Algorithms and Data Structures (II)

Gabriel Istrate

June 2, 2025

Minimum Spanning Tree

- 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***

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***

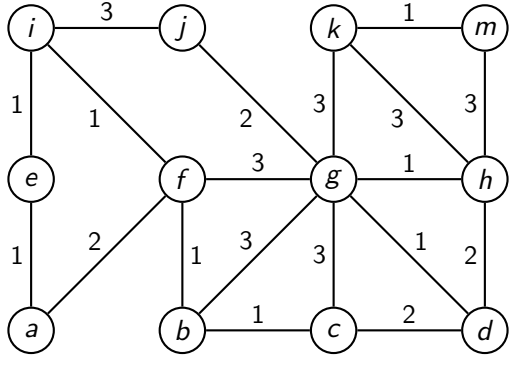
Minimum Spanning Tree

- 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

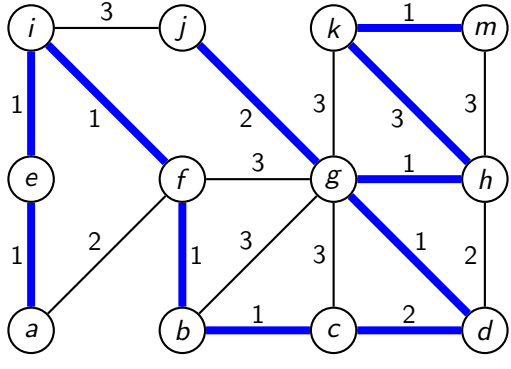
$$w(T) = \sum_{(u,v) \in T} w(u, v)$$

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

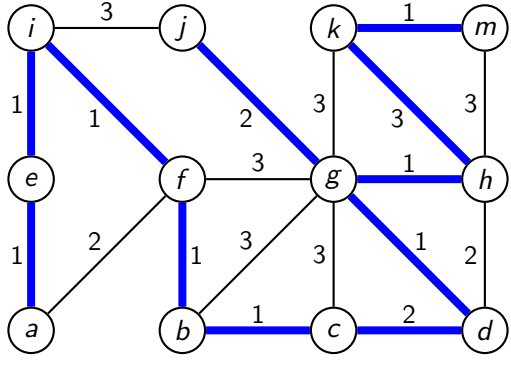
Example



Example

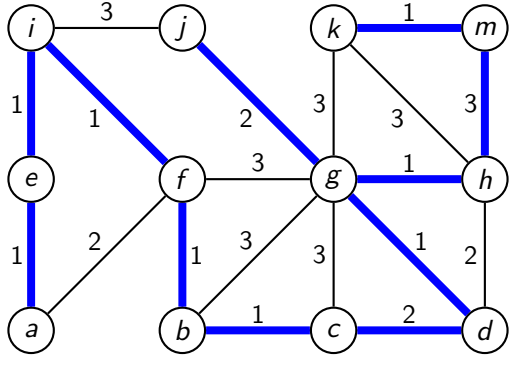


Example



- 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

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*

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

Refined Idea

```
Generic-MST(G, w)1   A = ∅
                 2   while A is not a spanning tree
                 3       find a safe edge e = (u, v)
                 4       A = A ∪ {e}
```

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

Refined Idea

```
Generic-MST(G, w)1  A = ∅
                 2  while A is not a spanning tree
                 3      find a safe edge e = (u, v)
                 4      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

Refined Idea

