Minimal Spanning Trees

Antonio Carzaniga

Faculty of Informatics
Università della Svizzera italiana

May 12, 2025

Outline

- MST problem
- Generic algorithm
- Prim and Kruskal

- Given a weighted graph $G = (V, E)$
  - with "weight" function $w : E \longrightarrow \mathbb{R}$

Minimum Spanning Tree

- Given a weighted graph $G = (V, E)$
  - with "weight" function $w : E \longrightarrow \mathbb{R}$
- Find an *acyclic* subset $T \subseteq E$
  - a ***tree***

- Given a weighted graph $G = (V, E)$
  - with "weight" function $w : E \longrightarrow \mathbb{R}$
- Find an *acyclic* subset $T \subseteq E$
  - a ***tree***
- T "spans" the entire graph G (i.e., touches all vertexes)
  - a ***spanning tree***

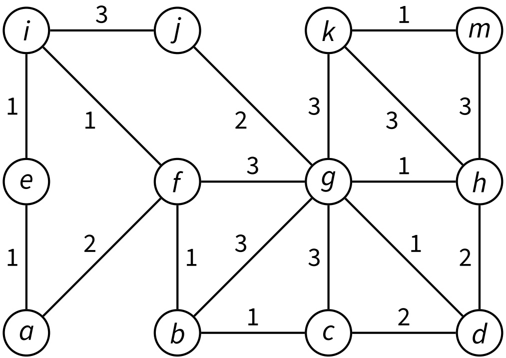
Minimum Spanning Tree

- Given a weighted graph $G = (V, E)$
  - with "weight" function $w : E \longrightarrow \mathbb{R}$
- Find an *acyclic* subset $T \subseteq E$
  - a ***tree***
- T "spans" the entire graph G (i.e., touches all vertexes)
  - a ***spanning tree***
- T's total weight of the tree is minimal

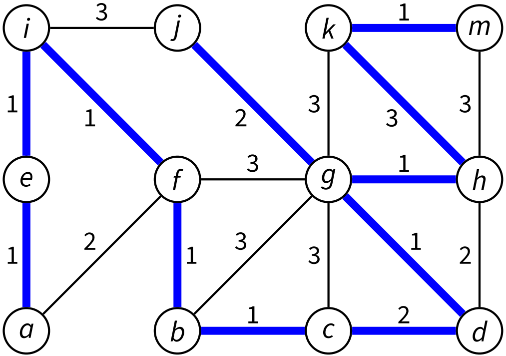
$$w(T) = \sum_{(u,v) \in T} w(u, v)$$

  - a ***minimum-weight spanning tree***, or "minimum spanning tree"

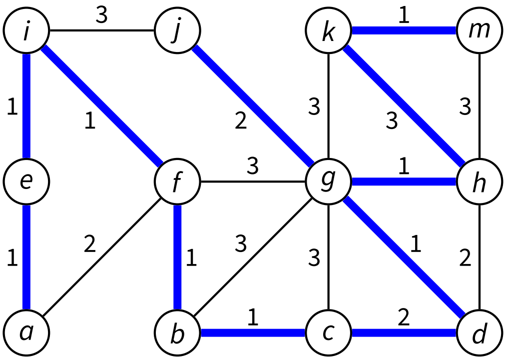
Example



Example

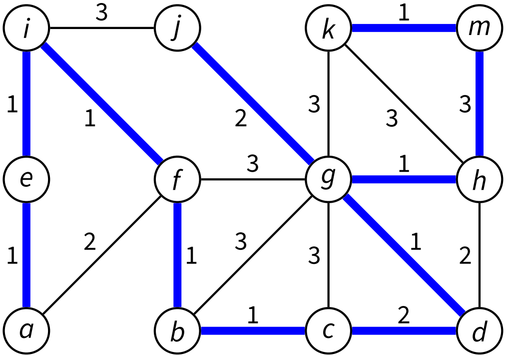


Example



- There may be more than one minimum spanning tree

Example



- There may be more than one minimum spanning tree

General Idea

General Idea

- Build T by adding one edge at a time

General Idea

- Build T by adding one edge at a time
- Each time, add $e \in E$ such that
  - e is the ***lightest edge***
  - e ***does not create a cycle***
- Stop when we have covered all vertexes

- Build T by adding one edge at a time
- Each time, add $e \in E$ such that
  - e is the ***lightest edge***
  - e ***does not create a cycle***
- Stop when we have covered all vertexes
- This is a *“greedy” algorithm*

General Idea

- Build T by adding one edge at a time
- Each time, add $e \in E$ such that
  - e is the ***lightest edge***
  - e ***does not create a cycle***
- Stop when we have covered all vertexes
- This is a *“greedy” algorithm*
- Does it work?

```
GENERIC-MST(G, w)
A = ∅
while A is not a spanning tree
      find a safe edge e = (u, v)
      A = A ∪ {e}
```

Refined Idea

```
GENERIC-MST(G, w)
A = ∅
while A is not a spanning tree
        find a safe edge e = (u, v)
        A = A ∪ {e}
```

- ***Invariant:*** A is a subset of a minimum spanning tree

```
GENERIC-MST(G, w)
A = ∅
while A is not a spanning tree
        find a safe edge e = (u, v)
        A = A ∪ {e}
```

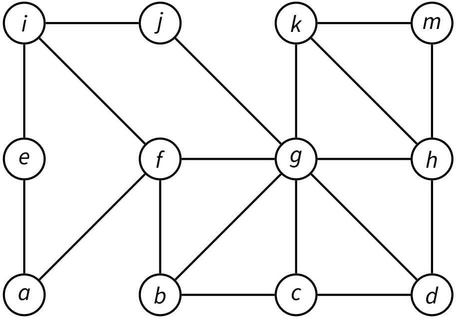
- ***Invariant:*** A is a subset of a minimum spanning tree
- A ***safe edge*** is an edge that maintains the invariant
  - e is such that, if A is a subset of a minimum spanning tree, then $A \cup \{e\}$ is also a subset of a minimum spanning tree

```
GENERIC-MST(G, w)
A = ∅
while A is not a spanning tree
      find a safe edge e = (u, v)
      A = A ∪ {e}
```

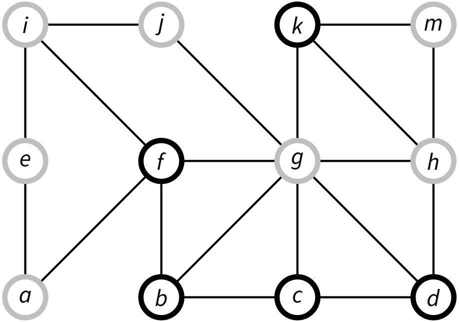
- ***Invariant:*** A is a subset of a minimum spanning tree
- A ***safe edge*** is an edge that maintains the invariant
  - e is such that, if A is a subset of a minimum spanning tree, then $A \cup \{e\}$ is also a subset of a minimum spanning tree
  - more or less the *definition* of a greedy algorithm

