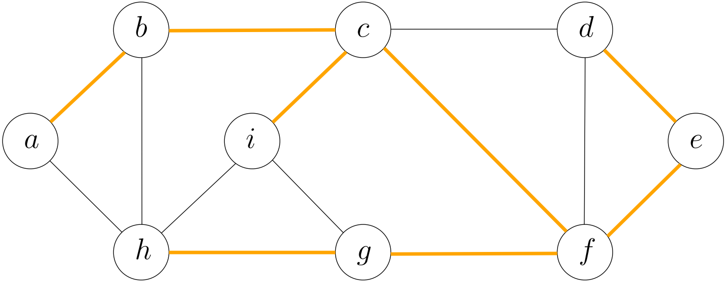
Lecturer: Kelin Luo

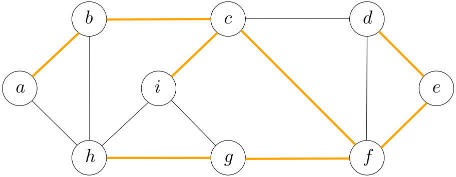
Department of Computer Science and Engineering
University at Buffalo

Outline

Spanning Tree

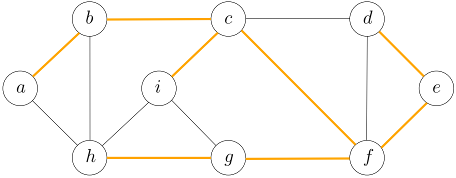
Def. Given a connected graph $G = (V, E)$, a spanning tree $T = (V, F)$ of G is a sub-graph of G that is a tree including all vertices V.





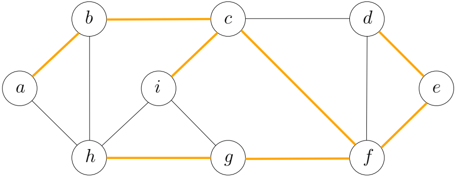
Lemma Let $T = (V, F)$ be a subgraph of $G = (V, E)$. The following statements are equivalent:

- T is a spanning tree of G;



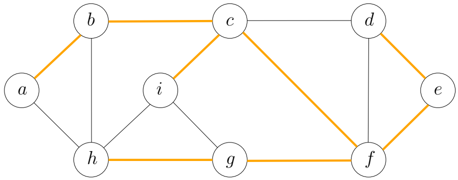
Lemma Let $T = (V, F)$ be a subgraph of $G = (V, E)$. The following statements are equivalent:

- T is a spanning tree of G;
- T is acyclic and connected;



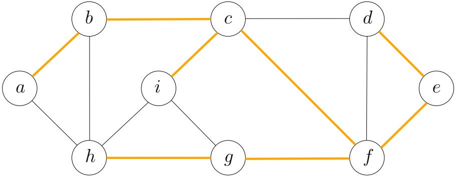
Lemma Let $T = (V, F)$ be a subgraph of $G = (V, E)$. The following statements are equivalent:

- T is a spanning tree of G;
- T is acyclic and connected;
- T is connected and has $n - 1$ edges;



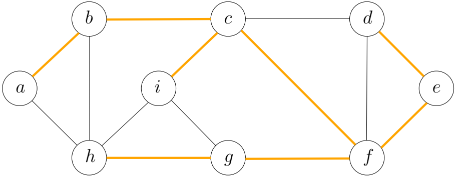
Lemma Let $T = (V, F)$ be a subgraph of $G = (V, E)$. The following statements are equivalent:

- T is a spanning tree of G;
- T is acyclic and connected;
- T is connected and has $n - 1$ edges;
- T is acyclic and has $n - 1$ edges;



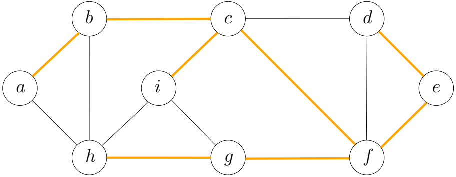
Lemma Let $T = (V, F)$ be a subgraph of $G = (V, E)$. The following statements are equivalent:

- T is a spanning tree of G;
- T is acyclic and connected;
- T is connected and has $n - 1$ edges;
- T is acyclic and has $n - 1$ edges;
- T is minimally connected: removal of any edge disconnects it;



Lemma Let $T = (V, F)$ be a subgraph of $G = (V, E)$. The following statements are equivalent:

- T is a spanning tree of G;
- T is acyclic and connected;
- T is connected and has $n - 1$ edges;
- T is acyclic and has $n - 1$ edges;
- T is minimally connected: removal of any edge disconnects it;
- T is maximally acyclic: addition of any edge creates a cycle;



Lemma Let $T = (V, F)$ be a subgraph of $G = (V, E)$. The following statements are equivalent:

- T is a spanning tree of G;
- T is acyclic and connected;
- T is connected and has $n - 1$ edges;
- T is acyclic and has $n - 1$ edges;
- T is minimally connected: removal of any edge disconnects it;
- T is maximally acyclic: addition of any edge creates a cycle;
- T has a unique simple path between every pair of nodes.

- How to find a spanning tree?
    - BFS

- How to find a spanning tree?
  - BFS
  - DFS

Minimum Spanning Tree (MST) Problem

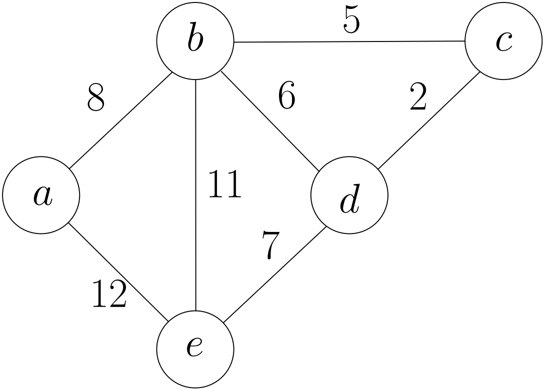
Input: Graph $G = (V, E)$ and edge weights $w : E \to \mathbb{R}$

Output: the spanning tree T of G with the minimum total weight

Minimum Spanning Tree (MST) Problem

Input: Graph $G = (V, E)$ and edge weights $w : E \to \mathbb{R}$

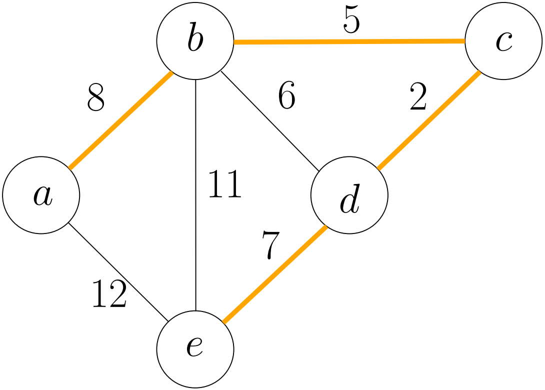
Output: the spanning tree T of G with the minimum total weight



