CSE 548 / AMS 542: Analysis of Algorithms

Prerequisites Review 6
( Greedy Algorithms: MST, SSSP, … )

Rezaul Chowdhury
Department of Computer Science
SUNY Stony Brook
Fall 2023
An Activity-Selection Problem

Suppose:

- You are given a set $S = \{a_1, a_2, ..., a_n\}$ of $n$ proposed activities that wish to use a resource, such as a lecture hall, which can serve only one activity at a time.
An Activity-Selection Problem

Suppose:

- You are given a set $S = \{a_1, a_2, \ldots, a_n\}$ of $n$ proposed activities that wish to use a resource, such as a lecture hall, which can serve only one activity at a time.
- Each activity $a_i$ has a start time $s_i$ and finish time $f_i$, where $0 \leq s_i < f_i < \infty$. If selected, activity $a_i$ takes place during the half-open time interval $[s_i, f_i)$.
- Activities $a_i$ and $a_j$ are compatible if the intervals $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap. That is, $a_i$ and $a_j$ are compatible if $s_i \geq f_j$ or $s_j \geq f_i$.

Goal: Select a maximum-size subset of mutually compatible activities.

Assume that the activities are sorted in monotonically non-decreasing order of finish time: $f_1 \leq f_2 \leq f_3 \leq \cdots \leq f_{n-1} \leq f_n$. 

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>$a_6$</th>
<th>$a_7$</th>
<th>$a_8$</th>
<th>$a_9$</th>
<th>$a_{10}$</th>
<th>$a_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>$f_i$</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>
An Activity-Selection Problem

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>$a_6$</th>
<th>$a_7$</th>
<th>$a_8$</th>
<th>$a_9$</th>
<th>$a_{10}$</th>
<th>$a_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>$f_i$</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
An Activity-Selection Problem

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>$a_6$</th>
<th>$a_7$</th>
<th>$a_8$</th>
<th>$a_9$</th>
<th>$a_{10}$</th>
<th>$a_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>$f_i$</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

A mutually compatible set of activities
An Activity-Selection Problem

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>$a_6$</th>
<th>$a_7$</th>
<th>$a_8$</th>
<th>$a_9$</th>
<th>$a_{10}$</th>
<th>$a_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>$f_i$</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>

A largest mutually compatible set of activities
An Activity-Selection Problem

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>$a_6$</th>
<th>$a_7$</th>
<th>$a_8$</th>
<th>$a_9$</th>
<th>$a_{10}$</th>
<th>$a_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>$f_i$</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>

Another largest mutually compatible set of activities
Activity-Selection: Greedy Choice

Let $S_k =$ the set of activities in $S$ that start after activity $a_k$ finishes.

**Theorem:** Consider any nonempty subproblem $S_k$ and let $a_m$ be an activity in $S_k$ with the earliest finish time. Then $a_m$ is included in some maximum-size subset of mutually compatible activities of $S_k$.

**Proof:** Let $A_k =$ a maximum-size subset of mutually compatible activities in $S_k$.

Let $a_j$ be the activity in $A_k$ with the earliest finish time.

If $a_j = a_m$, we are done.

If $a_j \neq a_m$, let $A_k' = A_k - \{a_j\} \cup \{a_m\}$.

The activities in $A_k'$ are disjoint because the activities in $A_k$ are disjoint, $a_j$ is the first activity in $A_k$ to finish, and $f_m \leq f_j$.

Since $|A_k'| = |A_k|$, we conclude that $A_k'$ is a maximum-size subset of mutually compatible activities of $S_k$, and it includes $a_m$. 

14
Greedy Activity Selection

An example set \( S \) of activities

<table>
<thead>
<tr>
<th>( a_i )</th>
<th>( a_1 )</th>
<th>( a_2 )</th>
<th>( a_3 )</th>
<th>( a_4 )</th>
<th>( a_5 )</th>
<th>( a_6 )</th>
<th>( a_7 )</th>
<th>( a_8 )</th>
<th>( a_9 )</th>
<th>( a_{10} )</th>
<th>( a_{11} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_i )</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>( f_i )</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>
**Greedy Activity Selection**

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>$a_6$</th>
<th>$a_7$</th>
<th>$a_8$</th>
<th>$a_9$</th>
<th>$a_{10}$</th>
<th>$a_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>$f_i$</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>

**Accept $a_1$**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_5$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_6$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_7$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_8$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_9$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_{10}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_{11}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Greedy Activity Selection

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>$a_6$</th>
<th>$a_7$</th>
<th>$a_8$</th>
<th>$a_9$</th>
<th>$a_{10}$</th>
<th>$a_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>$f_i$</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>

Overlap with $a_1$
Greedy Activity Selection

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>$a_6$</th>
<th>$a_7$</th>
<th>$a_8$</th>
<th>$a_9$</th>
<th>$a_{10}$</th>
<th>$a_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>$f_i$</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>

Reject $a_2$
**Greedy Activity Selection**

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>$a_6$</th>
<th>$a_7$</th>
<th>$a_8$</th>
<th>$a_9$</th>
<th>$a_{10}$</th>
<th>$a_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>$f_i$</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>

Overlap with $a_1$
Greedy Activity Selection

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>$a_6$</th>
<th>$a_7$</th>
<th>$a_8$</th>
<th>$a_9$</th>
<th>$a_{10}$</th>
<th>$a_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>$f_i$</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>

Reject $a_3$
Greedy Activity Selection

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>$a_6$</th>
<th>$a_7$</th>
<th>$a_8$</th>
<th>$a_9$</th>
<th>$a_{10}$</th>
<th>$a_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>$f_i$</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>

Accept $a_4$
Greedy Activity Selection

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>$a_6$</th>
<th>$a_7$</th>
<th>$a_8$</th>
<th>$a_9$</th>
<th>$a_{10}$</th>
<th>$a_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1 3</td>
<td>0 5</td>
<td>3 5</td>
<td>6 8</td>
<td>8 2</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_i$</td>
<td>4 5</td>
<td>6 7</td>
<td>9 9</td>
<td>10 11</td>
<td>12 14</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Overlap with $a_4$
# Greedy Activity Selection

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>$a_6$</th>
<th>$a_7$</th>
<th>$a_8$</th>
<th>$a_9$</th>
<th>$a_{10}$</th>
<th>$a_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>$f_i$</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reject $a_5$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_5$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_6$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_7$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_8$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_9$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_{10}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_{11}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Greedy Activity Selection

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>$a_6$</th>
<th>$a_7$</th>
<th>$a_8$</th>
<th>$a_9$</th>
<th>$a_{10}$</th>
<th>$a_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>$f_i$</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>

Overlap with $a_4$
Greedy Activity Selection

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>$a_6$</th>
<th>$a_7$</th>
<th>$a_8$</th>
<th>$a_9$</th>
<th>$a_{10}$</th>
<th>$a_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>$f_i$</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>

Reject $a_6$
**Greedy Activity Selection**

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>$a_6$</th>
<th>$a_7$</th>
<th>$a_8$</th>
<th>$a_9$</th>
<th>$a_{10}$</th>
<th>$a_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>$f_i$</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>

Overlap with $a_4$

![Overlap Diagram with $a_4$]
Greedy Activity Selection

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>$a_6$</th>
<th>$a_7$</th>
<th>$a_8$</th>
<th>$a_9$</th>
<th>$a_{10}$</th>
<th>$a_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>$f_i$</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>

Reject $a_7$
Greedy Activity Selection

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>$a_6$</th>
<th>$a_7$</th>
<th>$a_8$</th>
<th>$a_9$</th>
<th>$a_{10}$</th>
<th>$a_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>$f_i$</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>

Accept $a_8$
Greedy Activity Selection

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>$a_6$</th>
<th>$a_7$</th>
<th>$a_8$</th>
<th>$a_9$</th>
<th>$a_{10}$</th>
<th>$a_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>$f_i$</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>

Overlap with $a_8$
Greedy Activity Selection

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>$a_6$</th>
<th>$a_7$</th>
<th>$a_8$</th>
<th>$a_9$</th>
<th>$a_{10}$</th>
<th>$a_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>$f_i$</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>

Reject $a_9$
Greedy Activity Selection

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>$a_6$</th>
<th>$a_7$</th>
<th>$a_8$</th>
<th>$a_9$</th>
<th>$a_{10}$</th>
<th>$a_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>$f_i$</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>

Overlap with $a_8$
Greedy Activity Selection

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>$a_6$</th>
<th>$a_7$</th>
<th>$a_8$</th>
<th>$a_9$</th>
<th>$a_{10}$</th>
<th>$a_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>$f_i$</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>

Reject $a_{10}$
Greedy Activity Selection

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>$a_6$</th>
<th>$a_7$</th>
<th>$a_8$</th>
<th>$a_9$</th>
<th>$a_{10}$</th>
<th>$a_{11}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>$f_i$</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>

Accept $a_{11}$
Greedy Activity-Selection

\textsc{Greedy-Activity-Selector} (s, f)

1. \( n \leftarrow \text{s.length} \)
2. \( A \leftarrow \{a_1\} \)
3. \( k \leftarrow 1 \)
4. \textbf{for} \( m \leftarrow 2 \) \textbf{to} \( n \) \textbf{do}
5. \textbf{if} \( s[m] \geq f[k] \) \textbf{then}
6. \( A \leftarrow A \cup \{a_m\} \)
7. \( k \leftarrow m \)
8. \textbf{return} \( A \)

Running time = \( \Theta(n) \)
A topological sort of a DAG (i.e., directed acyclic graph) $G = (V, E)$ is a linear ordering of all its vertices such that if $G$ contains an edge $(u, v)$, then $u$ appears before $v$ in the ordering.