```
Generic-MST(G, w)1  A = ∅
                 2  while A is not a spanning tree
                 3      find a safe edge e = (u, v)
                 4      A = A ∪ {e}
```

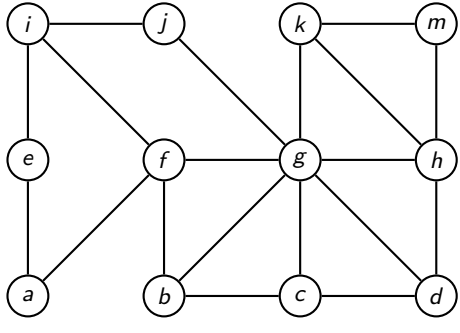
- ***Invariant:*** A is a subset of a minimum spanning tree
- A ***safe edge*** is an edge that maintains the invariant
  - e is such that, if A is a subset of a minimum spanning tree, then $A \cup \{e\}$ is also a subset of a minimum spanning tree
  - more or less the *definition* of a greedy algorithm

Preliminary Definitions

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

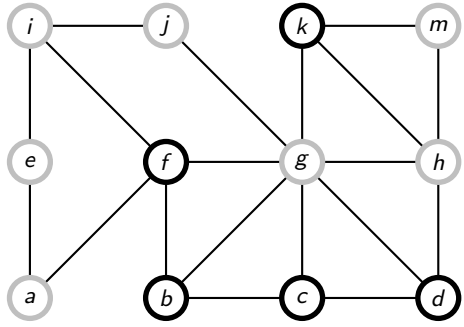
Preliminary Definitions

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



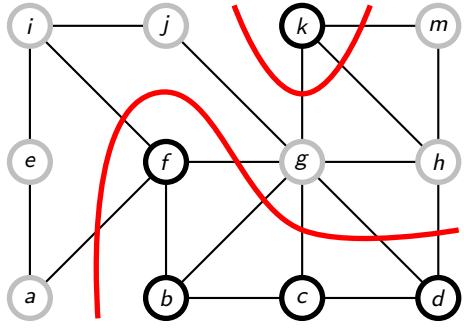
Preliminary Definitions

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



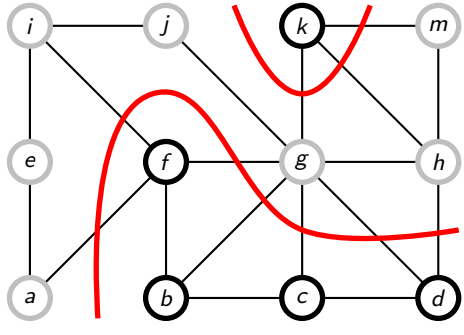
Preliminary Definitions

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



Preliminary Definitions

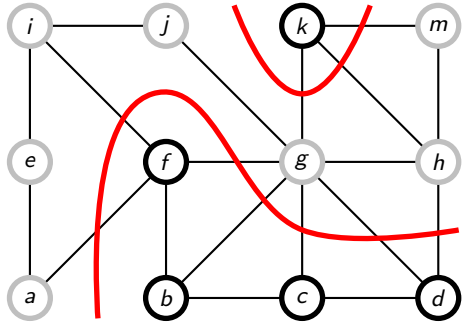
- A ***cut*** $(S, V - S)$ of an undirected graph $G = (V, E)$ is a *partition* of V



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

Preliminary Definitions

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

Finding a Safe Edge

```
Generic-MST(G, w)1  A = ∅
                2  while A is not a spanning tree
                3      find a safe edge e = (u, v)
                4      A = A ∪ {e}
```

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

Finding a Safe Edge

