MST-Kruskal($G$, $w$)

1: $F \leftarrow \emptyset$
2: $S \leftarrow \{\{v\} : v \in V\}$
3: sort the edges of $E$ in non-decreasing order of weights $w$
4: for each edge $(u, v) \in E$ in the order do
5: \hspace{1cm} $S_u \leftarrow$ the set in $S$ containing $u$
6: \hspace{1cm} $S_v \leftarrow$ the set in $S$ containing $v$
7: \hspace{1cm} if $S_u \neq S_v$ then
8: \hspace{1.5cm} $F \leftarrow F \cup \{(u, v)\}$
9: \hspace{1.5cm} $S \leftarrow S \setminus \{S_u\} \setminus \{S_v\} \cup \{S_u \cup S_v\}$
10: return $(V, F)$
Running Time of Kruskal’s Algorithm

MST-Kruskal\((G, w)\)

1. \(F \leftarrow \emptyset\)
2. \(S \leftarrow \{\{v\} : v \in V\}\)
3. sort the edges of \(E\) in non-decreasing order of weights \(w\)
4. for each edge \((u, v) \in E\) in the order do
5. \(S_u \leftarrow\) the set in \(S\) containing \(u\)
6. \(S_v \leftarrow\) the set in \(S\) containing \(v\)
7. if \(S_u \neq S_v\) then
8. \(F \leftarrow F \cup \{(u, v)\}\)
9. \(S \leftarrow S \setminus \{S_u\} \setminus \{S_v\} \cup \{S_u \cup S_v\}\)
10. return \((V, F)\)

Use union-find data structure to support 2, 5, 6, 7, 9.
Union-Find Data Structure

- $V$: ground set
- We need to maintain a partition of $V$ and support following operations:
  - Check if $u$ and $v$ are in the same set of the partition
  - Merge two sets in partition
\begin{itemize}
  \item $V = \{1, 2, 3, \cdots, 16\}$
  \item Partition: $\{2, 3, 5, 9, 10, 12, 15\}$, $\{1, 7, 13, 16\}$, $\{4, 8, 11\}$, $\{6, 14\}$
  \item $par[i]$: parent of $i$, $(par[i] = \bot$ if $i$ is a root). 
\end{itemize}
Q: how can we check if \( u \) and \( v \) are in the same set?

A: Check if root\((u)\) = root\((v)\).

- root\((u)\): the root of the tree containing \( u \).
- Merge the trees with root \( r \) and \( r_0 \): \( parent[r] = r_0 \).
Q: how can we check if $u$ and $v$ are in the same set?
Q: how can we check if \( u \) and \( v \) are in the same set?

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

$\text{root}(u)$: the root of the tree containing $u$

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

$\text{root}(u)$: the root of the tree containing $u$

Merge the trees with root $r$ and $r'$: $\text{par}[r] \leftarrow r'$. 
Union-Find Data Structure

\[ \text{root}(v) \]

1: if \( \text{par}[v] = \bot \) then
2: return \( v \)
3: else
4: return \( \text{root}(\text{par}[v]) \)
Union-Find Data Structure

\textbf{root}(v)

1: \textbf{if} \ par[v] = \bot \ \textbf{then}
2: \quad \textbf{return} \ v
3: \textbf{else}
4: \quad \textbf{return} \ \text{root}(par[v])

- Problem: the tree might too deep; running time might be large
Union-Find Data Structure

root(v)

1: if par[v] = \bot then
2: return v
3: else
4: return root(par[v])

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

\texttt{root}(v)

1: \textbf{if} \( \text{par}[v] = \perp \) \textbf{then}
2: \hspace{1em} \textbf{return} \hspace{0.5em} v
3: \textbf{else}
4: \hspace{1em} \textbf{return} \hspace{0.5em} \text{root}(\text{par}[v])

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

Improvement: all vertices in the path directly point to the root, saving time in the future.