We can view a topological sort of a graph as an ordering of its vertices along a horizontal line so that all directed edges go from left to right.
**Greedy-Topological-Sort (G)**

1. \( n \leftarrow |G.V| \)
2. \( i \leftarrow 0 \)
3. while \( i < n \) do
4. find a node \( x \in G.V \) with no incoming edges
5. assign number \( i \) to \( x \)
6. \( i \leftarrow i + 1 \)
7. remove \( x \) with all its outgoing edges from \( G \)
**Topological Sort**

**GREEDY-TOPOLOGICAL-SORT (G)**

1. \( n \leftarrow |G.V| \)
2. \( i \leftarrow 0 \)
3. while \( i < n \) do
4. find a node \( x \in G.V \) with no incoming edges
5. assign number \( i \) to \( x \)
6. \( i \leftarrow i + 1 \)
7. remove \( x \) with all its outgoing edges from \( G \)

choose node \( a \) with no incoming edges
**Topological Sort**

**GREEDY-TOPOLOGICAL-SORT (G)**

1. \( n \leftarrow |G.V| \)
2. \( i \leftarrow 0 \)
3. \( \text{while } i < n \text{ do} \)
4. \( \text{find a node } x \in G.V \) with no incoming edges
5. \( \text{assign number } i \text{ to } x \)
6. \( i \leftarrow i + 1 \)
7. \( \text{remove } x \text{ with all its outgoing edges from } G \)

**Diagram:**
- Assign number \( i = 0 \) to node \( a \)
- \( i \leftarrow 1 \) (increment \( i \))
**Topological Sort**

**GREEDY-TOPOLOGICAL-SORT (G)**

1. \( n \leftarrow |G.V| \)
2. \( i \leftarrow 0 \)
3. **while** \( i < n \) **do**
4. find a node \( x \in G.V \) with no incoming edges
5. assign number \( i \) to \( x \)
6. \( i \leftarrow i + 1 \)
7. remove \( x \) with all its outgoing edges from \( G \)
**Greedy-Topological-Sort** \( (G) \)

1. \( n \leftarrow |G.V| \)
2. \( i \leftarrow 0 \)
3. **while** \( i < n \) **do**
4. find a node \( x \in G.V \) with no incoming edges
5. assign number \( i \) to \( x \)
6. \( i \leftarrow i + 1 \)
7. remove \( x \) with all its outgoing edges from \( G \)

---

**Choose node \( b \) with no incoming edges**

- Choose node \( b \) with no incoming edges.
**Topological Sort**

**GREEDY-TOPOLOGICAL-SORT (G)**

1. \( n \leftarrow |G.V| \)
2. \( i \leftarrow 0 \)
3. \textbf{while} \( i < n \) 
4. find a node \( x \in G.V \) with no incoming edges
5. assign number \( i \) to \( x \)
6. \( i \leftarrow i + 1 \)
7. remove \( x \) with all its outgoing edges from \( G \)

assign number \( i = 1 \) to node \( b \)
\( i \leftarrow 2 \) (increment \( i \))
**GREEDY-TOPOLOGICAL-SORT (G)**

1. \( n \leftarrow |G.V| \)
2. \( i \leftarrow 0 \)
3. \textbf{while} \( i < n \) \textbf{do}
4. \hspace{10pt} \textbf{find a node} \( x \in G.V \) \textbf{with no incoming edges}
5. \hspace{10pt} \textbf{assign number} \( i \) \textbf{to} \( x \)
6. \hspace{10pt} \( i \leftarrow i + 1 \)
7. \hspace{10pt} \textbf{remove} \( x \) \textbf{with all its outgoing edges from} \( G \)

**remove node b with all its outgoing edges**
**Greedy-Topological-Sort (G)**

1. \( n \leftarrow |G.V| \)
2. \( i \leftarrow 0 \)
3. while \( i < n \) do
4.   find a node \( x \in G.V \) with no incoming edges
5.   assign number \( i \) to \( x \)
6.   \( i \leftarrow i + 1 \)
7.   remove \( x \) with all its outgoing edges from \( G \)

Choose node \( c \) with no incoming edges
**Topological Sort**

**GREEDY-TOPOLOGICAL-SORT (G)**

1. \( n \leftarrow |G.V| \)
2. \( i \leftarrow 0 \)
3. **while** \( i < n \) **do**
   4. find a node \( x \in G.V \) with no incoming edges
   5. assign number \( i \) to \( x \)
   6. \( i \leftarrow i + 1 \)
   7. remove \( x \) with all its outgoing edges from \( G \)

Diagram:

- Assign number \( i = 2 \) to node \( c \)
- \( i \leftarrow 3 \) (increment \( i \))
**Greedy Topological Sort (G)**

1. \( n \leftarrow |G.V| \)
2. \( i \leftarrow 0 \)
3. **while** \( i < n \) **do**
4. find a node \( x \in G.V \) with no incoming edges
5. assign number \( i \) to \( x \)
6. \( i \leftarrow i + 1 \)
7. remove \( x \) with all its outgoing edges from \( G \)

**Topological Sort**

remove node \( c \) with all its outgoing edges
**Greedy-Topological-Sort (G)**

1. \( n \leftarrow |G.V| \)
2. \( i \leftarrow 0 \)
3. while \( i < n \) do
4.   find a node \( x \in G.V \) with no incoming edges
5.   assign number \( i \) to \( x \)
6.   \( i \leftarrow i + 1 \)
7.   remove \( x \) with all its outgoing edges from \( G \)

---

**choose node \( d \) with no incoming edges**

```
choose node d with no incoming edges
```

![Diagram](image)
**Greedy-Topological-Sort (G)**

1. \( n \leftarrow |G.V| \)
2. \( i \leftarrow 0 \)
3. while \( i < n \) do
4.    find a node \( x \in G.V \) with no incoming edges
5.    assign number \( i \) to \( x \)
6.    \( i \leftarrow i + 1 \)
7.    remove \( x \) with all its outgoing edges from \( G \)

**Topological Sort**

assign number \( i = 3 \) to node \( d \)
\( i \leftarrow 4 \) (increment \( i \))
**Topological Sort**

**GREEDY-TOPOLOGICAL-SORT (G)**

1. \( n \leftarrow |G.V| \)
2. \( i \leftarrow 0 \)
3. \( \text{while } i < n \text{ do} \)
4. \( \text{find a node } x \in G.V \text{ with no incoming edges} \)
5. \( \text{assign number } i \text{ to } x \)
6. \( i \leftarrow i + 1 \)
7. \( \text{remove } x \text{ with all its outgoing edges from } G \)
GREEDY-TOPOLOGICAL-SORT (G)

1. \( n \leftarrow |G.V| \)
2. \( i \leftarrow 0 \)
3. while \( i < n \) do
4. find a node \( x \in G.V \) with no incoming edges
5. assign number \( i \) to \( x \)
6. \( i \leftarrow i + 1 \)
7. remove \( x \) with all its outgoing edges from \( G \)

**Example:**

Choose node \( e \) with no incoming edges.
**Greedy-Topological-Sort** $(G)$

1. $n \leftarrow |G.V|$
2. $i \leftarrow 0$
3. while $i < n$ do
4. find a node $x \in G.V$ with no incoming edges
5. assign number $i$ to $x$
6. $i \leftarrow i + 1$
7. remove $x$ with all its outgoing edges from $G$

**Assign number** $i = 4$ **to node** $e$

$i \leftarrow 5$ (increment $i$)
**GREEDY-TOPOLOGICAL-SORT (G)**

1. \( n \leftarrow |G.V| \)
2. \( i \leftarrow 0 \)
3. while \( i < n \) do
4.   find a node \( x \in G.V \) with no incoming edges
5.   assign number \( i \) to \( x \)
6.   \( i \leftarrow i + 1 \)
7.   remove \( x \) with all its outgoing edges from \( G \)
**Topological Sort**

**GREEDY-TOPOLOGICAL-SORT (G)**

1. \( n \leftarrow |G.V| \)
2. \( i \leftarrow 0 \)
3. while \( i < n \) do
4.   find a node \( x \in G.V \) with no incoming edges
5.   assign number \( i \) to \( x \)
6.   \( i \leftarrow i + 1 \)
7.   remove \( x \) with all its outgoing edges from \( G \)

choose node \( g \) with no incoming edges
**Greedy-Topological-Sort (G)**

1. \( n \leftarrow |G.V| \)
2. \( i \leftarrow 0 \)
3. **while** \( i < n \) **do**
   4. find a node \( x \in G.V \) with no incoming edges
   5. assign number \( i \) to \( x \)
   6. \( i \leftarrow i + 1 \)
   7. remove \( x \) with all its outgoing edges from \( G \)

**Assign number** \( i = 5 \) **to node** \( g \)

\( i \leftarrow 6 \) (increment \( i \))
**GREEDY-TOPOLOGICAL-SORT (G)**

1. $n \leftarrow |G.V|$
2. $i \leftarrow 0$
3. **while** $i < n$ **do**
4. find a node $x \in G.V$ with no incoming edges
5. assign number $i$ to $x$
6. $i \leftarrow i + 1$
7. remove $x$ with all its outgoing edges from $G$

**remove node $g$ with all its outgoing edges**
**Greedy-Topological-Sort (G)**

1. \( n \leftarrow |G.V| \)
2. \( i \leftarrow 0 \)
3. while \( i < n \) do
4. find a node \( x \in G.V \) with no incoming edges
5. assign number \( i \) to \( x \)
6. \( i \leftarrow i + 1 \)
7. remove \( x \) with all its outgoing edges from \( G \)

**Topological Sort**

Choose node \( f \) with no incoming edges
**Greedy-Topological-Sort** \((G)\)

1. \(n \leftarrow |G.V|\)
2. \(i \leftarrow 0\)
3. **while** \(i < n\) **do**
4. find a node \(x \in G.V\) with no incoming edges
5. assign number \(i\) to \(x\)
6. \(i \leftarrow i + 1\)
7. remove \(x\) with all its outgoing edges from \(G\)
**Greedy-Topological-Sort (G)**

1. \( n \leftarrow |G.V| \)
2. \( i \leftarrow 0 \)
3. \textbf{while} \( i < n \) \textbf{do}
4. \quad find a node \( x \in G.V \) with no incoming edges
5. \quad assign number \( i \) to \( x \)
6. \quad \( i \leftarrow i + 1 \)
7. \quad remove \( x \) with all its outgoing edges from \( G \)
**Topological Sort**

**Greedy-Topological-Sort (G)**

1. \( n \leftarrow |G.V| \)
2. \( i \leftarrow 0 \)
3. **while** \( i < n \) **do**
4. find a node \( x \in G.V \) with no incoming edges
5. assign number \( i \) to \( x \)
6. \( i \leftarrow i + 1 \)
7. remove \( x \) with all its outgoing edges from \( G \)

**choose node h with no incoming edges**

![Diagram of a graph with nodes a, b, c, d, e, f, g, and h labeled with numbers 0 to 6. Edge directions and numbers are shown.]
**Topological Sort**

**GREEDY-TOPOLOGICAL-SORT (G)**

1. \( n \leftarrow |G.V| \)
2. \( i \leftarrow 0 \)
3. while \( i < n \) do
   4. find a node \( x \in G.V \) with no incoming edges
   5. assign number \( i \) to \( x \)
   6. \( i \leftarrow i + 1 \)
   7. remove \( x \) with all its outgoing edges from \( G \)

assign number \( i = 7 \) to node \( h \)

\( i \leftarrow 8 \) (increment \( i \))
**Topological Sort**

**GREEDY-TOPOLOGICAL-SORT (G)**

1. \( n \leftarrow |G.V| \)
2. \( i \leftarrow 0 \)
3. while \( i < n \) do
4.   find a node \( x \in G.V \) with no incoming edges
5.   assign number \( i \) to \( x \)
6.   \( i \leftarrow i + 1 \)
7.   remove \( x \) with all its outgoing edges from \( G \)

remove node \( h \) with all its outgoing edges
**Topological Sort**

**GREEDY-TOPOLOGICAL-SORT (G)**

1. \( n \leftarrow |G.V| \)
2. \( i \leftarrow 0 \)
3. while \( i < n \) do
4. find a node \( x \in G.V \) with no incoming edges
5. assign number \( i \) to \( x \)
6. \( i \leftarrow i + 1 \)
7. remove \( x \) with all its outgoing edges from \( G \)
**Topological Sort**