Minimum Spanning Tree (MST) Problem

Input: Graph $G = (V, E)$ and edge weights $w : E \to \mathbb{R}$

Output: the spanning tree T of G with the minimum total weight



Recall: Steps of Designing A Greedy Algorithm

- Design a “reasonable” strategy
- Prove that the reasonable strategy is “safe” (key, usually done by “exchanging argument”)
- Show that the remaining task after applying the strategy is to solve a (many) smaller instance(s) of the same problem (usually trivial)

Def. A choice is “safe” if there is an optimum solution that is “consistent” with the choice

Recall: Steps of Designing A Greedy Algorithm

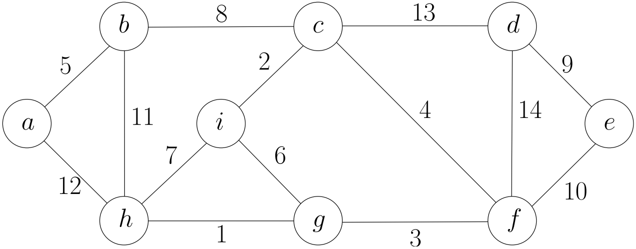
- Design a "reasonable" strategy
- Prove that the reasonable strategy is "safe" (key, usually done by "exchanging argument")
- Show that the remaining task after applying the strategy is to solve a (many) smaller instance(s) of the same problem (usually trivial)

Def. A choice is "safe" if there is an optimum solution that is "consistent" with the choice

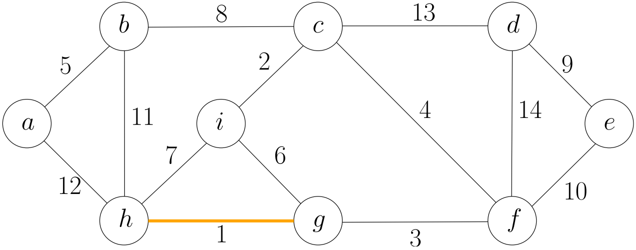
Two Classic Greedy Algorithms for MST

- Kruskal's Algorithm
- Prim's Algorithm

Outline



Q: Which edge can be safely included in the MST?



Q: Which edge can be safely included in the MST?

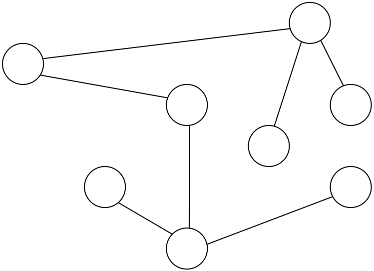
A: The edge with the smallest weight (lightest edge).

Lemma It is safe to include the lightest edge: there is a minimum spanning tree, that contains the lightest edge.

Lemma It is safe to include the lightest edge: there is a minimum spanning tree, that contains the lightest edge.

Proof.

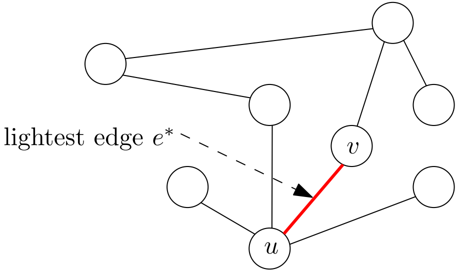
- Take a minimum spanning tree T



Lemma It is safe to include the lightest edge: there is a minimum spanning tree, that contains the lightest edge.

Proof.

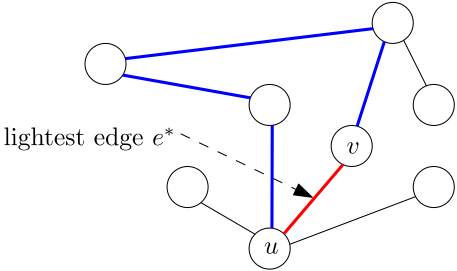
- Take a minimum spanning tree T
- Assume the lightest edge e^* is not in T



Lemma It is safe to include the lightest edge: there is a minimum spanning tree, that contains the lightest edge.

Proof.

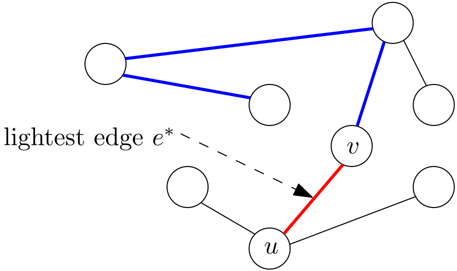
- Take a minimum spanning tree T
- Assume the lightest edge e^* is not in T
- There is a unique path in T connecting u and v



Lemma It is safe to include the lightest edge: there is a minimum spanning tree, that contains the lightest edge.

Proof.

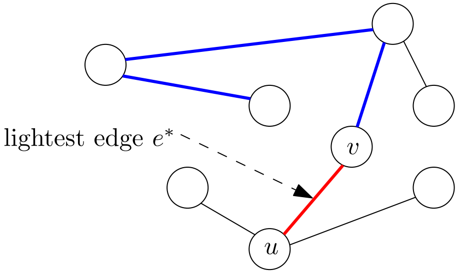
- Take a minimum spanning tree T
- Assume the lightest edge e^* is not in T
- There is a unique path in T connecting u and v
- Remove any edge e in the path to obtain tree T'



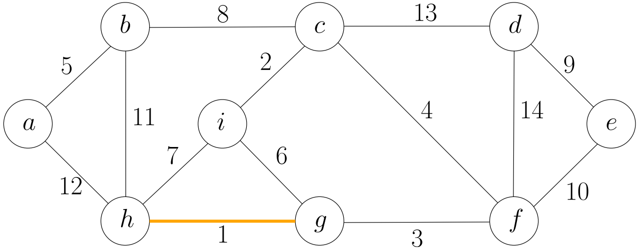
Lemma It is safe to include the lightest edge: there is a minimum spanning tree, that contains the lightest edge.

Proof.

