CSE 548: Analysis of Algorithms

Prerequisites Review 6
(Greedy Algorithms)

Rezaul A. Chowdhury
Department of Computer Science
SUNY Stony Brook
Fall 2019
An Activity-Selection Problem

Suppose:
- You are given a set \( S = \{a_1, a_2, \ldots, a_n\} \) of \( n \) proposed \textit{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 \textit{start time} \( s_i \) and \textit{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 \textit{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 increasing order of finish time: \( f_1 \leq f_2 \leq f_3 \leq \cdots \leq f_{n-1} \leq f_n \).
An Activity-Selection Problem

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$i$</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>
</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>$i$</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>
</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 mutually compatible set of activities
An Activity-Selection Problem

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$i$</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>
</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>$i$</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>
</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: Optimal Substructure

Let $S_{ij} =$ set of activities that start after $a_i$ finishes and finishes before $a_j$ starts

$A_{ij} =$ a maximum set of mutually compatible activities in $S_{ij}$, which includes some activity $a_k$

Now by including $a_k$ in an optimal solution we are left with the following two subproblems:

- finding mutually compatible activities in $S_{ik}$
- finding mutually compatible activities in $S_{kj}$

Let $A_{ik} = A_{ij} \cap S_{ik}$ and $A_{kj} = A_{ij} \cap S_{kj}$.

Then $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$ and $|A_{ij}| = |A_{ik}| + |A_{kj}| + 1$.

The cut-and-paste argument shows that the optimal solution $A_{ij}$ must also include optimal solutions to subproblems for $S_{ik}$ and $S_{kj}$.
**Activity-Selection: Recurrence Relation**

We have, \( A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj} \) and \( |A_{ij}| = |A_{ik}| + |A_{kj}| + 1. \)

Let \( c[i, j] = \) size of an optimal solution for the set \( S_{ij}. \)

Then

\[
c[i, j] = \begin{cases} 
0, & \text{if } S_{ij} = \emptyset, \\
\max_{a_k \in S_{ij}} \{c[i, k] + c[k, j] + 1\}, & \text{if } S_{ij} \neq \emptyset.
\end{cases}
\]

Hence, we can use either recursion with memorization or bottom-up dynamic programming to solve the problem in \( \Theta(n^3) \) time.

Can we do better?
Activity-Selection: Improvement (Greedy Choice)

\[
c[i, j] = \begin{cases} 
0, & \text{if } S_{ij} = \emptyset, \\
\max_{a_k \in S_{ij}} \{c[i, k] + c[k, j] + 1\}, & \text{if } S_{ij} \neq \emptyset.
\end{cases}
\]

Instead of iterating over all \(a_k \in S_{ij}\) and checking solutions to subproblems for \(S_{ik}\) and \(S_{kj}\) to find the optimal \(a_k\), can we find the optimal \(a_k\) without even solving the subproblems?

Observe that among the activities we choose for our solution, one must be the first one to finish. Intuitively, therefore, we should choose the activity in the input with the earliest finish time, since that would leave the resource available for as many of the activities that follow it as possible.
**Activity-Selection: Improvement (Greedy Choice)**

Let’s consider choosing the activity in the input with the earliest finish time.

Since the activities set in the input $S = \{a_1, a_2, \ldots, a_n\}$ sorted in monotonically increasing order of finish time, i.e., $f_1 \leq f_2 \leq f_3 \leq \ldots \leq f_{n-1} \leq f_n$, we should choose $a_1$ to be in our solution.

Let $S_k = \{a_i \in S|s_i \geq f_k\}$, i.e., the set of activities that start after activity $a_k$ finishes.

If we make the greedy choice of activity $a_1$, then $S_1$ remains as the only subproblem to solve.

Optimal substructure tells us that if $a_1$ is in the optimal solution, then an optimal solution to the original problem consists of activity $a_1$ and all the activities in an optimal solution to the subproblem $S_1$.

