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 $\delta(s, v)$, which is the length of the shortest pathfrom s to v. $\Pi[v]$ is the predecessor of v in the shortest path from s to v.

Basic operation in shortest path computation is the *relaxation operation*

$$
\begin{aligned}
&\text{RELAX}(u, v, w) \\
&\quad \text{if } d[v] > d[u] + w(u, v) \\
&\quad\quad \text{then } d[v] \leftarrow d[u] + w(u, v) \\
&\quad\quad\quad \Pi[v] \leftarrow u
\end{aligned}
$$

Relaxation is Safe

Lemma: The relaxation algorithm maintains the invariant that $d[v] \geq \delta(s, v)$ for all $v \in V$.

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. $\square$

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.
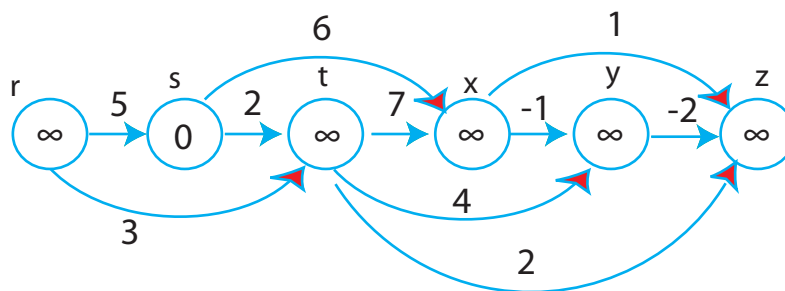 $\Theta(V + E)$ time

Example:

Figure 1: Shortest Path using Topological Sort.

Vertices sorted left to right in topological order

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

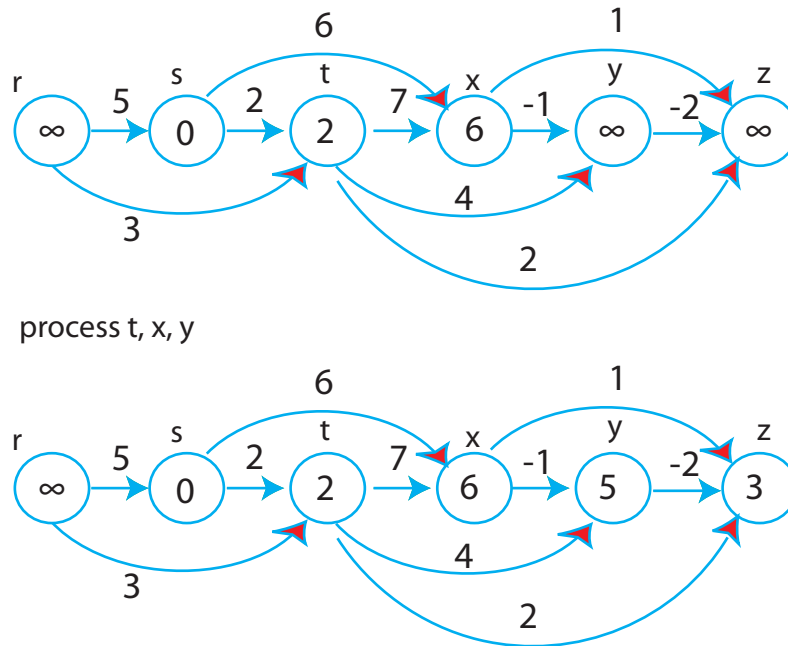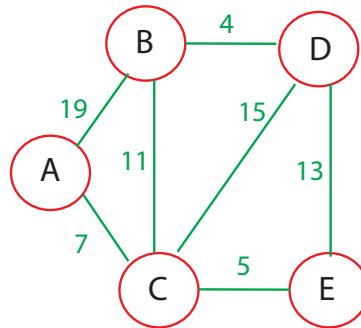Process s: $t : \infty \to 2$ $x : \infty \to 6$ (see top of Figure 2)

process t, x, y

Figure 2: Preview of Dynamic Programming

DIJKSTRA Demo

Figure 3: Dijkstra Demonstration with Balls and String.

Dijkstra's Algorithm

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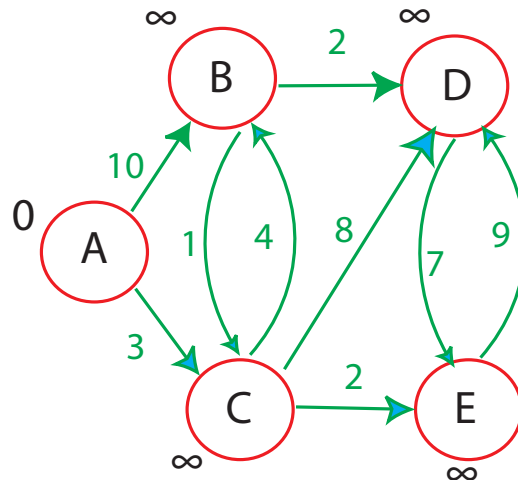
Pseudo-code

Dijkstra (G, W, s) //uses priority queue Q

        Initialize (G, s)

        $S \leftarrow \phi$

        $Q \leftarrow V[G]$ //Insert into Q

        while $Q \neq \phi$

                do $u \leftarrow$ EXTRACT-MIN(Q) //deletes u from Q

                $S = S \cup \{u\}$

                for each vertex $v \; \epsilon \;$ Adj$[u]$

                      do RELAX (u, v, w) $\leftarrow$ this is an implicit DECREASE_KEY operation

Example

$$S = \{\ \}\qquad\{\ A\ \ B\ \ C\ \ D\ \ E\ \} =\qquad Q$$

S = { }	{ A	B	C	D	E } =	Q
S = { A }	(0)	∞	∞	∞	∞	
S = { A, C }	0	10	(3)	∞	∞	← after relaxing edges from A
S = { A, C }	0	7	3	11	(5)	← after relaxing edges from C
S = { A, C, E }	0	(7)	3	11	5	
S = { A, C , E, B}	0	7	3	9	5	← after relaxing edges from B

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: $\Theta(V \lg V + E)$

6.006 Introduction to Algorithms
Fall 2011