CS 253: Algorithms

Chapter 24

Shortest Paths
Shortest Path Problems

• How can we find the shortest route between two points on a road map?

• Model the problem as a graph problem:
  ◦ Road map is a weighted graph:
    vertices = cities
    edges = road segments between cities
    edge weights = road distances
  ◦ Goal: find a shortest path between two vertices (cities)
Shortest Path Problem

- **Input:**
  - Directed graph $G = (V, E)$
  - Weight function $w : E \rightarrow \mathbb{R}$

- **Weight of path** $p = \langle v_0, v_1, \ldots, v_k \rangle$
  \[ w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i) \]

- **Shortest-path weight** from $u$ to $v$:
  \[ \delta(u, v) = \min \left\{ w(p) : u \xrightarrow{p} v \text{ if there exists a path from } u \text{ to } v \right\} \]
  \[ \infty \text{ otherwise} \]

- **Note:** there might be multiple shortest paths from $u$ to $v$
Negative-Weight Edges

- Negative-weight edges may form negative-weight cycles
- If such cycles are reachable from the source, then $\delta(s, v)$ is not properly defined!
  - Keep going around the cycle, and get $w(s, v) = -\infty$ for all $v$ on the cycle

Therefore, negative-weight edges will not be considered here
Cycles

- Can shortest paths contain cycles?
  No!

- Negative-weight cycles
  - Shortest path is not well defined (**we will not consider this case**)

- If there is a positive-weight cycle, we can get a shorter path by removing the cycle.

- Zero-weight cycles?
  - No reason to use them
  - Can remove them and obtain a path with the same weight
Shortest-Paths Notation

For each vertex $v \in V$:

- $\delta(s, v)$: shortest-path weight
- $d[v]$: shortest-path weight estimate
  - Initially, $d[v] = \infty$
  - $d[v] \rightarrow \delta(s,v)$ as algorithm progresses
- $\pi[v] = \text{predecessor}$ of $v$ on a shortest path from $s$
  - If no predecessor, $\pi[v] = \text{NIL}$
  - $\pi$ induces a tree—shortest-path tree
Initialization

Alg.: INITIALIZE-SINGLE-SOURCE(V, s)

1. \textbf{for} each \( v \in V \)
2. \textbf{do} \( d[v] \leftarrow \infty \)
3. \( \pi[v] \leftarrow \text{NIL} \)
4. \( d[s] \leftarrow 0 \)

- All the shortest-paths algorithms start with INITIALIZE-SINGLE-SOURCE
Relaxation Step

- **Relaxing** an edge \((u, v)\) = testing whether we can improve the shortest path to \(v\) found so far by going through \(u\)

  If \(d[v] > d[u] + w(u, v)\)
  
  we can improve the shortest path to \(v\)
  
  \[\Rightarrow d[v] = d[u] + w(u, v)\]
  
  \[\Rightarrow \pi[v] \leftarrow u\]

After relaxation:

\[d[v] \leq d[u] + w(u, v)\]
Dijkstra’s Algorithm

- Single-source shortest path problem:
  - No negative-weight edges: $w(u, v) > 0, \forall (u, v) \in E$
- Each edge is relaxed **only once**!
- Maintains two sets of vertices:

\[ d[v] = \delta(s, v) \quad d[v] > \delta(s, v) \]
Dijkstra’s Algorithm (cont.)

- Vertices in \((V - S)\) reside in a min-priority queue
  - Keys in Q are estimates of shortest-path weights \(d[u]\)
- Repeatedly select a vertex \(u \in (V - S)\), with the minimum shortest-path estimate \(d[u]\)
- Relax all edges leaving \(u\)

**STEPS**

1) Extract a vertex \(u\) from Q (i.e., \(u\) has the highest priority)
2) Insert \(u\) to \(S\)
3) Relax all edges leaving \(u\)
4) Update Q
Dijkstra \((G, w, s)\)

\[
S = <> \quad Q = <s, t, x, z, y>
\]

\[
S = <s> \quad Q = <y, t, x, z>
\]
Example (cont.)

\[ S = \langle s, y \rangle \quad Q = \langle z, t, x \rangle \]

\[ S = \langle s, y, z \rangle \quad Q = \langle t, x \rangle \]
Example (cont.)

\[ S = \langle s, y, z, t \rangle \quad Q = \langle x \rangle \]

\[ S = \langle s, y, z, t, x \rangle \quad Q = \langle \rangle \]
Dijkstra \((G, w, s)\)

