Algorithms: Design and Analysis, Part II

The Bellman-Ford Algorithm

Space Optimization

Quiz

Question: how much space does the basic Bellman-ford algorithm require? [pick the strongest true statement.] [m = # of edges, n = # of vertices]

[✓ (A) $\Theta(n^2)$] — $\Theta(1)$ for each of n^2 subproblems

(B) $\Theta(mn)$

(C) $\Theta(n^3)$

(D) $\Theta(m^2)$

Predecessor Pointers

$$A[i,v] = \min \begin{cases} A[i-1,v] \\ \min_{(w,v) \in E} \left(A[i-1,w] + c_{wv} \right) \end{cases}$$

Note: only need the $A[i-1,v]$'s to compute the $A[i,v]$'s.

⇒ only need $O(n)$ to remember the current and last rounds of subproblems [only $O(1)$ per destination!]

Concern: without a filled-in table, how do we reconstruct the actual shortest paths?

Exercise: find analogous optimizations for our previous DP algorithms

Computing Predecessor Pointers

Idea: Compute a second table B, where $B[i,v]$ = 2nd-to-last vertex on a shortest $s \rightarrow v$ path with $\leq i$ edges. (or NULL if no such paths exist)

("predecessor pointers")

Reconstruction: Assume the input graph G has no negative cycles and we correctly compute the $B[i,v]$'s.

Then: tracing back predecessor pointers – the $B[n-1,v]$'s – from v to s yields a shortest s-v path.

= last hop of a shortest s-v path

[correctness from optimal substructure of shortest paths]

Computing Predecessor Pointers

Recall: $$A[i,v] = \min \begin{cases} (1)\ A[i-1,v] \\ (2)\ \min_{(w,v)\in E} \{A[i-1,w] + c_{wv}\} \end{cases}$$

Base case: $B[0,v] =$ NULL for all $v \in V$

To compute $B[i,v]$ with $i>0$:

Case 1: $B[i,v] = B[i-1,v]$

Case 2: $B[i,v] =$ the vertex w achieving the minimum (i.e., the new last hop)

Correctness: computation of $A[i,v]$ is brute-force search through the $(1+\text{in-deg}(v))$ possible optimal solutions, $B[i,v]$ is just caching the last hop of the winner.

To reconstruct a negative-cost cycle:

use depth-first search to check for a cycle of predecessor pointers after each round (must be a negative cost cycle).

details omitted