Shortest Paths

Consider digraph $G = (V, E)$ with edge weight $w(e)$ associated with each edge $e$ ($w: E \rightarrow \mathbb{R}$).

The weight of some path $p = v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_k$ is $w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$.

E.g.,

\[
\begin{align*}
V_1 & \rightarrow V_2 \rightarrow V_3 \rightarrow V_4 \rightarrow V_5 \\
4 & \rightarrow 2 \rightarrow -5 & 1 \\
& & & & -2
\end{align*}
\]

$w(p) = -2$

Shortest path from $u$ to $v$ is a path of minimum weight from $u$ to $v$. The shortest path weight is the weight of such a path: $\delta(u, v) = \min \{ w(p) : p \text{ is a path from } u \text{ to } v \}$.

Also, $\delta(u, v) = +\infty$ if no path from $u$ to $v$ exists.

One subtlety:

\[
\begin{align*}
\delta(u, v) &= -\infty \\
\text{(Increasing/decreasing weights)}
\end{align*}
\]

Optimal substructure:

**Theorem:** A subpath of a shortest path is also a shortest path.

**Proof:**

By cut-and-paste, if a shorter $x \rightarrow y$ path existed, we could insert it into the $u \rightarrow v$ path and produce a shorter $u \rightarrow v$ path, contradicting the given that $u \rightarrow v$ was a shortest path.
**Triangle Inequality**

**Theorem:** For all $u, v, x \in V$, $d(u, v) \leq d(u, x) + d(x, v)$

**Proof:**

If triangle inequality violated, then $u, x, v$ lie a shorter path than $u, x, v$.

Contradiction statement that $d(u, v)$ corresponds to a shortest path.

---

**Adjacency-List Representation**

```
A -> B, D
B -> C, D
C -> B, D
D -> A, B, C
E -> C
```

Size = |E| for directed graph
Size = 2|E| for undirected graph

---

**Min-Priority Queue**

A data structure for maintaining a set $S$ of elements, each with an associated value (key), supporting:

* Insert $(S, x)$ inserts the element $x$ into $S$.
* Minimum ($S$) returns element with smallest key.
* Extract-Min ($S$) returns and removes element with smallest key.
* Decrease-Key ($S, x, k$) decreases the value of element $x$'s key to $k$. 

---

**LIB 2**
Single-source shortest paths problem

Goal: From a given source vertex \( s \in V \), find the shortest-path weights \( \delta(s,v) \) for all \( v \in V \).
Here we assume \( w(u,v) \geq 0 \), so \( \delta(s,v) \geq 0 > -\infty \).

Dijkstra's Algorithm (only valid for non-negative weights)

Idea: Greedy Algorithm

1. Maintain set \( S \) of vertices whose shortest-path distances from \( s \) are known.
2. At each step, add to \( S \) the vertex \( u \in V - S \) whose distance estimate from \( s \) is minimum.
3. Update distance estimates of vertices adjacent to \( u \).

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

\[
\begin{align*}
 d[s] & \leftarrow 0 \\
 d[V] & \leftarrow -\infty \text{ for each } v \in V - \{s\} \\
 S & \leftarrow \emptyset \\
 Q & \leftarrow V \quad (\text{priority queue of vertices keyed by } d) \\
 \text{while } & Q \neq \emptyset \\
 \quad & \text{do } u \leftarrow \text{Extract-Min}(Q) \\
 \quad & S \leftarrow S \cup \{u\} \\
 \quad & \text{for each } v \in \text{Adj}[u] \\
 \quad & \quad \text{do if } d[v] > d[u] + w(u,v) \\
 \quad & \quad \quad \text{then } d[v] \leftarrow d[u] + w(u,v) \\
 \quad & \quad \text{relaxation step} \\
 \quad & \quad \text{Implicit DECREASE-KEY}
\end{align*}
\]
Example:

\[ S = \{ A \} \]

Correctness (Part I)

**Lemma:** Invariant \( d[w] \geq \delta(s, v) \quad \forall v \in V \) at all times.

**Proof:**

\[ d[w] = 0 \text{ and } d[w] = +\infty \text{ for } v \neq s ; \quad \delta(s, s) = 0 \text{ and } \delta(s, v) \leq \infty \quad \forall v, \text{ so } \delta \text{ is consistent.} \]

Suppose invariant fails, that \( v \) is the first vertex with \( d[w] < \delta(s, v) \) and \( u \) is the vertex that caused \( d[w] \) to change by \( d[w] = d[w] + \omega(u, v) \).

