6.006 Introduction to Algorithms
Spring 2008

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:

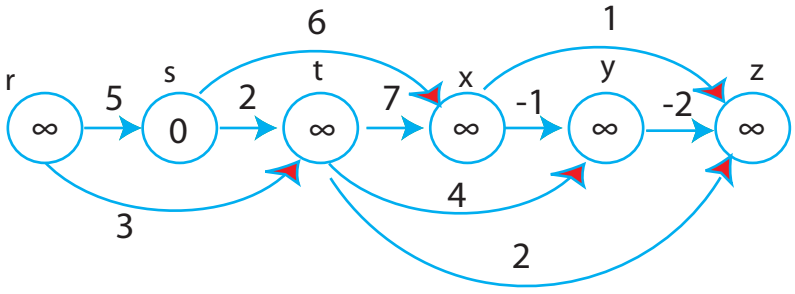


Figure 1: Shortest Path using Topological Sort

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)

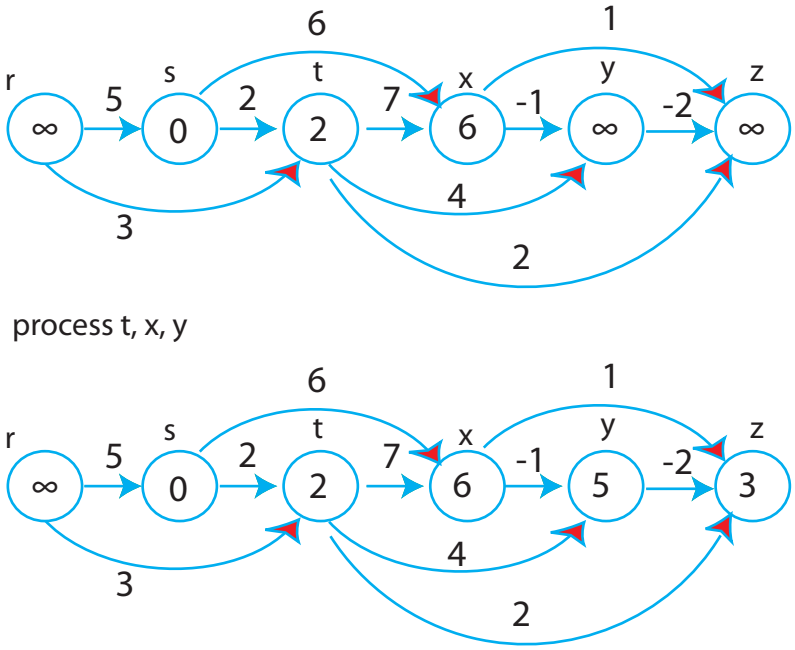


Figure 2: Preview of Dynamic Programming

Dijkstra's Algorithm

For each edge (u, v) ϵ 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 minimum shortest path estimate, add u to S, relax all edges out of u.

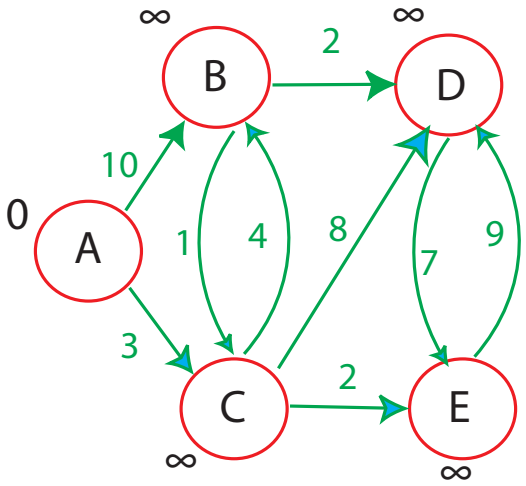
Pseudo-code

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

Recall

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

Example



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 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: $\theta(V.V + E.1) = \theta(V^2 + E) = \theta(V^2)$

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