\texttt{root}(v)

1: \textbf{if} \( \text{par}[v] = \perp \) \textbf{then}
2: \hspace{1em} \textbf{return} \hspace{0.5em} v
3: \textbf{else}
4: \hspace{1em} \text{par}[v] \leftarrow \text{root}(\text{par}[v])
5: \hspace{1em} \textbf{return} \hspace{0.5em} \text{par}[v]
**root(v)**

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

**root(v)**

1: if \( \text{par}[v] = \bot \) then
2: \hspace{1em} return \( v \)
3: else
4: \hspace{1em} \text{par}[v] \leftarrow \text{root}(\text{par}[v])
5: \hspace{1em} return \text{par}[v]
MST-Kruskal(G, w)

1: \( F \leftarrow \emptyset \)
2: \( S \leftarrow \{\{v\} : v \in V\} \)
3: sort the edges of \( E \) in non-decreasing order of weights \( w \)
4: for each edge \((u, v) \in E\) in the order do
5: \( S_u \leftarrow \) the set in \( S \) containing \( u \)
6: \( S_v \leftarrow \) the set in \( S \) containing \( v \)
7: if \( S_u \neq S_v \) then
8: \( F \leftarrow F \cup \{(u, v)\} \)
9: \( S \leftarrow S \setminus \{S_u\} \setminus \{S_v\} \cup \{S_u \cup S_v\} \)
10: return \((V, F)\)
MST-Kruskal($G, w$)

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

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

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

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

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

**Lemma**  An edge $e \in E$ is not in the MST, if and only if there is cycle $C$ in $G$ in which $e$ is the heaviest edge.
Assumption  Assume all edge weights are different.

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

- $(i, g)$ is not in the MST because of cycle $(i, c, f, g)$
Assumption  Assume all edge weights are different.

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

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

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

2. Single Source Shortest Paths
   - Dijkstra’s Algorithm

3. Shortest Paths in Graphs with Negative Weights

4. All-Pair Shortest Paths and Floyd-Warshall
Two Methods to Build a MST

1. Start from $F \leftarrow \emptyset$, and add edges to $F$ one by one until we obtain a spanning tree.
Two Methods to Build a MST

1. Start from $F \leftarrow \emptyset$, and add edges to $F$ one by one until we obtain a spanning tree.

2. Start from $F \leftarrow E$, and remove edges from $F$ one by one until we obtain a spanning tree.

Q: Which edge can be safely excluded from the MST?

A: The heaviest non-bridge edge.

Def. A bridge is an edge whose removal disconnects the graph.
Two Methods to Build a MST

1. Start from $F \leftarrow \emptyset$, and add edges to $F$ one by one until we obtain a spanning tree.

2. Start from $F \leftarrow E$, and remove edges from $F$ one by one until we obtain a spanning tree.

Q: Which edge can be safely excluded from the MST?
Two Methods to Build a MST

1. Start from $F \leftarrow \emptyset$, and add edges to $F$ one by one until we obtain a spanning tree.

2. Start from $F \leftarrow E$, and remove edges from $F$ one by one until we obtain a spanning tree.

Q: Which edge can be safely excluded from the MST?

A: The heaviest non-bridge edge.
Two Methods to Build a MST

1. Start from $F \leftarrow \emptyset$, and add edges to $F$ one by one until we obtain a spanning tree.

2. Start from $F \leftarrow E$, and remove edges from $F$ one by one until we obtain a spanning tree.

Q: Which edge can be safely excluded from the MST?

A: The heaviest non-bridge edge.

Def. A bridge is an edge whose removal disconnects the graph.
Lemma  It is safe to exclude the heaviest non-bridge edge: there is a MST that does not contain the heaviest non-bridge edge.
Reverse Kruskal’s Algorithm

\textbf{MST-Greedy}(\(G, w\))

1: \(F \leftarrow E\)
2: sort \(E\) in non-increasing order of weights
3: \textbf{for} every \(e\) in this order \textbf{do}
4: \hspace{1em} \textbf{if} \ ((V, F \setminus \{e\})\text{ is connected} \textbf{then}
5: \hspace{2em} F \leftarrow F \setminus \{e\}
6: \textbf{return} \ ((V, F))
Reverse Kruskal’s Algorithm: Example
Reverse Kruskal’s Algorithm: Example
Reverse Kruskal’s Algorithm: Example
Reverse Kruskal’s Algorithm: Example