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

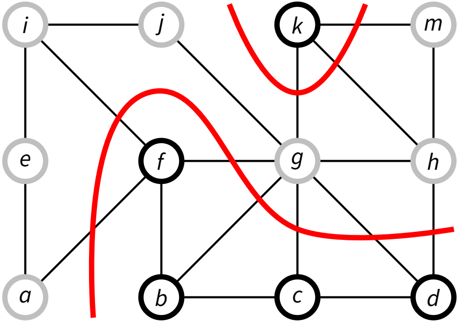
- A ***cut*** $(S, V - S)$ of an undirected graph $G = (V, E)$ is a *partition* of V



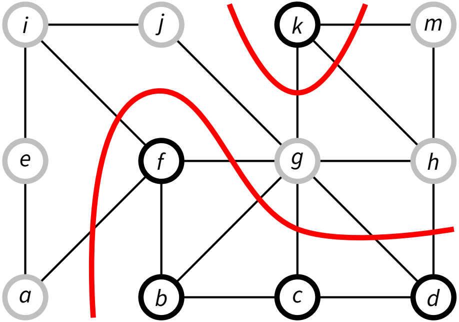
- A ***cut*** $(S, V - S)$ of an undirected graph $G = (V, E)$ is a *partition* of V



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

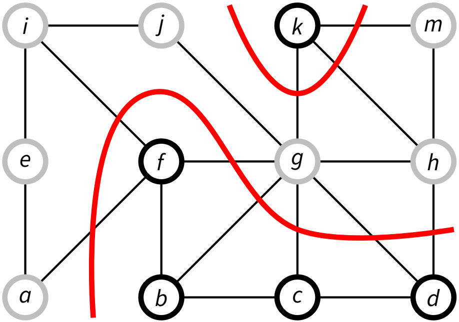


- A ***cut*** $(S, V - S)$ of an undirected graph $G = (V, E)$ is a *partition* of V



- An edge $e = (u, v)$ *crosses* the cut $(S, V - S)$ if $u \in S$ and $v \in V - S$, or vice-versa

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



- An edge $e = (u, v)$ *crosses* the cut $(S, V - S)$ if $u \in S$ and $v \in V - S$, or vice-versa

- A cut $(S, V - S)$ *respects* a set of edges A if no edge in A crosses the cut

Finding a Safe Edge

```
GENERIC-MST(G, w)
A = ∅
while A is not a spanning tree
    find a safe edge e = (u, v)
    A = A ∪ {e}
```

Finding a Safe Edge

```
GENERIC-MST(G, w)
A = ∅
while A is not a spanning tree
    find a safe edge e = (u, v)
    A = A ∪ {e}
```

- Let $(S, V - S)$ be a cut of G that respects A

Finding a Safe Edge

GENERIC-MST(G, w)

```
A = ∅
while A is not a spanning tree
      find a safe edge e = (u, v)
      A = A ∪ {e}
```

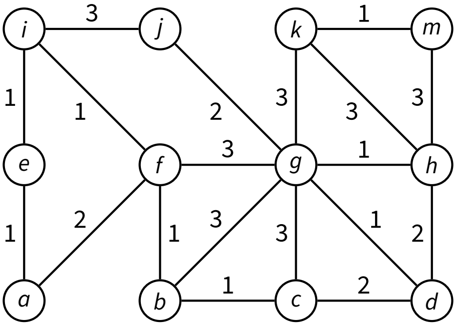
- Let $(S, V - S)$ be a cut of G that respects A
- A minimum-weight edge e that crosses $(S, V - S)$ is a safe edge

Example

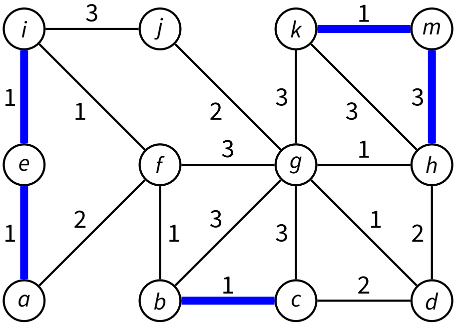
- Let $(S, V - S)$ be a cut of G that respects A
- A minimum-weight edge e that crosses $(S, V - S)$ is a safe edge

Example

- Let $(S, V - S)$ be a cut of G that respects A
- A minimum-weight edge e that crosses $(S, V - S)$ is a safe edge

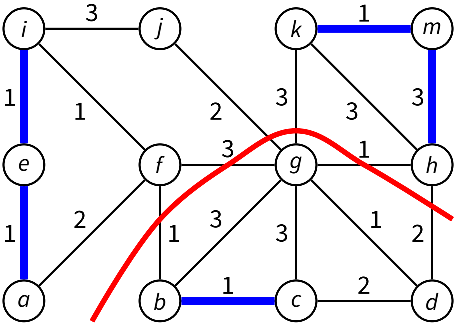


- Let $(S, V - S)$ be a cut of G that respects A
- A minimum-weight edge e that crosses $(S, V - S)$ is a safe edge



- $A = \{(a, e), (e, i), (b, c), (k, m), (m, h)\}$

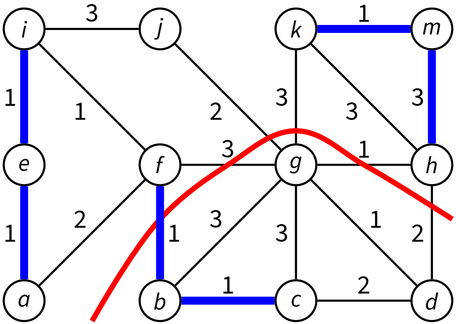
- Let $(S, V - S)$ be a cut of G that respects A
- A minimum-weight edge e that crosses $(S, V - S)$ is a safe edge



- $A = \{(a, e), (e, i), (b, c), (k, m), (m, h)\}$
- Choosing the cut $(\{a, e, f, h, i, j, k, m\}, \{b, c, d, g\})$

Example

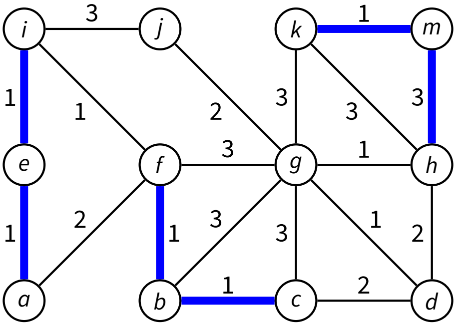
- Let $(S, V - S)$ be a cut of G that respects A
- A minimum-weight edge e that crosses $(S, V - S)$ is a safe edge



- $A = \{(a, e), (e, i), (b, c), (k, m), (m, h)\}$
- Choosing the cut $(\{a, e, f, h, i, j, k, m\}, \{b, c, d, g\})$
- (f, b) is a safe edge

Example

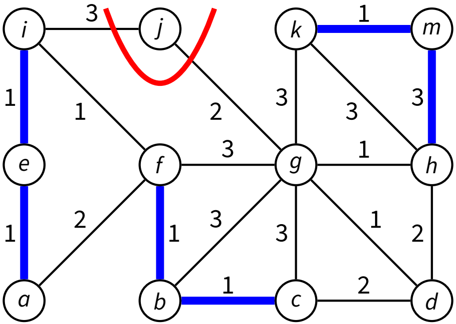
- Let $(S, V - S)$ be a cut of G that respects A
- A minimum-weight edge e that crosses $(S, V - S)$ is a safe edge