```
Generic-MST(G, w)1  A = ∅
                 2  while A is not a spanning tree
                 3      find a safe edge e = (u, v)
                 4      A = A ∪ {e}
```

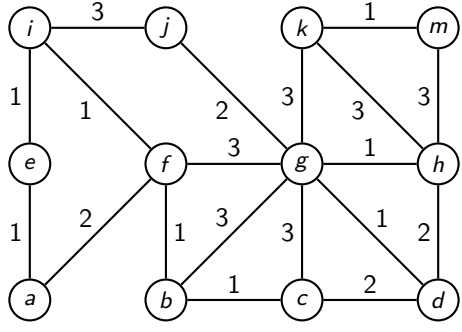
- Let $(S, V - S)$ be a cut of G that respects A
- A minimum-weight edge e that crosses $(S, V - S)$ is a safe edge

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

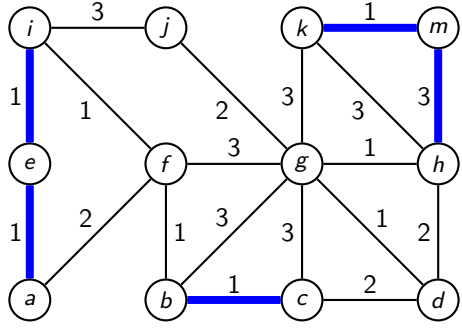
Example

- 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



Example

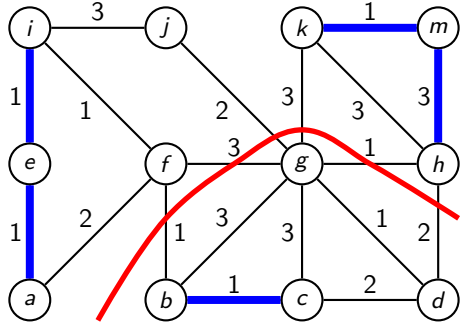
- Let $(S, V - S)$ be a cut of G that respects A
- A minimum-weight edge e that crosses $(S, V - S)$ is a safe edge



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

Example

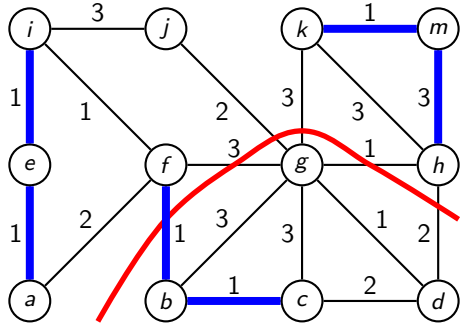
- Let $(S, V - S)$ be a cut of G that respects A
- A minimum-weight edge e that crosses $(S, V - S)$ is a safe edge



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

Example

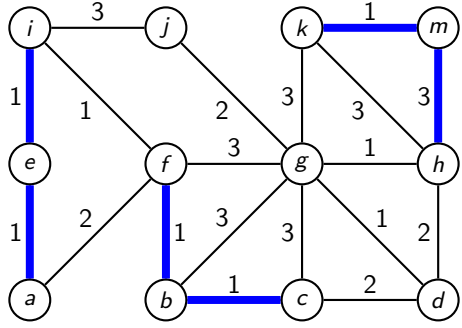
- Let $(S, V - S)$ be a cut of G that respects A
- A minimum-weight edge e that crosses $(S, V - S)$ is a safe edge



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

Example

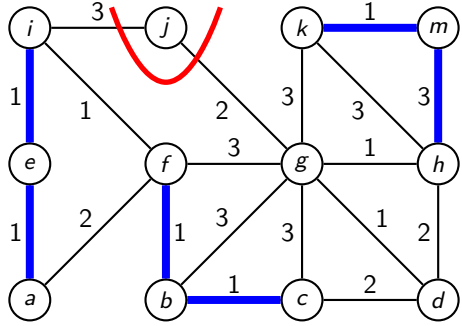
- Let $(S, V - S)$ be a cut of G that respects A
- A minimum-weight edge e that crosses $(S, V - S)$ is a safe edge



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

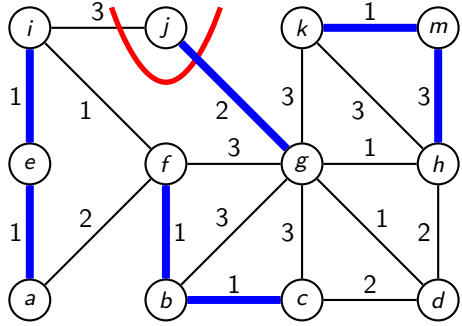
- Let $(S, V - S)$ be a cut of G that respects A
- A minimum-weight edge e that crosses $(S, V - S)$ is a safe edge



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

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

Concrete Algorithms

- 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

Disjoint-Set Data Structure

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

Disjoint-Set Data Structure

- *Make-Set*(x) creates a set containing x
- *Find-Set*(x) returns the *representative* of the set containing x
  - $x, y \in S \Longrightarrow$ *Find-Set*$(x) =$ *Find-Set*(y)
  - $x \in S_1 \wedge y \in S_2 \wedge S_1 \neq S_2 \Longrightarrow$ *Find-Set*$(x) \neq$ *Find-Set*(y)

Disjoint-Set Data Structure

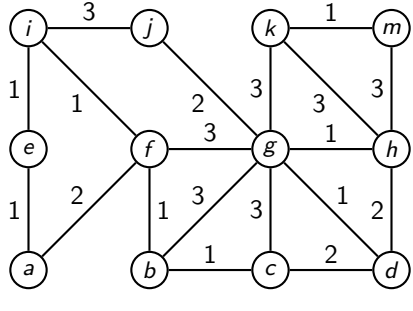
- *Make-Set*(x) creates a set containing x
- *Find-Set*(x) returns the *representative* of the set containing x
  - $x, y \in S \Longrightarrow$ *Find-Set*$(x) =$ *Find-Set*(y)
  - $x \in S_1 \wedge y \in S_2 \wedge S_1 \neq S_2 \Longrightarrow$ *Find-Set*$(x) \neq$ *Find-Set*(y)
- *Union*(x, y) joins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

Kruskal's Algorithm

```
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)
```

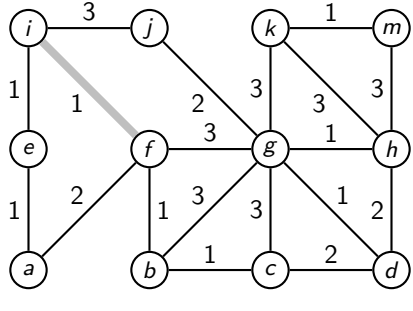
Kruskal's Algorithm

```
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)
```



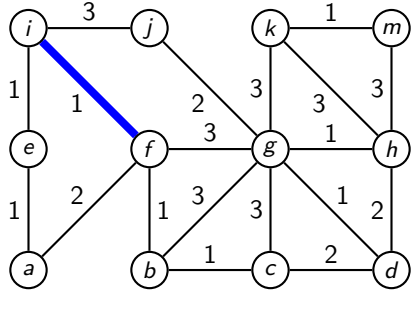
Kruskal's Algorithm

```
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)
```



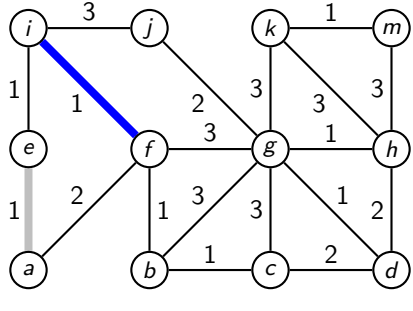
Kruskal's Algorithm

```
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)
```



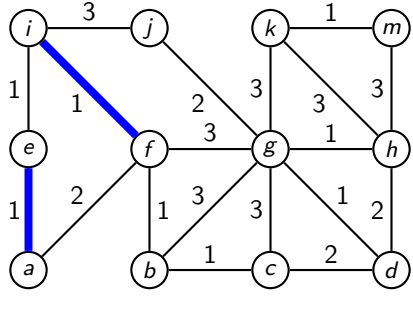
Kruskal's Algorithm

