Greedy Algorithm

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

1: \(F \leftarrow \emptyset\)
2: sort edges in \(E\) in non-decreasing order of weights \(w\)
3: for each edge \((u, v)\) in the order do
4: if \(u\) and \(v\) are not connected by a path of edges in \(F\) then
5: \(F \leftarrow F \cup \{(u, v)\}\)
6: return \((V, F)\)
Sets: \{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g\}, \{h\}, \{i\}
Kruskal’s Algorithm: Example

Sets: \{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g\}, \{h\}, \{i\}
Sets: \{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g, h\}, \{i\}
Kruskal’s Algorithm: Example

Sets: \{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g, h\}, \{i\}
Kruskal’s Algorithm: Example

Sets: \{a\}, \{b\}, \{c, i\}, \{d\}, \{e\}, \{f\}, \{g, h\}
Kruskal’s Algorithm: Example

Sets: \{a\}, \{b\}, \{c, i\}, \{d\}, \{e\}, \{f\}, \{g, h\}
Kruskal’s Algorithm: Example

Sets: \{a\}, \{b\}, \{c, i\}, \{d\}, \{e\}, \{f, g, h\}
Kruskal’s Algorithm: Example

Sets: \{a\}, \{b\}, \{c, i\}, \{d\}, \{e\}, \{f, g, h\}
Sets:  \( \{a\} \),  \( \{b\} \),  \( \{c, i, f, g, h\} \),  \( \{d\} \),  \( \{e\} \)
Kruskal’s Algorithm: Example

Sets: \{a\}, \{b\}, \{c, i, f, g, h\}, \{d\}, \{e\}
Kruskal’s Algorithm: Example

Sets: \{a, b\}, \{c, i, f, g, h\}, \{d\}, \{e\}
Sets: \{a, b\}, \{c, i, f, g, h\}, \{d\}, \{e\}
Sets: \( \{a, b, c, i, f, g, h\}, \{d\}, \{e\} \)
Sets: \( \{a, b, c, i, f, g, h\}, \{d\}, \{e\} \)
Kruskal’s Algorithm: Example

Sets: \{a, b, c, i, f, g, h\}, \{d, e\}
Kruskal’s Algorithm: Example

Sets: \{a, b, c, i, f, g, h\}, \{d, e\}
Kruskal’s Algorithm: Example

Sets: \{a, b, c, i, f, g, h, d, e\}
Kruskal’s Algorithm: Efficient Implementation of Greedy Algorithm

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

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 $(V, F)$
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
\[ V = \{1, 2, 3, \ldots, 16\} \]

Partition: \( \{2, 3, 5, 9, 10, 12, 15\}, \{1, 7, 13, 16\}, \{4, 8, 11\}, \{6, 14\} \)

- \( par[i] \): parent of \( i \), \( (par[i] = \bot \text{ if } i \text{ is a root}) \).
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 $r'$:

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

Union-Find Data Structure

![Diagram of Union-Find data structure with nodes and edges illustrating set relationships.]
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

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

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

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

Improvement: all vertices in the path directly point to the root, saving time in the future.
Union-Find Data Structure

**root(v)**

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

1: if \( par[v] = \perp \) then
2: return \( v \)
3: else
4: \( par[v] \leftarrow \text{root}(par[v]) \)
5: return \( par[v] \)
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: $par[v] \leftarrow \bot$
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: $par[u'] \leftarrow v'$
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: \(par[v] \leftarrow \bot\)
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. \(par[u'] \leftarrow v'\)
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 \(3\) = \(O(m\log n)\).
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

**MST-Greedy**(\(G, w\))

1. \(F \leftarrow E\)
2. sort \(E\) in non-increasing order of weights
3. for every \(e\) in this order do
4. if \((V, F \setminus \{e\})\) is connected then
5. \(F \leftarrow F \setminus \{e\}\)
6. return \((V, F)\)
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

Diagram:

- Vertices: a, b, c, d, e, f, g, i, h
- Edges with weights:
  - a to b: 5
  - b to c: 8
  - b to i: 7
  - i to c: 2
  - c to g: 6
  - g to f: 3
  - f to e: 10
  - f to d: 9
  - d to e: 4
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

Diagram:

- Vertices: a, b, c, d, e, f, g, h, i
- Edges with weights:
  - a to b: 5
  - b to c: 8
  - c to i: 2
  - i to g: 3
  - g to f: 10
  - f to e: 9
  - d to e: 4

The diagram represents a graph where each vertex is connected to others with specific weights.