- $A = \{(a, e), (e, i), (b, c), (k, m), (m, h)\}$
- Choosing the cut $(\{a, e, f, h, i, j, k, m\}, \{b, c, d, g\})$
- (f, b) is a safe edge; (g, h) is a safe edge, too

Example

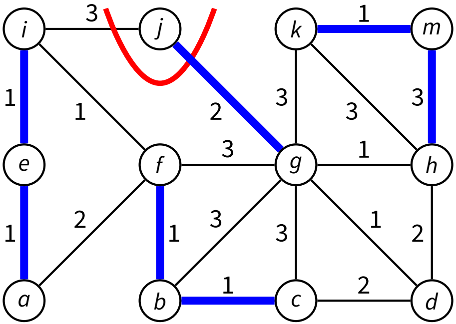
- Let $(S, V - S)$ be a cut of G that respects A
- A minimum-weight edge e that crosses $(S, V - S)$ is a safe edge



- $A = \{(a, e), (e, i), (b, c), (k, m), (m, h)\}$
- Choosing the cut $(\{a, e, f, h, i, j, k, m\}, \{b, c, d, g\})$
- (f, b) is a safe edge; (g, h) is a safe edge, too

Example

- Let $(S, V - S)$ be a cut of G that respects A
- A minimum-weight edge e that crosses $(S, V - S)$ is a safe edge



- $A = \{(a, e), (e, i), (b, c), (k, m), (m, h)\}$
- Choosing the cut $(\{a, e, f, h, i, j, k, m\}, \{b, c, d, g\})$
- (f, b) is a safe edge; (g, h) is a safe edge, too

Concrete Algorithms

- Kruskal's algorithm (1956)
  - based on the generic minimum-spanning-tree algorithm
  - incrementally builds a ***forest*** A

- Kruskal's algorithm (1956)
  - based on the generic minimum-spanning-tree algorithm
  - incrementally builds a ***forest*** A
- Prim's algorithm (1957)
  - based on the generic minimum-spanning-tree algorithm
  - incrementally builds a ***single tree*** A

- *Make-Set*(x) creates a set containing x

- *Make-Set*(x) creates a set containing x
- *Find-Set*(x) returns the *representative* of the set containing x
  - $x, y \in S \Rightarrow \textit{Find-Set}(x) = \textit{Find-Set}(y)$
  - $x \in S_1 \land y \in S_2 \land S_1 \neq S_2 \Rightarrow \textit{Find-Set}(x) \neq \textit{Find-Set}(y)$

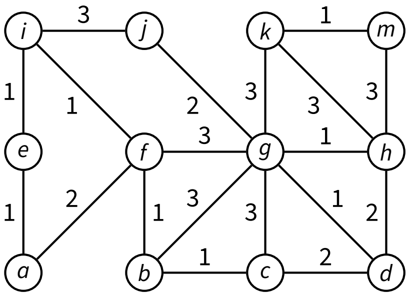
- *Make-Set*(x) creates a set containing x
- *Find-Set*(x) returns the *representative* of the set containing x
  - $x, y \in S \Rightarrow \textit{Find-Set}(x) = \textit{Find-Set}(y)$
  - $x \in S_1 \wedge y \in S_2 \wedge S_1 \neq S_2 \Rightarrow \textit{Find-Set}(x) \neq \textit{Find-Set}(y)$
- *Union*(x, y) joins the sets containing x and y

Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
  A = ∅
  for each vertex v ∈ V(G)
      MAKE-SET(v)                    // disjoint-set data structure
  sort E in non-decreasing order by weight w
  for each edge (u, v) ∈ E, taken in non-decreasing order by w
      if FIND-SET(u) ≠ FIND-SET(v)
          A = A ∪ {(u, v)}
          UNION(u, v)
```

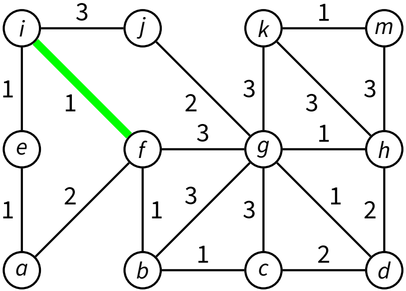
Kruskal’s Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                          // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```



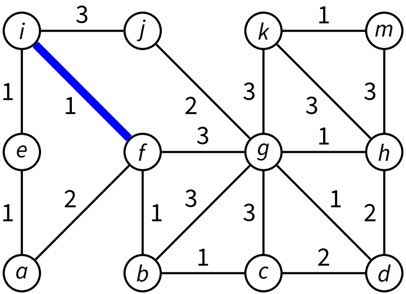
Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                          // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```



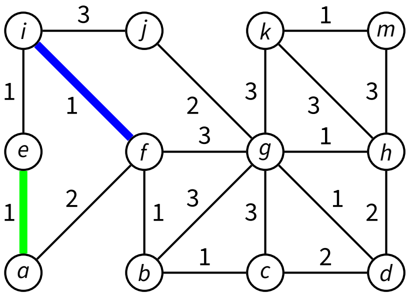
Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                         // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```



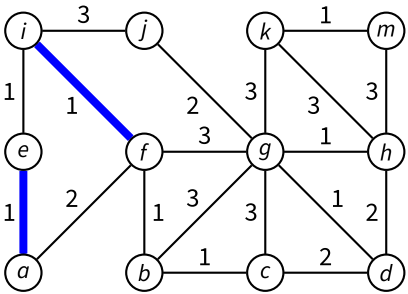
Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                    // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```



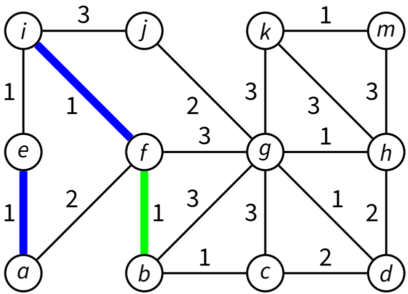
Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                        // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```



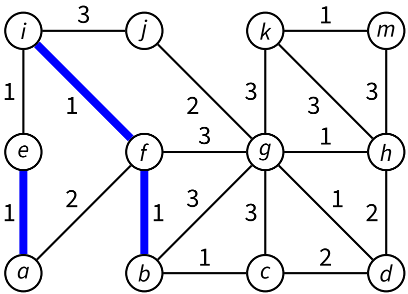
Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                    // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```



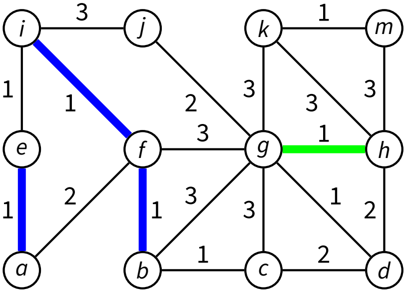
Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                         // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```



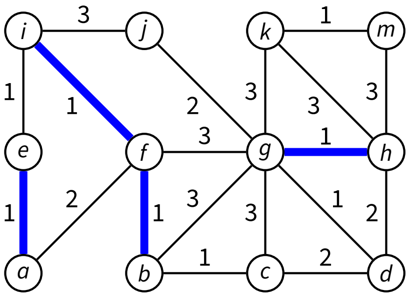
Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
        MAKE-SET(v)                          // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
        if FIND-SET(u) ≠ FIND-SET(v)
                A = A ∪ {(u, v)}
                UNION(u, v)
```

