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>
Assumption  Assume all edge weights are different.

Lemma  \((u, v)\) is in MST, if and only if there exists a cut \((U, V \setminus U)\), such that \((u, v)\) is the lightest edge between \(U\) and \(V \setminus U\).
**Assumption**  Assume all edge weights are different.

**Lemma**  \((u,v)\) is in MST, if and only if there exists a cut \((U, V \setminus U)\), such that \((u,v)\) is the lightest edge between \(U\) and \(V \setminus U\).

\[\begin{array}{c}
(a, b) \quad 5 \\
(b, c) \quad 8 \\
(c, f) \quad 13 \\
(f, g) \quad 2 \\
(g, h) \quad 7 \\
(h, a) \quad 11 \\
\end{array}\]

\((c, f)\) is in MST because of cut \((\{a, b, c, i\}, V \setminus \{a, b, c, i\})\)
**Assumption** Assume all edge weights are different.

**Lemma** \((u, v)\) is in MST, if and only if there exists a cut \((U, V \setminus U)\), such that \((u, v)\) is the lightest edge between \(U\) and \(V \setminus U\).

- \((c, f)\) is in MST because of cut \((\{a, b, c, i\}, V \setminus \{a, b, c, i\})\)
- \((i, g)\) is not in MST because no such cut exists
“Evidence” for $e \in \text{MST}$ or $e \notin \text{MST}$

**Assumption**  Assume all edge weights are different.

- $e \in \text{MST} \iff$ there is a cut in which $e$ is the lightest edge
- $e \notin \text{MST} \iff$ there is a cycle in which $e$ is the heaviest edge
“Evidence” for $e \in \text{MST}$ or $e \notin \text{MST}$

**Assumption**  Assume all edge weights are different.

- $e \in \text{MST} \iff$ there is a cut in which $e$ is the lightest edge
- $e \notin \text{MST} \iff$ there is a cycle in which $e$ is the heaviest edge

Exactly one of the following is true:

- There is a cut in which $e$ is the lightest edge
- There is a cycle in which $e$ is the heaviest edge
Assumption Assume all edge weights are different.

- \( e \in \text{MST} \iff \) there is a cut in which \( e \) is the lightest edge
- \( e \notin \text{MST} \iff \) there is a cycle in which \( e \) is the heaviest edge

Exactly one of the following is true:

- There is a cut in which \( e \) is the lightest edge
- There is a cycle in which \( e \) is the heaviest edge

Thus, the minimum spanning tree is unique with assumption.
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
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Graph</th>
<th>Weights</th>
<th>SS?</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple DP</td>
<td>DAG</td>
<td>$\mathbb{R}$</td>
<td>SS</td>
<td>$O(n + m)$</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>U/D</td>
<td>$\mathbb{R}_{\geq 0}$</td>
<td>SS</td>
<td>$O(n \log n + m)$</td>
</tr>
<tr>
<td>Bellman-Ford</td>
<td>U/D</td>
<td>$\mathbb{R}$</td>
<td>SS</td>
<td>$O(nm)$</td>
</tr>
<tr>
<td>Floyd-Warshall</td>
<td>U/D</td>
<td>$\mathbb{R}$</td>
<td>AP</td>
<td>$O(n^3)$</td>
</tr>
</tbody>
</table>

- $\text{DAG} = \text{directed acyclic graph}$
- $\text{U} = \text{undirected}$
- $\text{D} = \text{directed}$
- $\text{SS} = \text{single source}$
- $\text{AP} = \text{all pairs}$
**s-t Shortest Paths**