```
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)
```



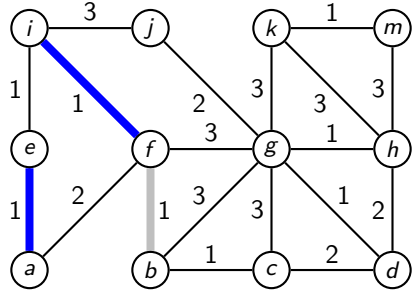
Kruskal's Algorithm

```
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)
```



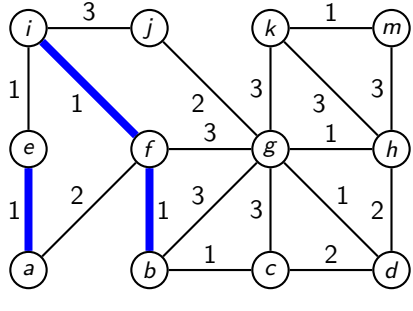
Kruskal's Algorithm

```
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)
```



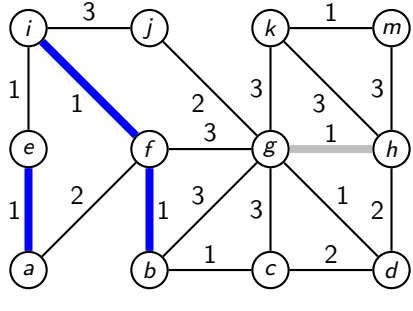
Kruskal's Algorithm

```
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)
```



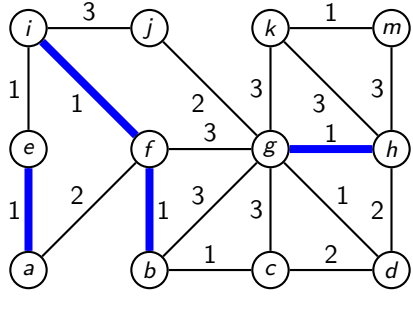
Kruskal's Algorithm

```
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)
```



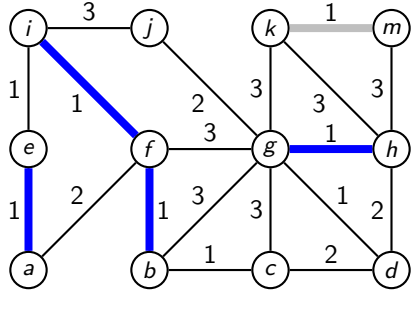
Kruskal's Algorithm

```
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)
```



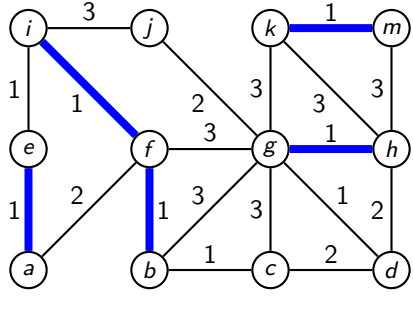
Kruskal's Algorithm

```
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)
```



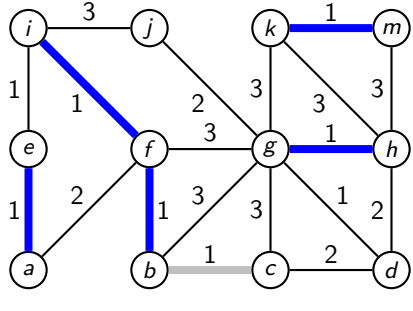
Kruskal's Algorithm

```
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)
```



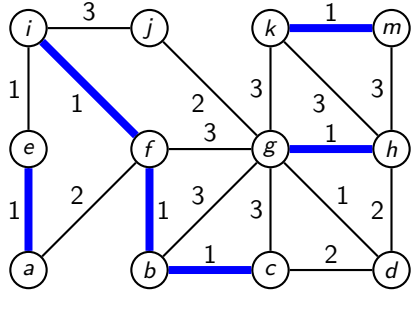
Kruskal's Algorithm

```
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)
```



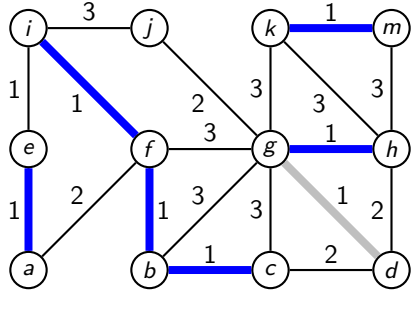
Kruskal's Algorithm

```
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)
```



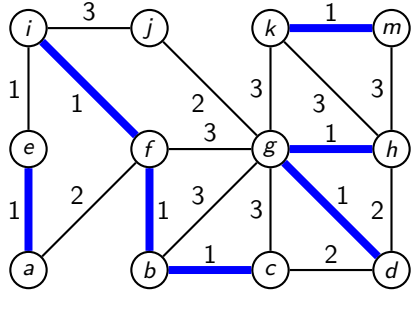
Kruskal's Algorithm