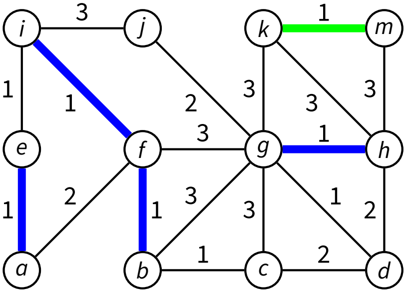


MST-KRUSKAL(G, w)

$A = \emptyset$
for each vertex $v \in V(G)$
    MAKE-SET(v) // *disjoint-set* data structure
sort E in non-decreasing order by weight w
for each edge $(u, v) \in E$, taken in non-decreasing order by w
    if FIND-SET$(u) \neq$ **FIND-SET**(v)
        $A = A \cup \{(u, v)\}$
        UNION(u, v)

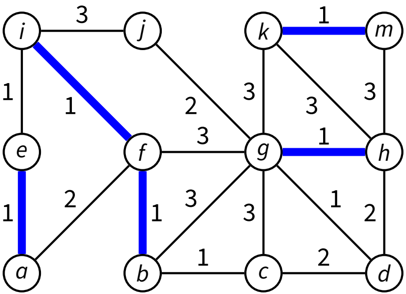


```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                      // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```



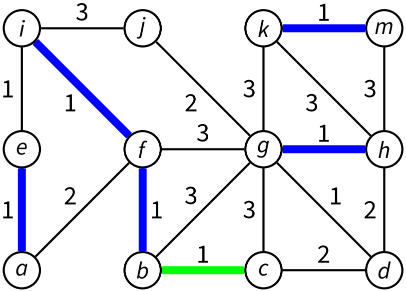
Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                         // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```

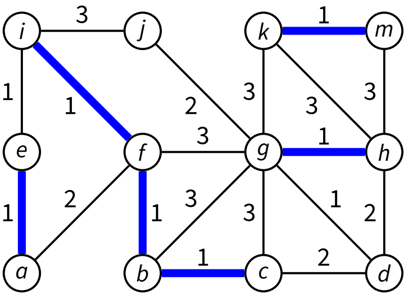


Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                    // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```



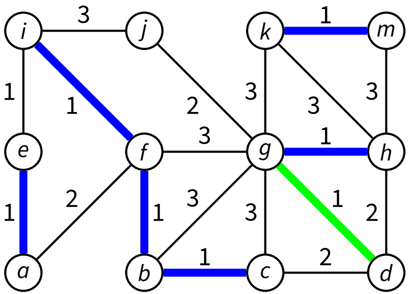
```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
        MAKE-SET(v)                      // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
        if FIND-SET(u) ≠ FIND-SET(v)
                A = A ∪ {(u, v)}
                UNION(u, v)
```



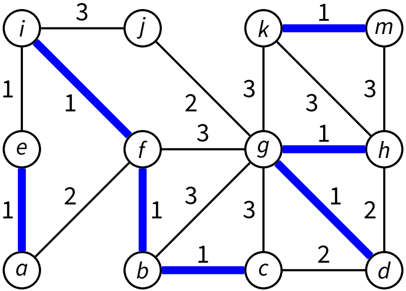
Kruskal's Algorithm

MST-KRUSKAL(G, w)

```
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                      // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```

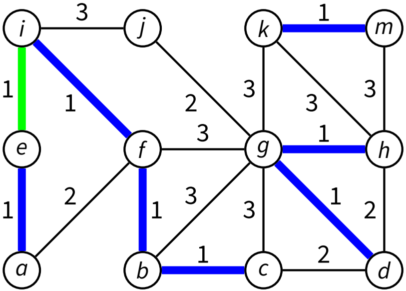


```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                        // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```



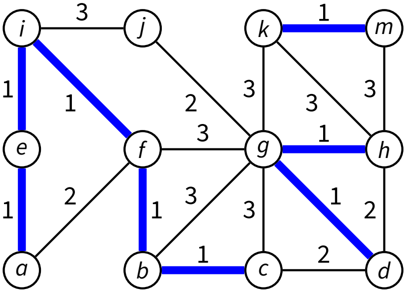
Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                        // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```



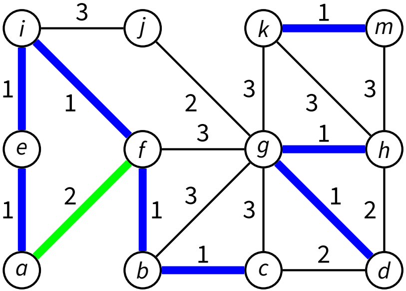
Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                          // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```



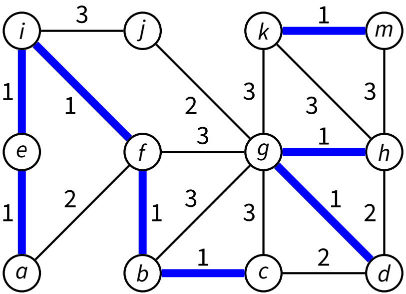
Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                         // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```



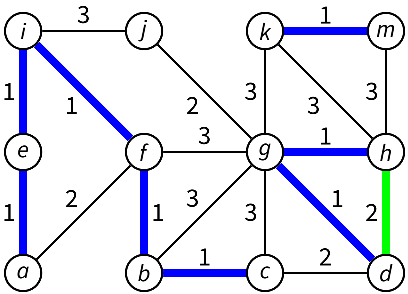
Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                       // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```



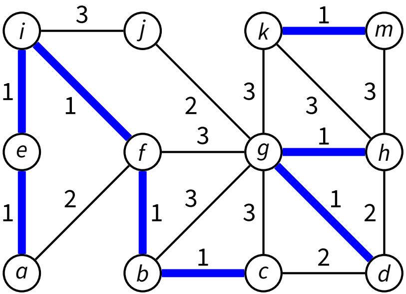
Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                          // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```



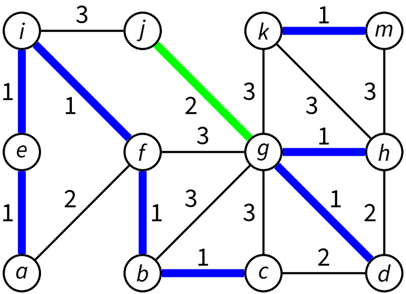
Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                         // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```



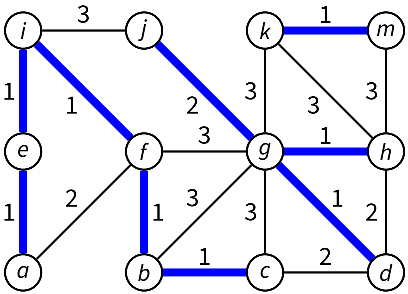
Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                           // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```

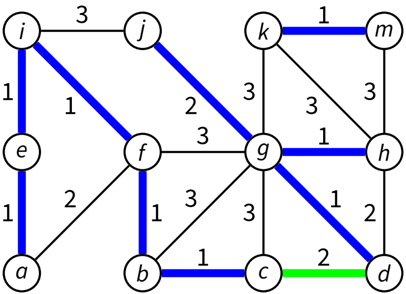


Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                  // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```

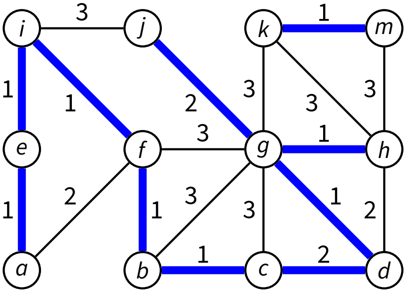