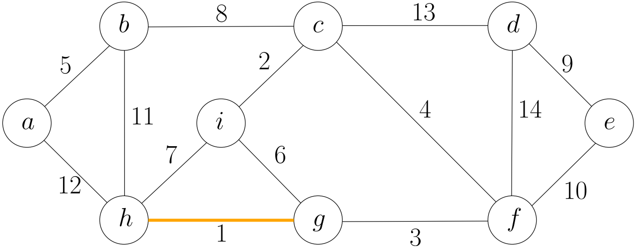
- Take a minimum spanning tree T
- Assume the lightest edge e^* is not in T
- There is a unique path in T connecting u and v
- Remove any edge e in the path to obtain tree T'
- $w(e^*) \le w(e) \implies w(T') \le w(T)$: T' is also a MST $\square$



Is the Residual Problem Still a MST Problem?

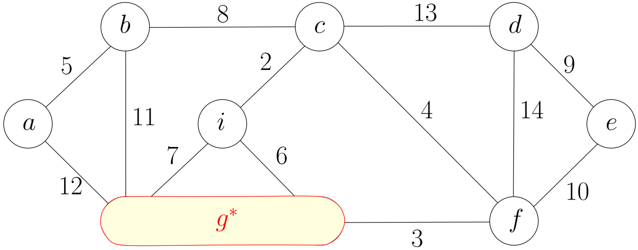


Is the Residual Problem Still a MST Problem?



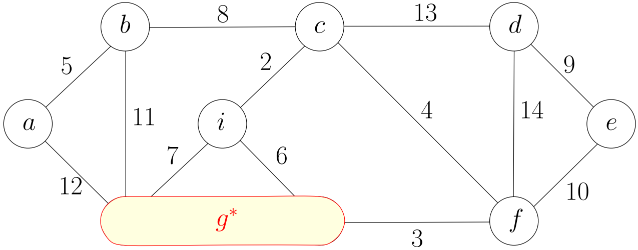
- Residual problem: find the minimum spanning tree that contains edge (g, h)

Is the Residual Problem Still a MST Problem?



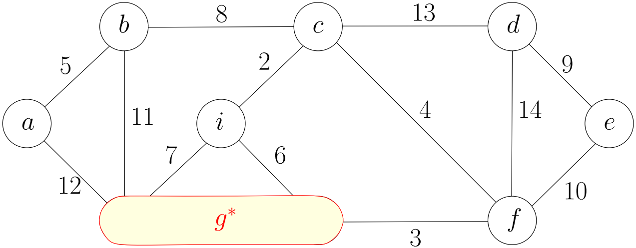
- Residual problem: find the minimum spanning tree that contains edge (g, h)
- Contract the edge (g, h)

Is the Residual Problem Still a MST Problem?

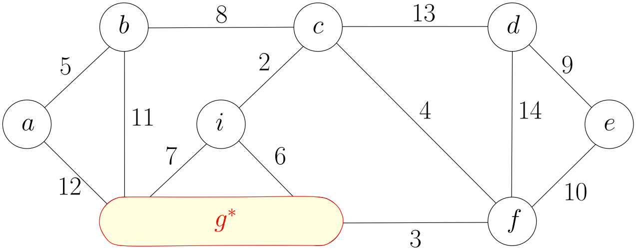


- Residual problem: find the minimum spanning tree that contains edge (g, h)
- Contract the edge (g, h)
- Residual problem: find the minimum spanning tree in the contracted graph

Contraction of an Edge (u, v)

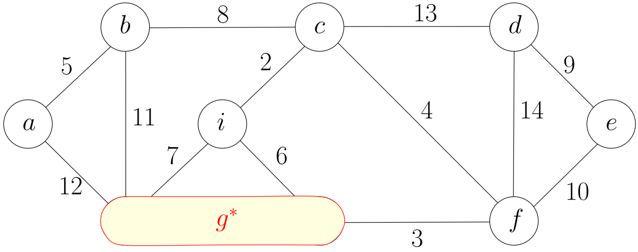


Contraction of an Edge (u, v)



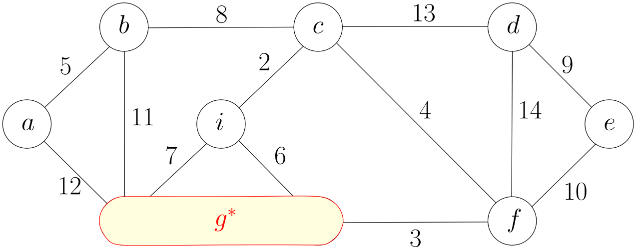
- Remove u and v from the graph, and add a new vertex u^*

Contraction of an Edge (u, v)



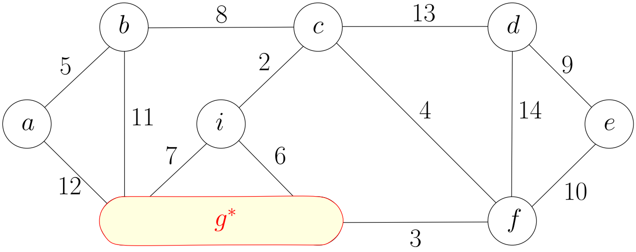
- Remove u and v from the graph, and add a new vertex u^*
- Remove all edges (u, v) from E

Contraction of an Edge (u, v)



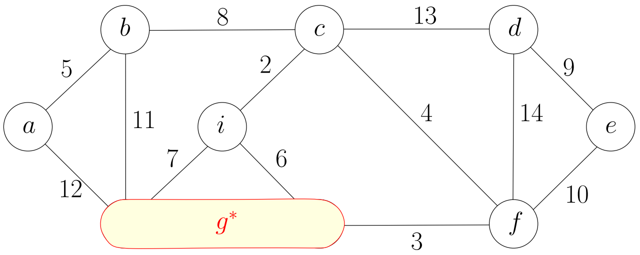
- Remove u and v from the graph, and add a new vertex u^*
- Remove all edges (u, v) from E
- For every edge $(u, w) \in E, w \neq v$, change it to (u^*, w)

Contraction of an Edge (u, v)



- Remove u and v from the graph, and add a new vertex u^*
- Remove all edges (u, v) from E
- For every edge $(u, w) \in E, w \neq v$, change it to (u^*, w)
- For every edge $(v, w) \in E, w \neq u$, change it to (u^*, w)

Contraction of an Edge (u, v)



- Remove u and v from the graph, and add a new vertex u^*
- Remove all edges (u, v) from E
- For every edge $(u, w) \in E, w \neq v$, change it to (u^*, w)
- For every edge $(v, w) \in E, w \neq u$, change it to (u^*, w)
- May create parallel edges! E.g. : two edges (i, g^*)

Greedy Algorithm

Repeat the following step until G contains only one vertex:

1. Choose the lightest edge e^*, add e^* to the spanning tree
2. Contract e^* and update G be the contracted graph

Greedy Algorithm

Repeat the following step until G contains only one vertex:

1. Choose the lightest edge e^*, add e^* to the spanning tree
2. Contract e^* and update G be the contracted graph

Q: What edges are removed due to contractions?

