Outline

1. **Minimum Spanning Tree**
   - Kruskal’s Algorithm
   - Reverse-Kruskal’s Algorithm
   - Prim’s Algorithm

2. **Single Source Shortest Paths**
   - Dijkstra’s Algorithm

3. **Shortest Paths in Graphs with Negative Weights**

4. **All-Pair Shortest Paths and Floyd-Warshall**
Recall the greedy strategy for Kruskal’s algorithm: choose the edge with the smallest weight.
Recall the greedy strategy for Kruskal’s algorithm: choose the edge with the smallest weight.

Greedy strategy for Prim’s algorithm: choose the lightest edge incident to $a$. 
Recall the greedy strategy for Kruskal’s algorithm: choose the edge with the smallest weight.

Greedy strategy for Prim’s algorithm: choose the lightest edge incident to $a$.
Lemma  It is safe to include the lightest edge incident to $a$. 
Lemma: It is safe to include the lightest edge incident to \( a \).

Proof.

- Let \( T \) be a MST
- Consider all components obtained by removing \( a \) from \( T \)
**Lemma** It is safe to include the lightest edge incident to \( a \).

Proof.

- Let \( T \) be a MST.
- Consider all components obtained by removing \( a \) from \( T \).
- Let \( e^* \) be the lightest edge incident to \( a \) and \( e^* \) connects \( a \) to component \( C \).
**Lemma** It is safe to include the lightest edge incident to $a$.

**Proof.**

- Let $T$ be a MST
- Consider all components obtained by removing $a$ from $T$
- Let $e^*$ be the lightest edge incident to $a$ and $e^*$ connects $a$ to component $C$
- Let $e$ be the edge in $T$ connecting $a$ to $C$
**Lemma** It is safe to include the lightest edge incident to $a$.

**Proof.**

- Let $T$ be a MST
- Consider all components obtained by removing $a$ from $T$
- Let $e^*$ be the lightest edge incident to $a$ and $e^*$ connects $a$ to component $C$
- Let $e$ be the edge in $T$ connecting $a$ to $C$
- $T' = T \setminus \{e\} \cup \{e^*\}$ is a spanning tree with $w(T') \leq w(T)$
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Prim’s Algorithm: Example

The image shows a graph with nodes labeled a, b, c, d, e, f, g, h, and i, connected by edges with weights. The graph starts with node a and proceeds to b, c, d, f, e, g, and h, highlighting the Prim’s algorithm process.
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Prim’s Algorithm: Example
Greedy Algorithm

MST-Greedy1(\(G, w\))

1: \(S \leftarrow \{s\}\), where \(s\) is arbitrary vertex in \(V\)
2: \(F \leftarrow \emptyset\)
3: while \(S \neq V\) do
4: \((u, v) \leftarrow\) lightest edge between \(S\) and \(V \setminus S\), where \(u \in S\) and \(v \in V \setminus S\)
5: \(S \leftarrow S \cup \{v\}\)
6: \(F \leftarrow F \cup \{(u, v)\}\)
7: return \((V, F)\)
Greedy Algorithm

MST-Greedy1(G, w)

1: \( S \leftarrow \{s\} \), where \( s \) is arbitrary vertex in \( V \)
2: \( F \leftarrow \emptyset \)
3: \textbf{while} \( S \neq V \) \textbf{do}
4: \((u, v) \leftarrow \text{lightest edge between } S \text{ and } V \setminus S \),
   \hspace{1cm} \text{where } u \in S \text{ and } v \in V \setminus S
5: \( S \leftarrow S \cup \{v\} \)
6: \( F \leftarrow F \cup \{(u, v)\} \)
7: \textbf{return} \((V, F)\)

Running time of naive implementation: \( O(nm) \)
Prim’s Algorithm: Efficient Implementation of Greedy Algorithm

For every $v \in V \setminus S$ maintain

- $d[v] = \min_{u \in S: \{u, v\} \in E} w(u, v)$: the weight of the lightest edge between $v$ and $S$
- $\pi[v] = \arg \min_{u \in S: \{u, v\} \in E} w(u, v)$: $(\pi[v], v)$ is the lightest edge between $v$ and $S$
Prim’s Algorithm: Efficient Implementation of Greedy Algorithm

For every \( v \in V \setminus S \) maintain

- \( d[v] = \min_{u \in S : (u,v) \in E} w(u,v) \): the weight of the lightest edge between \( v \) and \( S \)
- \( \pi[v] = \arg\min_{u \in S : (u,v) \in E} w(u,v) \): \( (\pi[v],v) \) is the lightest edge between \( v \) and \( S \)

