LECTURE 16
Greedy Algorithms (and Graphs)
- Graph representation
- Minimum spanning trees
- Optimal substructure
- Greedy choice
- Prim’s greedy MST algorithm

Prof. Charles E. Leiserson
Graphs (review)

Definition. A directed graph (digraph) $G = (V, E)$ is an ordered pair consisting of
- a set $V$ of \textit{vertices} (singular: \textit{vertex}),
- a set $E \subseteq V \times V$ of \textit{edges}.

In an \textit{undirected graph} $G = (V, E)$, the edge set $E$ consists of \textit{unordered} pairs of vertices.

In either case, we have $|E| = O(V^2)$. Moreover, if $G$ is connected, then $|E| \geq |V| - 1$, which implies that $\lg |E| = \Theta(\lg V)$.

(Review CLRS, Appendix B.)
Adjacency-matrix representation

The *adjacency matrix* of a graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$, is the matrix $A[1 \ldots n, 1 \ldots n]$ given by

$$A[i, j] = \begin{cases} 
1 & \text{if } (i, j) \in E, \\
0 & \text{if } (i, j) \notin E.
\end{cases}$$
Adjacency-matrix representation

The **adjacency matrix** of a graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$, is the matrix $A[1 \ldots n, 1 \ldots n]$ given by

$$A[i, j] = \begin{cases} 1 & \text{if } (i, j) \in E, \\ 0 & \text{if } (i, j) \not\in E. \end{cases}$$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

$\Theta(V^2)$ storage \implies **dense** representation.
Adjacency-list representation

An *adjacency list* of a vertex $v \in V$ is the list $Adj[v]$ of vertices adjacent to $v$.

$Adj[1] = \{2, 3\}$
$Adj[2] = \{3\}$
$Adj[3] = \{\}$
$Adj[4] = \{3\}$
Adjacency-list representation

An **adjacency list** of a vertex \( v \in V \) is the list \( Adj[v] \) of vertices adjacent to \( v \).

- \( Adj[1] = \{2, 3\} \)
- \( Adj[2] = \{3\} \)
- \( Adj[3] = \{} \)
- \( Adj[4] = \{3\} \)

For undirected graphs, \(|Adj[v]| = \text{degree}(v)\).
For digraphs, \(|Adj[v]| = \text{out-degree}(v)\).
Adjacency-list representation

An **adjacency list** of a vertex \( v \in V \) is the list \( \text{Adj}[v] \) of vertices adjacent to \( v \).

\[
\begin{align*}
\text{Adj}[1] &= \{2, 3\} \\
\text{Adj}[2] &= \{3\} \\
\text{Adj}[3] &= \{} \\
\text{Adj}[4] &= \{3\}
\end{align*}
\]

For undirected graphs, \( |\text{Adj}[v]| = \text{degree}(v) \).
For digraphs, \( |\text{Adj}[v]| = \text{out-degree}(v) \).

**Handshaking Lemma:** \( \sum_{v \in V} = 2|E| \) for undirected graphs \( \Rightarrow \) adjacency lists use \( \Theta(V + E) \) storage — a **sparse** representation (for either type of graph).
Minimum spanning trees

**Input:** A connected, undirected graph $G = (V, E)$ with weight function $w : E \rightarrow \mathbb{R}$.

- For simplicity, assume that all edge weights are distinct. (CLRS covers the general case.)
Minimum spanning trees

**Input:** A connected, undirected graph \( G = (V, E) \) with weight function \( w : E \to \mathbb{R} \).

- For simplicity, assume that all edge weights are distinct. (CLRS covers the general case.)

**Output:** A *spanning tree* \( T \) — a tree that connects all vertices — of minimum weight:

\[
w(T) = \sum_{(u,v) \in T} w(u, v).
\]
Example of MST
Example of MST
Optimal substructure

MST $T$:

(Other edges of $G$ are not shown.)
**Optimal substructure**

**MST** $T$:

(Other edges of $G$ are not shown.)

Remove any edge $(u, v) \in T$. 
Optimal substructure

MST $T$:

(Other edges of $G$
are not shown.)

Remove any edge $(u, v) \in T$. 
Optimal substructure

MST $T$: (Other edges of $G$ are not shown.)

Remove any edge $(u, v) \in T$. Then, $T$ is partitioned into two subtrees $T_1$ and $T_2$. 
Optimal substructure

MST $T$:

(Other edges of $G$ are not shown.)

Remove any edge $(u, v) \in T$. Then, $T$ is partitioned into two subtrees $T_1$ and $T_2$.