```
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)
```



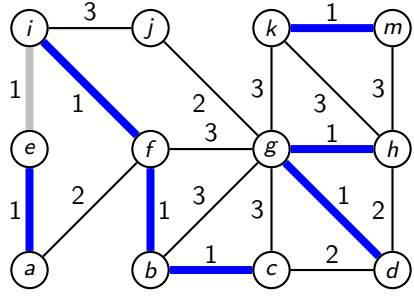
Kruskal's Algorithm

```
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)
```



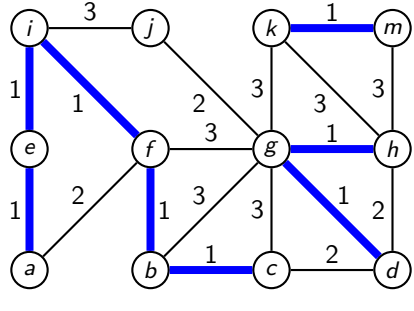
Kruskal's Algorithm

```
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)
```



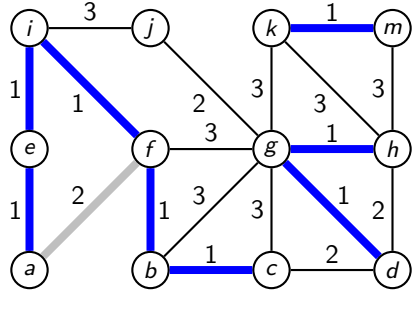
Kruskal's Algorithm

```
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)
```



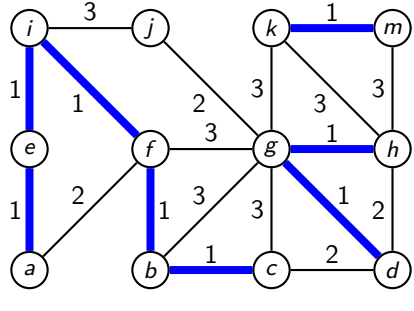
Kruskal's Algorithm

```
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)
```



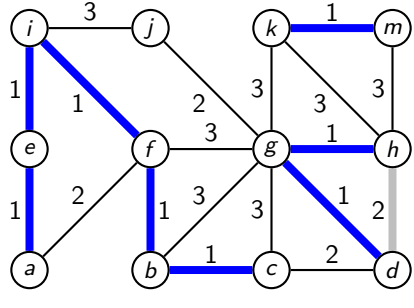
Kruskal's Algorithm

```
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)
```



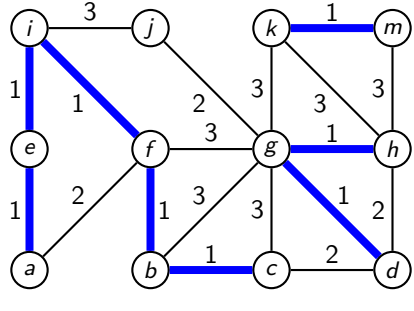
Kruskal's Algorithm

```
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)
```



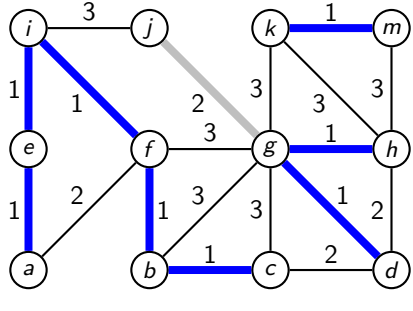
Kruskal's Algorithm

```
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)
```



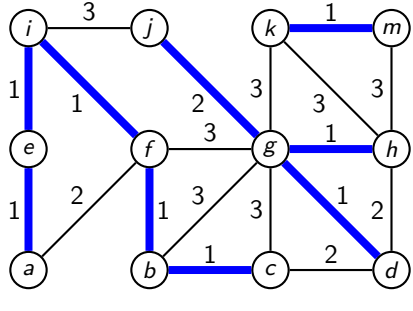
Kruskal's Algorithm

```
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)
```



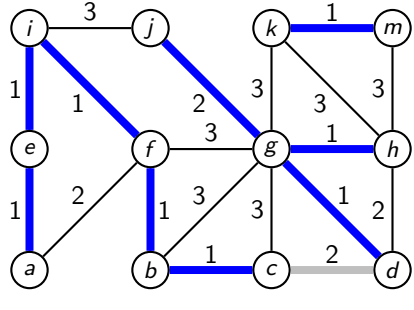
Kruskal's Algorithm

```
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)
```



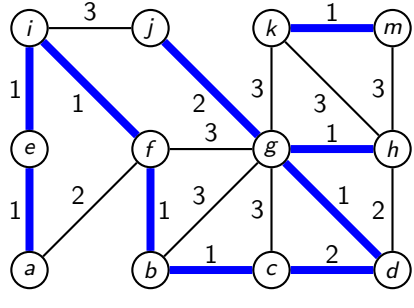
Kruskal's Algorithm