**Greedy-Topological-Sort (G)**

1. \( n \leftarrow |G.V| \)
2. \( i \leftarrow 0 \)
3. while \( i < n \) do
4.   find a node \( x \in G.V \) with no incoming edges
5.   assign number \( i \) to \( x \)
6.   \( i \leftarrow i + 1 \)
7.   remove \( x \) with all its outgoing edges from \( G \)

assign number \( i = 8 \) to node \( i \)

\( i \leftarrow 9 \) (increment \( i \))
**Greedy Topological Sort**

\[ \text{Greedy-Topological-Sort} \ (G) \]

1. \( n \leftarrow |G.V| \)
2. \( i \leftarrow 0 \)
3. \textbf{while} \( i < n \) \textbf{do}
4. \textbf{find a node} \( x \in G.V \) \textbf{with no incoming edges}
5. \textbf{assign number} \( i \) \textbf{to} \( x \)
6. \( i \leftarrow i + 1 \)
7. \textbf{remove} \( x \) \textbf{with all its outgoing edges from} \( G \)

**Diagram:**

- Node labels: 0, 1, 2, 3, 4, 5, 6, 7, 8
- Edges and labels indicate dependencies between nodes.
**Greedy-Topological-Sort (G)**

1. \( n \leftarrow |G.V| \)
2. \( i \leftarrow 0 \)
3. **while** \( i < n \) **do**
4. find a node \( x \in G.V \) with no incoming edges
5. assign number \( i \) to \( x \)
6. \( i \leftarrow i + 1 \)
7. remove \( x \) with all its outgoing edges from \( G \)

![Diagram of Topological Sort](image)

---

**Done!**

---

**Diagram of Topological Sort**

- Graph \( G \) with nodes labeled from 0 to 8.
- Nodes are ordered topologically from left to right.
- Edge labels indicate direction of edges.

---

**Nodes and Edges**

- Nodes: a, b, c, d, e, g, f, h, i
- Edges: directed from lower to higher numbers.
Let $n = |G.V|$ and $m = |G.E|$. Then the running time of the algorithm is $O(n + m)$. 

**Greedy-Topological-Sort (G)**

1. $n \leftarrow |G.V|$
2. $i \leftarrow 0$
3. while $i < n$ do
4. find a node $x \in G.V$ with no incoming edges
5. assign number $i$ to $x$
6. $i \leftarrow i + 1$
7. remove $x$ with all its outgoing edges from $G$
The Minimum Spanning Tree (MST) Problem

We are given a weighted connected undirected graph $G = (V, E)$ with vertex set $V$ and edge set $E$, and a weight function $w$ such that for each edge $(u, v) \in E$, $w(u, v)$ represents its weight.

An acyclic subset $T \subseteq E$ that connects all vertices of $V$ must form a tree, which we call a spanning tree since it “spans” the graph $G$.

A spanning tree of $G$ can be found easily in $O(n + m)$ time, where $n = |V|$ and $m = |E|$, using a breadth-first search (BFS) or a depth-first search (DFS).

The minimum-spanning-tree (MST) problem asks us to find a spanning tree $T$ whose total weight $w(T) = \sum_{(u, v) \in T} w(u, v)$ is minimized.
The Minimum Spanning Tree (MST) Problem

A weighted undirected graph
The Minimum Spanning Tree (MST) Problem

A weighted undirected graph

Its MST (in red) of total weight 37
MST: Greedy Strategy for Growing an MST

We are given a weighted connected undirected graph \( G = (V, E) \) with vertex set \( V \) and edge set \( E \), and a weight function \( w \) such that for each edge \((u, v) \in E\), \( w(u, v) \) represents its weight.

Suppose set \( A \subset E \) is a subset of some MST of \( G \).

Now if edge \((u, v) \in E\) but edge \((u, v) \notin A\), we call \((u, v)\) a **safe edge** provided \( A \cup \{u, v\} \) is also a subset of an MST of \( G \).
MST: Greedy Strategy for Growing an MST

Red edges form an MST. Let’s call it $T$

Let $A = \{(i, c), (c, f), (f, g), (g, h)\}$

Clearly, $A \subset T$. 

MST: Greedy Strategy for Growing an MST

Red edges form an MST. Let’s call it $T$

Let $A = \{(i, c), (c, f), (f, g), (g, h)\}$

Clearly, $A \subset T$.

Edge $(b, c)$ is safe because $A \cup \{(b, c)\} \subseteq T$. 
MST: Greedy Strategy for Growing an MST

Red edges form an MST. Let’s call it $T$

Let $A = \{(i, c), (c, f), (f, g), (g, h)\}$

Clearly, $A \subset T$.

Edge $(c, d)$ is safe because $A \cup \{(c, d)\} \subseteq T$. 
MST: Greedy Strategy for Growing an MST

Red edges form an MST. Let's call it $T$.

Let $A = \{(i, c), (c, f), (f, g), (g, h)\}$.

Clearly, $A \subseteq T$.

Edge $(a, b)$ is safe because $A \cup \{(a, b)\} \subseteq T$. 
MST: Greedy Strategy for Growing an MST

Red edges form an MST. Let’s call it $T$.

Let $A = \{(i, c), (c, f), (f, g), (g, h)\}$

Clearly, $A \subset T$.

Edge $(d, e)$ is safe because $A \cup \{(d, e)\} \subseteq T$. 
MST: Greedy Strategy for Growing an MST

Red edges form an MST. Let’s call it $T$

Let $A = \{(i, c), (c, f), (f, g), (g, h)\}$

Clearly, $A \subset T$.

Edge $(h, i)$ is NOT safe because $A \cup \{(h, i)\}$ is NOT part of any MST of the given graph.
MST: Greedy Strategy for Growing an MST

Let's call it $T$. Let $A = \{(i, c), (c, f), (f, g), (g, h)\}$

Clearly, $A \subset T$.

Edge $(d, f)$ is NOT safe because $A \cup \{(d, f)\}$ is NOT part of any MST of the given graph.
MST: Greedy Strategy for Growing an MST

Red edges form an MST. Let’s call it \( T \)

Let \( A = \{(i, c), (c, f), (f, g), (g, h)\} \)

Clearly, \( A \subset T \).

Edge \((a, h)\) is safe because though \( A \cup \{(a, h)\} \) is not a subset of \( T \), it is a subset of another MST.
MST: Greedy Strategy for Growing an MST

Generic-MST(\(G = (V, E), w\))

1. \(A \leftarrow \emptyset\)
2. while \(A\) does not form a spanning tree of \(G\) do
3. find an edge \((u, v) \in E\) that is safe for \(A\)
4. \(A \leftarrow A \cup \{(u, v)\}\)
5. return \(A\)
MST: Finding Safe Edges

A **cut** \((S, V \setminus S)\) of an undirected graph \(G = (V, E)\) is a partition of \(V\).

We say that an edge \((u, v) \in E\) **crosses** the cut \((S, V \setminus S)\) if one of its endpoints is in \(S\) and the other is in \(V \setminus S\).

Green vertices belong to set \(S\), i.e., \(S = \{a, b, d, e\}\).
White vertices belong to set \(V - S\), i.e., \(V - S = \{c, f, g, h, i\}\).
The red line represent the cut \((S, V - S)\).
Dotted edges are the cut edges, i.e., they cross the red line.
MST: Finding Safe Edges

A cut \textit{respects} a set $A$ of edges if no edge in $A$ crosses the cut.

An edge is a \textit{light edge} crossing a cut if its weight is the minimum of any edge crossing the cut. Multiple light edges can cross a cut.

Let the blue thick edges form the set $A$, i.e.,

$$A = \{(a, b), (c, f), (c, i), (f, g), (g, h)\}.$$

Then edge $(c, d)$ is a light edge crossing the cut.
MST: Finding Safe Edges

A cut respects a set $A$ of edges if no edge in $A$ crosses the cut.

An edge is a light edge crossing a cut if its weight is the minimum of any edge crossing the cut. Multiple light edges can cross a cut.

The entire set $A$ can be on the same side of the cut, e.g.,

$$A = \{(c, f), (c, i), (f, g), (g, h)\}.$$ 

Still edge $(c, d)$ is a light edge crossing the cut.
MST: Finding Safe Edges

A cut respects a set \( A \) of edges if no edge in \( A \) crosses the cut.

An edge is a light edge crossing a cut if its weight is the minimum of any edge crossing the cut. Multiple light edges can cross a cut.

Consider a different cut as shown above.

Consider the same set \( A = \{(c, f), (c, i), (f, g), (g, h)\} \).

Now both \((a, h)\) and \((b, c)\) are light edges crossing the cut.
**MST: Finding Safe Edges**

**Theorem:** Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function $w$ defined on $E$. Let $A$ be a subset of $E$ that is included in some minimum spanning tree for $G$, and let $(S, V \setminus S)$ be any cut of $G$ that respects $A$, and let $(u, v)$ be a light edge crossing $(S, V \setminus S)$. Then, edge $(u, v)$ is safe for $A$. 
**THEOREM:** ... ... ... Let $A \subset E$ is included in some MST $T$ of $G$, and let $(S, V \setminus S)$ be any cut of $G$ that respects $A$, and let $(u, v)$ be a light edge crossing $(S, V \setminus S)$. Then, edge $(u, v)$ is safe for $A$.

**PROOF IDEA:**

Let $(u, v)$ be a light edge crossing the cut.
Let’s assume $(u, v) \notin T$, as otherwise we are done.
**MST: Finding Safe Edges**

**THEOREM:** ... ... Let $A \subseteq E$ is included in some MST $T$ of $G$, and let $(S, V \setminus S)$ be any cut of $G$ that respects $A$, and let $(u, v)$ be a light edge crossing $(S, V \setminus S)$. Then, edge $(u, v)$ is safe for $A$.

**PROOF IDEA:**

As $T$ is a spanning tree, some edge $(x, y) \in T$ must also cross the cut.
**MST: Finding Safe Edges**

**THEOREM:** Let $A \subset E$ is included in some MST $T$ of $G$, and let $(S, V \setminus S)$ be any cut of $G$ that respects $A$, and let $(u, v)$ be a light edge crossing $(S, V \setminus S)$. Then, edge $(u, v)$ is safe for $A$.

**PROOF IDEA:**

As $T$ is a spanning tree, some edge $(x, y) \in T$ must also cross the cut. Let’s add edge $(u, v)$ to $T$. That must form a cycle in $T \cup \{(u, v)\}$. So, $T \cup \{(u, v)\}$ is not a tree.
**THEOREM:** ... ... ... Let \( A \subset E \) is included in some MST \( T \) of \( G \), and let \((S, V \setminus S)\) be any cut of \( G \) that respects \( A \), and let \((u, v)\) be a light edge crossing \((S, V \setminus S)\). Then, edge \((u, v)\) is safe for \( A \).

