Lecture 17: Shortest Paths III - Dijkstra and Special Cases

Lecture Overview

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

Readings

CLRS, Sections 24.2-24.3

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 vehicles 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 $s$: $t: \infty \rightarrow 2 \quad x: \infty \rightarrow 6$ (see top of Figure 2)
Figure 2: Preview of Dynamic Programming

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

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

\[
\text{RELAX}(u, v, w)
\]

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

Example

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

Figure 3: Dijkstra Execution

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

Correctness: Each time a vertex \(u\) is added to set \(S\), we have \(d[u] = \delta(s, u)\)
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: \( \theta(VV + E) = \theta(V^2 + E) = \theta(V^2) \)

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: \( \theta(V lg V + E) \)