**Theorem.** The subtree $T_1$ is an MST of $G_1 = (V_1, E_1)$, the subgraph of $G$ induced by the vertices of $T_1$:

$V_1 = \text{vertices of } T_1$,

$E_1 = \{(x, y) \in E : x, y \in V_1\}$.

Similarly for $T_2$. 
Proof of optimal substructure

Proof. Cut and paste:

\[ w(T) = w(u, v) + w(T_1) + w(T_2). \]

If \( T_1' \) were a lower-weight spanning tree than \( T_1 \) for \( G_1 \), then \( T' = \{(u, v)\} \cup T_1' \cup T_2 \) would be a lower-weight spanning tree than \( T \) for \( G \). □
Proof of optimal substructure

**Proof.** Cut and paste:

\[ w(T) = w(u, v) + w(T_1) + w(T_2). \]

If \( T_1' \) were a lower-weight spanning tree than \( T_1 \) for \( G_1 \), then \( T' = \{(u, v)\} \cup T_1' \cup T_2 \) would be a lower-weight spanning tree than \( T \) for \( G \).

Do we also have overlapping subproblems?

- Yes.
Proof of optimal substructure

*Proof.* Cut and paste:

\[ w(T) = w(u, v) + w(T_1) + w(T_2). \]

If \( T'_1 \) were a lower-weight spanning tree than \( T_1 \) for \( G_1 \), then \( T' = \{(u, v)\} \cup T'_1 \cup T_2 \) would be a lower-weight spanning tree than \( T \) for \( G \). □

Do we also have overlapping subproblems?

• Yes.

Great, then dynamic programming may work!

• Yes, but MST exhibits another powerful property which leads to an even more efficient algorithm.
Hallmark for “greedy” algorithms

**Greedy-choice property**

*A locally optimal choice is globally optimal.*
Hallmark for “greedy” algorithms

**Greedy-choice property**
*A locally optimal choice is globally optimal.*

**Theorem.** Let $T$ be the MST of $G = (V, E)$, and let $A \subseteq V$. Suppose that $(u, v) \in E$ is the least-weight edge connecting $A$ to $V - A$. Then, $(u, v) \in T$ or some other edge of equal weight is $\in T$
Proof of theorem

Proof. Suppose \((u, v) \notin T\) nor an edge of equal weight. Cut and paste.

\(T:\)

\(\in A\)

\(\in V - A\)

\((u, v) = \text{least-weight edge connecting } A \text{ to } V - A\)
Proof of theorem

**Proof.** Suppose \((u, v) \notin T\) nor an edge of equal weight.

Cut and paste.

\(T:\)

- \(\in A\)
- \(\in V - A\)

\((u, v) = \text{least-weight edge connecting } A \text{ to } V - A\)

Consider the unique simple path from \(u\) to \(v\) in \(T\).
Proof of theorem

Proof. Suppose \((u, v) \notin T\) nor an edge of equal weight. Cut and paste.

\[ T: \]

- \(\in A\)
- \(\in V - A\)

\((u, v) = \text{least-weight edge connecting } A \text{ to } V - A\)

Consider the unique simple path from \(u\) to \(v\) in \(T\).

Swap \((u, v)\) with the first edge on this path that connects a vertex in \(A\) to a vertex in \(V - A\).
Proof of theorem

Proof. Suppose \((u, v) \notin T\) nor an edge of equal weight. Cut and paste.

\(T':\)

- \(\in A\)
- \(\in V - A\)

Consider the unique simple path from \(u\) to \(v\) in \(T\). Swap \((u, v)\) with the first edge on this path that connects a vertex in \(A\) to a vertex in \(V - A\).

A lighter-weight spanning tree than \(T\) results.
Prim’s algorithm

**IDEA:** Maintain \( V - A \) as a priority queue \( Q \). Key each vertex in \( Q \) with the weight of the least-weight edge connecting it to a vertex in \( A \).

\[
\begin{align*}
Q & \leftarrow V \\
\text{key}[v] & \leftarrow \infty \text{ for all } v \in V \\
\text{key}[s] & \leftarrow 0 \text{ for some arbitrary } s \in V \\
\textbf{while} \ Q \neq \emptyset \\
\quad \textbf{do} \ u \leftarrow \text{EXTRACT-MIN}(Q) \\
\quad \quad \textbf{for each} \ v \in \text{Adj}[u] \\
\quad \quad \quad \textbf{do if} \ v \in Q \text{ and } w(u, v) < \text{key}[v] \\
\quad \quad \quad \quad \textbf{then} \ \text{key}[v] \leftarrow w(u, v) \quad \triangleright \text{DECREASE-KEY} \\
\quad \quad \quad \quad \pi[v] \leftarrow u \\
\end{align*}
\]