**PROOF IDEA:**

We can break the cycle by removing edge \((x, y)\) from \( T \cup \{(u, v)\} \).
Let \( T' = T - \{(x, y)\} \cup \{(u, v)\} \).

Observe that \( T' \) is now a spanning tree of \( G \).
**MST: Finding Safe Edges**

**Theorem:** Let $A \subseteq E$ be included in some MST $T$ of $G$, and let $(S, V \setminus S)$ be any cut of $G$ that respects $A$, and let $(u, v)$ be a light edge crossing $(S, V \setminus S)$. Then, edge $(u, v)$ is safe for $A$.

**Proof Idea:**

Now, $w(T') = w(T - \{(x, y)\} \cup \{(u, v)\})$

$$= w(T) - w((x, y)) + w((u, v)) \leq w(T)$$

But we assumed that $T$ is an MST of $G$, and so $w(T) \leq w(T')$
**THEOREM:** Let $A \subseteq E$ is included in some MST $T$ of $G$, and let $(S, V \setminus S)$ be any cut of $G$ that respects $A$, and let $(u, v)$ be a light edge crossing $(S, V \setminus S)$. Then, edge $(u, v)$ is safe for $A$.

**PROOF IDEA:**

Since, $w(T') \leq w(T)$ and $w(T) \leq w(T')$, we have $w(T') = w(T)$. So, $T'$ must also be an MST of $G$. 
**MST: Finding Safe Edges**

**Theorem:** Let \( A \subseteq E \) be included in some MST \( T \) of \( G \), and let \((S, V \setminus S)\) be any cut of \( G \) that respects \( A \), and let \((u, v)\) be a light edge crossing \((S, V \setminus S)\). Then, edge \((u, v)\) is safe for \( A \).

**Proof Idea:**

Since \( A \subseteq T \) and \((x, y) \notin A\), we have \( A \subseteq T' \).

Thus, \( A \cup \{(u, v)\} \subseteq T' \).

Since \( T' \) is an MST of \( G \), edge \((u, v)\) is safe for \( A \).
**MST: Finding Safe Edges**

**Theorem:** Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function $w$ defined on $E$. Let $A$ be a subset of $E$ that is included in some minimum spanning tree for $G$, and let $(S, V \setminus S)$ be any cut of $G$ that respects $A$, and let $(u, v)$ be a light edge crossing $(S, V \setminus S)$. Then, edge $(u, v)$ is safe for $A$.

**Corollary:** Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function $w$ defined on $E$. Let $A$ be a subset of $E$ that is included in some minimum spanning tree for $G$, and let $C = (V_C, E_C)$ be a connected component (tree) in the forest $G_A = (V, A)$. If $(u, v)$ is a light edge crossing from $C$ to some other component of $G_A$, then edge $(u, v)$ is safe for $A$. 
**MST: Prim’s Algorithm**

\[ \text{MST-Prim} \ ( G = (V, E), \ w, \ r ) \]

1. \( \text{for each vertex } v \in G. V \ \text{do} \)
2. \( v.d \leftarrow \infty \)
3. \( v.\pi \leftarrow \text{NIL} \)
4. \( r.d \leftarrow 0 \)
5. \( \text{Min-Heap } Q \leftarrow \emptyset \)
6. \( \text{for each vertex } v \in G. V \ \text{do} \)
7. \( \text{INSERT}( Q, v ) \)
8. \( \text{while } Q \neq \emptyset \ \text{do} \)
9. \( u \leftarrow \text{EXTRACT-MIN}( Q ) \)
10. \( \text{for each } (u,v) \in G. E \ \text{do} \)
11. \( \text{if } v \in Q \ \text{and } w(u,v) < v.d \ \text{then} \)
12. \( v.d \leftarrow w(u,v) \)
13. \( v.\pi \leftarrow u \)
14. \( \text{DECREASE-KEY}( Q, v, w(u,v) ) \)
MST: Prim’s Algorithm

Initial State

- **a**. \( d = \infty \)
  - **a**. \( \pi = NIL \)
- **b**. \( d = \infty \)
  - **b**. \( \pi = NIL \)
- **c**. \( d = \infty \)
  - **c**. \( \pi = NIL \)
- **d**. \( d = \infty \)
  - **d**. \( \pi = NIL \)
- **e**. \( d = \infty \)
  - **e**. \( \pi = NIL \)
- **h**. \( d = \infty \)
  - **h**. \( \pi = NIL \)
- **g**. \( d = \infty \)
  - **g**. \( \pi = NIL \)
- **f**. \( d = \infty \)
  - **f**. \( \pi = NIL \)
MST: Prim’s Algorithm

Initial State

- $b. \text{d} = \infty$
- $b. \pi = \text{NIL}$
- $c. \text{d} = \infty$
- $c. \pi = \text{NIL}$
- $d. \text{d} = \infty$
- $d. \pi = \text{NIL}$
- $a. \text{d} = 0$
- $a. \pi = \text{NIL}$
- $i. \text{d} = \infty$
- $i. \pi = \text{NIL}$
- $h. \text{d} = \infty$
- $h. \pi = \text{NIL}$
- $g. \text{d} = \infty$
- $g. \pi = \text{NIL}$
- $f. \text{d} = \infty$
- $f. \pi = \text{NIL}$
- $e. \text{d} = \infty$
- $e. \pi = \text{NIL}$
Step 1: add vertex \( a \) to MST

\begin{itemize}
\item \( b. d = \infty \), \( b. \pi = \text{NIL} \)
\item \( c. d = \infty \), \( c. \pi = \text{NIL} \)
\item \( d. d = \infty \), \( d. \pi = \text{NIL} \)
\item \( a. d = 0 \), \( a. \pi = \text{NIL} \)
\item \( i. d = \infty \), \( i. \pi = \text{NIL} \)
\item \( h. d = \infty \), \( h. \pi = \text{NIL} \)
\item \( g. d = \infty \), \( g. \pi = \text{NIL} \)
\item \( f. d = \infty \), \( f. \pi = \text{NIL} \)
\item \( e. d = \infty \), \( e. \pi = \text{NIL} \)
\end{itemize}
Step 1': update neighbors of a

\begin{align*}
S &= \{a\} \\
\text{Cut} &= (S, V - S)
\end{align*}
MST: Prim’s Algorithm

Step 2: add vertex $b$ through edge $(a, b)$

$S = \{a\}$
Cut $= (S, V - S)$

$(a, b)$ is the light edge crossing the cut
**Step 2’**: update neighbors of \( b \)

\[
\begin{align*}
S & = \{a, b\} \\
\text{Cut} & = (S, V - S)
\end{align*}
\]
MST: Prim’s Algorithm

Step 3: add vertex c through edge \((b, c)\)

\[ S = \{a, b\} \]

Cut = \((S, V - S)\)

\((b, c)\) is a light edge crossing the cut
MST: Prim’s Algorithm

Step 3’: update neighbors of $c$

$S = \{a, b, c\}$
Cut = $(S, V - S)$
Step 4: add vertex $i$ through edge $(c, i)$

$S = \{a, b, c\}$

Cut = $(S, V - S)$

$(c, i)$ is the light edge crossing the cut
**MST: Prim’s Algorithm**

Step 4’: update neighbors of \( i \)

\[ S = \{a, b, c, i\} \]

Cut = \((S, V - S)\)
MST: Prim’s Algorithm

Step 5: add vertex $f$ through edge $(c, f)$

$S = \{a, b, c, i\}$

Cut = $(S, V - S)$

$(c, f)$ is the light edge crossing the cut
Step 5’: update neighbors of $f$

$S = \{a, b, c, i, f\}$
Cut = $(S, V - S)$
**MST: Prim’s Algorithm**

**Step 6: add vertex** \( g \) **through edge** \( (f, g) \)

\[ S = \{a, b, c, i, f\} \]

Cut = \((S, V - S)\)

\((c, g)\) is the light edge crossing the cut
Step 6': update neighbors of $g$

$S = \{a, b, c, i, f, g\}$
Cut = $(S, V - S)$
MST: Prim’s Algorithm

Step 7: add vertex $h$ through edge $(g, h)$

$S = \{a, b, c, i, f, g\}$

Cut = $(S, V - S)$

$(g, h)$ is the light edge crossing the cut
**MST: Prim’s Algorithm**

**Step 7’: update neighbors of h**

\[ S = \{a, b, c, i, f, g, h\} \]

\[ \text{Cut} = (S, V - S) \]
Step 8: add vertex $d$ through edge $(c, d)$

$S = \{a, b, c, i, f, g, h\}$

Cut = $(S, V - S)$

$(c, d)$ is the light edge crossing the cut
MST: Prim’s Algorithm

Step 8’: update neighbors of $d$

$S = \{a, b, c, i, f, g, h, d\}$
Cut = $(S, V - S)$
MST: Prim’s Algorithm

Step 9: add vertex $e$ through edge $(d, e)$

$S = \{a, b, c, i, f, g, h, d\}$

Cut = $(S, V - S)$

$(d, e)$ is the light edge crossing the cut
Step 9′: update neighbors of e

\[ S = \{a, b, c, i, f, g, h, d, e\} \]
MST: Prim’s Algorithm

Done

Total weight = 37
MST: Prim’s Algorithm

Let \( n = |V| \) and \( m = |E| \)

# INSERTS = \( n \)

# EXTRACT-MINS = \( n \)

# DECREASE-KEYS \( \leq m \)

Total cost

\[
\leq n(\text{cost}_{\text{Insert}} + \text{cost}_{\text{Extract-Min}}) + m(\text{cost}_{\text{Decrease-Key}})
\]
MST: Prim’s Algorithm

\[\text{Let } n = |V| \text{ and } m = |E|\]

For Binary Heap (worst-case costs):
\[\begin{align*}
\text{cost}_{\text{Insert}} &= O(\log n) \\
\text{cost}_{\text{Extract-Min}} &= O(\log n) \\
\text{cost}_{\text{Decrease-Key}} &= O(\log n)
\end{align*}\]

\[\therefore \text{Total cost (worst-case)} = O((m + n) \log n)\]
MST: Prim’s Algorithm

Let \( n = |V| \) and \( m = |E| \)

For Fibonacci Heap (amortized):

\[
\begin{align*}
\text{cost}_{\text{Insert}} &= O(1) \\
\text{cost}_{\text{Extract-Min}} &= O(\log n) \\
\text{cost}_{\text{Decrease-Key}} &= O(1)
\end{align*}
\]

∴ Total cost (amortized)
\[
= O(m + n \log n)
\]
A Disjoint-Set Data Structure
(for Kruskal’s MST Algorithm)

A disjoint-set data structure maintains a collection of disjoint dynamic sets. Each set is identified by a representative which must be a member of the set.