```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                          // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```



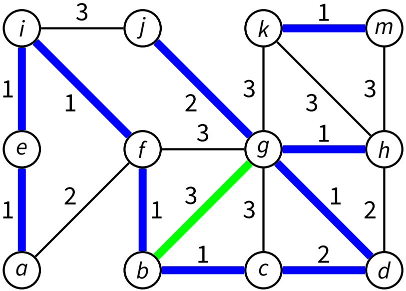
Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
        MAKE-SET(v)                              // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
        if FIND-SET(u) ≠ FIND-SET(v)
                A = A ∪ {(u, v)}
                UNION(u, v)
```



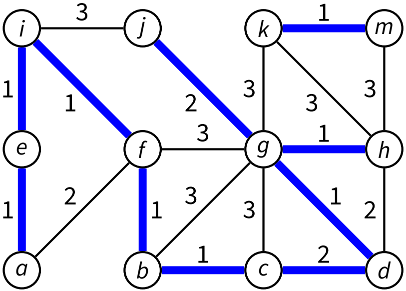
Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                    // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```

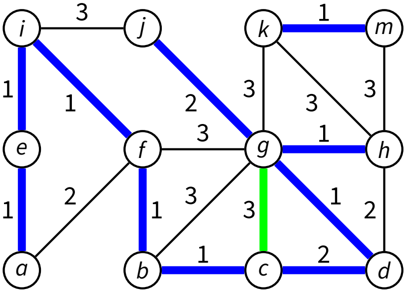


Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
        MAKE-SET(v)                         // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
        if FIND-SET(u) ≠ FIND-SET(v)
                A = A ∪ {(u, v)}
                UNION(u, v)
```

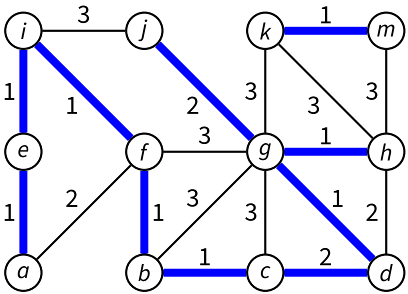


```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
        MAKE-SET(v)                         // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
        if FIND-SET(u) ≠ FIND-SET(v)
                A = A ∪ {(u, v)}
                UNION(u, v)
```



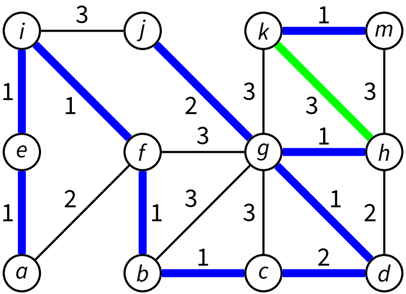
Kruskal’s Algorithm

```
MST-KRUSKAL(G, w)
  A = ∅
  for each vertex v ∈ V(G)
      MAKE-SET(v)                          // disjoint-set data structure
  sort E in non-decreasing order by weight w
  for each edge (u, v) ∈ E, taken in non-decreasing order by w
      if FIND-SET(u) ≠ FIND-SET(v)
          A = A ∪ {(u, v)}
          UNION(u, v)
```



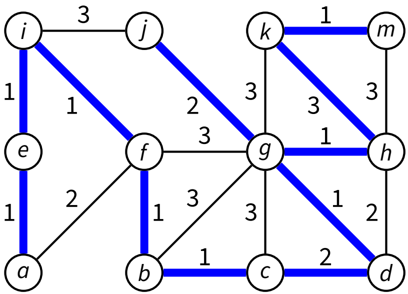
Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                      // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```

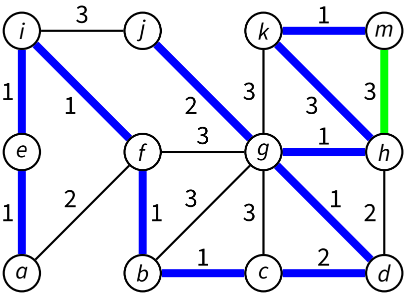


Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                    // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```

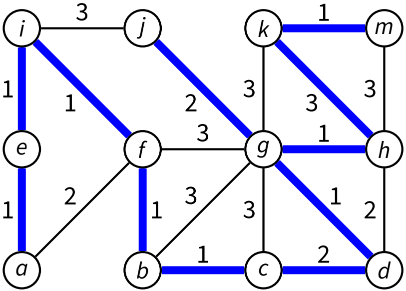


```
MST-KRUSKAL(G, w)
  A = ∅
  for each vertex v ∈ V(G)
      MAKE-SET(v)                    // disjoint-set data structure
  sort E in non-decreasing order by weight w
  for each edge (u, v) ∈ E, taken in non-decreasing order by w
      if FIND-SET(u) ≠ FIND-SET(v)
          A = A ∪ {(u, v)}
          UNION(u, v)
```



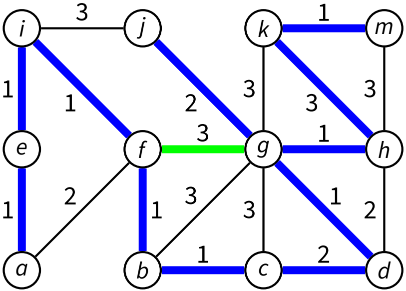
Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                          // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```



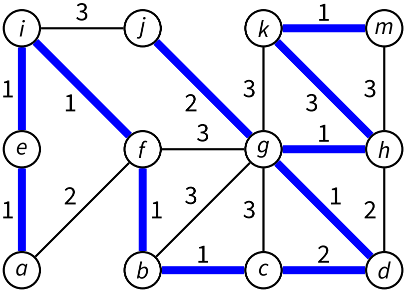
Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                    // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```



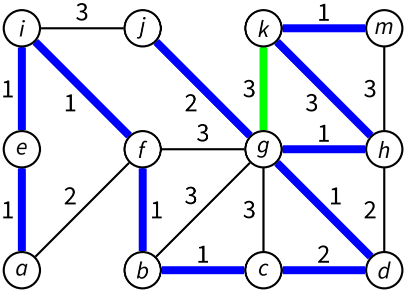
Kruskal’s Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                    // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```



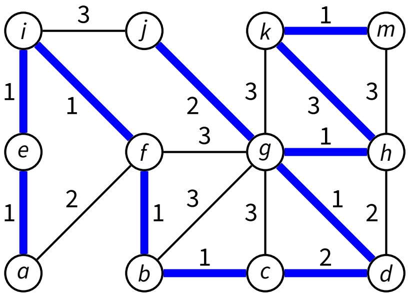
Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                          // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```



