Kruskal’s MST Algorithm

**Greedy decision:** Add the edge with smallest weight, that does not create a cycle.
Lemma: Suppose that $S$ is a subset of nodes from $G = (V, E)$. Then, the cheapest edge $e$ between $S$ and $V \setminus S$ is part of every MST.

Proof: Any MST of $G$ must include some edge between $S$ and $V \setminus S$ (otherwise it would not be a tree).

Let $e$ be the cheapest edge between $S$ and $V \setminus S$.

Suppose $T$ is a ST that does not include $e$. $T \cup \{e\}$ must have a cycle and that cycle must have another edge $e'$ between $S$ and $V \setminus S$.

Remove $e'$ to form $T' = T \cup \{e\} \setminus \{e'\}$.

$T'$ is a tree (removing edge from cycle cannot disconnect graph)
$T'$ spans $V$ (same number of edges as ST $T$)
weight($T'$) = weight($T$) + weight($e$) − weight($e'$).

$\Rightarrow$ weight($T'$) < weight($T$), since weight($e$) < weight($e'$).

$\Rightarrow$ $T'$ is a cheaper ST.

So, $e$ is part of every MST.
Kruskal’s MST Algorithm

**Greedy decision:** Add the edge with smallest weight, that does not create a cycle.

**Proof of optimality:** Let $G = (V, E)$, and $T \subseteq E$ be the set of edges resulting from Kruskal’s algorithm.

??

**Lemma:** The cheapest edge between $S \subseteq V$ and $V \setminus S$ is part of every MST.
Kruskal’s MST Algorithm

Greedy decision: Add the edge with smallest weight, that does not create a cycle.

Proof of optimality:

1. Consider iteration where $(u, v)$ added to solution.

**Lemma:** The cheapest edge between $S \subseteq V$ and $V \setminus S$ is part of every MST.
Kruskal’s MST Algorithm

Greedy decision: Add the edge with smallest weight, that does not create a cycle.

Proof of optimality:

1. Consider iteration where \((u, v)\) added to solution.
2. Break into \(S\) and \(V \setminus S\) based on \((u, v)\).

Let \(S\) be: \(v\) and all nodes already connected to \(v\).

Lemma: The cheapest edge between \(S \subseteq V\) and \(V \setminus S\) is part of every MST.
Kruskal’s MST Algorithm

Greedy decision: Add the edge with smallest weight, that does not create a cycle.

Proof of optimality:

1. Consider iteration where \((u, v)\) added to solution.
2. Break into \(S\) and \(V \setminus S\) based on \((u, v)\).
3. Cheapest edge between \(S\) and \(V \setminus S\) must be in optimal solution.

Lemma: The cheapest edge between \(S \subseteq V\) and \(V \setminus S\) is part of every MST.
Kruskal’s MST Algorithm

**Greedy decision:** Add the edge with smallest weight, that does not create a cycle.

**Proof of optimality:** Let $G = (V, E)$, and $T \subseteq E$ be the set of edges resulting from Kruskal’s algorithm.

Consider the iteration that some edge $e = (u, v)$ is added by Kruskal’s algorithm. Let $S$ be the set of: $v$ and all nodes already connected to $v$. Clearly $v \in S$ and $u \in V \setminus S$ (otherwise adding $e$ would have created a cycle). We are picking the cheapest such edge (otherwise the cheaper edge would have been selected since it would not have created a cycle either). By the cut property lemma, this edge must be part of the MST.

**Lemma:** The cheapest edge between $S \subseteq V$ and $V \setminus S$ is part of every MST.
Prim’s MST Algorithm

**Greedy decision:** Beginning at any node, add the node that can be connected as cheaply as possible to the tree we are building.
Prim’s MST Algorithm

Greedy decision: Beginning at any node, add the node that can be connected as cheaply as possible to the tree we are building.
Prim’s MST Algorithm

**Greedy decision:** Beginning at any node, add the node that can be connected as cheaply as possible to the tree we are building.
Greedy decision: Beginning at any node, add the node that can be connected as cheaply as possible to the tree we are building.
Prim’s MST Algorithm

**Greedy decision:** Beginning at any node, add the node that can be connected as cheaply as possible to the tree we are building.
**Prim’s MST Algorithm**

**Greedy decision:** Beginning at any node, add the node that can be connected as cheaply as possible to the tree we are building.
Prim’s MST Algorithm

**Greedy decision:** Beginning at any node, add the node that can be connected as cheaply as possible to the tree we are building.
Prim’s MST Algorithm

**Greedy decision:** Beginning at any node, add the node that can be connected as cheaply as possible to the tree we are building.

**Proof of validity & optimality:** Let \( G = (V, E) \), and \( T \subseteq E \) be the set of edges resulting from Prim’s algorithm.

\( T \) is a tree because ??
Prim’s MST Algorithm

**Greedy decision:** Beginning at any node, add the node that can be connected as cheaply as possible to the tree we are building.