The collection is maintained under the following operations:

**MAKE-SET( x ):** create a new set \( \{x\} \) containing only element \( x \).

   Element \( x \) becomes the representative of the set.

**FIND( x ):** returns a pointer to the representative of the set containing \( x \)

**UNION( x, y ):** replace the dynamic sets \( S_x \) and \( S_y \) containing \( x \) and \( y \), respectively, with the set \( S_x \cup S_y \)
A Disjoint-Set Data Structure (union by rank)

**MAKE-SET (x)**
1. \( \pi(x) \leftarrow x \)
2. \( \text{rank}(x) \leftarrow 0 \)

**LINK (x, y)**
1. \( \text{if rank}(x) > \text{rank}(y) \text{ then } \pi(y) \leftarrow x \)
2. \( \text{else } \pi(x) \leftarrow y \)
3. \( \text{if rank}(x) = \text{rank}(y) \text{ then } \text{rank}(y) \leftarrow \text{rank}(y) + 1 \)

**UNION (x, y)**
1. \( \text{LINK (FIND (x), FIND (y))} \)

**FIND (x)**
1. \( \text{if } x \neq \pi(x) \text{ then return FIND (} \pi(x) \text{)} \)
2. \( \text{else return } x \)
A Disjoint-Set Data Structure
(union by rank)

**Theorem:** A sequence of $N$ MAKE-SET, UNION and FIND operations of which exactly $n \leq N$ are MAKE-SET operations takes $O(N \log n)$ time to execute.
LEMMA: Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function $w$ defined on $E$. Let $C$ be a cycle of $G$ with a unique heaviest edge $\hat{e} \in E$. Then $\hat{e}$ cannot be part of any MST of $G$. 
**MST: Another Useful Lemma**

**Lemma:** Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function $w$ defined on $E$. Let $C$ be a cycle of $G$ with a unique heaviest edge $\hat{e} \in E$. Then $\hat{e}$ cannot be part of any MST of $G$.

**Proof:**

Let $\hat{e}$ be part of some spanning tree $T$ of $G$. 

![Graph with weights](image)
**MST: Another Useful Lemma**

**Lemma:** Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function $w$ defined on $E$. Let $C$ be a cycle of $G$ with a unique heaviest edge $\hat{e} \in E$. Then $\hat{e}$ cannot be part of any MST of $G$.

**Proof:**

Let $\hat{e}$ be part of some spanning tree $T$ of $G$. Let’s remove $\hat{e}$ from $T$. Then $T$ will get split into two components. There must be an edge $\bar{e} \in C$ that reconnects the two components.
**MST: Another Useful Lemma**

**Lemma:** Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function $w$ defined on $E$. Let $C$ be a cycle of $G$ with a unique heaviest edge $\hat{e} \in E$. Then $\hat{e}$ cannot be part of any MST of $G$.

**Proof:**

Let’s add $\bar{e}$ to $T - \{\hat{e}\}$, and let $T' = T \cup \{\bar{e}\} - \{\hat{e}\}$. Then $T'$ is a spanning tree of $G$.

Since $w(\hat{e}) > w(\bar{e})$, we get, $w(T') = w(T) - w(\hat{e}) + w(\bar{e}) < w(T)$.

So, $T$ cannot be an MST of $G$!
**MST: Kruskal’s Algorithm**

**MST-Kruskal** \((G = (V, E), w)\)

1. \(A \leftarrow \emptyset\)
2. \textbf{for} each vertex \(v \in G.V\) \textbf{do}
3. \hspace{1em} \textbf{MAKE-SET}(v)
4. \hspace{1em} sort the edges of \(G.E\) into nondecreasing order by weight \(w\)
5. \textbf{for} each edge \((u, v) \in G.E\) taken in nondecreasing order by weight \textbf{do}
6. \hspace{1em} \textbf{if} \ FIND(u) \neq FIND(v) \ \textbf{then}
7. \hspace{2em} \hspace{1em} \(A \leftarrow A \cup \{(u, v)\}\)
8. \hspace{1em} \textbf{UNION}(u, v)
9. \hspace{1em} \textbf{return} A
**MST: Kruskal’s Algorithm**

**Initial State:**

**Disjoint-Set Data Structure (union by rank only):**

\[ \text{MAKE-SET}(x), \ x \in \{a, b, c, d, e, f, g, h, i\} \]
MST: Kruskal’s Algorithm

(1) edge \((h, g)\):

\[
S = \{\text{component (connected through red edges) containing } h\} = \{h\}
\]

Cut = \((S, V - S)\)

\((h, g)\) is the light edge crossing the cut

Disjoint-Set Data Structure (union by rank only):

\[
\text{FIND}(h)\text{ returns } h, \quad \text{FIND}(g)\text{ returns } g
\]
MST: Kruskal’s Algorithm

(1) edge \((h, g)\):

\[
S = \{\text{component (connected through red edges) containing } h\} = \{h\}
\]

Cut = \((S, V - S)\)

\((h, g)\) is the light edge crossing the cut

Disjoint-Set Data Structure
(union by rank only):

\[
\text{UNION( } h, g \text{ )}
\]
MST: Kruskal’s Algorithm

(2) edge \((i, c)\):

\[
S = \{ \text{component (connected through red edges) containing } i \} = \{i\}
\]

\[
\text{Cut} = (S, V - S)
\]

\((i, c)\) is the light edge crossing the cut

Disjoint-Set Data Structure (union by rank only):

\(\text{FIND}(i)\) returns \(i\), \(\text{FIND}(c)\) returns \(c\)
**MST: Kruskal’s Algorithm**

(2) edge \((i, c)\):

\[
S = \{\text{component (connected through red edges) containing } i\} = \{i\}
\]

Cut = \((S, V - S)\)

\((i, c)\) is the light edge crossing the cut

**Disjoint-Set Data Structure**
(union by rank only):

\[
\text{UNION}(i, c)
\]
MST: Kruskal’s Algorithm

(3) edge \((g, f)\):

\[
S = \{\text{component (connected through red edges) containing } g\} = \{h, g\}
\]

Cut = \((S, V - S)\)

\((g, f)\) is the light edge crossing the cut

**Disjoint-Set Data Structure**

(union by rank only):

\[\text{FIND}(g) \text{ returns } g, \quad \text{FIND}(f) \text{ returns } f\]
MST: Kruskal’s Algorithm

(3) edge \((g, f)\):

\[ S = \{\text{component (connected through red edges) containing } g\} = \{h, g\} \]

Cut = \((S, V - S)\)

\((g, f)\) is the light edge crossing the cut

Disjoint-Set
Data Structure
(union by rank only):

\[ \text{UNION}(g, f) \]
MST: Kruskal’s Algorithm

(4) edge \((a, b)\):

\[ S = \{\text{component (connected through red edges) containing } a\} = \{a\} \]

Cut = \((S, V - S)\)

\((a, b)\) is the light edge crossing the cut

Disjoint-Set Data Structure (union by rank only):

\[ \text{FIND}(a) \text{ returns } a, \quad \text{FIND}(b) \text{ returns } b \]
MST: Kruskal’s Algorithm

(4) edge \((a, b)\):

\[
S = \{\text{component (connected through red edges) containing } a\} = \{a\}
\]

\[
\text{Cut} = (S, V - S)
\]

\((a, b)\) is the light edge crossing the cut

Disjoint-Set Data Structure
(union by rank only):

```
UNION(a, b)
```
MST: Kruskal’s Algorithm

(5) edge \((c, f)\):

\[
S = \{\text{component (connected through red edges) containing } c\} = \{c, i\}
\]

Cut = \((S, V - S)\)

\((c, f)\) is the light edge crossing the cut

---

**Disjoint-Set Data Structure**

(union by rank only):

\[
\begin{align*}
\text{FIND}(c) \text{ returns } c, & \quad \text{FIND}(f) \text{ returns } g \\
\end{align*}
\]
MST: Kruskal’s Algorithm

(5) edge \((c, f)\):

\[ S = \{ \text{component (connected through red edges) containing } c \} = \{c, i\} \]

Cut = \((S, V - S)\)

\((c, f)\) is the light edge crossing the cut

Disjoint-Set
Data Structure
(union by rank only):

\[
\begin{align*}
\text{UNION}(c, f)
\end{align*}
\]
MST: Kruskal’s Algorithm

(6) edge \((i, g)\):

\[
S = \{\text{component (connected through red edges) containing } i\} = \{i, c, f, g, h\}
\]

Cut = \((S, V - S)\)

\((i, g)\) creates a cycle by connecting two nodes of \(S\), and it is the heaviest edge on that cycle

---

Disjoint-Set Data Structure
(union by rank only):

\[
\text{FIND}(i) \text{ returns } g, \quad \text{FIND}(g) \text{ returns } g
\]
**MST: Kruskal’s Algorithm**

(6) edge \((i, g)\):

\[ S = \{ \text{component (connected through red edges) containing } i \} = \{i, c, f, g, h\} \]

\[ \text{Cut} = (S, V - S) \]

\((i, g)\) creates a cycle by connecting two nodes of \(S\), and it is the heaviest edge on that cycle.

**Disjoint-Set Data Structure**

(union by rank only):

- Node \(a\) with rank 0
- Node \(b\) with rank 1
- Node \(c\) with rank 0
- Node \(d\) with rank 0
- Node \(e\) with rank 0
- Node \(f\) with rank 0
- Node \(g\) with rank 0
- Node \(h\) with rank 0
- Node \(i\) with rank 0
**MST: Kruskal’s Algorithm**

(7) edge \((c, d)\):

\[
S = \{\text{component (connected through red edges) containing } c\} = \{i, c, f, g, h\}
\]

Cut = \((S, V - S)\)

\((c, d)\) is the light edge crossing the cut

---

**Disjoint-Set Data Structure**

(union by rank only):

\(\text{FIND}(c)\) returns \(g\), \(\text{FIND}(d)\) returns \(d\)
MST: Kruskal’s Algorithm

(7) edge \( (c, d) \):

\[ S = \{ \text{component (connected through red edges) containing } c \} = \{ i, c, f, g, h \} \]

Cut = \((S, V - S)\)

\( (c, d) \) is the light edge crossing the cut

Disjoint-Set
Data Structure
(union by rank only):

\[ \text{UNION}(c, d) \]
MST: Kruskal’s Algorithm

(8) edge \((i, h)\):

\[ S = \{ \text{component (connected through red edges) containing } i \} = \{i, c, f, g, h\} \]

\[ \text{Cut} = (S, V - S) \]

\((i, h)\) creates a cycle by connecting two nodes of \(S\), and it is the heaviest edge on that cycle.

Disjoint-Set Data Structure (union by rank only):

\[ \text{FIND}(i) \text{ returns } g, \quad \text{FIND}(h) \text{ returns } g \]
(8) edge \((i, h)\):  

\[ S = \{ \text{component (connected through red edges) containing } i \} = \{i, c, f, g, h\} \]

Cut = \((S, V - S)\)

\((i, h)\) creates a cycle by connecting two nodes of \(S\), and it is the heaviest edge on that cycle.