Greedy Algorithm

Repeat the following step until G contains only one vertex:

1. Choose the lightest edge e^*, add e^* to the spanning tree
2. Contract e^* and update G be the contracted graph

Q: What edges are removed due to contractions?

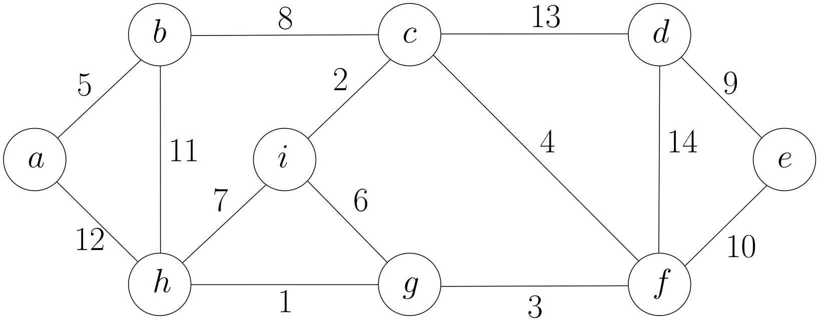
A: Edge (u, v) is removed if and only if there is a path connecting u and v formed by edges we selected

Greedy Algorithm

MST-Greedy(G, w)

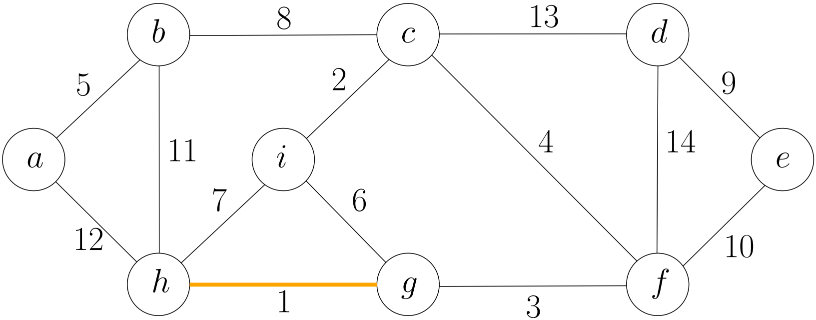
$F \leftarrow \emptyset$
sort edges in E in non-decreasing order of weights w
for each edge (u, v) in the order **do**
    if u and v are not connected by a path of edges in F **then**
        $F \leftarrow F \cup \{(u, v)\}$
return (V, F)

Kruskal's Algorithm: Example



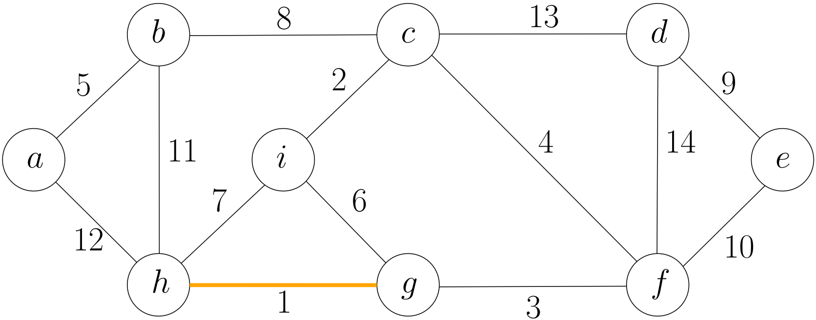
Sets: $\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g\}, \{h\}, \{i\}$

Kruskal's Algorithm: Example



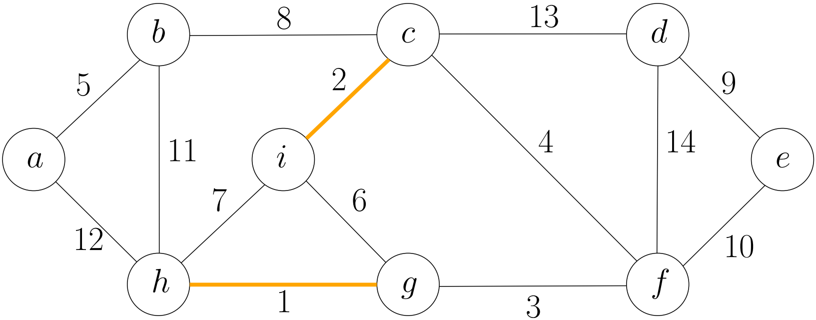
Sets: $\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g\}, \{h\}, \{i\}$

Kruskal's Algorithm: Example



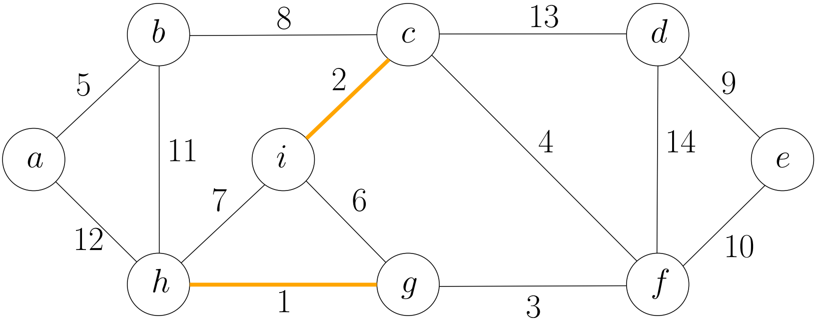
Sets: $\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g, h\}, \{i\}$

Kruskal's Algorithm: Example



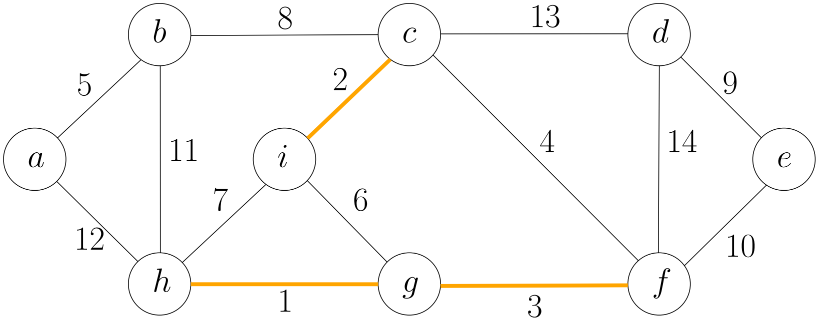
Sets: $\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g, h\}, \{i\}$