1. \textsc{initialize-single-source}(\(V, s\)) \hspace{2cm} \(\Theta(V)\)
2. \(S \leftarrow s\)
3. \(Q \leftarrow V[G]\) \hspace{2cm} \(O(V)\) build min-heap
4. \textbf{while} \(Q \neq \emptyset\) \hspace{2cm} Executed \(O(V)\) times
5. \hspace{1cm} \textbf{do} \(u \leftarrow \textsc{extract-min}(Q)\) \hspace{2cm} \(O(\lg V)\)
6. \hspace{2cm} \(S \leftarrow S \cup \{u\}\)
7. \hspace{2cm} \textbf{for each vertex} \(v \in \text{Adj}[u]\) \hspace{2cm} \(O(E)\) times \((\text{max})\)
8. \hspace{3cm} \textbf{do} \textsc{relax}(u, v, w) \hspace{2cm} \(O(\text{ElgV})\)
9. \hspace{2cm} \text{Update} \(Q\) (\textsc{decrease-key}) \hspace{2cm} \(O(\lg V)\)

\textbf{Running time:} \hspace{1cm} \(O(V\lg V + E\lg V) = O(E\lg V)\)
Dijkstra’s SSSP Algorithm (adjacency matrix)

new \( L[i] = \min\{ L[i], L[k] + W[k, i] \} \)

where \( k \) is the newly-selected intermediate node and \( W[.] \) is the distance between \( k \) and \( i \)
SSSP cont.

new \( L[i] = \text{Min}\{ L[i], L[k] + W[k, i] \} \)

where \( k \) is the newly-selected intermediate node and \( W[.] \) is the distance between \( k \) and \( i \)
new $L[i] = \text{Min}\{ L[i], \ L[k] + W[k, i] \}$

where $k$ is the newly-selected intermediate node and $W[.]$ is the distance between $k$ and $i$
\[ L[.] = \begin{bmatrix}
1 & 0 & 3 & 1 & 5 & 3 \\
\end{bmatrix} \]

new \[ L[.] = \begin{bmatrix}
1 & 0 & 3 & 1 & 4 & 3 \\
\end{bmatrix} \]
\[ L[.] = \begin{pmatrix}
1 & 0 & 3 & 1 & 4 & 3 \\
1 & 0 & 3 & 1 & 4 & 3
\end{pmatrix} \]

new \( L[.] = \begin{pmatrix}
1 & 0 & 3 & 1 & 4 & 3 \\
1 & 0 & 3 & 1 & 4 & 3
\end{pmatrix} \]
Running time: 

$O(V^2)$ (adjacency matrix)  

$O(E\lg V)$ (adjacency list)

Which one is better?
All-Pairs Shortest Paths

Given:
- Directed graph \( G = (V, E) \)
- Weight function \( w : E \rightarrow \mathbb{R} \)

Compute:
- The shortest paths between all pairs of vertices in a graph
- Result: an \( n \times n \) matrix of shortest-path distances \( \delta(u, v) \)

We can run Dijkstra’s algorithm once from each vertex:
- \( O(VE \lg V) \) with binary heap and adjacency-list representation
- if the graph is dense \( \Rightarrow O(V^3 \lg V) \)
- We can achieve \( O(V^3) \) by using an adjacency-matrix.
Problem 1

- We are given a directed graph $G=(V, E)$ on which each edge $(u, v)$ has an associated value $r(u, v)$, which is a real number in the range $0 \leq r(u, v) \leq 1$ that represents the **reliability** of a communication channel from vertex $u$ to vertex $v$.

- We interpret $r(u, v)$ as the probability that the channel from $u$ to $v$ will not fail, and we assume that these probabilities are independent.

- Give an efficient algorithm to find the most reliable path between two given vertices.
Problem 1 (cont.)

- \( r(u,v) = \Pr(\text{channel from } u \text{ to } v \text{ will not fail}) \)
- Assuming that the probabilities are independent, the reliability of a path \( p=<v_1,v_2,\ldots,v_k> \) is:
  \[
  r(v_1,v_2) \cdot r(v_2,v_3) \cdots r(v_{k-1},v_k)
  \]

**Solution 1:** modify Dijkstra’s algorithm

- Perform relaxation as follows:
  \[
  \text{if } d[v] < d[u] \cdot w(u,v) \text{ then } \]
  \[
  d[v] = d[u] \cdot w(u,v)
  \]

- Use “EXTRACT_MAX” instead of “EXTRACT_MIN”
Problem 1 (cont.)

- **Solution 2:** use Dijkstra’s algorithm without any modifications!
  - We want to find the channel with the highest reliability, i.e.,
    \[
    \max_p \prod_{(u,v) \in p} r(u, v)
    \]
  - But Dijkstra’s algorithm computes
    \[
    \min_p \sum_{(u,v) \in p} w(u, v)
    \]
  - Take the log
    \[
    \lg(\max_p \prod_{(u,v) \in p} r(u, v)) = \max_p \sum_{(u,v) \in p} \lg(r(u, v))
    \]
Problem 1 (cont.)

- Turn this into a minimization problem by taking the negative:

\[-\min_p \sum_{(u,v) \in p} \lg(r(u, v)) = \min_p \sum_{(u,v) \in p} -\lg(r(u, v))\]

- Run Dijkstra’s algorithm using

\[w(u, v) = -\lg(r(u, v))\]