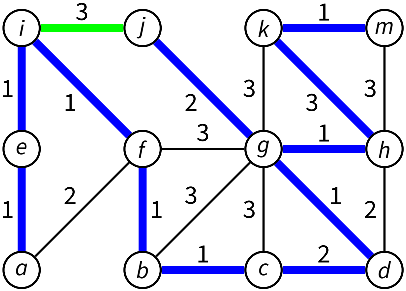
Kruskal’s Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                          // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```



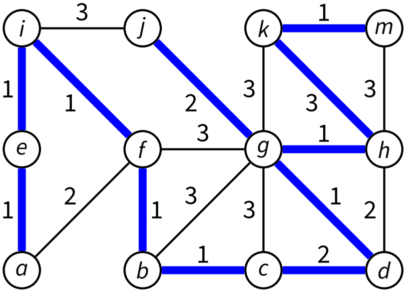
Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
        MAKE-SET(v)                       // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
        if FIND-SET(u) ≠ FIND-SET(v)
                A = A ∪ {(u, v)}
                UNION(u, v)
```



Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
    MAKE-SET(v)                    // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
```



Complexity of MST-KRUSKAL

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ V(G)
        MAKE-SET(v)                          // disjoint-set data structure
sort E in non-decreasing order by weight w
for each edge (u, v) ∈ E, taken in non-decreasing order by w
        if FIND-SET(u) ≠ FIND-SET(v)
                A = A ∪ {(u, v)}
                UNION(u, v)
```

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ V(G)
3      MAKE-SET(v)                      // disjoint-set data structure
4  sort E in non-decreasing order by weight w
5  for each edge (u, v) ∈ E, taken in non-decreasing order by w
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
```

- $|V|$ times **MAKE-SET** (loop of line 2–3)

Complexity of MST-KRUSKAL

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ V(G)
3      MAKE-SET(v)                        // disjoint-set data structure
4  sort E in non-decreasing order by weight w
5  for each edge (u, v) ∈ E, taken in non-decreasing order by w
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
```

- $|V|$ times **MAKE-SET** (loop of line 2–3)
- $O(|E| \log |E|)$ for sorting E (line 4)

Complexity of MST-Kruskal

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ V(G)
3      MAKE-SET(v)                    // disjoint-set data structure
4  sort E in non-decreasing order by weight w
5  for each edge (u, v) ∈ E, taken in non-decreasing order by w
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
```

- $|V|$ times **Make-Set** (loop of line 2–3)
- $O(|E| \log |E|)$ for sorting E (line 4)
- $2|E|$ times **Find-Set**

Complexity of MST-Kruskal

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ V(G)
3      MAKE-SET(v)                          // disjoint-set data structure
4  sort E in non-decreasing order by weight w
5  for each edge (u, v) ∈ E, taken in non-decreasing order by w
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
```

- $|V|$ times **Make-Set** (loop of line 2–3)
- $O(|E| \log |E|)$ for sorting E (line 4)
- $2|E|$ times **Find-Set**
- $O(|E|)$ times **Union**

- Build builds *T* incrementally
  - in each iteration, add a node *v* to *T* through an edge that connecs *v* with *T*

- Build builds *T* incrementally
  - in each iteration, add a node *v* to *T* through an edge that connecs *v* with *T*
- Variables storing values known at each iteration

- Build builds *T* incrementally
  - in each iteration, add a node *v* to *T* through an edge that connecs *v* with *T*
- Variables storing values known at each iteration
  - *T*, tree being built

- Build builds T incrementally
  - in each iteration, add a node v to T through an edge that connecs v with T
- Variables storing values known at each iteration
  - T, tree being built
  - $W[v]$ best known cost/weight of connecting v with T

Prim's Algorithm

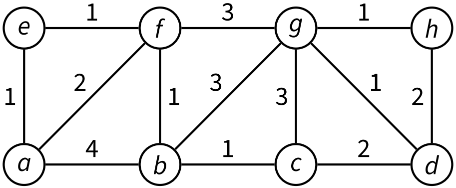
- Build builds T incrementally
  - in each iteration, add a node v to T through an edge that connecs v with T
- Variables storing values known at each iteration
  - T, tree being built
  - $W[v]$ best known cost/weight of connecting v with T
  - $P[v]$, node $u \in T$ such that the edge (u, v) is the least-cost edge connecting v with T

Prim's Algorithm

```
MST-PRIM(G, u, w)
T = (∅, ∅)
for each vertex v ∈ V(G)
    W[v] = ∞
    P[v] = NIL
W[u] = 0
while V(T) ≠ V(G)
    find u ∉ V(T) such that W[u] is minimal
    T = T ∪ {u} // add u to T
    for all v ∈ Adj(u) \ V(T)
        if w(u, v) < W[v]
            W[v] = w(u, v)
            P[v] = u
```

Prim's Algorithm

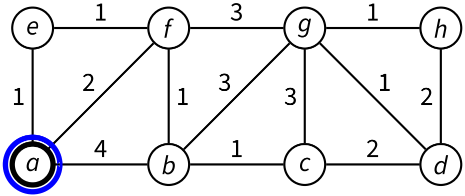
```
MST-PRIM(G, u, w)
T = (∅, ∅)
for each vertex v ∈ V(G)
    W[v] = ∞
    P[v] = NIL
W[u] = 0
while V(T) ≠ V(G)
    find u ∉ V(T) such that W[u] is minimal
    T = T ∪ {u} // add u to T
    for all v ∈ Adj(u) \ V(T)
        if w(u, v) < W[v]
            W[v] = w(u, v)
            P[v] = u
```



v	W	P	T
a	0		
b	∞		
c	∞		
d	∞		
e	∞		
f	∞		
g	∞		
h	∞		

Prim's Algorithm

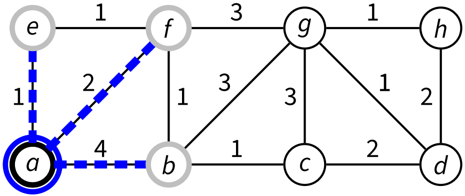
```
MST-PRIM(G, u, w)
T = (∅, ∅)
for each vertex v ∈ V(G)
    W[v] = ∞
    P[v] = NIL
W[u] = 0
while V(T) ≠ V(G)
    find u ∉ V(T) such that W[u] is minimal
    T = T ∪ {u} // add u to T
    for all v ∈ Adj(u) \ V(T)
        if w(u, v) < W[v]
            W[v] = w(u, v)
            P[v] = u
```



v	W	P	T
a	0		✓
b	∞		
c	∞		
d	∞		
e	∞		
f	∞		
g	∞		
h	∞		

Prim's Algorithm

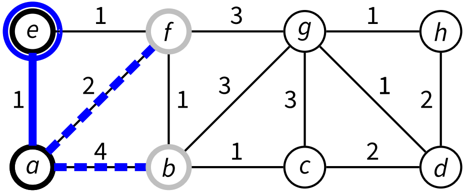
```
MST-PRIM(G, u, w)
T = (∅, ∅)
for each vertex v ∈ V(G)
        W[v] = ∞
        P[v] = NIL
W[u] = 0
while V(T) ≠ V(G)
        find u ∉ V(T) such that W[u] is minimal
        T = T ∪ {u} // add u to T
        for all v ∈ Adj(u) \ V(T)
                if w(u, v) < W[v]
                        W[v] = w(u, v)
                        P[v] = u
```



v	W	P	T
a	0		✓
b	4	a	
c	∞		
d	∞		
e	1	a	
f	2	a	
g	∞		
h	∞		

Prim's Algorithm

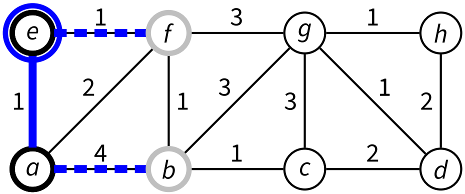
```
MST-PRIM(G, u, w)
T = (∅, ∅)
for each vertex v ∈ V(G)
        W[v] = ∞
        P[v] = NIL
W[u] = 0
while V(T) ≠ V(G)
        find u ∉ V(T) such that W[u] is minimal
        T = T ∪ {u} // add u to T
        for all v ∈ Adj(u) \ V(T)
                if w(u, v) < W[v]
                        W[v] = w(u, v)
                        P[v] = u
```



v	W	P	T
a	0		✓
b	4	a	
c	∞		
d	∞		
e	1	a	✓
f	2	a	
g	∞		
h	∞		

Prim's Algorithm

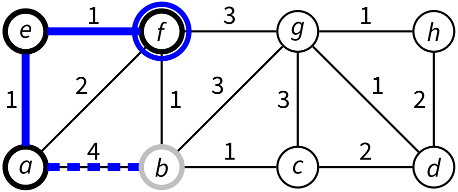
```
MST-PRIM(G, u, w)
T = (∅, ∅)
for each vertex v ∈ V(G)
    W[v] = ∞
    P[v] = NIL
W[u] = 0
while V(T) ≠ V(G)
    find u ∉ V(T) such that W[u] is minimal
    T = T ∪ {u} // add u to T
    for all v ∈ Adj(u) \ V(T)
        if w(u, v) < W[v]
            W[v] = w(u, v)
            P[v] = u
```



v	W	P	T
a	0		✓
b	4	a	
c	∞		
d	∞		
e	1	a	✓
f	1	e	
g	∞		
h	∞		

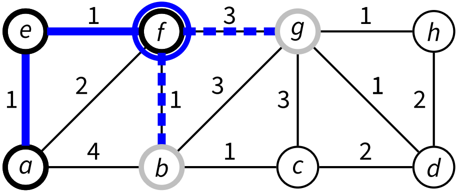
Prim's Algorithm