```
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)
```



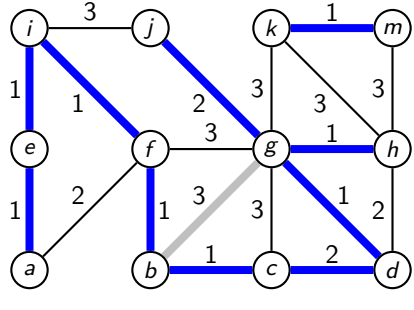
Kruskal's Algorithm

```
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)
```



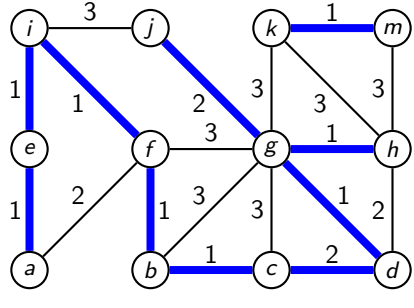
Kruskal's Algorithm

```
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)
```



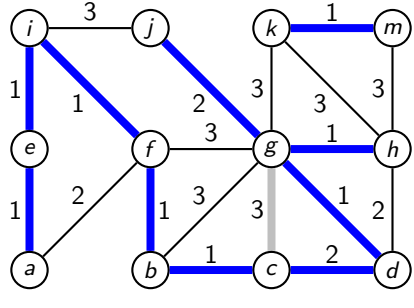
Kruskal's Algorithm

```
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)
```



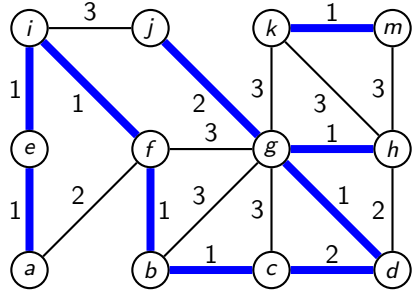
Kruskal's Algorithm

```
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)
```



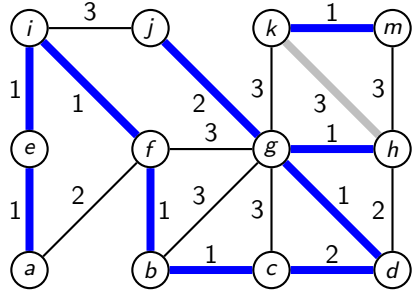
Kruskal's Algorithm

```
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)
```



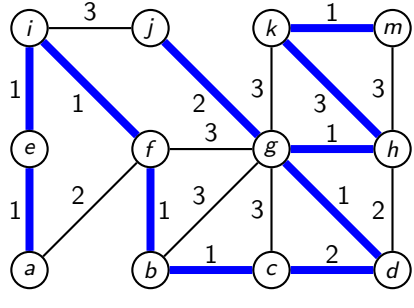
Kruskal's Algorithm

```
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)
```



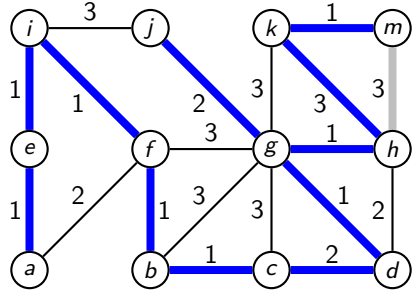
Kruskal's Algorithm

```
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)
```



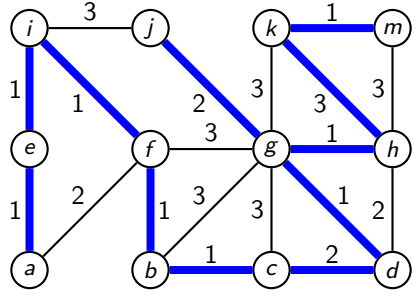
Kruskal's Algorithm

```
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)
```



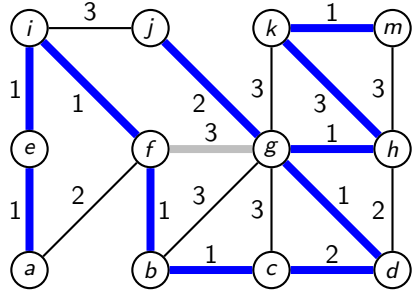
Kruskal's Algorithm

```
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)
```



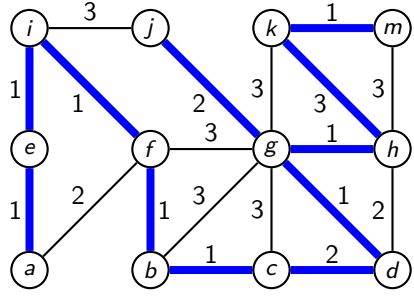
Kruskal's Algorithm