**Input:** (directed or undirected) graph $G = (V, E)$, $s, t \in V$

$w : E \rightarrow \mathbb{R}_{\geq 0}$

**Output:** shortest path from $s$ to $t$
**s-t Shortest Paths**

**Input:** (directed or undirected) graph \( G = (V, E), \ s, t \in V \)
\[ w : E \to \mathbb{R}_{\geq 0} \]

**Output:** shortest path from \( s \) to \( t \)
**s-t Shortest Paths**

**Input:** (directed or undirected) graph $G = (V, E)$, $s, t \in V$

$$w : E \rightarrow \mathbb{R}_{\geq 0}$$

**Output:** shortest path from $s$ to $t$
Single Source Shortest Paths

**Input:** (directed or undirected) graph $G = (V, E)$, $s \in V$

$w : E \rightarrow \mathbb{R}_{\geq 0}$

**Output:** shortest paths from $s$ to all other vertices $v \in V$
Single Source Shortest Paths

**Input:** (directed or undirected) graph $G = (V, E)$, $s \in V$

$w : E \rightarrow \mathbb{R}_{\geq 0}$

**Output:** shortest paths from $s$ to all other vertices $v \in V$

Reason for Considering Single Source Shortest Paths Problem

- We do not know how to solve $s$-$t$ shortest path problem more efficiently than solving single source shortest path problem
Single Source Shortest Paths

**Input:** (directed or undirected) graph $G = (V, E)$, $s \in V$

$w : E \rightarrow \mathbb{R}_{\geq 0}$

**Output:** shortest paths from $s$ to all other vertices $v \in V$

Reason for Considering Single Source Shortest Paths Problem

- We do not know how to solve $s$-$t$ shortest path problem more efficiently than solving single source shortest path problem

- Shortest paths in directed graphs is more general than in undirected graphs: we can replace every undirected edge with two anti-parallel edges of the same weight
Single Source Shortest Paths

**Input:** (directed or undirected) graph \( G = (V, E) \), \( s \in V \)

\[ w : E \to \mathbb{R}_{\geq 0} \]

**Output:** shortest paths from \( s \) to all other vertices \( v \in V \)

Reason for Considering Single Source Shortest Paths

- We do not know how to solve \( s-t \) shortest path problem more efficiently than solving single source shortest path problem
- Shortest paths in directed graphs is more general than in undirected graphs: we can replace every undirected edge with two anti-parallel edges of the same weight
Single Source Shortest Paths

**Input:** directed graph $G = (V, E)$, $s \in V$

$w : E \rightarrow \mathbb{R}_{\geq 0}$

**Output:** shortest paths from $s$ to all other vertices $v \in V$

Reason for Considering Single Source Shortest Paths

- We do not know how to solve $s$-$t$ shortest path problem more efficiently than solving single source shortest path problem
- Shortest paths in directed graphs is more general than in undirected graphs: we can replace every undirected edge with two anti-parallel edges of the same weight
Single Source Shortest Paths

**Input:** directed graph $G = (V, E)$, $s \in V$

$w : E \rightarrow \mathbb{R}_{\geq 0}$

**Output:** $\pi[v], v \in V \setminus s$: the parent of $v$ in shortest path tree

$d[v], v \in V \setminus s$: the length of shortest path from $s$ to $v$
Q: How to compute shortest paths from $s$ when all edges have weight 1?
Q: How to compute shortest paths from $s$ when all edges have weight 1?

A: Breadth first search (BFS) from source $s$
Q: How to compute shortest paths from $s$ when all edges have weight 1?

A: Breadth first search (BFS) from source $s$
Q: How to compute shortest paths from $s$ when all edges have weight 1?

A: Breadth first search (BFS) from source $s$
Q: How to compute shortest paths from $s$ when all edges have weight 1?

A: Breadth first search (BFS) from source $s$
Q: How to compute shortest paths from $s$ when all edges have weight 1?

A: Breadth first search (BFS) from source $s$
Q: How to compute shortest paths from $s$ when all edges have weight 1?

A: Breadth first search (BFS) from source $s$
Assumption  Weights $w(u, v)$ are integers (w.l.o.g.).
Assumption  Weights $w(u, v)$ are integers (w.l.o.g).

- An edge of weight $w(u, v)$ is equivalent to a path of $w(u, v)$ unit-weight edges.
Assumption  Weights $w(u, v)$ are integers (w.l.o.g).

- An edge of weight $w(u, v)$ is equivalent to a path of $w(u, v)$ unit-weight edges

![Diagram showing equivalent paths]

Shortest Path Algorithm by Running BFS

1: replace $(u, v)$ of length $w(u, v)$ with a path of $w(u, v)$ unit-weight edges, for every $(u, v) \in E$
2: run BFS
3: $\pi[v] \leftarrow$ vertex from which $v$ is visited
4: $d[v] \leftarrow$ index of the level containing $v$
**Assumption** Weights $w(u, v)$ are integers (w.l.o.g).