**Disjoint-Set Data Structure**  
(union by rank only):
(9) edge \((a, h)\): 

\[ S = \{\text{component (connected through red edges) containing } a\} = \{a, b\} \]

Cut = \((S, V - S)\)

\((a, h)\) is a light edge crossing the cut

**Disjoint-Set Data Structure** (union by rank only):

\(
\begin{align*}
\text{FIND}(a) & \text{ returns } b, \quad \text{FIND}(h) \text{ returns } g
\end{align*}
\)
**MST: Kruskal’s Algorithm**

(9) edge \((a, h)\):

\[
S = \{\text{component (connected through red edges) containing } a\} = \{a, b\}
\]

Cut = \((S, V - S)\)

\((a, h)\) is a light edge crossing the cut

---

**Disjoint-Set Data Structure**

(union by rank only):

\[
\text{UNION}(a, h)
\]
MST: Kruskal’s Algorithm

(10) edge \((b, c)\):

\[ \begin{align*}
S &= \{ \text{component (connected through red edges) containing } b \} = \{a, b, c, d, f, g, h, i\} \\
\text{Cut} &= (S, V - S)
\end{align*} \]

\((b, c)\) creates a cycle by connecting two nodes of \(S\), and it is a heaviest edge on that cycle.

**Disjoint-Set Data Structure**

(union by rank only):

\[ \text{FIND}(b) \text{ returns } g, \quad \text{FIND}(c) \text{ returns } g \]
MST: Kruskal’s Algorithm

(10) edge \((b, c)\):

\[
S = \{\text{component (connected through red edges) containing } b\} = \{a, b, c, d, f, g, h, i\}
\]

\[
\text{Cut} = (S, V - S)
\]

\((b, c)\) creates a cycle by connecting two nodes of \(S\), and it is a heaviest edge on that cycle

Disjoint-Set

Data Structure

(union by rank only)
MST: Kruskal’s Algorithm

(11) edge \((d, e)\):

\[ S = \{ \text{component (connected through red edges) containing } d \} = \{a, b, c, d, f, g, h, i\} \]

\[ \text{Cut} = (S, V - S) \]

\((d, e)\) is a light edge crossing the cut

**Disjoint-Set Data Structure (union by rank only):**

FIND\((d)\) returns \(g\), FIND\((e)\) returns \(e\)
MST: Kruskal’s Algorithm

(11) edge \((d, e)\):

\[ S = \{\text{component (connected through red edges) containing } d\} = \{a, b, c, d, f, g, h, i\} \]

Cut = \((S, V - S)\)

\((d, e)\) is a light edge crossing the cut

Disjoint-Set
Data Structure
(union by rank only) :
MST: Kruskal’s Algorithm

(12) edge \((e, f)\):

\[ S = \{\text{component (connected through red edges) containing } e\} = \{a, b, c, d, e, f, g, h, i\} \]

\[ \text{Cut} = (S, V - S) \]

\((e, f)\) creates a cycle by connecting two nodes of \(S\), and it is the heaviest edge on that cycle

**Disjoint-Set Data Structure**
(union by rank only):

\(\text{FIND}(e)\) returns \(g\), \(\text{FIND}(f)\) returns \(g\)
MST: Kruskal’s Algorithm

(12) edge \((e, f)\):

\[ S = \{\text{component (connected through red edges) containing } e\} = \{a, b, c, d, e, f, g, h, i\} \]

\[ \text{Cut} = (S, V - S) \]

\((e, f)\) creates a cycle by connecting two nodes of \(S\), and it is the heaviest edge on that cycle

Disjoint-Set Data Structure
(union by rank only):

\[
\begin{array}{ccccccccc}
\text{a} & \text{b} & \text{c} & \text{d} & \text{e} & \text{f} & \text{g} & \text{i} & \text{h} \\
\hline
0 & 1 & 0 & 0 & 0 & 0 & 2 & 0 & 0
\end{array}
\]
MST: Kruskal’s Algorithm

(13) edge \((b, h)\):

\[ S = \{\text{component (connected through red edges) containing } b\} = \{a, b, c, d, e, f, g, h, i\} \]

\[ \text{Cut} = (S, V - S) \]

\((b, h)\) creates a cycle by connecting two nodes of \(S\), and it is the heaviest edge on that cycle

---

Disjoint-Set
Data Structure
(union by rank only):

\(\text{FIND}(b)\) returns \(g\), \(\text{FIND}(h)\) returns \(g\)
MST: Kruskal’s Algorithm

(13) edge \((b, h)\):

\[ S = \{\text{component (connected through red edges) containing } b\} = \{a, b, c, d, e, f, g, h, i\} \]

Cut = \((S, V - S)\)

\((b, h)\) creates a cycle by connecting two nodes of \(S\), and it is the heaviest edge on that cycle

Disjoint-Set
Data Structure
(union by rank only) :

---
MST: Kruskal’s Algorithm

(14) edge \((d, f)\):

\[ S = \{ \text{component (connected through red edges) containing } d \} = \{a, b, c, d, e, f, g, h, i\} \]

\[ \text{Cut} = (S, V - S) \]

\((d, f)\) creates a cycle by connecting two nodes of \(S\), and it is the heaviest edge on that cycle.

Disjoint-Set Data Structure (union by rank only):

\[ \text{FIND}(d) \text{ returns } g, \quad \text{FIND}(f) \text{ returns } g \]
MST: Kruskal’s Algorithm

(14) edge \((d, f)\):

\[ S = \{\text{component (connected through red edges) containing } d\} = \{a, b, c, d, e, f, g, h, i\}\]

\[
\text{Cut} = (S, V - S)
\]

\((d, f)\) creates a cycle by connecting two nodes of \(S\), and it is the heaviest edge on that cycle

Disjoint-Set

Data Structure

(union by rank only):
MST: Kruskal’s Algorithm

(14) edge \((d, f)\):

Total weight = 37

Disjoint-Set Data Structure
(union by rank only):
Let \( n = |V| \) and \( m = |E| \). Since \( G \) is connected, we have \( m \geq n - 1 \). Then the sorting in step 4 can be done in \( O(m \log m) \) time.

\#disjoint-set operations performed, \( N = 2m + 2n - 1 \), of which

\#MAKE-SET: \( n \),  \#FIND: \( 2m \),  \#UNION: \( n - 1 \)

So, total time taken by disjoint-set operations = \( O((n + m) \log n) \)

Hence, MST-Kruskal’s running time = \( O(m \log m) \)
The Single-Source Shortest Paths (SSSP) Problem

We are given a weighted, directed graph $G = (V, E)$ with vertex set $V$ and edge set $E$, and a non-negative weight function $w$ such that for each edge $(u, v) \in E$, $w(u, v)$ represents its weight.

We are also given a source vertex $s \in V$.

Our goal is to find a shortest path (i.e., a path of the smallest total edge weight) from $s$ to each vertex $v \in V$. 
Lemma: [subpaths of shortest paths are shortest paths] Given a weighted, directed graph $G = (V, E)$ with weight function $w: E \to \mathbb{R}$, let $p = v_1 v_2 \ldots v_k$ be a shortest path from vertex $v_1$ to vertex $v_k$ and, for any $i$ and $j$ such that $1 \leq i \leq j \leq k$, let $p_{ij} = v_i v_{i+1} \ldots v_j$ be the subpath of $p$ from vertex $v_i$ to vertex $v_j$. Then $p_{ij}$ is a shortest path from $v_i$ to $v_j$. 

Intuition behind Dijkstra’s SSSP Algorithm 

Lemma: [subpaths of shortest paths are shortest paths] Given a weighted, directed graph $G = (V, E)$ with weight function $w: E \to \mathbb{R}$, let $p = v_1 v_2 \ldots v_k$ be a shortest path from vertex $v_1$ to vertex $v_k$ and, for any $i$ and $j$ such that $1 \leq i \leq j \leq k$, let $p_{ij} = v_i v_{i+1} \ldots v_j$ be the subpath of $p$ from vertex $v_i$ to vertex $v_j$. Then $p_{ij}$ is a shortest path from $v_i$ to $v_j$. 

166
**Intuition behind Dijkstra’s SSSP Algorithm**

Lemma: [subpaths of shortest paths are shortest paths] … … …

Let $p = v_1 v_2 \ldots v_k$ be a shortest path from $v_1$ to $v_k$ and, for any $i$ and $j$ such that $1 \leq i \leq j \leq k$, let $p_{ij} = v_i v_{i+1} \ldots v_j$ be the subpath of $p$ from $v_i$ to $v_j$. Then $p_{ij}$ is a shortest path from $v_i$ to $v_j$.

**Proof:** Let’s decompose $p$ as follows.

Then weight of path $p$, $w(p) = w(p_{1i}) + w(p_{ij}) + w(p_{jk})$. 

![Diagram of shortest paths](image)
**Intuition behind Dijkstra’s SSSP Algorithm**

**Lemma:** [subpaths of shortest paths are shortest paths] ...

Let $p = v_1v_2 \ldots v_k$ be a shortest path from $v_1$ to $v_k$ and, for any $i$ and $j$ such that $1 \leq i \leq j \leq k$, let $p_{ij} = v_i v_{i+1} \ldots v_j$ be the subpath of $p$ from $v_i$ to $v_j$. Then $p_{ij}$ is a shortest path from $v_i$ to $v_j$.

**Proof:** Let’s decompose $p$ as follows.

```
\begin{align*}
    \text{Then weight of path } p, \ w(p) &= w(p_{1i}) + w(p_{ij}) + w(p_{jk}). \\
    \text{If } p_{ij} \text{ is not a shortest path, let } p'_{ij} \text{ be a shorter path from } v_i \text{ to } v_j. \\
    \therefore w(p_{1i}) + w(p'_{ij}) + w(p_{jk}) &< w(p_{1i}) + w(p_{ij}) + w(p_{jk}) = w(p),
\end{align*}
```

which contradicts our assumption that $p$ is a shortest $v_1$ to $v_k$ path.
**Intuition behind Dijkstra’s SSSP Algorithm**

**Observations:** Let $v \in V$ and $v \neq s$.

Consider any shortest path $p_{s,v}$ from $s$ to $v$.

Path $p_{s,v}$ must reach $v$ through a node $u$ from which $v$ has an incoming edge, i.e., $(u, v) \in E$.

Let $p_{s,u}$ be the subpath of $p_{s,v}$ that goes from $s$ to $u$.

Since subpaths of shortest paths are also shortest paths, $p_{s,u}$ must be a shortest path from $s$ to $u$.

So, once we know $p_{s,u}$, we can append $(u, v)$ to it to find $p_{s,v}$.

But two questions!
Intuition behind Dijkstra’s SSSP Algorithm

First question: $v$ can have multiple incoming edges. How do we know which of them lies on $p_{s,v}$?