But is the intuition correct?
**Activity-Selection: Improvement (Greedy Choice)**

**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, since we have shown that $a_m$ is in some maximum-size subset of mutually compatible activities of $S_k$.

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 $a_m$. 
**Activity-Selection: Recursive Algorithm**

**RECURSIVE-ACTIVITY-SELECTOR** \( (s, f, k, n) \)

1. \( m \leftarrow k + 1 \)
2. while \( m \leq n \) and \( s[m] < f[k] \)
3. \( m \leftarrow m + 1 \)
4. if \( m \leq n \) then
5. \( \text{return } \{a_m\} \cup \text{RECURSIVE-ACTIVITY-SELECTOR} \ (s, f, m, n) \)
6. else return \( \emptyset \)
Greedy Activity Selection

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$i$</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>
</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>

Diagram of activities:
- $a_1$, $a_2$, $a_3$, $a_4$, $a_5$, $a_6$, $a_7$, $a_8$, $a_9$, $a_{10}$, $a_{11}$
# Greedy Activity Selection

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$i$</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>
</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>

The activities $a_1$ to $a_{11}$ are shown in a Gantt chart, with $s_i$ and $f_i$ indicating the start and finish times respectively.
# Greedy Activity Selection

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$i$</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>
</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>

![Gantt Chart](chart.png)
Greedy Activity Selection

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$i$</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>
</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>

![Diagram of activities]

The diagram illustrates the activities $a_1$ to $a_{11}$ with their start times $s_i$ and finish times $f_i$. The selected activities are shaded in green, representing the optimal set of activities that can be scheduled without overlapping.
Greedy Activity Selection

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$i$</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>
</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>

Diagram:

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

a_1
a_2
a_3
a_4
a_5
a_6
a_7
a_8
a_9
a_{10}
a_{11}
```
## Greedy Activity Selection

### An example set $S$ of activities

<table>
<thead>
<tr>
<th>$i$</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>
</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>

Diagram:

- $a_1$: [Green Bar]
- $a_2$: [Dashed Bar]
- $a_3$: [Dashed Bar]
- $a_4$: [Green Bar]
- $a_5$: [Dashed Bar]
- $a_6$: [Dashed Bar]
- $a_7$: [Dashed Bar]
- $a_8$: [Green Bar]
- $a_9$: [Dashed Bar]
- $a_{10}$: [Dashed Bar]
- $a_{11}$: [Dashed Bar]
Greedy Activity Selection

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$i$</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>
</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>

Diagram of activities:

- $a_1$ from 1 to 5
- $a_2$ from 4 to 6
- $a_3$ from 2 to 7
- $a_4$ from 1 to 5
- $a_5$ from 3 to 8
- $a_6$ from 4 to 7
- $a_7$ from 2 to 8
- $a_8$ from 1 to 9
- $a_9$ from 3 to 11
- $a_{10}$ from 4 to 12
- $a_{11}$ from 5 to 13
Greedy Activity Selection

An example set $S$ of activities

<table>
<thead>
<tr>
<th>$i$</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>
</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>
Activity-Selection: Iterative Algorithm

\[
\text{GREEDY-ACTIVITY-SELECTOR} \ (s, f)
\]

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

Running time = \( \Theta(n) \)
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.

Our goal is to find an acyclic subset $T \subseteq E$ that connects all vertices of $V$ and whose total weight $w(T) = \sum_{(u,v) \in T} w(u, v)$ is minimized.

Since $T$ is acyclic and connects all of the vertices, it must form a tree, which we call a spanning tree since it “spans” the graph $G$.

We call the problem of determining the tree $T$ the minimum-spanning-tree problem.
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
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 $A \subseteq E$ 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**
MST: Greedy Strategy for Growing an MST

Generic-MST ( G = (V, E), w )
1. A ← ∅
2. while A does not form a spanning tree of G do
3. find an edge (u, v) ∈ E that is safe for A
4. A ← A ∪ {(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 \(V \setminus S\).

We say that 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.

Note that there can be more than one light edge crossing a cut in the case of ties.

More generally, we say that an edge is a light edge satisfying a given property if its weight is the minimum of any edge satisfying the property.
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.
Blue thick edges form set $A$, i.e.,
\[ A = \{(a, b), (c, f), (c, i), (f, g), (g, h)\}. \]
**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 $C$ to some other component of $G_A$, then edge $(u, v)$ is safe for $A$. 
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 (for Kruskal’s MST Algorithm)

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

**FIND** \( (x) \)
1. \( \text{if } x \neq \pi(x) \text{ then } \pi(x) \leftarrow \text{FIND} \left( \pi(x) \right) \)
2. \( \text{return } \pi(x) \)
A Disjoint-Set Data Structure (for Kruskal’s MST Algorithm)

**Theorem:** A sequence of $N$ **MAKE-SET**, **UNION** and **FIND** operations of which exactly $n \leq N$ are **MAKE-SET** operations takes $O(N\alpha(n))$ times to execute, where $\alpha(n)$ is the extremely slowly growing *Inverse Ackermann Function* which has a value no larger than 3 for all practical values of $n$.

**Proof:** We will prove this later in the semester.
MST: Kruskal’s Algorithm

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

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

Initial State

![Graph Diagram](image-url)
MST: Kruskal’s Algorithm

(1) edge \((h, g)\)
MST: Kruskal’s Algorithm

(1) edge \((h, g)\)
MST: Kruskal’s Algorithm

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

![Graph with labeled edges]
MST: Kruskal’s Algorithm

(2) edge \((i, c)\)
MST: Kruskal’s Algorithm

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

\[
\begin{align*}
\text{a} &\quad 11 &\quad \text{b} \\
\text{h} &\quad 8 &\quad \text{c} &\quad 8 &\quad \text{d} &\quad 7 \\
\text{g} &\quad 1 &\quad \text{e} &\quad 10 \\
\end{align*}
\]
MST: Kruskal’s Algorithm

(3) edge \((g, f)\)
MST: Kruskal’s Algorithm

(4) edge \((a, b)\)
MST: Kruskal’s Algorithm