Graph with labeled edges:
Reverse Kruskal’s Algorithm: Example
Reverse Kruskal’s Algorithm: Example
Reverse Kruskal’s Algorithm: Example
Reverse Kruskal’s Algorithm: Example
Reverse Kruskal’s Algorithm: Example
Reverse Kruskal’s Algorithm: Example
Reverse Kruskal’s Algorithm: Example
Reverse Kruskal’s Algorithm: Example
Reverse Kruskal’s Algorithm: Example
Reverse Kruskal’s Algorithm: Example
Reverse Kruskal’s Algorithm: Example
Reverse Kruskal’s Algorithm: Example
Reverse Kruskal’s Algorithm: Example
Reverse Kruskal’s Algorithm: Example
Reverse Kruskal’s Algorithm: Example
Outline

1. **Minimum Spanning Tree**
   - Kruskal’s Algorithm
   - Reverse-Kruskal’s Algorithm
   - Prim’s Algorithm

2. **Single Source Shortest Paths**
   - Dijkstra’s Algorithm

3. **Shortest Paths in Graphs with Negative Weights**

4. **All-Pair Shortest Paths and Floyd-Warshall**
Recall the greedy strategy for Kruskal’s algorithm: choose the edge with the smallest weight.
Recall the greedy strategy for Kruskal’s algorithm: choose the edge with the smallest weight.

Greedy strategy for Prim’s algorithm: choose the lightest edge incident to $a$. 

**Design Greedy Strategy for MST**
Recall the greedy strategy for Kruskal’s algorithm: choose the edge with the smallest weight.

Greedy strategy for Prim’s algorithm: choose the lightest edge incident to a.
Lemma  It is safe to include the lightest edge incident to \( a \).
Lemma  It is safe to include the lightest edge incident to $a$. 

Proof.
- Let $T$ be a MST
- Consider all components obtained by removing $a$ from $T$
**Lemma**  It is safe to include the lightest edge incident to \( a \).

**Proof.**

- Let \( T \) be a MST.
- Consider all components obtained by removing \( a \) from \( T \).
- Let \( e^* \) be the lightest edge incident to \( a \) and \( e^* \) connects \( a \) to component \( C \).
Lemma  It is safe to include the lightest edge incident to \( a \).

Proof.

- Let \( T \) be a MST
- Consider all components obtained by removing \( a \) from \( T \)
- Let \( e^* \) be the lightest edge incident to \( a \) and \( e^* \) connects \( a \) to component \( C \)
- Let \( e \) be the edge in \( T \) connecting \( a \) to \( C \)
Lemma It is safe to include the lightest edge incident to $a$.

Proof.
- Let $T$ be a MST
- Consider all components obtained by removing $a$ from $T$
- Let $e^*$ be the lightest edge incident to $a$ and $e^*$ connects $a$ to component $C$
- Let $e$ be the edge in $T$ connecting $a$ to $C$
- $T' = T \setminus \{e\} \cup \{e^*\}$ is a spanning tree with $w(T') \leq w(T)$
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Prim’s Algorithm: Example

Diagram of a weighted graph with Prim's algorithm applied.
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Greedy Algorithm

**MST-Greedy1**($G, w$)

1. $S \leftarrow \{s\}$, where $s$ is arbitrary vertex in $V$
2. $F \leftarrow \emptyset$
3. **while** $S \neq V$ **do**
   4. $(u, v) \leftarrow$ lightest edge between $S$ and $V \setminus S$, where $u \in S$ and $v \in V \setminus S$
   5. $S \leftarrow S \cup \{v\}$
   6. $F \leftarrow F \cup \{(u, v)\}$
5. **return** $(V, F)$