```
MST-PRIM(G, u, w)
T = (∅, ∅)
for each vertex v ∈ V(G)
    W[v] = ∞
    P[v] = NIL
W[u] = 0
while V(T) ≠ V(G)
    find u ∉ V(T) such that W[u] is minimal
    T = T ∪ {u} // add u to T
    for all v ∈ Adj(u) \ V(T)
        if w(u, v) < W[v]
            W[v] = w(u, v)
            P[v] = u
```



v	W	P	T
a	0		✓
b	4	a	
c	∞		
d	∞		
e	1	a	✓
f	1	e	✓
g	∞		
h	∞		

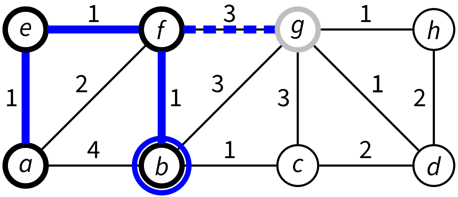
```
MST-PRIM(G, u, w)
T = (∅, ∅)
for each vertex v ∈ V(G)
        W[v] = ∞
        P[v] = NIL
W[u] = 0
while V(T) ≠ V(G)
        find u ∉ V(T) such that W[u] is minimal
        T = T ∪ {u} // add u to T
        for all v ∈ Adj(u) \ V(T)
                if w(u, v) < W[v]
                        W[v] = w(u, v)
                        P[v] = u
```



v	W	P	T
a	0		✓
b	1	f	
c	∞		
d	∞		
e	1	a	✓
f	1	e	✓
g	3	f	
h	∞		

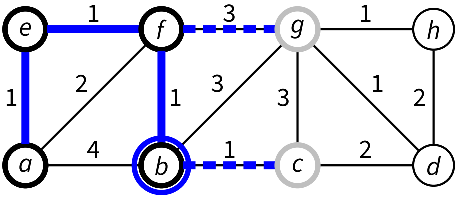
Prim's Algorithm