(4) edge \((a, b)\)
MST: Kruskal’s Algorithm

(5) edge $(c, f)$
MST: Kruskal’s Algorithm

(5) edge \((c, f)\)
MST: Kruskal’s Algorithm

(6) edge \((i, g)\)
MST: Kruskal’s Algorithm

(6) edge \((i, g)\)
MST: Kruskal’s Algorithm

(7) edge \((c, d)\)
MST: Kruskal’s Algorithm

(7) edge (c, d)
MST: Kruskal’s Algorithm

(8) edge \((i, h)\)
MST: Kruskal’s Algorithm

(8) edge \((i, h)\)
MST: Kruskal’s Algorithm

(9) edge \((a, h)\)
(9) edge \((a, h)\)
MST: Kruskal’s Algorithm

(10) edge \((b, c)\)
MST: Kruskal’s Algorithm

(10) edge \((b, c)\)
MST: Kruskal’s Algorithm

(11) edge \((d, e)\)
MST: Kruskal’s Algorithm

(11) edge \((d, e)\)
MST: Kruskal’s Algorithm

(12) edge \((e, f)\)
MST: Kruskal’s Algorithm

(12) edge \((e, f)\)
MST: Kruskal’s Algorithm

(13) edge \((b, h)\)
MST: Kruskal’s Algorithm

(13) edge \((b, h)\)
MST: Kruskal’s Algorithm

(14) edge \((d, f)\)
MST: Kruskal’s Algorithm

(14) edge \((d, f)\)
MST: Kruskal’s Algorithm

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

Total weight = 37
**MST: Kruskal’s Algorithm**

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

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. \( \text{if FIND-SET}(u) \neq \text{FIND-SET}(v) \ \text{then} \)
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-SET: \( 2m \),  \#UNION: \( n - 1 \)

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

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

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

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

Initial State

Graph with nodes and edges labeled with weights.
Step 1: add vertex $a$ to MST
MST: Prim’s Algorithm

Step 1’: update neighbors of $a$
MST: Prim’s Algorithm

Step 2: add vertex $b$ through edge $(a, b)$
**MST: Prim’s Algorithm**

**Step 2’: update neighbors of** \( b \)
MST: Prim’s Algorithm

Step 3: add vertex $c$ through edge $(b, c)$
MST: Prim’s Algorithm

Step 3’: update neighbors of c
Step 4: add vertex $i$ through edge $(c, i)$
MST: Prim’s Algorithm

Step 4′: update neighbors of $i$
MST: Prim’s Algorithm

Step 5: add vertex $f$ through edge $(c, f)$
Step 5': update neighbors of $f$
Step 6: add vertex $g$ through edge $(f, g)$
**MST: Prim’s Algorithm**

Step 6': update neighbors of $g$
Step 7: add vertex $h$ through edge $(g, h)$
MST: Prim’s Algorithm

Step 7’: update neighbors of $h$
MST: Prim’s Algorithm

Step 8: add vertex d through edge (c, d)
MST: Prim’s Algorithm

Step 8’: update neighbors of $d$
MST: Prim’s Algorithm

Step 9: add vertex $e$ through edge $(d, e)$
Step 9′: update neighbors of $e$
MST: Prim’s Algorithm

Done

Total weight = 37
**MST: Prim’s Algorithm**

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

1. for each vertex \(v \in G.V\) do
2. \(v.d \leftarrow \infty\)
3. \(v.\pi \leftarrow \text{NIL}\)
4. \(r.d \leftarrow 0\)
5. Min-Heap \(Q \leftarrow \emptyset\)
6. for each vertex \(v \in G.V\) do
7. \(\text{INSERT}(Q, v)\)
8. while \(Q \neq \emptyset\) do
9. \(u \leftarrow \text{EXTRACT-MIN}(Q)\)
10. for each \((u,v) \in G.E\) 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. \(\text{DECREASE-KEY}(Q, v, w(u,v))\)

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

\# \text{INSERTS} = n
\# \text{EXTRACT-MINS} = n
\# \text{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**

Let \( n = |V| \) 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

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

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

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

For Fibonacci Heap ( amortized ):

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

\[ \therefore \text{Total cost ( amortized )} \]
\[ = O(m + n \log n) \]
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 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 \( v \in V \).
SSSP: Relaxation

**INITIALIZE-SINGLE-SOURCE** ( $G = (V, E), \ s$ )

1. for each vertex $v \in G.V$ do
2. $v \cdot d \leftarrow \infty$
3. $v \cdot \pi \leftarrow NIL$
4. $s \cdot d \leftarrow 0$

**RELAX** ( $u, v, w$ )

1. if $u \cdot d + w(u, v) < v \cdot d$ then
2. $v \cdot d \leftarrow u \cdot d + w(u, v)$
3. $v \cdot \pi \leftarrow u$
SSSP: Properties of Shortest Paths and Relaxation

The **weight** $w(p)$ of path $p = \langle v_0, v_1, \ldots, v_k \rangle$ is the sum of the weights of its constituent edges:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