```
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)
```



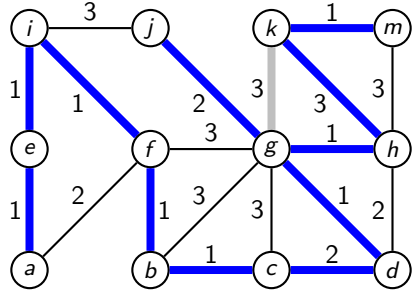
Kruskal's Algorithm

```
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)
```



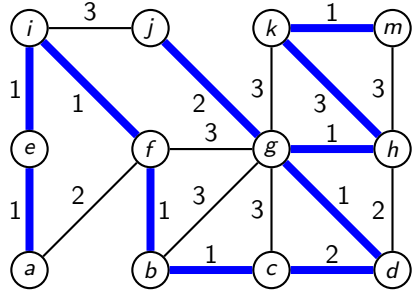
Kruskal's Algorithm

```
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)
```



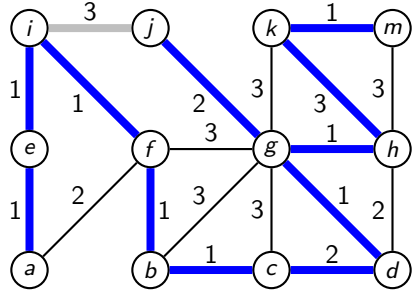
Kruskal's Algorithm

```
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)
```



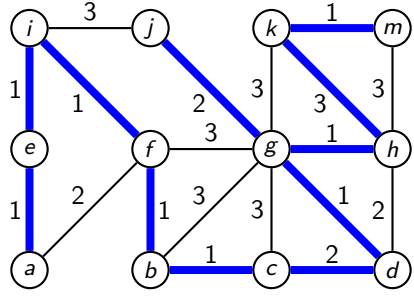
Kruskal's Algorithm

```
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)
```



Kruskal's Algorithm

```
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)
```



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)
```

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)

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)
```

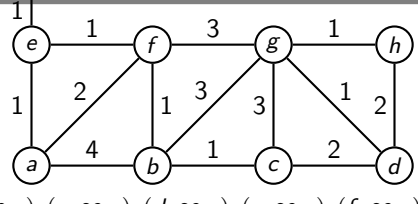
- $|V|$ times **Make-Set** (loop of line 2–3)
- $O(|E| \log |E|)$ for sorting E (line 4)
- $2|E|$ times **Find-Set**
- $O(|E|)$ times **Union**

Prim's Algorithm

```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = NIL
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```

Prim's Algorithm

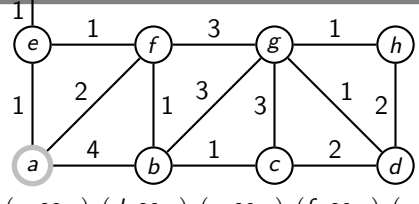
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = NIL
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(a, 0, \cdot), (b, \infty, \cdot), (c, \infty, \cdot), (d, \infty, \cdot), (e, \infty, \cdot), (f, \infty, \cdot), (g, \infty, \cdot), (h, \infty, \cdot)\}$

Prim's Algorithm

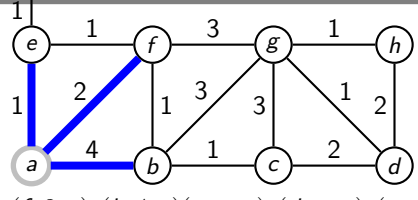
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = NIL
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(b, \infty, \cdot), (c, \infty, \cdot), (d, \infty, \cdot), (e, \infty, \cdot), (f, \infty, \cdot), (g, \infty, \cdot), (h, \infty, \cdot)\}$

Prim's Algorithm

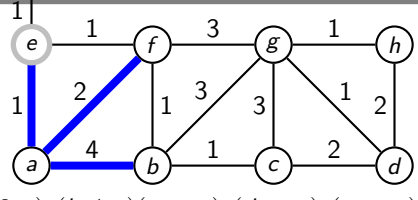
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = NIL
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(e, 1, a), (f, 2, a), (b, 4, a)(c, \infty, \cdot), (d, \infty, \cdot), (g, \infty, \cdot), (h, \infty, \cdot)\}$

Prim's Algorithm

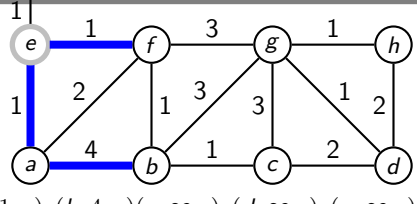
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = NIL
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(f, 2, a), (b, 4, a)(c, \infty, \cdot), (d, \infty, \cdot), (g, \infty, \cdot), (h, \infty, \cdot)\}$

Prim's Algorithm

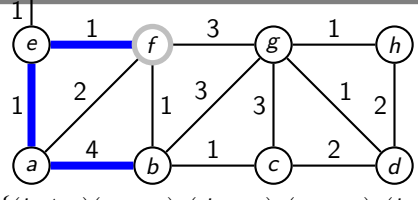
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = NIL
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(f, 1, e), (b, 4, a)(c, \infty, \cdot), (d, \infty, \cdot), (g, \infty, \cdot), (h, \infty, \cdot)\}$

