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

RELAX(u, v, w)
if d[v] > d[u] + w(u, v)
then d[v] ← d[u] + w(u, v)
Π[v] ← u

Triangle Inequality:

Theorem: For all u, v, x ∈ X, we have

δ(u, v) ≤ δ(u, x) + δ(x, v)

Proof: See Figure [1]

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 $d[u] \geq \delta(s, u)$. By the triangle inequality, $\delta(s, v) \leq \delta(s, u) + \delta(u, v)$. This means that $\delta(s, v) \leq d[u] + w(u, v)$, since $d[u] \geq \delta(s, u)$ and $w(u, v) \geq \delta(u, v)$. 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:**

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 \( s : \infty \rightarrow 2 \) \( x : \infty \rightarrow 6 \) (see top of Figure 3)

![Diagram showing shortest paths](image)

Figure 3: Preview of Dynamic Programming

**DIJKSTRA Demo**

Use gravity to compute shortest paths! Set up a shortest path problem using balls and strings – see Figure 4. Assuming you built it correctly (a big assumption!), if you lift the start vertex (ball) up high, the other vertices should line up in the order in which the Dijkstra algorithm will process vertices.
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

\[
\text{Dijkstra} \ (G, W, s) \quad \text{//uses priority queue } Q
\]
\[
\text{Initialize } (G, s) \\
S \leftarrow \emptyset \\
Q \leftarrow V[G] \quad \text{//Insert into } Q \\
\text{while } Q \neq \emptyset \\
\quad \text{do } u \leftarrow \text{EXTRACT-MIN}(Q) \quad \text{//deletes } u \text{ from } Q \\
\quad S = S \cup \{u\} \\
\quad \text{for each vertex } v \in \text{Adj}[u] \\
\qquad \text{do RELAX } (u, v, w) \quad \text{//this is an implicit DECREASE\_KEY operation}
\]
Example

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

Figure 5: 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

\[ \Theta(v) \text{ inserts into priority queue} \]
\[ \Theta(v) \text{ EXTRACT\_MIN operations} \]
\[ \Theta(E) \text{ DECREASE\_KEY operations} \]

Array impl:

\[ \Theta(v) \text{ time for extract min} \]
\[ \Theta(1) \text{ for decrease key} \]
Total: \( \Theta(VV + E) = \Theta(V^2 + E) = \Theta(V^2) \)

Binary min-heap:

\[ \Theta(lg V) \text{ for extract min} \]
\[ \Theta(lg V) \text{ for decrease key} \]
Total: \( \Theta(V lg V + E lg V) \)

Fibonacci heap (not covered in 6.006):

\[ \Theta(lg V) \text{ for extract min} \]
\[ \Theta(1) \text{ for decrease key} \]
amortized cost
Total: \( \Theta(V lg V + E) \)