At the end, \( \{(v, \pi[v])\} \) forms the MST.
Example of Prim’s algorithm

\[ \in A \]

\[ \in V - A \]
Example of Prim’s algorithm

∈ A
∈ V − A
Example of Prim’s algorithm

\( \in A \)

\( \in V - A \)
Example of Prim’s algorithm

\[ \in A \]

\[ \in V - A \]
Example of Prim’s algorithm

\[ \in A \]
\[ \in V - A \]
Example of Prim’s algorithm

\( \in A \)

\( \in V - A \)
Example of Prim’s algorithm
Example of Prim’s algorithm
Example of Prim’s algorithm

\[ \in A \quad \in V - A \]
Example of Prim’s algorithm

\[ \in A \quad \in V - A \]
Example of Prim’s algorithm

\[ \in A \] 
\[ \in V - A \]
Example of Prim’s algorithm

\[ \in A \quad \in V - A \]
Example of Prim’s algorithm

\[ \in A \]

\[ \in V - A \]
Analysis of Prim

\[ Q \leftarrow V \]

\[ key[v] \leftarrow \infty \text{ for all } v \in V \]

\[ key[s] \leftarrow 0 \text{ for some arbitrary } s \in V \]

\textbf{while} \( Q \neq \emptyset \)

\textbf{do} \( u \leftarrow \text{EXTRACT-MIN}(Q) \)

\textbf{for} each \( v \in Adj[u] \)

\textbf{do if} \( v \in Q \text{ and } w(u, v) < key[v] \)

\textbf{then} \( key[v] \leftarrow w(u, v) \)

\( \pi[v] \leftarrow u \)
Analysis of Prim

\[ \Theta(V) \]

**total**

\[
\begin{align*}
Q & \leftarrow V \\
key[v] & \leftarrow \infty \text{ for all } v \in V \\
key[s] & \leftarrow 0 \text{ for some arbitrary } s \in V \\
\text{while } Q \neq \emptyset \\
do & u \leftarrow \text{EXTRACT-MIN}(Q) \\
\text{for each } v \in \text{Adj}[u] \\
do & \text{if } v \in Q \text{ and } w(u, v) < \key[v] \\
\text{then } & \key[v] \leftarrow w(u, v) \\
\pi[v] & \leftarrow u
\end{align*}
\]
Analysis of Prim

\[ \begin{align*}
\Theta(V) & \text{ total} \\
|V| & \text{ times}
\end{align*} \]

\[ Q \leftarrow V \]
\[ \text{key}[v] \leftarrow \infty \text{ for all } v \in V \]
\[ \text{key}[s] \leftarrow 0 \text{ for some arbitrary } s \in V \]

\begin{algorithm}
\caption{EXTRACT-MIN(Q)}
\end{algorithm}

\begin{algorithm}
\caption{Prim's Algorithm}
\end{algorithm}

\textbf{while} \( Q \neq \emptyset \)
\begin{align*}
\text{do } u & \leftarrow \text{EXTRACT-MIN}(Q) \\
\text{for each } v \in \text{Adj}[u] \text{ do if } v \in Q \text{ and } w(u, v) < \text{key}[v] \text{ then } \text{key}[v] & \leftarrow w(u, v) \\
\text{then } \text{key}[v] & \leftarrow w(u, v) \\
\pi[v] & \leftarrow u
\end{align*}
Analysis of Prim

\[ Q \leftarrow V \]
\[ \text{key}[^v] \leftarrow \infty \text{ for all } v \in V \]
\[ \text{key}[^s] \leftarrow 0 \text{ for some arbitrary } s \in V \]

\[ \text{while } Q \neq \emptyset \]
\[ \text{do } u \leftarrow \text{EXTRACT-MIN}(Q) \]

\[ \text{for each } v \in \text{Adj}[u] \]
\[ \text{do if } v \in Q \text{ and } w(u, v) < \text{key}[v] \]
\[ \text{then } \text{key}[v] \leftarrow w(u, v) \]
\[ \pi[v] \leftarrow u \]
Analysis of Prim

\[ \Theta(V) \] total
\[ Q \leftarrow V \]
\[ \text{key}[v] \leftarrow \infty \text{ for all } v \in V \]
\[ \text{key}[s] \leftarrow 0 \text{ for some arbitrary } s \in V \]
\[ \text{while } Q \neq \emptyset \]
\[ \text{do } u \leftarrow \text{EXTRACT-MIN}(Q) \]
\[ \text{for each } v \in \text{Adj}[u] \]
\[ \text{do if } v \in Q \text{ and } w(u, v) < \text{key}[v] \]
\[ \text{then } \text{key}[v] \leftarrow w(u, v) \]
\[ \pi[v] \leftarrow u \]