Prim's Algorithm

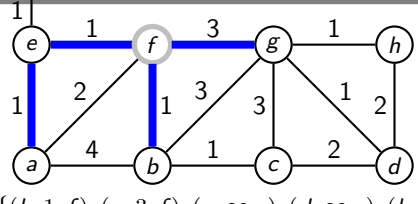
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = NIL
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(b, 4, a)(c, \infty, \cdot), (d, \infty, \cdot), (g, \infty, \cdot), (h, \infty, \cdot)\}$

Prim's Algorithm

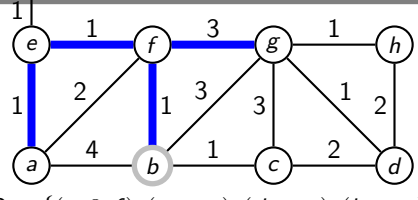
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = NIL
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(b, 1, f), (g, 3, f), (c, \infty, \cdot), (d, \infty, \cdot), (h, \infty, \cdot)\}$

Prim's Algorithm

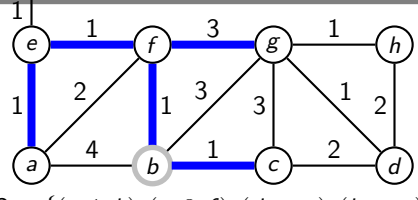
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = NIL
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(g, 3, f), (c, \infty, \cdot), (d, \infty, \cdot), (h, \infty, \cdot)\}$

Prim's Algorithm

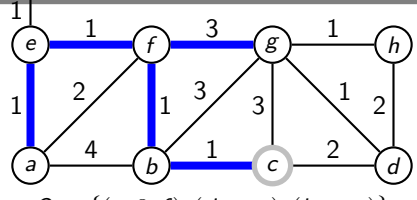
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = NIL
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(c, 1, b), (g, 3, f), (d, \infty, \cdot), (h, \infty, \cdot)\}$

Prim's Algorithm

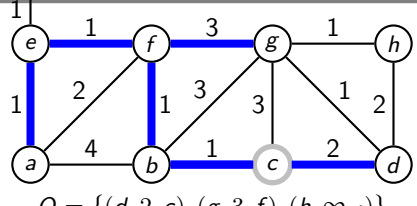
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = NIL
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(g, 3, f), (d, \infty, \cdot), (h, \infty, \cdot)\}$

Prim's Algorithm

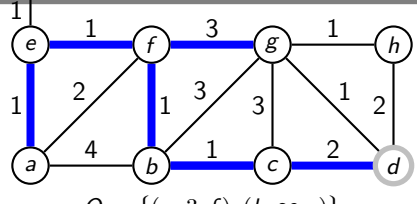
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = NIL
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(d, 2, c), (g, 3, f), (h, \infty, \cdot)\}$

Prim's Algorithm

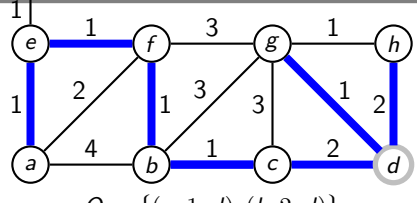
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = NIL
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(g, 3, f), (h, \infty, \cdot)\}$

Prim's Algorithm

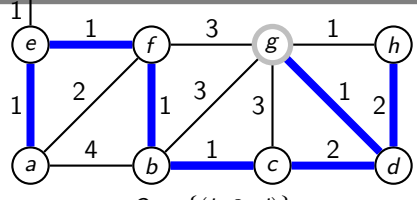
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = NIL
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(g, 1, d), (h, 2, d)\}$

Prim's Algorithm

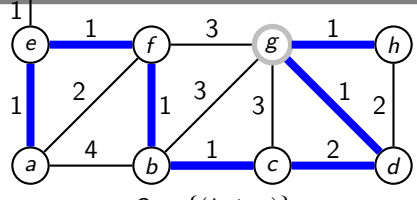
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = NIL
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(h, 2, d)\}$

Prim's Algorithm

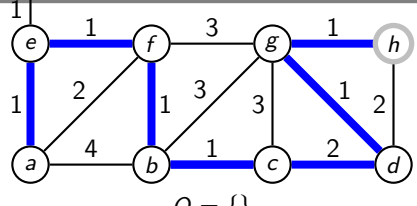
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = NIL
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(h, 1, g)\}$

Prim's Algorithm

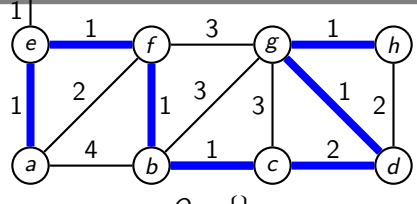
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = NIL
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{\}$

Prim's Algorithm

```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = NIL
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{\}$