Then \( d[w] = d[w] + \omega(u, v) \)

\[ \leq \delta(s, u) + d[w] \quad \text{triangle inequality} \]

\[ \leq \delta(s, u) + \omega(u, v) \quad \text{shortest path} \]

\[ \leq d[w] + \omega(u, v) \quad \text{by previous lemma} \]

Then \( d[w] < d[w] + \omega(u, v) \) violates

Correctness (Part II)

**Theorem:** When Dijkstra's algorithm terminates, \( d[w] = \delta(s, v) \quad \forall v \in V \)

**Proof:** \( d[w] \) doesn't change once added to \( S \), so suffices to show true when added

Suppose \( u \) is first vertex added to \( s \) to \( S \), for which \( d[w] \neq \delta(s, u) \)

\[ d[w] = \delta(s, u) \quad \text{by previous lemma} \]

Let \( p \) be a shortest path from \( s \) to \( u \) \( [w(p) = \delta(s, u)] \)

Consider first place \( p \) enters \( S \) \( [\text{via edge } (x, y)] \)

\( (y \text{ is first vertex along } p \text{ in } V - S, x \text{ is predecessor of } y \text{ along } p) \)
Because \( u \) is first violation, \( d[s,u] = \delta(s,u) \).

When \( x \) was added to \( S \), we relaxed \( (o,y) \) and set
\[
d[x,y] = \delta(s,x) + \omega(x,y) = \delta(s,y)
\]
because subpaths of shortest paths are shortest paths.

Thus
\[
d[x,y] = \delta(s,y) \leq \delta(s,u) \leq d[u,l] \tag{623.5}
\]
sub-path previous lemma

But \( d[u,l] \leq d[x,y] \) by Dijkstra's choice of \( u \)
\( \Leftarrow \) emphasizes need for greedy step

So \( d[x,y] = \delta(s,y) = \delta(s,u) \)

Contradiction.

---

**Analysis**

\[d[u,l] \rightarrow \infty \text{ for each } v \in V-\{s\} \quad \text{ \( O(V) \)}\]

\[
\begin{align*}
&\text{while} \ Q \neq \emptyset \quad \text{do} \quad u \gets \text{Extract-Min}(Q) \\
&\quad S \gets S \cup \{u\} \\
&\quad \text{for each } v \in \text{Adj}[u] \\
&\qquad \text{do if } \ d[v,l] > d[u,l] + \omega(u,v) \\
&\qquad \quad \text{then } \ d[v,l] \gets d[u,l] + \omega(u,v)
\end{align*}
\]

**DECREASE-KEY : \( O(1) \)** \( \text{ worst-case aggregate analysis } \)

**Time = \( O(V) \cdot \text{Extract-Min} + O(E) \cdot \text{DECREASE-KEY} \)**

(Same as Prim's MST algorithm)

<table>
<thead>
<tr>
<th>( Q )</th>
<th>Extract-Min</th>
<th>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) ) for all vertices reachable</td>
</tr>
<tr>
<td>binary heap</td>
<td>( O(lg V) )</td>
<td>( O(lg V) )</td>
<td>( O(E) )</td>
</tr>
<tr>
<td>Fibonacci heap</td>
<td>( O(lg V) ) amortized</td>
<td>( O(lg V) ) amortized</td>
<td>( O(E+VlgV) ) worst-case</td>
</tr>
</tbody>
</table>
Unweighted Graphs

Suppose \( w(u, v) = 1 \) \( \forall (u, v) \in E \). Then Dijkstra's algorithm can be improved using simple FIFO queue in place of priority queue.

(Breadth-First-Search) first-in first-out

\[
\begin{align*}
\text{BFS}(G, s) & \\
d[S] & = 0 & \\
d[u] & = \infty \text{ for each } u \in V - \{s\} & \\
Q & = \{s\} & \\
\text{while } Q \neq \emptyset & \\
& \quad \text{do } u \leftarrow \text{Dequeue}(Q) & \\
& \quad \text{for each } v \in \text{Adj}[u] & \\
& \quad \quad \text{do if } d[v] = \infty & \\
& \quad \quad \quad \text{then } d[v] \leftarrow d[u] + 1 & \\
& \quad \text{Enqueue}(Q, v) & \\
\end{align*}
\]

Analysis
Time: \( O(V + E) \) All queue operations are \( O(1) \); there is no Decrease-Key.

Example:

\[
\begin{align*}
& \text{Q: A, B, D, E, F, K} \\
& \text{0 1 1 2 2 3 3 4 4} \\
\end{align*}
\]

Correctness of BFS

Key Idea: FIFO queue in BFS mimics priority queue in Dijkstra.

Invariant: \( v \) immediately after \( u \) in queue \( \Rightarrow d[v] \) is either \( d[u] \) or \( d[u] + 1 \).