**Proof of validity & optimality:** Let $G = (V, E)$, and $T \subseteq E$ be the set of edges resulting from Prim’s algorithm.

$T$ is a tree because it consists of a single connected component and we never add an additional edge to an already connected node (i.e. no cycles).

$T$ is a spanning tree because ??
Prim’s MST Algorithm

Greedy decision: Beginning at any node, add the node that can be connected as cheaply as possible to the tree we are building.

Proof of validity & optimality: Let $G = (V, E)$, and $T \subseteq E$ be the set of edges resulting from Prim’s algorithm.

$T$ is a tree because it consists of a single connected component and we never add an additional edge to an already connected node (i.e. no cycles).

$T$ is a spanning tree because we added nodes (edges) until there were no nodes left to add.

$T$ is an MST because ??
Prim’s MST Algorithm

**Greedy decision:** Beginning at any node, add the node that can be connected as cheaply as possible to the tree we are building.

**Proof of validity & optimality:** Let $G = (V, E)$, and $T \subseteq E$ be the set of edges resulting from Prim’s algorithm.

$T$ is a tree because it consists of a single connected component and we never add an additional edge to an already connected node (i.e. no cycles).

$T$ is a spanning tree because we added nodes (edges) until there were no nodes left to add.

$T$ is an MST because at each step we add the cheapest edge between the current tree and the remaining nodes (cut property).
Activity Selection

Goal: Assign courses to a classroom.
Activity Selection

Input:
- $S = \{a_1, a_2, ..., a_n\}$ – set of courses that need rooms.
- $a_i = (s_i, f_i)$ – start and finish times for each course.
Activity Selection

Input:
• $S = \{a_1, a_2, ..., a_n\}$ – set of courses that need rooms.
• $a_i = (s_i, f_i)$ – start and finish times for each course.

Rules:
• $a_i$ and $a_j$ are compatible if $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap.
Activity Selection

Input:
• $S = \{a_1, a_2, ..., a_n\}$ – set of courses that need rooms.
• $a_i = (s_i, f_i)$ – start and finish times for each course.

Rules:
• $a_i$ and $a_j$ are compatible if $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap.

Goal: Select a maximum sized subset of mutually compatible courses.
Activity Selection

Input:
• $S = \{a_1, a_2, \ldots, a_n\}$ – set of courses that need rooms.
• $a_i = (s_i, f_i)$ – start and finish times for each course.

Rules:
• $a_i$ and $a_j$ are compatible if $[s_i, f_i]$ and $[s_j, f_j]$ do not overlap.

Goal: Select a maximum sized subset of mutually compatible courses.

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>$f_i$</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
</tr>
</tbody>
</table>
Activity Selection

Input:
• $S = \{a_1, a_2, ..., a_n\}$ – set of courses that need rooms.
• $a_i = (s_i, f_i)$ – start and finish times for each course.

Rules:
• $a_i$ and $a_j$ are compatible if $[s_i, f_i]$ and $[s_j, f_j]$ do not overlap.

Goal: Select a maximum sized subset of mutually compatible courses.
Activity Selection

Input:
• $S = \{a_1, a_2, ..., a_n\}$ – set of courses that need rooms.
• $a_i = (s_i, f_i)$ – start and finish times for each course.

Rules:
• $a_i$ and $a_j$ are compatible if $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap.

Goal: Select a maximum sized subset of mutually compatible courses.

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>$f_i$</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
</tr>
</tbody>
</table>
Activity Selection

Input:
• \( S = \{a_1, a_2, \ldots, a_n\} \) – set of courses that need rooms.
• \( a_i = (s_i, f_i) \) – start and finish times for each course.

Rules:
• \( a_i \) and \( a_j \) are compatible if \([s_i, f_i)\) and \([s_j, f_j)\) do not overlap.

Goal: Select a maximum sized subset of mutually compatible courses.

\[
\begin{array}{c|ccccc}
 i & 1 & 2 & 3 & 4 & 5 \\
 s_i & 1 & 3 & 4 & 5 & 7 \\
 f_i & 3 & 5 & 6 & 7 & 9 \\
\end{array}
\]
Activity Selection

Input:
- $S = \{a_1, a_2, ..., a_n\}$ – set of courses that need rooms.
- $a_i = (s_i, f_i)$ – start and finish times for each course.

Rules:
- $a_i$ and $a_j$ are compatible if $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap.

Goal: Select a maximum sized subset of mutually compatible courses.

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>$f_i$</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
</tr>
</tbody>
</table>

Greedy selection criteria?
- Earliest compatible finish time.

Algorithm Outline?
Activity Selection

activity_selection(activities A)
    sort_by_finish(A)
    selected = {A[1]}
    last_added = 1
    for i = 2 to A.length
        if A[i].start ≥ A[last_added].finish
            selected = selected ∪ {A[i]}
            last_added = i
    return selected

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>s_i</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>f_i</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
</tr>
</tbody>
</table>