Kruskal's Algorithm: Example



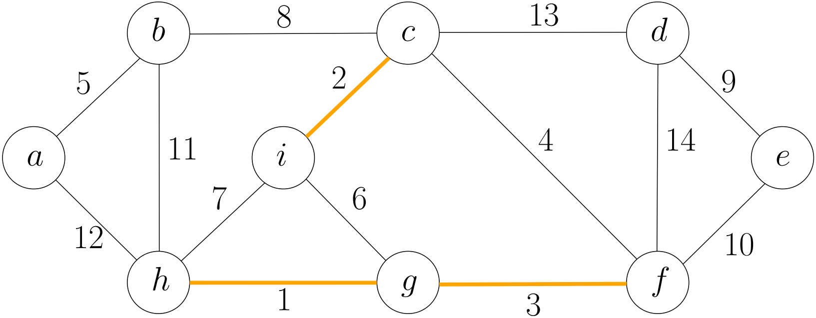
Sets: $\{a\}, \{b\}, \{c, i\}, \{d\}, \{e\}, \{f\}, \{g, h\}$

Kruskal's Algorithm: Example



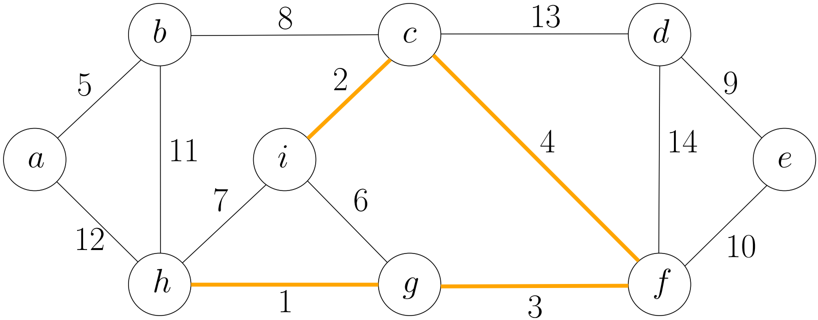
Sets: $\{a\}, \{b\}, \{c, i\}, \{d\}, \{e\}, \{f\}, \{g, h\}$

Kruskal's Algorithm: Example



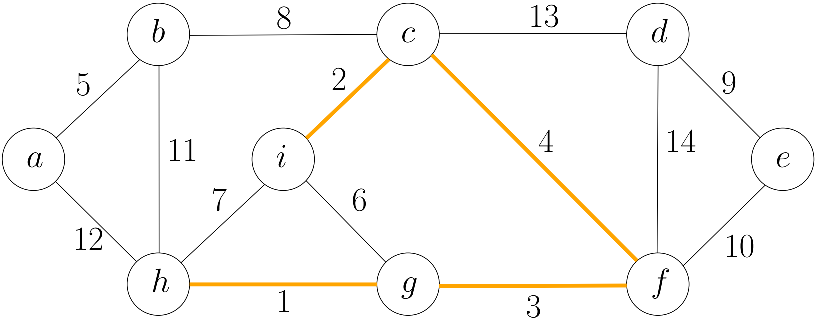
Sets: $\{a\}, \{b\}, \{c, i\}, \{d\}, \{e\}, \{f, g, h\}$

Kruskal's Algorithm: Example



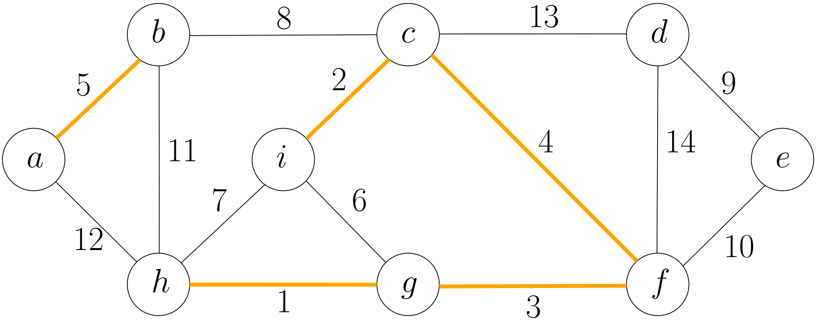
Sets: $\{a\}, \{b\}, \{c, i\}, \{d\}, \{e\}, \{f, g, h\}$

Kruskal's Algorithm: Example



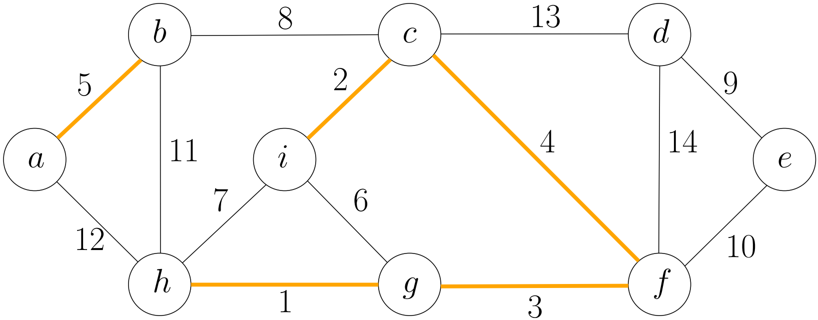
Sets: $\{a\}, \{b\}, \{c, i, f, g, h\}, \{d\}, \{e\}$

Kruskal's Algorithm: Example



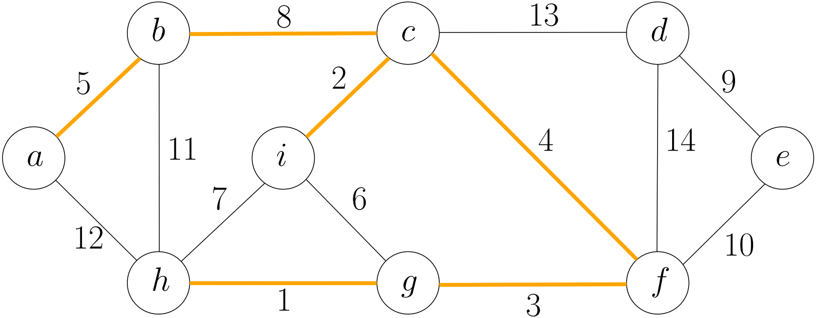
Sets: $\{a\}, \{b\}, \{c, i, f, g, h\}, \{d\}, \{e\}$