Suppose, $v$ has $k$ incoming edges $(u_1, v), (u_2, v), \ldots, (u_k, v)$.

The solution is to maintain a tentative shortest $s$ to $v$ distance $d[v]$ initialized to $\infty$, and update $d[v]$ to $\min\{d[v], w(p_{s,u}) + w(u, v)\}$ when we find the shortest path $p_{s,u}$ to each $u \in \{u_1, u_2, \ldots, u_k\}$.
Intuition behind Dijkstra’s SSSP Algorithm

Second question: When do we know that $d[v] = \delta(s, v)$, where $\delta(s, v)$ is the shortest distance from $s$ to $v$?

Find shortest paths to vertices in non-decreasing order of $\delta(s, \cdot)$.

We start with vertex $s$ because we know $\delta(s, s) = 0$.

Since edge weights are non-negative, any $u$ with $\delta(s, u) > \delta(s, v)$ cannot be on $p_{s,v}$.

So, if $d[v]$ is the smallest among all vertices to which we are yet to find shortest distances, we know that $d[v] = \delta(s, v)$. 
Dijkstra’s SSSP Algorithm with a Min-Heap

(SSSP: Single-Source Shortest Paths)

**Input:** Weighted graph $G = (V, E)$ with vertex set $V$ and edge set $E$, a weight function $w$, and a source vertex $s \in G[V]$.

**Output:** For all $v \in G[V]$, $v.d$ is set to the shortest distance from $s$ to $v$.

```
Dijkstra-SSSP ( G = (V, E), w, s )
1.    for each vertex v ∈ G.V do
2.        v.d ← ∞
3.        v.π ← NIL
4.    s.d ← 0
5.    Min-Heap Q ← ∅
6.    for each vertex v ∈ G.V do
7.        INSERT( Q, v )
8.    while Q ≠ ∅ do
9.        u ← EXTRACT-MIN( Q )
10.       for each (u,v) ∈ G.E do
11.           if u.d + w(u,v) < v.d then
12.              v.d ← u.d + w(u,v)
13.              v.π ← u
14.            DECREASE-KEY( Q, v, u.d + w(u,v) )
```
SSSP: Dijkstra’s Algorithm

Initial State (with initial tentative distances)
SSSP: Dijkstra’s Algorithm

Step 1: add vertex $s$ to SPT
SSSP: Dijkstra’s Algorithm

Step 1’: update neighbors of $s$
SSSP: Dijkstra’s Algorithm

Step 2: add vertex $y$ through edge $(s, y)$
SSSP: Dijkstra’s Algorithm

Step 2’: update neighbors of $y$
SSSP: Dijkstra’s Algorithm

Step 3: add vertex z through edge (y, z)
SSSP: Dijkstra’s Algorithm

Step 3′: update neighbors of $z$
SSSP: Dijkstra’s Algorithm

Step 4: add vertex $t$ through edge $(y, t)$
SSSP: Dijkstra’s Algorithm

Step 4’: update neighbors of \( t \)
SSSP: Dijkstra’s Algorithm

Step 5: add vertex $x$ through edge $(t, x)$
SSSP: Dijkstra’s Algorithm

Step 5’: update neighbors of $x$
SSSP: Dijkstra’s Algorithm

Done
One undirected edge $\Rightarrow$ Two directed edges

SSSP: Dijkstra’s Algorithm
SSSP: Dijkstra’s Algorithm

Initial State (with initial tentative distances)

- **b.** \(d = \infty\)  \(\pi = \text{NIL}\)
- **c.** \(d = \infty\)  \(\pi = \text{NIL}\)
- **d.** \(d = \infty\)  \(\pi = \text{NIL}\)
- **a.** \(d = \infty\)  \(\pi = \text{NIL}\)
- **i.** \(d = \infty\)  \(\pi = \text{NIL}\)
- **h.** \(d = \infty\)  \(\pi = \text{NIL}\)
- **g.** \(d = \infty\)  \(\pi = \text{NIL}\)
- **f.** \(d = \infty\)  \(\pi = \text{NIL}\)
- **e.** \(d = \infty\)  \(\pi = \text{NIL}\)
SSSP: Dijkstra’s Algorithm

Initial State (with initial tentative distances)

- \( b.d = \infty \), \( b.\pi = NIL \)
- \( c.d = \infty \), \( c.\pi = NIL \)
- \( d.d = \infty \), \( d.\pi = NIL \)
- \( a.d = 0 \), \( a.\pi = NIL \)
- \( i.d = \infty \), \( i.\pi = NIL \)
- \( h.d = \infty \), \( h.\pi = NIL \)
- \( g.d = \infty \), \( g.\pi = NIL \)
- \( f.d = \infty \), \( f.\pi = NIL \)
SSSP: Dijkstra’s Algorithm

Initial State (with initial tentative distances)

- **a**. \(d = 0\), \(\pi = \text{NIL}\)
- **b**. \(d = \infty\), \(\pi = \text{NIL}\)
- **c**. \(d = \infty\), \(\pi = \text{NIL}\)
- **d**. \(d = \infty\), \(\pi = \text{NIL}\)
- **e**. \(d = \infty\), \(\pi = \text{NIL}\)
- **f**. \(d = \infty\), \(\pi = \text{NIL}\)
- **g**. \(d = \infty\), \(\pi = \text{NIL}\)
- **h**. \(d = \infty\), \(\pi = \text{NIL}\)
- **i**. \(d = \infty\), \(\pi = \text{NIL}\)
SSSP: Dijkstra’s Algorithm

Step 1: add vertex $a$ to SPT

- $b. d = \infty$
- $b. \pi =\text{NIL}$
- $c. d = \infty$
- $c. \pi =\text{NIL}$
- $d. d = \infty$
- $d. \pi =\text{NIL}$
- $a. d = 0$
- $a. \pi =\text{NIL}$
- $i. d = \infty$
- $i. \pi =\text{NIL}$
- $h. d = \infty$
- $h. \pi =\text{NIL}$
- $g. d = \infty$
- $g. \pi =\text{NIL}$
- $f. d = \infty$
- $f. \pi =\text{NIL}$
- $e. d = \infty$
- $e. \pi =\text{NIL}$
SSSP: Dijkstra’s Algorithm

Step 1’: update neighbors of \( a \)

- \( b.\pi = a \)
- \( b.d = 4 \)
- \( c.\pi = NIL \)
- \( c.d = \infty \)
- \( d.\pi = NIL \)
- \( d.d = \infty \)
- \( h.\pi = a \)
- \( h.d = 8 \)
- \( g.\pi = NIL \)
- \( g.d = \infty \)
- \( f.\pi = NIL \)
- \( f.d = \infty \)
- \( i.\pi = NIL \)
- \( i.d = \infty \)
**SSSP: Dijkstra’s Algorithm**

**Step 2: add vertex** $b$ **through edge** $(a, b)$

- **a.** $d = 0$
- **a.** $\pi = NIL$
- **b.** $d = 4$
- **b.** $\pi = a$
- **c.** $d = \infty$
- **c.** $\pi = NIL$
- **d.** $d = \infty$
- **d.** $\pi = NIL$
- **e.** $d = \infty$
- **e.** $\pi = NIL$
- **f.** $d = \infty$
- **f.** $\pi = NIL$
- **g.** $d = \infty$
- **g.** $\pi = NIL$
- **h.** $d = 8$
- **h.** $\pi = a$

Graph with nodes $a, b, c, d, e, f, g, h$ and edges with weights.
SSSP: Dijkstra’s Algorithm

Step 2’: update neighbors of \( b \)

- \( b.d = 4 \), \( b.\pi = a \)
- \( c.d = 12 \), \( c.\pi = b \)
- \( d.d = \infty \), \( d.\pi = NIL \)
- \( a.d = 0 \), \( a.\pi = NIL \)
- \( h.d = 8 \), \( h.\pi = a \)
- \( g.d = \infty \), \( g.\pi = NIL \)
- \( f.d = \infty \), \( f.\pi = NIL \)

Graph with nodes and edges:
- Node \( a \):
  - Distance: 0
  - Previous Node: NIL
- Node \( b \):
  - Distance: 4
  - Previous Node: \( a \)
- Node \( c \):
  - Distance: 12
  - Previous Node: \( b \)
- Node \( d \):
  - Distance: \infty
  - Previous Node: NIL
- Node \( e \):
  - Distance: \infty
  - Previous Node: NIL
- Node \( f \):
  - Distance: \infty
  - Previous Node: NIL
- Node \( g \):
  - Distance: \infty
  - Previous Node: NIL
- Node \( h \):
  - Distance: 8
  - Previous Node: \( a \)

Connections:
- \( a \) to \( b \) with distance 4
- \( a \) to \( h \) with distance 8
- \( b \) to \( c \) with distance 8
- \( b \) to \( d \) with distance \infty
- \( b \) to \( f \) with distance \infty
- \( c \) to \( d \) with distance 7
- \( c \) to \( i \) with distance 2
- \( d \) to \( e \) with distance 9
- \( d \) to \( f \) with distance 14
- \( f \) to \( e \) with distance 10
- \( i \) to \( g \) with distance 6
- \( i \) to \( h \) with distance 7
- \( i \) to \( e \) with distance 14
- \( g \) to \( h \) with distance 2
- \( g \) to \( f \) with distance \infty
- \( g \) to \( e \) with distance 14
- \( h \) to \( i \) with distance 7
- \( h \) to \( f \) with distance \infty
- \( i \) to \( c \) with distance 2
SSSP: Dijkstra’s Algorithm

Step 3: add vertex $h$ through edge $(a, h)$
SSSP: Dijkstra’s Algorithm

Step 3’: update neighbors of \( h \)

- \( b.d = 4 \)  \( b.\pi = a \)
- \( c.d = 12 \)  \( c.\pi = b \)
- \( d.d = \infty \)  \( d.\pi = NIL \)
- \( a.d = 0 \)  \( a.\pi = NIL \)
- \( h.d = 8 \)  \( h.\pi = a \)
- \( i.d = 15 \)  \( i.\pi = h \)
- \( g.d = 9 \)  \( g.\pi = h \)
- \( f.d = \infty \)  \( f.\pi = NIL \)
SSSP: Dijkstra’s Algorithm

Step 4: add vertex $g$ through edge $(h, g)$

- $b.d = 4$
- $b.\pi = a$
- $c.d = 12$
- $c.\pi = b$
- $d.d = \infty$
- $d.\pi = NIL$
- $a.d = 0$
- $a.\pi = NIL$
- $h.d = 8$
- $h.\pi = a$
- $i.d = 15$
- $i.\pi = h$
- $g.d = 9$
- $g.\pi = h$
- $f.d = \infty$
- $f.\pi = NIL$
- $e.d = \infty$
- $e.\pi = NIL$
SSSP: Dijkstra’s Algorithm

