LECTURE 11
Greedy Algorithms (and Graphs)
- Graph representation
- Minimum spanning trees
- Optimal substructure
- Greedy choice
- Prim’s greedy MST algorithm
Graphs (review)

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

In an *undirected graph* $G = (V, E)$, the edge set $E$ consists of *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 $\Rightarrow$ **dense** representation.
Adjacency-list representation

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

$\begin{align*}
\text{Adj}[1] &= \{2, 3\} \\
\text{Adj}[2] &= \{3\} \\
\text{Adj}[3] &= \emptyset \\
\text{Adj}[4] &= \{3\}
\end{align*}$
Adjacency-list representation

An **adjacency list** of a vertex \( v \in V \) is the list \( Adj[v] \) 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, \( |Adj[v]| = \text{degree}(v) \).
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 \( v \).

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 $w : E \rightarrow \mathbb{R}$.

- 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 $T_2$.  

Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson
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, $(u, v) \in T$. 

Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson
Proof of theorem

Proof. Suppose \((u, v) \notin T\). 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 \((u, v) \notin T\). Cut and paste.

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

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

Proof. Suppose \((u, v) \notin T\). Cut and paste.

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 \((u, v) \not\in T\). Cut and paste.

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 \( A \).

\[
Q \leftarrow V \\
key[v] \leftarrow \infty \text{ for all } v \in V \\
key[s] \leftarrow 0 \text{ for some arbitrary } s \in V \\
\text{while } Q \neq \emptyset \\
\quad \text{ do } u \leftarrow \text{EXTRACT-MIN}(Q) \\
\quad \quad \text{ for each } v \in \text{Adj}[u] \\
\quad \quad \quad \text{ do if } v \in Q \text{ and } w(u, v) < key[v] \\
\quad \quad \quad \quad \text{ then } key[v] \leftarrow w(u, v) \quad \triangleright \text{DECREASE-KEY} \\
\quad \quad \quad \quad \pi[v] \leftarrow u \\
\text{At the end, } \{(v, \pi[v])\} \text{ forms the MST.}
\]
Example of Prim’s algorithm
Example of Prim’s algorithm
Example of Prim’s algorithm
Example of Prim’s algorithm

\[ \begin{array}{c}
\in A \\
\in V - A
\end{array} \]
Example of Prim’s algorithm

\[ \in A \quad \in V - A \]

\[ 5 \quad 6 \quad 5 \quad 12 \quad 6 \quad 5 \quad 12 \]

\[ 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \]

\[ 10 \quad 14 \quad 8 \quad 7 \quad 15 \quad 10 \quad 15 \]

Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson
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

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

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

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

\[\text{\(\Theta(V)\) total}\]

\[
\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 \\
  \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) \\
  \text{\(\pi[v]\) } &\leftarrow u
\end{align*}
\]
Analysis of Prim

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

while \( Q \neq \emptyset \)

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

\( \Theta(V) \) total

\(|V| \) times
Analysis of Prim

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

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

\( \Theta(V) \) times
\( |V| \) times

degree(u) times

Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson
Analysis of Prim

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

while \( Q \neq \emptyset \)

\[ u \leftarrow \text{EXTRACT-MIN}(Q) \]

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.

\[ \Theta(V) \]

\[ |V| \]

\[ \text{degree}(u) \]
Analysis of Prim

\[ \Theta(V) \]

\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
\end{align*}

while \( Q \neq \emptyset \)

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

Handshaking Lemma \( \Rightarrow \Theta(E) \) implicit 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)

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

\[
\begin{array}{cccc}
Q & T_{\text{EXTRACT-MIN}} & T_{\text{DECREASE-KEY}} & \text{Total} \\
\end{array}
\]
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>
</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>
</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>$O(E + V \lg V)$</td>
</tr>
<tr>
<td></td>
<td>amortized</td>
<td>amortized</td>
<td>worst case</td>
</tr>
</tbody>
</table>
MST algorithms

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

- **Input:** A set of $n$ proposed *activities* that wish to use a resource, such as a lecture hall, which can be used by only one activity at a time. Each activity has a *start time* and a *finish time*.

- **Output:** To select a maximum-size subset of mutually compatible activities.
The Optimal Structure

• The optimal substructure of the problem:
  – An optimal solution $A_{ij}$ to $S_{ij}$ includes activities $a_k$. Then the solution $A_{ik}$ to $S_{ik}$ and $A_{kj}$ to $S_{kj}$ used within this optimal solution to $S_{ij}$ must be optimal as well.
Recursive Solution

\[ c[i, j] = 0 \quad \text{if} \quad S_{ij} = \emptyset, \]

or

\[ \max \{ c[i, k] + c[k, j] + 1 \} \quad \text{otherwise}. \]
Greedy Solution

• **Theorem 16.1** : Consider any nonempty subproblem $S_{ij}$, and let $a_m$ be the activities in $S_{ij}$ with the earliest finish time
  – Activities $a_m$ is used in some maximum-size subset of mutually compatible activities of $S_{ij}$.
  – The subproblem $S_{im}$ is empty, so that choosing $a_m$ leaves the subproblem $S_{mj}$ as the only one that may be nonempty.
Greedy Solution

Proof. (1) Suppose that $A_{ij}$ is a maximum-size subset of mutually compatible activities of $S_{ij}$, let us order the activities in $A_{ij}$ in monotonically increasing order of finish time. Let $a_k$ is the first activity in $A_{ij}$. If $a_k = a_m$, we are done. If $a_k = a_m$, we construct $A'_{ij} = A_{ij} - \{a_k\} U \{a_m\}$. The activities in $A'_{ij}$ are disjoint, since the activities in $A_{ij}$ are, $a_k$ is the first activity to finish, and $f_m < f_k$.  

Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson
Greedy Solution

Proof. (2) Suppose that $S_{im}$ is nonempty, so that there is some activity $a_k$ such that $f_i \leq s_k < f_k \leq s_m < f_m$. Then $a_k$ is also in $S_{ij}$ and it has an earlier finish time than $a_m$, which contradicts our choice of $a_m$. 

Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson
Greedy Algorithm

**GREEDY-ACTIVITY-SELECTOR***(s,f)***

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