In every iteration

- Pick \( u \in V \setminus S \) with the smallest \( d[u] \) value
- Add \( (\pi[u],u) \) to \( F \)
- Add \( u \) to \( S \), update \( d \) and \( \pi \) values.
Prim’s Algorithm

**MST-Prim**\((G, w)\)

1: \(s \leftarrow \text{arbitrary vertex in } G\)
2: \(S \leftarrow \emptyset, d(s) \leftarrow 0 \text{ and } d[v] \leftarrow \infty \text{ for every } v \in V \setminus \{s\}\)
3: while \(S \neq V\) do
4: \(u \leftarrow \text{vertex in } V \setminus S \text{ with the minimum } d[u]\)
5: \(S \leftarrow S \cup \{u\}\)
6: for each \(v \in V \setminus S\) such that \((u, v) \in E\) do
7: if \(w(u, v) < d[v]\) then
8: \(d[v] \leftarrow w(u, v)\)
9: \(\pi[v] \leftarrow u\)
10: return \(\{(u, \pi[u]) | u \in V \setminus \{s\}\}\)
Example
Example
Example

(5, a)

(12, a)
Example
Example
Example

A graph with labeled vertices and edges, showing connections among nodes a, b, c, d, e, f, g, h, and i.
Example

\begin{itemize}
\item $(13, c)$
\item $(11, b)$
\item $(2, c)$
\item $(4, c)$
\end{itemize}
Example
Example

\begin{figure}
\centering
\begin{tikzpicture}
\node[shape=circle,draw=black] (a) at (0,0) {$a$};
\node[shape=circle,draw=black] (b) at (1,1) {$b$};
\node[shape=circle,draw=black] (c) at (2,1) {$c$};
\node[shape=circle,draw=black] (i) at (1,-1) {$i$};
\node[shape=circle,draw=black] (h) at (-1,0) {$h$};
\node[shape=circle,draw=black] (g) at (0,-2) {$g$};
\node[shape=circle,draw=black] (d) at (3,0) {$d$};
\node[shape=circle,draw=black] (f) at (3,-2) {$f$};
\node[shape=circle,draw=black] (e) at (4,0) {$e$};

\draw[->,thick](a) to node [above]{$5$} (h);
\draw[->,thick](b) to node [left]{$8$} (c);
\draw[->,thick](i) to node [left]{$2$} (c);
\draw[->,thick](h) to node [below]{$7$} (i);
\draw[->,thick](i) to node [below]{$6$} (g);
\draw[->,thick](h) to node [right]{$11$} (a);
\draw[->,thick](a) to node [below]{$12$} (h);
\draw[->,thick](b) to node [below]{$11$} (i);
\draw[->,thick](g) to node [below]{$1$} (f);
\draw[->,thick](c) to node [above]{$13$} (d);
\draw[->,thick](f) to node [below]{$3$} (e);
\draw[->,thick](e) to node [above]{$10$} (f);
\draw[->,thick](d) to node [below]{$9$} (e);

\node at (1.5,1) {(13, $c$)};
\node at (-1.5,-1) {(11, $b$)};
\node at (1.5,-1) {(2, $c$)};
\end{tikzpicture}
\end{figure}
Example
Example
Example
Example
Example
Example
Example
Example

(13, c)

(10, f)
Example

(1, g)

(10, f)

(13, c)
Example
Example
Example
Example
Example
Example
Prim’s Algorithm

For every $v \in V \setminus S$ maintain

- $d[v] = \min_{u \in S : (u,v) \in E} w(u,v)$:
  - the weight of the lightest edge between $v$ and $S$
- $\pi[v] = \arg \min_{u \in S : (u,v) \in E} w(u,v)$:
  - $(\pi[v], v)$ is the lightest edge between $v$ and $S$

In every iteration

- Pick $u \in V \setminus S$ with the smallest $d[u]$ value
- Add $(\pi[u], u)$ to $F$
- Add $u$ to $S$, update $d$ and $\pi$ values.
Prim’s Algorithm

For every \( v \in V \setminus S \) maintain

- \( d[v] = \min_{u \in S : (u, v) \in E} w(u, v) \): the weight of the lightest edge between \( v \) and \( S \)
- \( \pi[v] = \arg \min_{u \in S : (u, v) \in E} w(u, v) \): \( (\pi[v], v) \) is the lightest edge between \( v \) and \( S \)