Running time of naive implementation: $O(n^2)$
**Greedy Algorithm**

**MST-Greedy1**($G, w$)

1. $S \leftarrow \{s\}$, where $s$ is arbitrary vertex in $V$
2. $F \leftarrow \emptyset$
3. **while** $S \neq V$ **do**
4. $(u, v) \leftarrow$ lightest edge between $S$ and $V \setminus S$,
   where $u \in S$ and $v \in V \setminus S$
5. $S \leftarrow S \cup \{v\}$
6. $F \leftarrow F \cup \{(u, v)\}$
7. **return** $(V, F)$

- Running time of naive implementation: $O(nm)$
Prim’s Algorithm: Efficient Implementation of Greedy Algorithm

For every \( v \in V \setminus S \) maintain
- \( d[v] = \min_{u \in S: (u,v) \in E} w(u, v): \) the weight of the lightest edge between \( v \) and \( S \)
- \( \pi[v] = \arg \min_{u \in S: (u,v) \in E} w(u, v): \) \( (\pi[v], v) \) is the lightest edge between \( v \) and \( S \)
Prim’s Algorithm: Efficient Implementation of Greedy Algorithm

For every $v \in V \setminus S$ maintain

- $d[v] = \min_{u \in S: (u,v) \in E} w(u, v)$: the weight of the lightest edge between $v$ and $S$
- $\pi[v] = \arg \min_{u \in S: (u,v) \in E} w(u, v)$: $(\pi[v], v)$ is the lightest edge between $v$ and $S$

In every iteration

- Pick $u \in V \setminus S$ with the smallest $d[u]$ value
- Add $(\pi[u], u)$ to $F$
- Add $u$ to $S$, update $d$ and $\pi$ values.
Prim’s Algorithm

MST-Prim($G, w$)

1: $s \leftarrow$ arbitrary vertex in $G$
2: $S \leftarrow \emptyset$, $d(s) \leftarrow 0$ and $d[v] \leftarrow \infty$ for every $v \in V \setminus \{s\}$
3: while $S \neq V$ do
4: \hspace{1em} $u \leftarrow$ vertex in $V \setminus S$ with the minimum $d[u]$
5: \hspace{1em} $S \leftarrow S \cup \{u\}$
6: \hspace{1em} for each $v \in V \setminus S$ such that $(u, v) \in E$ do
7: \hspace{2em} if $w(u, v) < d[v]$ then
8: \hspace{3em} $d[v] \leftarrow w(u, v)$
9: \hspace{3em} $\pi[v] \leftarrow u$
10: return $\{(u, \pi[u]) | u \in V \setminus \{s\}\}$
Example
Example
Example

Graph with labeled edges:

- Edge (5, a) connects nodes b and c.
- Edge (12, a) connects nodes h and i.

Nodes and edges labeled with numbers 1 to 14.
Example

\[(5, a)\]
Example

\[(5, a)\]

\[(12, a)\]
Example

\begin{itemize}
\item \textbf{a}\hspace{1cm}b
\item \textbf{h}\hspace{1cm}i\hspace{1cm}c\hspace{1cm}d\hspace{1cm}e
\item \textbf{(8, b)}\hspace{1cm}13\hspace{1cm}9\hspace{1cm}14\hspace{1cm}10
\item \textbf{2}\hspace{1cm}4\hspace{1cm}14\hspace{1cm}10
\item \textbf{11, b)}\hspace{1cm}12\hspace{1cm}7\hspace{1cm}6\hspace{1cm}3
\end{itemize}
Example
Example

\[
\begin{array}{c}
(8, b) \\
(11, b)
\end{array}
\]
Example
Example
Example

Graph with edges labeled with tuples:

- (11, b)
- (2, c)
- (13, c)
- (4, c)
- (14, b)
- (13, b)

Nodes labeled with lowercase letters:
- a
- b
- c
- d
- e
- f
- g
- h
- i
Example
Example
Example
Example
Example
Example
Example
Example

\begin{itemize}
\item \((1, g)\)
\item \((13, c)\)
\item \((10, f)\)
\end{itemize}
Example

![Graph Image]