```
MST-PRIM(G, u, w)
T = (∅, ∅)
for each vertex v ∈ V(G)
        W[v] = ∞
        P[v] = NIL
W[u] = 0
while V(T) ≠ V(G)
        find u ∉ V(T) such that W[u] is minimal
        T = T ∪ {u} // add u to T
        for all v ∈ Adj(u) \ V(T)
                if w(u, v) < W[v]
                        W[v] = w(u, v)
                        P[v] = u
```



v	W	P	T
a	0		✓
b	1	f	✓
c	∞		
d	∞		
e	1	a	✓
f	1	e	✓
g	3	f	
h	∞		

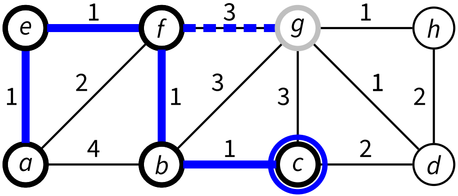
```
MST-PRIM(G, u, w)
T = (∅, ∅)
for each vertex v ∈ V(G)
    W[v] = ∞
    P[v] = NIL
W[u] = 0
while V(T) ≠ V(G)
    find u ∉ V(T) such that W[u] is minimal
    T = T ∪ {u} // add u to T
    for all v ∈ Adj(u) \ V(T)
        if w(u, v) < W[v]
            W[v] = w(u, v)
            P[v] = u
```



v	W	P	T
a	0		✓
b	1	f	✓
c	1	b	
d	∞		
e	1	a	✓
f	1	e	✓
g	3	f	
h	∞		

Prim's Algorithm

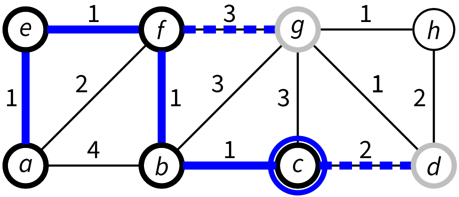
```
MST-PRIM(G, u, w)
T = (∅, ∅)
for each vertex v ∈ V(G)
    W[v] = ∞
    P[v] = NIL
W[u] = 0
while V(T) ≠ V(G)
    find u ∉ V(T) such that W[u] is minimal
    T = T ∪ {u} // add u to T
    for all v ∈ Adj(u) \ V(T)
        if w(u, v) < W[v]
            W[v] = w(u, v)
            P[v] = u
```



v	W	P	T
a	0		✓
b	1	f	✓
c	1	b	✓
d	∞		
e	1	a	✓
f	1	e	✓
g	3	f	
h	∞		

Prim's Algorithm

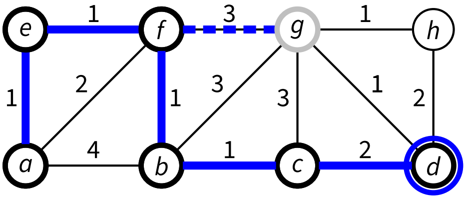
```
MST-PRIM(G, u, w)
T = (∅, ∅)
for each vertex v ∈ V(G)
    W[v] = ∞
    P[v] = NIL
W[u] = 0
while V(T) ≠ V(G)
    find u ∉ V(T) such that W[u] is minimal
    T = T ∪ {u} // add u to T
    for all v ∈ Adj(u) \ V(T)
        if w(u, v) < W[v]
            W[v] = w(u, v)
            P[v] = u
```



v	W	P	T
a	0		✓
b	1	f	✓
c	1	b	✓
d	2	c	
e	1	a	✓
f	1	e	✓
g	3	f	
h	∞		

Prim's Algorithm

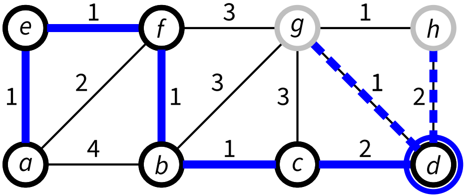
```
MST-PRIM(G, u, w)
T = (∅, ∅)
for each vertex v ∈ V(G)
    W[v] = ∞
    P[v] = NIL
W[u] = 0
while V(T) ≠ V(G)
    find u ∉ V(T) such that W[u] is minimal
    T = T ∪ {u} // add u to T
    for all v ∈ Adj(u) \ V(T)
        if w(u, v) < W[v]
            W[v] = w(u, v)
            P[v] = u
```



v	W	P	T
a	0		✓
b	1	f	✓
c	1	b	✓
d	2	c	✓
e	1	a	✓
f	1	e	✓
g	3	f	
h	∞		

Prim's Algorithm

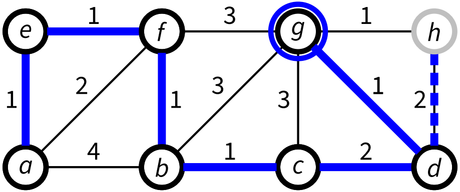
```
MST-PRIM(G, u, w)
T = (∅, ∅)
for each vertex v ∈ V(G)
        W[v] = ∞
        P[v] = NIL
W[u] = 0
while V(T) ≠ V(G)
        find u ∉ V(T) such that W[u] is minimal
        T = T ∪ {u} // add u to T
        for all v ∈ Adj(u) \ V(T)
                if w(u, v) < W[v]
                        W[v] = w(u, v)
                        P[v] = u
```



v	W	P	T
a	0		✓
b	1	f	✓
c	1	b	✓
d	2	c	✓
e	1	a	✓
f	1	e	✓
g	1	d	
h	2	d	

Prim's Algorithm

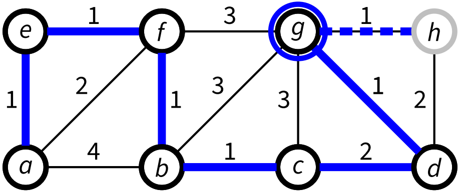
```
MST-PRIM(G, u, w)
T = (∅, ∅)
for each vertex v ∈ V(G)
        W[v] = ∞
        P[v] = NIL
W[u] = 0
while V(T) ≠ V(G)
        find u ∉ V(T) such that W[u] is minimal
        T = T ∪ {u} // add u to T
        for all v ∈ Adj(u) \ V(T)
                if w(u, v) < W[v]
                        W[v] = w(u, v)
                        P[v] = u
```



v	W	P	T
a	0		✓
b	1	f	✓
c	1	b	✓
d	2	c	✓
e	1	a	✓
f	1	e	✓
g	1	d	✓
h	2	d	

MST-PRIM(G, u, w)

$T = (\emptyset, \emptyset)$
for each vertex $v \in V(G)$
    $W[v] = \infty$
    $P[v] = \text{NIL}$
$W[u] = 0$
while $V(T) \neq V(G)$
    find $u \notin V(T)$ such that $W[u]$ is minimal
    $T = T \cup \{u\}$ // add u to T
    for all $v \in Adj(u) \setminus V(T)$
        if $w(u, v) < W[v]$
            $W[v] = w(u, v)$
            $P[v] = u$

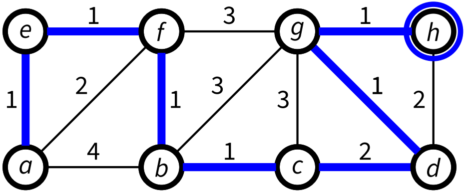


v	W	P	T
a	0		✓
b	1	f	✓
c	1	b	✓
d	2	c	✓
e	1	a	✓
f	1	e	✓
g	1	d	✓
h	1	g	

Prim’s Algorithm

MST-PRIM(G, u, w)

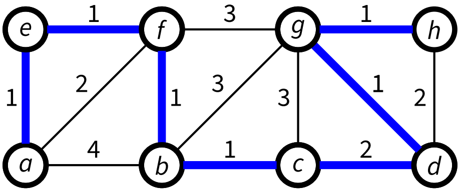
```
T = (∅, ∅)
for each vertex v ∈ V(G)
        W[v] = ∞
        P[v] = NIL
W[u] = 0
while V(T) ≠ V(G)
        find u ∉ V(T) such that W[u] is minimal
        T = T ∪ {u} // add u to T
        for all v ∈ Adj(u) \ V(T)
                if w(u, v) < W[v]
                        W[v] = w(u, v)
                        P[v] = u
```



v	W	P	T
a	0		✓
b	1	f	✓
c	1	b	✓
d	2	c	✓
e	1	a	✓
f	1	e	✓
g	1	d	✓
h	1	g	✓

Prim's Algorithm

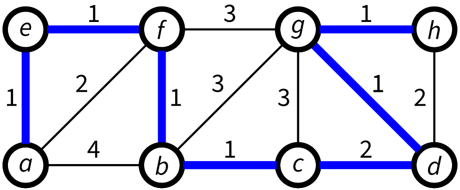
```
MST-PRIM(G, u, w)
T = (∅, ∅)
for each vertex v ∈ V(G)
    W[v] = ∞
    P[v] = NIL
W[u] = 0
while V(T) ≠ V(G)
    find u ∉ V(T) such that W[u] is minimal
    T = T ∪ {u} // add u to T
    for all v ∈ Adj(u) \ V(T)
        if w(u, v) < W[v]
            W[v] = w(u, v)
            P[v] = u
```



v	W	P	T
a	0		✓
b	1	f	✓
c	1	b	✓
d	2	c	✓
e	1	a	✓
f	1	e	✓
g	1	d	✓
h	1	g	✓

Prim's Algorithm

```
MST-PRIM(G, u, w)
T = (∅, ∅)
for each vertex v ∈ V(G)
    W[v] = ∞
    P[v] = NIL
W[u] = 0
while V(T) ≠ V(G)
    find u ∉ V(T) such that W[u] is minimal
    T = T ∪ {u} // add u to T
    for all v ∈ Adj(u) \ V(T)
        if w(u, v) < W[v]
            W[v] = w(u, v)
            P[v] = u
```



v	W	P	T
a	0		✓
b	1	f	✓
c	1	b	✓
d	2	c	✓
e	1	a	✓
f	1	e	✓
g	1	d	✓
h	1	g	✓