In every iteration

- Pick \( u \in V \setminus S \) with the smallest \( d[u] \) value \( \text{extract min} \)
- Add \( (\pi[u], u) \) to \( F \)
- Add \( u \) to \( S \), update \( d \) and \( \pi \) values \( \text{decrease_key} \)

Use a priority queue to support the operations
Def. A priority queue is an abstract data structure that maintains a set $U$ of elements, each with an associated key value, and supports the following operations:

- $\text{insert}(v, \text{key}_\text{value})$: insert an element $v$, whose associated key value is $\text{key}_\text{value}$.
- $\text{decrease_key}(v, \text{new}_\text{key}_\text{value})$: decrease the key value of an element $v$ in queue to $\text{new}_\text{key}_\text{value}$
- $\text{extract_min}()$: return and remove the element in queue with the smallest key value
- ...
Prim’s Algorithm

MST-Prim\((G, w)\)

1: \( s \leftarrow \text{arbitrary vertex in } G \)
2: \( S \leftarrow \emptyset, d(s) \leftarrow 0 \) and \( d[v] \leftarrow \infty \) for every \( v \in V \setminus \{s\} \)
3:
4: \textbf{while} \( S \neq V \) \textbf{do}
5: \( u \leftarrow \text{vertex in } V \setminus S \) with the minimum \( d[u] \)
6: \( S \leftarrow S \cup \{u\} \)
7: \textbf{for} each \( v \in V \setminus S \) such that \((u, v) \in E\) \textbf{do}
8: \hspace{1em} \textbf{if} \( w(u, v) < d[v] \) \textbf{then}
9: \hspace{2em} \( d[v] \leftarrow w(u, v) \)
10: \hspace{2em} \( \pi[v] \leftarrow u \)
11: \textbf{return} \( \{(u, \pi[u])|u \in V \setminus \{s\}\} \)
Prim’s Algorithm Using Priority Queue

**MST-Prim**($G, w$)

1: \( s \leftarrow \) arbitrary vertex in \( G \)
2: \( S \leftarrow \emptyset, d(s) \leftarrow 0 \) and \( d[v] \leftarrow \infty \) for every \( v \in V \setminus \{s\} \)
3: \( Q \leftarrow \) empty queue, for each \( v \in V : Q.\text{insert}(v, d[v]) \)
4: \textbf{while} \( S \neq V \) \textbf{do}
5: \( u \leftarrow Q.\text{extract}_\text{min}() \)
6: \( S \leftarrow S \cup \{u\} \)
7: \textbf{for each} \( v \in V \setminus S \) \textbf{such that} \( (u, v) \in E \) \textbf{do}
8: \quad \textbf{if} \( w(u, v) < d[v] \) \textbf{then}
9: \quad \quad \( d[v] \leftarrow w(u, v), Q.\text{decrease}_\text{key}(v, d[v]) \)
10: \quad \( \pi[v] \leftarrow u \)
11: \textbf{return} \( \{(u, \pi[u])|u \in V \setminus \{s\}\} \)
Running Time of Prim’s Algorithm Using Priority Queue

\[ O(n) \times \text{(time for extract\_min)} + O(m) \times \text{(time for decrease\_key)} \]
Running Time of Prim’s Algorithm Using Priority Queue

\[ O(n) \times \text{(time for extract\_min)} + O(m) \times \text{(time for decrease\_key)} \]

<table>
<thead>
<tr>
<th>Concrete DS</th>
<th>extract_min</th>
<th>decrease_key</th>
<th>Overall Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td>(O(\log n))</td>
<td>(O(\log n))</td>
<td>(O(m \log n))</td>
</tr>
<tr>
<td>Fibonacci heap</td>
<td>(O(\log n))</td>
<td>(O(1))</td>
<td>(O(n \log n + m))</td>
</tr>
</tbody>
</table>
Running Time of Prim’s Algorithm Using Priority Queue

\[ O(n) \times \text{(time for extract\_min)} + O(m) \times \text{(time for decrease\_key)} \]

<table>
<thead>
<tr>
<th>concrete DS</th>
<th>extract_min</th>
<th>decrease_key</th>
<th>overall time</th>
</tr>
</thead>
<tbody>
<tr>
<td>heap</td>
<td>(O(\log n))</td>
<td>(O(\log n))</td>
<td>(O(m \log n))</td>
</tr>
<tr>
<td>Fibonacci heap</td>
<td>(O(\log n))</td>
<td>(O(1))</td>
<td>(O(n \log n + m))</td>
</tr>
</tbody>
</table>