- Node labels: a, b, c, d, e, f, g, h, i
- Edge labels: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14
- Edges: (1, g), (10, f), (13, c)

The graph shows a network with labeled nodes and edges.
Example

\begin{figure}
\centering
\includegraphics[width=\textwidth]{example_graph}
\end{figure}
Example
Example
Example
Example
Prim’s Algorithm

For every $v \in V \setminus S$ maintain
- $d[v] = \min_{u \in S : (u, v) \in E} w(u, v)$:
  the weight of the lightest edge between $v$ and $S$
- $\pi[v] = \arg \min_{u \in S : (u, v) \in E} w(u, v)$:
  $(\pi[v], v)$ is the lightest edge between $v$ and $S$

In every iteration
- Pick $u \in V \setminus S$ with the smallest $d[u]$ value
- Add $(\pi[u], u)$ to $F$
- Add $u$ to $S$, update $d$ and $\pi$ values.
Prim’s Algorithm

For every \( v \in V \setminus S \) maintain

- \( d[v] = \min_{u \in S : (u,v) \in E} w(u,v) \): the weight of the lightest edge between \( v \) and \( S \)
- \( \pi[v] = \arg\min_{u \in S : (u,v) \in E} w(u,v) \): \((\pi[v], v)\) is the lightest edge between \( v \) and \( S \)

In every iteration

- Pick \( u \in V \setminus S \) with the smallest \( d[u] \) value \(\text{extract}_\text{min}\)
- Add \((\pi[u], u)\) to \( F \)
- Add \( u \) to \( S \), update \( d \) and \( \pi \) values. \(\text{decrease}_\text{key}\)

Use a priority queue to support the operations
**Def.** A *priority queue* is an abstract data structure that maintains a set $U$ of elements, each with an associated key value, and supports the following operations:

- $\text{insert}(v, \text{key\_value})$: insert an element $v$, whose associated key value is $\text{key\_value}$.
- $\text{decrease\_key}(v, \text{new\_key\_value})$: decrease the key value of an element $v$ in queue to $\text{new\_key\_value}$
- $\text{extract\_min}()$: return and remove the element in queue with the smallest key value
- ...
Prim’s Algorithm

**MST-Prim**($G, w$)

1: $s \leftarrow$ arbitrary vertex in $G$
2: $S \leftarrow \emptyset$, $d(s) \leftarrow 0$ and $d[v] \leftarrow \infty$ for every $v \in V \setminus \{s\}$
3: 
4: \textbf{while} $S \neq V$ \textbf{do}
5: \quad $u \leftarrow$ vertex in $V \setminus S$ with the minimum $d[u]$
6: \quad $S \leftarrow S \cup \{u\}$
7: \quad \textbf{for} each $v \in V \setminus S$ such that $(u, v) \in E$ \textbf{do}
8: \quad \quad \textbf{if} $w(u, v) < d[v]$ \textbf{then}
9: \quad \quad \quad $d[v] \leftarrow w(u, v)$
10: \quad \quad \quad $\pi[v] \leftarrow u$
11: \quad \textbf{return} $\{(u, \pi[u]) | u \in V \setminus \{s\}\}$
MST-Prim($G, w$)

1: $s \leftarrow$ arbitrary vertex in $G$
2: $S \leftarrow \emptyset, d(s) \leftarrow 0$ and $d[v] \leftarrow \infty$ for every $v \in V \setminus \{s\}$
3: $Q \leftarrow$ empty queue, for each $v \in V$: $Q.insert(v, d[v])$
4: while $S \neq V$ do
5: \hspace{1em} $u \leftarrow Q.extract\_min()$
6: \hspace{1em} $S \leftarrow S \cup \{u\}$
7: \hspace{1em} for each $v \in V \setminus S$ such that $(u, v) \in E$ do
8: \hspace{2em} if $w(u, v) < d[v]$ then
9: \hspace{3em} $d[v] \leftarrow w(u, v), Q.decrease\_key(v, d[v])$
10: \hspace{3em} $\pi[v] \leftarrow u$
11: return $\{(u, \pi[u])|u \in V \setminus \{s\}\}$
Running Time of Prim’s Algorithm Using Priority Queue

\[ O(n) \times \text{(time for extract\_min)} + O(m) \times \text{(time for decrease\_key)} \]
Running Time of Prim’s Algorithm Using Priority Queue

\[ O(n) \times (\text{time for extract\_min}) + O(m) \times (\text{time for decrease\_key}) \]