Kruskal's Algorithm: Example



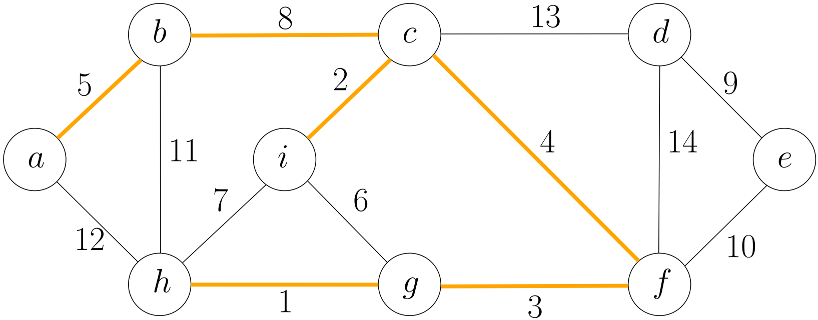
Sets: $\{a, b\}, \{c, i, f, g, h\}, \{d\}, \{e\}$

Kruskal's Algorithm: Example



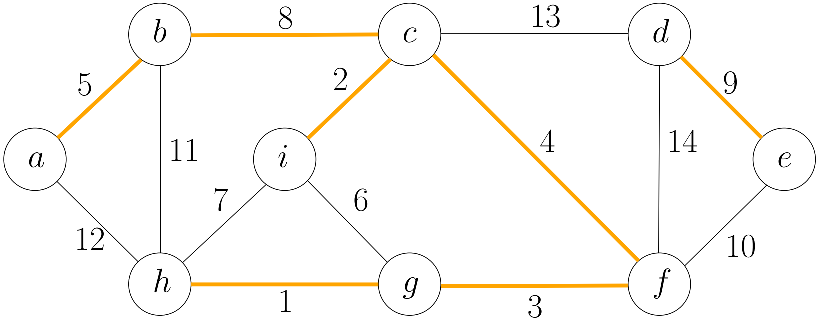
Sets: $\{a, b\}, \{c, i, f, g, h\}, \{d\}, \{e\}$

Kruskal's Algorithm: Example



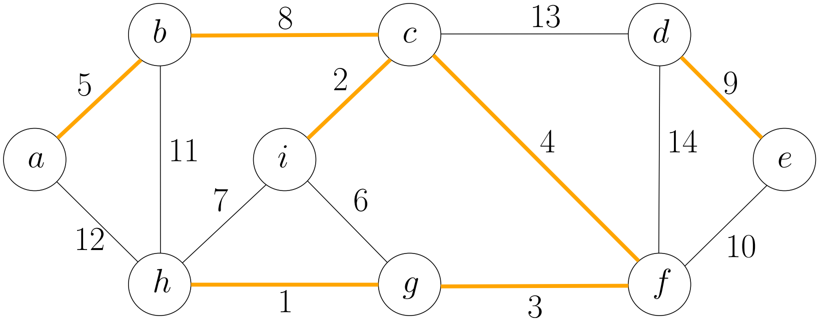
Sets: $\{a, b, c, i, f, g, h\}, \{d\}, \{e\}$

Kruskal's Algorithm: Example



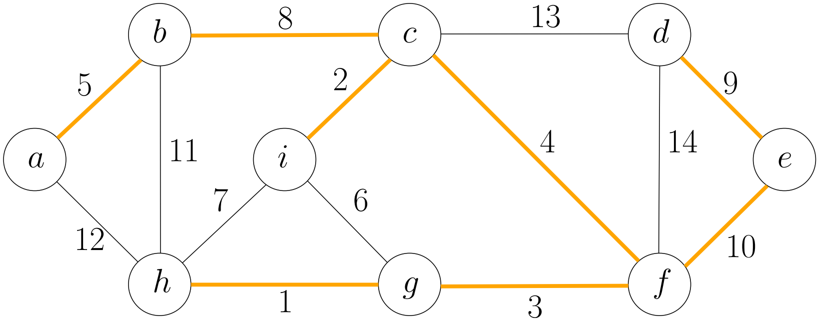
Sets: $\{a, b, c, i, f, g, h\}, \{d\}, \{e\}$

Kruskal's Algorithm: Example



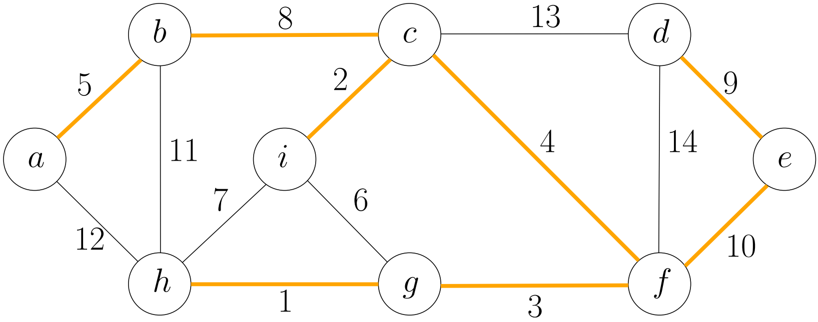
Sets: $\{a, b, c, i, f, g, h\}, \{d, e\}$

Kruskal's Algorithm: Example



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

Kruskal's Algorithm: Example



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

Kruskal's Algorithm: Efficient Implementation of Greedy Algorithm

MST-Kruskal(G, w)

$F \leftarrow \emptyset$
$\mathcal{S} \leftarrow \{\{v\} : v \in V\}$
sort the edges of E in non-decreasing order of weights w
for each edge $(u, v) \in E$ in the order **do**
    $S_u \leftarrow$ the set in $\mathcal{S}$ containing u
    $S_v \leftarrow$ the set in $\mathcal{S}$ containing v
    if $S_u \neq S_v$ **then**
        $F \leftarrow F \cup \{(u, v)\}$
        $\mathcal{S} \leftarrow \mathcal{S} \setminus \{S_u\} \setminus \{S_v\} \cup \{S_u \cup S_v\}$
return (V, F)

Running Time of Kruskal's Algorithm

MST-Kruskal(G, w)

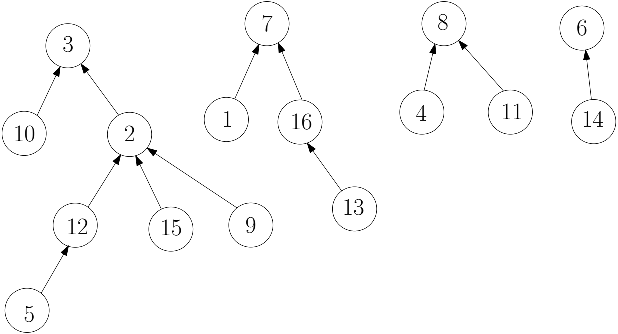
```
1:  F ← ∅
2:  𝒮 ← {{v} : v ∈ V}
3:  sort the edges of E in non-decreasing order of weights w
4:  for each edge (u, v) ∈ E in the order do
5:      S_u ← the set in 𝒮 containing u
6:      S_v ← the set in 𝒮 containing v
7:      if S_u ≠ S_v then
8:          F ← F ∪ {(u, v)}
9:          𝒮 ← 𝒮 \ {S_u} \ {S_v} ∪ {S_u ∪ S_v}
10: return (V, F)
```

Use union-find data structure to support 2, 5, 6, 7, 9.