Handshaking Lemma \implies \Theta(E) \text{ implicit DECREASE-KEY’s.}
Analysis of Prim

\[ \Theta(V) \]

\[ Q \leftarrow V \]
\[ key[v] \leftarrow \infty \text{ for all } v \in V \]
\[ key[s] \leftarrow 0 \text{ for some arbitrary } s \in V \]

\[ \text{while } Q \neq \emptyset \]
\[ \text{do } u \leftarrow \text{EXTRACT-MIN}(Q) \]
\[ \text{for each } v \in \text{Adj}[u] \]
\[ \text{do if } v \in Q \text{ and } w(u, v) < key[v] \]
\[ \text{then } key[v] \leftarrow w(u, v) \]
\[ \pi[v] \leftarrow u \]

Handshaking Lemma \( \Rightarrow \Theta(E) \) implicit \text{DECREASE-KEY}'s.

Time = \( \Theta(V) \cdot T_{\text{EXTRACT-MIN}} + \Theta(E) \cdot T_{\text{DECREASE-KEY}} \)
Analysis of Prim (continued)

\[ \text{Time} = \Theta(V) \cdot T_{\text{EXTRACT-MIN}} + \Theta(E) \cdot T_{\text{DECREASE-KEY}} \]
Analysis of Prim (continued)

Time = \( \Theta(V) \cdot T_{\text{Extract-Min}} + \Theta(E) \cdot T_{\text{Decrease-Key}} \)

\[ \begin{array}{ccc}
Q & T_{\text{Extract-Min}} & T_{\text{Decrease-Key}} & \text{Total} \\
\end{array} \]
Analysis of Prim (continued)

\[ \text{Time} = \Theta(V) \cdot T_{\text{Extract-Min}} + \Theta(E) \cdot T_{\text{Decrease-Key}} \]

<table>
<thead>
<tr>
<th>Q</th>
<th>( T_{\text{Extract-Min}} )</th>
<th>( T_{\text{Decrease-Key}} )</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>( O(V) )</td>
<td>( O(1) )</td>
<td>( O(V^2) )</td>
</tr>
</tbody>
</table>
Analysis of Prim (continued)

\[
\text{Time} = \Theta(V) \cdot T_{\text{Extract-Min}} + \Theta(E) \cdot T_{\text{Decrease-Key}}
\]

<table>
<thead>
<tr>
<th>( Q )</th>
<th>( T_{\text{Extract-Min}} )</th>
<th>( T_{\text{Decrease-Key}} )</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>( O(V) )</td>
<td>( O(1) )</td>
<td>( O(V^2) )</td>
</tr>
<tr>
<td>binary heap</td>
<td>( O(\lg V) )</td>
<td>( O(\lg V) )</td>
<td>( O(E \lg V) )</td>
</tr>
<tr>
<td>heap</td>
<td>( O(V) )</td>
<td>( O(1) )</td>
<td>( O(V^2) )</td>
</tr>
</tbody>
</table>
Analysis of Prim (continued)

Time = \(\Theta(V) \cdot T_{\text{Extract-Min}} + \Theta(E) \cdot T_{\text{Decrease-Key}}\)

<table>
<thead>
<tr>
<th>(Q)</th>
<th>(T_{\text{Extract-Min}})</th>
<th>(T_{\text{Decrease-Key}})</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>(O(V))</td>
<td>(O(1))</td>
<td>(O(V^2))</td>
</tr>
<tr>
<td>binary heap</td>
<td>(O(\lg V))</td>
<td>(O(\lg V))</td>
<td>(O(E \lg V))</td>
</tr>
<tr>
<td>Fibonacci heap</td>
<td>(O(\lg V))</td>
<td>(O(1))</td>
<td>(O(E + V \lg V)) worst case</td>
</tr>
<tr>
<td>amortized</td>
<td></td>
<td>(amortized)</td>
<td></td>
</tr>
</tbody>
</table>
MST algorithms

Kruskal’s algorithm (see CLRS):
• Uses the *disjoint-set data structure* (Lecture 10).
• Running time = \( O(E \lg V) \).
MST algorithms

Kruskal’s algorithm (see CLRS):
• Uses the *disjoint-set data structure* (Lecture 10).
• Running time = \(O(E \lg V)\).

Best to date:
• Karger, Klein, and Tarjan [1993].
• Randomized algorithm.
• \(O(V + E)\) expected time.