Lecture 16: Shortest Paths II - Dijkstra

Lecture Overview

- Review
- Shortest paths in DAGs
- Shortest paths in graphs without negative edges
- Dijkstra’s Algorithm

Readings

CLRS, Sections 24.2-24.3

Review

d[v] is the length of the current shortest path from starting vertex s. Through a process of relaxation, d[v] should eventually become δ(s, v), which is the length of the shortest path from s to v. Π[v] is the predecessor of v in the shortest path from s to v.

Basic operation in shortest path computation is the relaxation operation

\[
\text{RELAX}(u, v, w) \\
\text{if } d[v] > d[u] + w(u, v) \\
\text{then } d[v] \leftarrow d[u] + w(u, v) \\
\Pi[v] \leftarrow u
\]

Relaxation is Safe

Lemma: The relaxation algorithm maintains the invariant that d[v] ≥ δ(s, v) for all v ∈ V.

Proof: By induction on the number of steps.

Consider RELAX(u, v, w). By induction d[u] ≥ δ(s, u). By the triangle inequality, δ(s, v) ≤ δ(s, u) + δ(u, v). This means that δ(s, v) ≤ d[u] + w(u, v), since d[u] ≥ δ(s, u) and w(u, v) ≥ δ(u, v). So setting d[v] = d[u] + w(u, v) is safe. □
DAGs:
Can’t have negative cycles because there are no cycles!

1. Topologically sort the DAG. Path from $u$ to $v$ implies that $u$ is before $v$ in the linear ordering.

2. One pass over vertices in topologically sorted order relaxing each edge that leaves each vertex.
$\Theta(V + E)$ time

Example:

![Diagram of shortest path using topological sort]

Figure 1: Shortest Path using Topological Sort.

Vertices sorted left to right in topological order

Process $r$: stays $\infty$. All vertices to the left of $s$ will be $\infty$ by definition

Process $s$: $t: \infty \to 2$  $x: \infty \to 6$ (see top of Figure 2)
DIJKSTRA Demo
Dijkstra’s Algorithm

For each edge \((u, v) \in E\), assume \(w(u, v) \geq 0\), maintain a set \(S\) of vertices whose final shortest path weights have been determined. Repeatedly select \(u \in V - S\) with minimum shortest path estimate, add \(u\) to \(S\), relax all edges out of \(u\).

**Pseudo-code**

\[
\text{Dijkstra} \ (G, W, s) \quad \text{//uses priority queue Q}
\]

\[
\text{Initialize} \ (G, s)
\]

\[
S \leftarrow \phi
\]

\[
Q \leftarrow V[G] \quad \text{//Insert into Q}
\]

while \(Q \neq \phi\)

\[
do u \leftarrow \text{EXTRACT-MIN}(Q) \quad \text{//deletes u from Q}
\]

\[
S = S \cup \{u\}
\]

for each vertex \(v \in \text{Adj}[u]\)

\[
do \text{RELAX} \ (u, v, w) \quad \leftarrow \text{this is an implicit DECREASE\_KEY operation}
\]
Example

\[
\begin{align*}
S = \{ \} & \quad \{ A, B, C, D, E \} = Q \\
S = \{ A \} & \quad 0 \infty \infty \infty \infty \\
S = \{ A, C \} & \quad 0 10 3 \infty \infty \quad \text{after relaxing edges from A} \\
S = \{ A, C \} & \quad 0 7 3 11 5 \quad \text{after relaxing edges from C} \\
S = \{ A, C, E \} & \quad 0 7 3 11 5 \\
S = \{ A, C, E, B \} & \quad 0 7 3 9 5 \quad \text{after relaxing edges from B}
\end{align*}
\]

Figure 4: Dijkstra Execution

Strategy: Dijkstra is a greedy algorithm: choose closest vertex in \( V - S \) to add to set \( S \).

Correctness: We know relaxation is safe. The key observation is that each time a vertex \( u \) is added to set \( S \), we have \( d[u] = \delta(s, u) \).
**Dijkstra Complexity**

- $\Theta(v)$ inserts into priority queue
- $\Theta(v)$ EXTRACT_MIN operations
- $\Theta(E)$ DECREASE_KEY operations

Array impl:

- $\Theta(v)$ time for extra min
- $\Theta(1)$ for decrease key
- Total: $\Theta(V.V + E.1) = \Theta(V^2 + E) = \Theta(V^2)$

Binary min-heap:

- $\Theta(lg V)$ for extract min
- $\Theta(lg V)$ for decrease key
- Total: $\Theta(V lg V + E lg V)$

Fibonacci heap (not covered in 6.006):

- $\Theta(lg V)$ for extract min
- $\Theta(1)$ for decrease key
- amortized cost
- Total: $\Theta(V lg V + E)$