Union-Find Data Structure

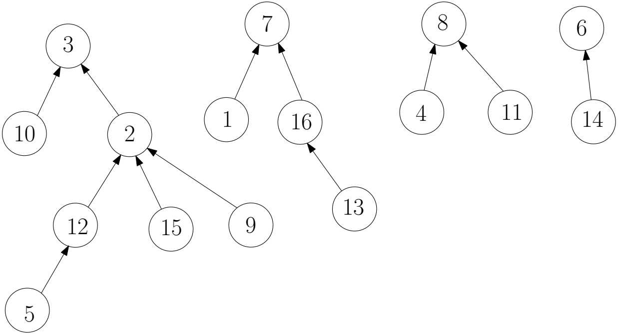
- V: ground set
- We need to maintain a partition of V and support following operations:
  - Check if u and v are in the same set of the partition
  - Merge two sets in partition

- $V = \{1, 2, 3, \cdots, 16\}$
- Partition: $\{2, 3, 5, 9, 10, 12, 15\}, \{1, 7, 13, 16\}, \{4, 8, 11\}, \{6, 14\}$

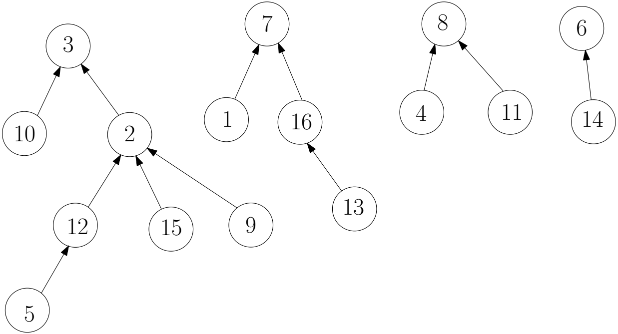


- $par[i]$: parent of i, ($par[i] = \perp$ if i is a root).

Union-Find Data Structure

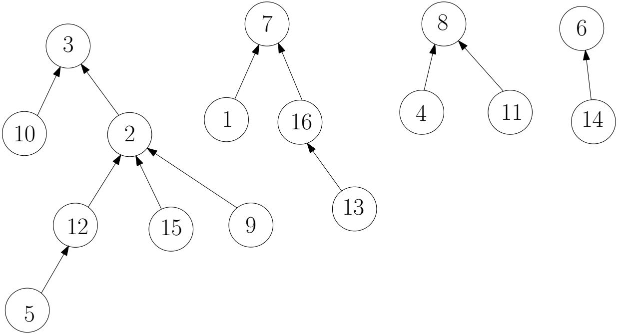


Union-Find Data Structure



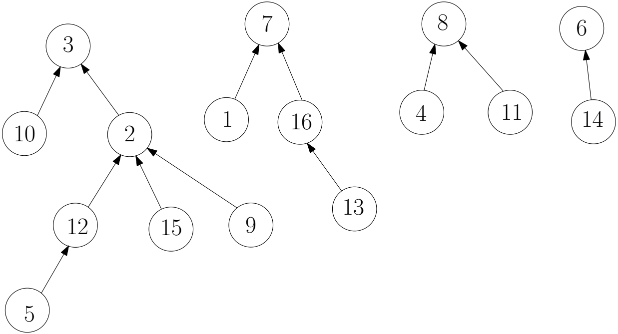
- Q: how can we check if u and v are in the same set?

Union-Find Data Structure



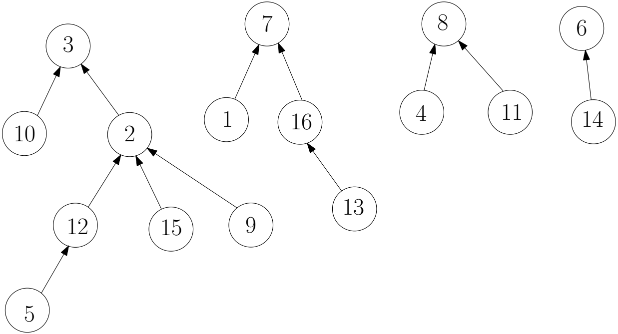
- Q: how can we check if u and v are in the same set?
- A: Check if root(u) = root(v).

Union-Find Data Structure



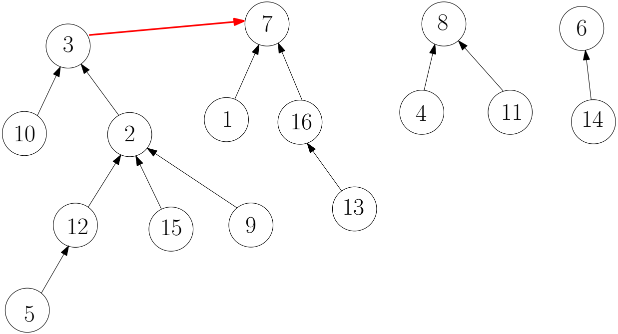
- Q: how can we check if u and v are in the same set?
- A: Check if root(u) = root(v).
- root(u): the root of the tree containing u

Union-Find Data Structure



- Q: how can we check if u and v are in the same set?
- A: Check if root(u) = root(v).
- root(u): the root of the tree containing u
- Merge the trees with root r and r': $par[r] \leftarrow r'$.

Union-Find Data Structure



- Q: how can we check if u and v are in the same set?
- A: Check if root(u) = root(v).
- root(u): the root of the tree containing u
- Merge the trees with root r and r': $par[r] \leftarrow r'$.

Union-Find Data Structure

root(v)

if $par[v] = \perp$ **then**
    return v
else
    return root($par[v]$)

Union-Find Data Structure

root(v)

```
if par[v] = ⊥ then
    return v
else
    return root(par[v])
```

- Problem: the tree might too deep; running time might be large

Union-Find Data Structure

root(v)

if $par[v] = \perp$ **then**
  return v
else
  return root($par[v]$)

- Problem: the tree might too deep; running time might be large
- Improvement: all vertices in the path directly point to the root, saving time in the future.