Step 4’: update neighbors of $g$

- $b$. $d = 4$
  - $b$. $\pi = a$
- $c$. $d = 12$
  - $c$. $\pi = b$
- $d$. $d = \infty$
  - $d$. $\pi = \text{NIL}$
- $a$. $d = 0$
  - $a$. $\pi = \text{NIL}$
- $h$. $d = 8$
  - $h$. $\pi = a$
- $g$. $d = 9$
  - $g$. $\pi = h$
- $i$. $d = 15$
  - $i$. $\pi = h$
- $e$. $d = \infty$
  - $e$. $\pi = \text{NIL}$
SSSP: Dijkstra’s Algorithm

Step 5: add vertex \( f \) through edge \((g, f)\)
SSSP: Dijkstra’s Algorithm

Step 5’: update neighbors of \( f \)

- \( b. d = 4 \), \( b. \pi = a \)
- \( c. d = 12 \), \( c. \pi = b \)
- \( d. d = 25 \), \( d. \pi = f \)
- \( a. d = 0 \), \( a. \pi = NIL \)
- \( i. d = 15 \), \( i. \pi = h \)
- \( h. d = 8 \), \( h. \pi = a \)
- \( g. d = 9 \), \( g. \pi = h \)
- \( f. d = 11 \), \( f. \pi = g \)
- \( e. d = 21 \), \( e. \pi = f \)
SSSP: Dijkstra’s Algorithm

Step 6: add vertex c through edge (b, c)
SSSP: Dijkstra’s Algorithm

**Step 6’: update neighbors of c**

- $b.d = 4$
  - $b.\pi = a$
- $c.d = 12$
  - $c.\pi = b$
- $d.d = 19$
  - $d.\pi = c$

![Graph diagram showing the updated neighbors of node c with distances and previous nodes]
SSSP: Dijkstra’s Algorithm

Step 7: add vertex $i$ through edge $(c, i)$
SSSP: Dijkstra’s Algorithm

Step 7’: update neighbors of i

- \( a.d = 0 \), \( a.\pi = NIL \)
- \( b.d = 4 \), \( b.\pi = a \)
- \( c.d = 12 \), \( c.\pi = b \)
- \( d.d = 19 \), \( d.\pi = c \)
- \( i.d = 14 \), \( i.\pi = c \)
- \( h.d = 8 \), \( h.\pi = a \)
- \( g.d = 9 \), \( g.\pi = h \)
- \( f.d = 11 \), \( f.\pi = g \)
- \( e.d = 21 \), \( e.\pi = f \)
SSSP: Dijkstra’s Algorithm

Step 8: add vertex $d$ through edge $(c, d)$

- $b.d = 4$, $b.\pi = a$
- $c.d = 12$, $c.\pi = b$
- $d.d = 19$, $d.\pi = c$
- $a.d = 0$, $a.\pi = NIL$
- $i.d = 14$, $i.\pi = c$
- $h.d = 8$, $h.\pi = a$
- $g.d = 9$, $g.\pi = h$
- $f.d = 11$, $f.\pi = g$
SSSP: Dijkstra’s Algorithm

Step 8’: update neighbors of \(d\)

- \(b. d = 4\), \(b. \pi = a\)
- \(c. d = 12\), \(c. \pi = b\)
- \(d. d = 19\), \(d. \pi = c\)

\(a. d = 0\), \(a. \pi = NIL\)

\(b. d = 4\), \(b. \pi = a\)
\(c. d = 12\), \(c. \pi = b\)
\(d. d = 19\), \(d. \pi = c\)

\(i. d = 14\), \(i. \pi = c\)

\(h. d = 8\), \(h. \pi = a\)
\(g. d = 9\), \(g. \pi = h\)
\(f. d = 11\), \(f. \pi = g\)

\(e. d = 21\), \(e. \pi = f\)
SSSP: Dijkstra’s Algorithm

Step 9: add vertex e through edge \((f, e)\)
SSSP: Dijkstra’s Algorithm

Step 9’: update neighbors of e

b. \( d = 4 \)
\( b. \pi = a \)

c. \( d = 12 \)
\( c. \pi = b \)

d. \( d = 19 \)
\( d. \pi = c \)

a. \( d = 0 \)
\( a. \pi = NIL \)

i. \( d = 14 \)
\( i. \pi = c \)

h. \( d = 8 \)
\( h. \pi = a \)

g. \( d = 9 \)
\( g. \pi = h \)

f. \( d = 11 \)
\( f. \pi = g \)

e. \( d = 21 \)
\( e. \pi = f \)
SSSP: Dijkstra’s Algorithm

Done

\(b.d = 4\)
\(b.\pi = a\)

\(c.d = 12\)
\(c.\pi = b\)

\(d.d = 19\)
\(d.\pi = c\)

\(a.d = 0\)
\(a.\pi = NIL\)

\(i.d = 14\)
\(i.\pi = c\)

\(e.d = 21\)
\(e.\pi = f\)
Dijkstra’s SSSP Algorithm with a Min-Heap
( SSSP: Single-Source Shortest Paths )

Input: Weighted graph $G = (V, E)$ with vertex set $V$ and edge set $E$, a weight function $w$, and a source vertex $s \in G[V]$.

Output: For all $v \in G[V]$, $v.d$ is set to the shortest distance from $s$ to $v$.

Let $n = |G[V]|$ and $m = |G[E]|$

# INSERTS $= n$
# EXTRACT-MINS $= n$
# DECREASE-KEYS $\leq m$

Total cost

\[ \leq n(cost_{\text{Insert}} + cost_{\text{Extract-Min}}) + m(cost_{\text{Decrease-Key}}) \]
Dijkstra’s SSSP Algorithm with a Min-Heap
(SSSP: Single-Source Shortest Paths)

Input: Weighted graph $G = (V, E)$ with vertex set $V$ and edge set $E$, a weight function $w$, and a source vertex $s \in G[V]$.  

Output: For all $v \in G[V]$, $v.d$ is set to the shortest distance from $s$ to $v$.

Let $n = |G[V]|$ and $m = |G[E]|$

For Binary Heap (worst-case costs):

- $cost_{\text{Insert}} = O(\log n)$
- $cost_{\text{Extract-Min}} = O(\log n)$
- $cost_{\text{Decrease-Key}} = O(\log n)$

\[ \therefore \text{Total cost (worst-case)} = O((m + n) \log n) \]
Dijkstra’s SSSP Algorithm with a Min-Heap  
*(SSSP: Single-Source Shortest Paths)*

**Input:** Weighted graph $G = (V, E)$ with vertex set $V$ and edge set $E$, a weight function $w$, and a source vertex $s \in G[V]$.

**Output:** For all $v \in G[V]$, $v.d$ is set to the shortest distance from $s$ to $v$.

---

**Dijkstra-SSSP** ($G = (V,E)$, $w$, $s$)
1. for each vertex $v \in G.V$ do  
2. \hspace{1cm} $v.d \leftarrow \infty$  
3. \hspace{1cm} $v.\pi \leftarrow NIL$  
4. \hspace{1cm} $s.d \leftarrow 0$  
5. \hspace{1cm} Min-Heap $Q \leftarrow \emptyset$  
6. for each vertex $v \in G.V$ do  
7. \hspace{2cm} INSERT($Q$, $v$)  
8. \hspace{1cm} while $Q \neq \emptyset$ do  
9. \hspace{2cm} $u \leftarrow \text{EXTRACT-MIN}(Q)$  
10. \hspace{2cm} for each $(u,v) \in G.E$ do  
11. \hspace{3cm} if $u.d + w(u,v) < v.d$ then  
12. \hspace{4cm} $v.d \leftarrow u.d + w(u,v)$  
13. \hspace{4cm} $v.\pi \leftarrow u$  
14. \hspace{4cm} DECREASE-KEY($Q$, $v$, $u.d + w(u,v)$)

Let $n = |G[V]|$ and $m = |G[E]|$

For Fibonacci Heap (amortized):

- $cost_{\text{Insert}} = O(1)$
- $cost_{\text{Extract-Min}} = O(\log n)$
- $cost_{\text{Decrease-Key}} = O(1)$

\[\therefore \text{Total cost (amortized)} = O(m + n \log n)\]
Optional
Kruskal’s MST algorithm
and a Union-Find data structure
with union by rank and path compression
A Disjoint-Set Data Structure (union by rank and path compression)

**MAKE-SET** \( x \)
1. \( \pi(x) \leftarrow x \)
2. \( \text{rank}(x) \leftarrow 0 \)

**LINK** \( x, y \)
1. if \( \text{rank}(x) > \text{rank}(y) \) then \( \pi(y) \leftarrow x \)
2. else \( \pi(x) \leftarrow y \)
3. if \( \text{rank}(x) = \text{rank}(y) \) then \( \text{rank}(y) \leftarrow \text{rank}(y) + 1 \)

**UNION** \( x, y \)
1. \( \text{LINK} \left( \text{FIND} \left( x \right), \text{FIND} \left( y \right) \right) \)

**FIND** \( x \)
1. if \( x \neq \pi(x) \) then \( \pi(x) \leftarrow \text{FIND} \left( \pi(x) \right) \)
2. return \( \pi(x) \)
A Disjoint-Set Data Structure
(union by rank and path compression)

**Theorem:** A sequence of \( N \) \texttt{MAKE-SET}, \texttt{UNION} and \texttt{FIND} operations of which exactly \( n \) \((\leq N)\) are \texttt{MAKE-SET} operations takes \( O(N\alpha(n)) \) time to execute, where \( \alpha(n) \) is the extremely slowly growing \textit{Inverse Ackermann Function} which has a value no larger than 3 for all practical values of \( n \).
# MST: Kruskal’s Algorithm (union by rank and path compression)

**Algorithm:**

\[ \text{MST-Kruskal}(G = (V, E), w) \]

1. \( A \leftarrow \emptyset \)
2. \( \text{for each vertex } v \in G.V \text{ do} \)
3. \( \text{MAKE-SET}(v) \)
4. \( \text{sort the edges of } G.E \text{ into nondecreasing order by weight } w \)
5. \( \text{for each edge } (u, v) \in G.E \text{ taken in nondecreasing order by weight do} \)
6. \( \text{if } \text{FIND}(u) \neq \text{FIND}(v) \text{ then} \)
7. \( A \leftarrow A \cup \{(u, v)\} \)
8. \( \text{UNION}(u, v) \)
9. \( \text{return } A \)

Let \( n = |V| \) and \( m = |E| \). Since \( G \) is connected, we have \( m \geq n - 1 \). Then the sorting in step 4 can be done in \( O(m \log m) \) time.

# disjoint-set operations performed, \( N = 2m + 2n - 1 \), of which

- \#MAKE-SET: \( n \)
- \#FIND: \( 2m \)
- \#UNION: \( n - 1 \)

So, total time taken by disjoint-set operations = \( O((n + m)\alpha(n)) \)

Hence, MST-Kruskal’s running time = \( O(m \log m) \)