We define the **shortest-path weight** $\delta(u, v)$ from $u$ to $v$ by

$$\delta(u, v) = \begin{cases} \min\{w(p) : p \text{ is } u \sim v\}, & \text{if there is a path from } u \text{ to } v, \\ \infty, & \text{otherwise.} \end{cases}$$

A **shortest path** from vertex $u$ to vertex $v$ is then defined as any path $p$ with weight $w(p) = \delta(u, v)$. 

91
SSSP: Properties of Shortest Paths and Relaxation

Triangle inequality (Lemma 24.10 of CLRS)
For any edge $(u, v) \in E$, we have $\delta(s, v) \leq \delta(s, u) + w(u, v)$.

Upper-bound inequality (Lemma 24.11 of CLRS)
We always have $v.d \geq \delta(s, v)$ for all vertices $v \in V$, and once $v.d$ achieves the value $\delta(u, v)$, it never changes.

No-path property (Corollary 24.12 of CLRS)
If there is no path from $s$ to $v$, then we always have $v.d = \delta(s, v) = \infty$.

Convergence property (Lemma 24.14 of CLRS)
If $s \leadsto u \to v$ is a shortest path in $G$ for some $u, v \in V$, and if $u.d = \delta(s, u)$ at any time prior to relaxing edge $(u, v)$, then $v.d = \delta(s, v)$ at all times afterward.
Path-relaxation property (Lemma 24.15 of CLRS)

If \( p = (v_0, v_1, \ldots, v_k) \) is a shortest path from \( s = v_0 \) to \( v_k \), and we relax the edges of \( p \) in the order \((v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)\), then \( v_k \cdot d = \delta(s, v_k) \). This property holds regardless of any other relaxation steps that occur, even if they are intermixed with relaxations on the edges of \( p \).

Predecessor-subgraph property (Lemma 24.17 of CLRS)

Once \( v \cdot d = \delta(s, v) \) for all \( v \in V \), the predecessor subgraph is a shortest-paths tree rooted at \( s \).
**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. $v.d \leftarrow \infty$
3. $v.\pi \leftarrow NIL$
4. $s.d \leftarrow 0$
5. Min-Heap $Q \leftarrow \emptyset$
6. for each vertex $v \in G.V$ do
7. \textsc{INSERT}( $Q$, $v$ )
8. while $Q \neq \emptyset$ do
9. $u \leftarrow \textsc{EXTRACT-MIN}( Q )$
10. for each $(u,v) \in G.E$ do
11. if $u.d + w(u,v) < v.d$ then
12. $v.d \leftarrow u.d + w(u,v)$
13. $v.\pi \leftarrow u$
14. \textsc{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

![Graph diagram with nodes and edges labeled with weights]
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
SSSP: Dijkstra’s Algorithm

One undirected edge $\Rightarrow$ Two directed edges
SSSP: Dijkstra’s Algorithm

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

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

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

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

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

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

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

Step 4: add vertex \( g \) through edge \((h, g)\)
SSSP: Dijkstra’s Algorithm

Step 4’: update neighbors of \( g \)
SSSP: Dijkstra’s Algorithm

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

Step 5’: update neighbors of f
SSSP: Dijkstra’s Algorithm

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

Step 6’: update neighbors of $c$
SSSP: Dijkstra’s Algorithm

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

Step 7’: update neighbors of i
SSSP: Dijkstra’s Algorithm

Step 8: add vertex $d$ through edge $(c, d)$
SSSP: Dijkstra’s Algorithm

Step 8’: update neighbors of d
SSSP: Dijkstra’s Algorithm

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

Step 9’: update neighbors of e
SSSP: Dijkstra’s Algorithm

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

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):

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

\[
\therefore \text{Total cost (worst-case)} = O((m + n) \log n)
\]
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. \( \text{for each vertex } v \in G.V \text{ do} \)
2. \( v.d \leftarrow \infty \)
3. \( v.\pi \leftarrow \text{NIL} \)
4. \( s.d \leftarrow 0 \)
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. \( \text{if } u.d + w(u,v) < v.d \text{ then} \)
12. \( v.d \leftarrow u.d + w(u,v) \)
13. \( v.\pi \leftarrow u \)
14. \( \text{DECREASE-KEY}(Q, v, u.d + w(u,v)) \)

---

Let \( n = |G[V]| \) and \( m = |G[E]| \)

For Fibonacci Heap (amortized):

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

\[\therefore \text{Total cost (amortized)} = O(m + n \log n)\]