- An edge of weight $w(u, v)$ is equivalent to a path of $w(u, v)$ unit-weight edges.

![Diagram of an edge and a path](image)

**Shortest Path Algorithm by Running BFS**

1. replace $(u, v)$ of length $w(u, v)$ with a path of $w(u, v)$ unit-weight edges, for every $(u, v) \in E$
2. run BFS
3. $\pi[v] \leftarrow$ vertex from which $v$ is visited
4. $d[v] \leftarrow$ index of the level containing $v$

- Problem: $w(u, v)$ may be too large!
Assumption  Weights $w(u, v)$ are integers (w.l.o.g).

- An edge of weight $w(u, v)$ is equivalent to a path of $w(u, v)$ unit-weight edges

\[
\begin{align*}
\text{Shortest Path Algorithm by Running BFS} \\
1: & \text{ replace } (u, v) \text{ of length } w(u, v) \text{ with a path of } w(u, v) \text{ unit-weight edges, for every } (u, v) \in E \\
2: & \text{ run BFS virtually} \\
3: & \pi[v] \leftarrow \text{ vertex from which } v \text{ is visited} \\
4: & d[v] \leftarrow \text{ index of the level containing } v
\end{align*}
\]

- Problem: $w(u, v)$ may be too large!
Shortest Path Algorithm by Running BFS Virtually

1: \( S \leftarrow \{s\}, d(s) \leftarrow 0 \)
2: \textbf{while } |S| \leq n \textbf{ do}
3: \text{find a } v \notin S \text{ that minimizes } \min_{u \in S: (u,v) \in E} \{d[u] + w(u, v)\}
4: \( S \leftarrow S \cup \{v\} \)
5: \( d[v] \leftarrow \min_{u \in S: (u,v) \in E} \{d[u] + w(u, v)\} \)
Virtual BFS: Example
Virtual BFS: Example

Time 0
Virtual BFS: Example

Time 2
Virtual BFS: Example

Time 4
Virtual BFS: Example

Time 7
Virtual BFS: Example

Time 9
Virtual BFS: Example

Time 10
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
Dijkstra’s Algorithm

Dijkstra\((G, w, s)\)

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

Running time = \( O(n^2) \)
Dijkstra's Algorithm

Dijkstra($G, w, s$)

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

- Running time $= O(n^2)$
Improved Running Time using Priority Queue

Dijkstra\((G, w, s)\)

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: \(Q \leftarrow \text{empty queue, for each } v \in V:\ Q.\text{insert}(v, d[v])\)
4: \textbf{while } S \neq V \textbf{ do}
5: \( u \leftarrow Q.\text{extract\_min()}\)
6: \( S \leftarrow S \cup \{u\}\)
7: \textbf{for each } v \in V \setminus S \text{ such that } (u, v) \in E \textbf{ do}
8: \quad \textbf{if } d[u] + w(u, v) < d[v] \textbf{ then}
9: \quad \quad d[v] \leftarrow d[u] + w(u, v), \ Q.\text{decrease\_key}(v, d[v])
10: \quad \pi[v] \leftarrow u
11: \textbf{return } (\pi, d)
Recall: Prim’s Algorithm for MST

\textbf{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: \( Q \leftarrow \text{empty queue, for each } v \in V: Q.\text{insert}(v, d[v]) \)
4: \( \textbf{while } S \neq V \textbf{ do} \)
5: \( u \leftarrow Q.\text{extract\_min}() \)
6: \( S \leftarrow S \cup \{u\} \)
7: \( \textbf{for each } v \in V \setminus S \text{ such that } (u, v) \in E \textbf{ do} \)
8: \( \textbf{if } w(u, v) < d[v] \textbf{ then} \)
9: \( d[v] \leftarrow w(u, v), Q.\text{decrease\_key}(v, d[v]) \)
10: \( \pi[v] \leftarrow u \)
11: \( \textbf{return } \{(u, \pi[u])|u \in V \setminus \{s\}\} \)
Improved Running Time

Running time:
$O(n) \times (\text{time for extract\_min}) + O(m) \times (\text{time for decrease\_key})$

<table>
<thead>
<tr>
<th>Priority-Queue</th>
<th>extract_min</th>
<th>decrease_key</th>
<th>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>