<table>
<thead>
<tr>
<th>concrete DS</th>
<th>extract_min</th>
<th>decrease_key</th>
<th>overall time</th>
</tr>
</thead>
<tbody>
<tr>
<td>heap</td>
<td>( O(\log n) )</td>
<td>( O(\log n) )</td>
<td>( O(m \log n) )</td>
</tr>
<tr>
<td>Fibonacci heap</td>
<td>( O(\log n) )</td>
<td>( O(1) )</td>
<td>( O(n \log n + m) )</td>
</tr>
</tbody>
</table>
Running Time of Prim’s Algorithm Using Priority Queue

\[ O(n) \times \text{(time for extract\_min)} + O(m) \times \text{(time for decrease\_key)} \]

<table>
<thead>
<tr>
<th>concrete DS</th>
<th>extract_min</th>
<th>decrease_key</th>
<th>overall time</th>
</tr>
</thead>
<tbody>
<tr>
<td>heap</td>
<td>(O(\log n))</td>
<td>(O(\log n))</td>
<td>(O(m \log n))</td>
</tr>
<tr>
<td>Fibonacci heap</td>
<td>(O(\log n))</td>
<td>(O(1))</td>
<td>(O(n \log n + m))</td>
</tr>
</tbody>
</table>
**Assumption** Assume all edge weights are different.

**Lemma** \((u, v)\) is in MST, if and only if there exists a cut \((U, V \setminus U)\), such that \((u, v)\) is the lightest edge between \(U\) and \(V \setminus U\).
Assumption  Assume all edge weights are different.

Lemma  \((u, v)\) is in MST, if and only if there exists a cut \((U, V \setminus U)\), such that \((u, v)\) is the lightest edge between \(U\) and \(V \setminus U\).

- \((c, f)\) is in MST because of cut \(\{a, b, c, i\}, V \setminus \{a, b, c, i\}\)
**Assumption**  Assume all edge weights are different.

**Lemma**  \((u, v)\) is in MST, if and only if there exists a cut \((U, V \setminus U)\), such that \((u, v)\) is the lightest edge between \(U\) and \(V \setminus U\).

- \((c, f)\) is in MST because of cut \(\{a, b, c, i\}, V \setminus \{a, b, c, i\}\)
- \((i, g)\) is not in MST because no such cut exists
“Evidence” for $e \in \text{MST}$ or $e \notin \text{MST}$

**Assumption** Assume all edge weights are different.

- $e \in \text{MST} \iff$ there is a cut in which $e$ is the lightest edge
- $e \notin \text{MST} \iff$ there is a cycle in which $e$ is the heaviest edge
“Evidence” for $e \in \text{MST}$ or $e \notin \text{MST}$

**Assumption** Assume all edge weights are different.

- $e \in \text{MST} \iff$ there is a cut in which $e$ is the lightest edge
- $e \notin \text{MST} \iff$ there is a cycle in which $e$ is the heaviest edge

Exactly one of the following is true:
- There is a cut in which $e$ is the lightest edge
- There is a cycle in which $e$ is the heaviest edge
“Evidence” for $e \in \text{MST}$ or $e \notin \text{MST}$

**Assumption**  Assume all edge weights are different.

- $e \in \text{MST} \iff$ there is a cut in which $e$ is the lightest edge
- $e \notin \text{MST} \iff$ there is a cycle in which $e$ is the heaviest edge

Exactly one of the following is true:

- There is a cut in which $e$ is the lightest edge
- There is a cycle in which $e$ is the heaviest edge

Thus, the minimum spanning tree is unique with assumption.