Union-Find Data Structure

root(v)

if $par[v] = \perp$ **then**
    return v
else
    return root($par[v]$)

root(v)

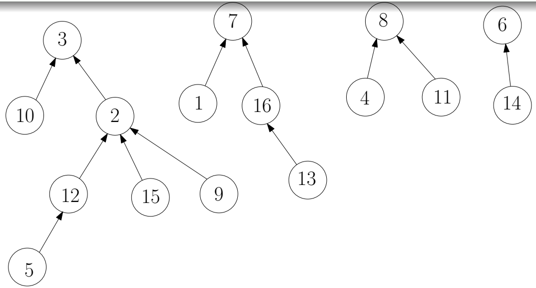
if par$[v] = \perp$ **then**
    return v
else
    $par[v] \leftarrow$ root($par[v]$)
return $par[v]$

- Problem: the tree might too deep; running time might be large
- Improvement: all vertices in the path directly point to the root, saving time in the future.

Union-Find Data Structure

root(v)

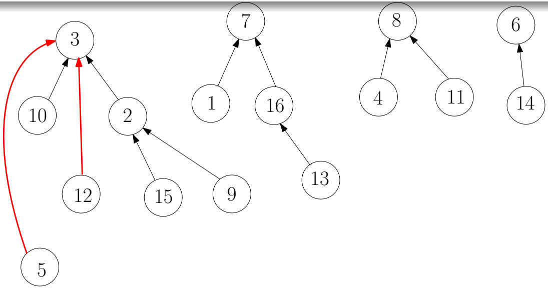
1: **if** $par[v] = \perp$ **then**
2:     **return** v
3: **else**
4:     $par[v] \leftarrow \text{root}(par[v])$
5:     **return** $par[v]$



Union-Find Data Structure

root(v)

> **if** $par[v] = \perp$ **then**
>   **return** v
> **else**
> $par[v] \leftarrow \text{root}(par[v])$
>   **return** $par[v]$



MST-Kruskal(G, w)

$F \leftarrow \emptyset$
$\mathcal{S} \leftarrow \{\{v\} : v \in V\}$
sort the edges of E in non-decreasing order of weights w
for each edge $(u, v) \in E$ in the order **do**
    $S_u \leftarrow$ the set in $\mathcal{S}$ containing u
    $S_v \leftarrow$ the set in $\mathcal{S}$ containing v
    if $S_u \neq S_v$ **then**
        $F \leftarrow F \cup \{(u, v)\}$
        $\mathcal{S} \leftarrow \mathcal{S} \setminus \{S_u\} \setminus \{S_v\} \cup \{S_u \cup S_v\}$
return (V, F)

MST-Kruskal(G, w)

```
F ← ∅
for every v ∈ V do: par[v] ← ⊥
sort the edges of E in non-decreasing order of weights w
for each edge (u, v) ∈ E in the order do
    u' ← root(u)
    v' ← root(v)
    if u' ≠ v' then
        F ← F ∪ {(u, v)}
        par[u'] ← v'
return (V, F)
```

MST-Kruskal(G, w)

1: $F \leftarrow \emptyset$
2: **for** every $v \in V$ **do**: $par[v] \leftarrow \perp$
3: sort the edges of E in non-decreasing order of weights w
4: **for** each edge $(u, v) \in E$ in the order **do**
5:     $u' \leftarrow \text{root}(u)$
6:     $v' \leftarrow \text{root}(v)$
7:     **if** $u' \neq v'$ **then**
8:         $F \leftarrow F \cup \{(u, v)\}$
9:         $par[u'] \leftarrow v'$
10: **return** (V, F)

- 2,5,6,7,9 takes time $O(m\alpha(n))$
- $\alpha(n)$ is very slow-growing: $\alpha(n) \leq 4$ for $n \leq 10^{80}$.

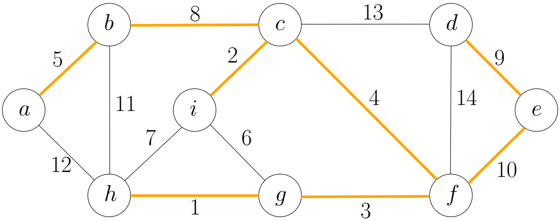
MST-Kruskal(G, w)

1: $F \leftarrow \emptyset$
2: **for** every $v \in V$ **do**: $par[v] \leftarrow \perp$
3: sort the edges of E in non-decreasing order of weights w
4: **for** each edge $(u, v) \in E$ in the order **do**
5:     $u' \leftarrow$ root(u)
6:     $v' \leftarrow$ root(v)
7:     **if** $u' \neq v'$ **then**
8:         $F \leftarrow F \cup \{(u, v)\}$
9:         $par[u'] \leftarrow v'$
10: **return** (V, F)

- 2,5,6,7,9 takes time $O(m\alpha(n))$
- $\alpha(n)$ is very slow-growing: $\alpha(n) \leq 4$ for $n \leq 10^{80}$.
- Running time $=$ time for 3 $= O(m \lg n)$.

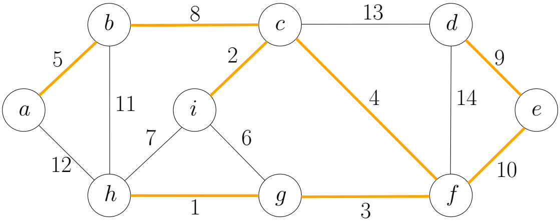
Assumption Assume all edge weights are different.

Lemma An edge $e \in E$ is not in the MST, if and only if there is cycle C in G in which e is the heaviest edge.



Assumption Assume all edge weights are different.

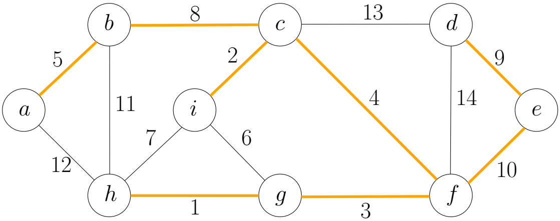
Lemma An edge $e \in E$ is not in the MST, if and only if there is cycle C in G in which e is the heaviest edge.



- (i, g) is not in the MST because of cycle (i, c, f, g)

Assumption Assume all edge weights are different.

Lemma An edge $e \in E$ is not in the MST, if and only if there is cycle C in G in which e is the heaviest edge.



- (i, g) is not in the MST because of cycle (i, c, f, g)
- (e, f) is in the MST because no such cycle exists

Outline

1. Minimum Spanning Tree
   - Kruskal's Algorithm
   - Reverse-Kruskal's Algorithm
   - Prim's Algorithm

Outline

1. Minimum Spanning Tree
    - Kruskal's Algorithm
    - Reverse-Kruskal's Algorithm
    - Prim's Algorithm