All-Pairs Shortest Paths (APSP)

Problem Definition

Algorithms: Design and Analysis, Part II

Problem Definition

Input: Directed graph $G = (V, E)$ with edge costs c_e for each edge $e \in E$, [No distinguished source vertex.]

Goal: Either

(A) Compute the length of a shortest $u \rightarrow v$ path for <u>all</u> pairs of vertices $u, v \in V$

OR

(B) Correctly report that G contains a negative cycle.

Quiz

Question: How many invocations of a single-source shortest-path subroutine are needed to solve the all-pairs shortest path problem? [$n = \#$ of vertices]

A) 1

B) $n-1$

C) n

D) n^2

Running time (nonnegative edge costs):

$$n \cdot \text{ Dijkstra } = O(nm \log n) = \begin{array}{l} O(n^2 \log n) \text{ if } m = \Theta(n) \\ O(n^3 \log n) \text{ if } m = \Theta(n^2) \end{array}$$

Running time (general edge costs):

$$n \cdot \text{ Bellman-Ford } = O(n^2 m) = \begin{array}{l} O(n^3) \text{ if } m = \Theta(n) \\ O(n^4) \text{ if } m = \Theta(n^2) \end{array}$$

All-Pairs Shortest Paths (APSP)

Optimal Substructure

Algorithms: Design and Analysis, Part II

Motivation

Floyd-Warshall algorithm: $O(n^3)$ algorithm for APSP.

- Works even with graphs with negative edge lengths.

Thus: (1) At least as good as n Bellman-Fords, better in dense graphs.

(2) In graphs with nonnegative edge costs, competitive with n Dijkstra's in dense graphs.

Important special case: Transitive closure of a binary (i.e., all-pairs reachability) relation.

Open question: Solve APSP significantly faster than $O(n^3)$ in dense graphs?

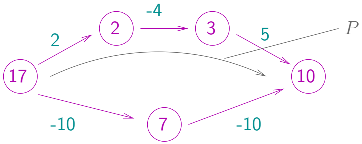
Optimal Substructure

Recall: Can be tricky to define ordering on subproblems in graph problems.

Key idea: Order the vertices $V = \{1, 2, \ldots, n\}$ arbitrarily. Let $V^{(k)} = \{1, 2, \ldots, k\}$.

Lemma: Suppose G has no negative cycle. Fix source $i \in V$, destination $j \in V$, and $k \in \{1, 2, \ldots, n\}$. Let P = shortest (cycle-free) i-j path with $\underline{\text{all internal nodes in } V^{(k)}}$.

Example: $[i = 17, j = 10, k = 5]$



Optimal Substructure (con'd)

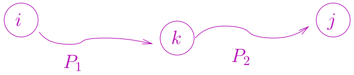
Optimal substructure lemma: Suppose G has no negative cost cycle. Let P be a shortest (cycle-free) i-j path with all internal nodes in $V^{(k)}$. Then:

Case 1: If k not internal to P, then P is a shortest (cycle-free) i-j path with all internal vertices in $V^{(k-1)}$.

Case 2: If k is internal to P, then:

P_1 = shortest (cycle-free) i-k path with all internal nodes in $V^{(k-1)}$ and

P_2 = shortest (cycle-free) k-j path with all internal nodes in $V^{(k-1)}$



Proof: Similar to Bellman-Ford opt substructure (you check!)

All-Pairs Shortest Paths (APSP)

Algorithms: Design and Analysis, Part II

The Floyd-Warshall Algorithm

Quiz

Setup: Let $A =$ 3-D array (indexed by i, j, k).

Intent: $A[i, j, k] =$ length of a shortest i-j path with all internal nodes in $\{1, 2, \ldots, k\}$ (or $+\infty$ if no such paths)

Question: What is $A[i, j, 0]$ if

(1) $i = j$ (2)$(i, j) \in E$ (3) $i \neq j$ and $(i, j) \notin E$

A) 0, 0, and $+\infty$

B) 0, c_{ij}, and c_{ij}

C) 0, c_{ij}, and $+\infty$

D) $+\infty$, c_{ij}, and $+\infty$

The Floyd-Warshall Algorithm

Let A = 3-D array (indexed by i, j, k)

Base cases: For all $i, j \in V$:

$$A[i,j,0] = \left\{ \begin{array}{l} 0 \text{ if } i = j \\ c_{ij} \text{ if } (i,j) \in E \\ +\infty \text{ if } i \neq j \text{ and } (i,j) \notin E \end{array} \right\}$$

For $k = 1$ to n

  For $i = 1$ to n

    For $j = 1$ to n

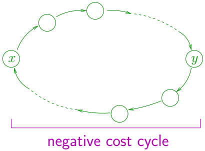
$$A[i,j,k] = \min \left\{ \begin{array}{ll} A[i,j,k-1] & \text{Case 1} \\ A[i,k,k-1] + A[k,j,k-1] & \text{Case 2} \end{array} \right\}$$

Correctness: From optimal substructure + induction, as usual.

Running time: $O(1)$ per subproblem, $O(n^3)$ overall.

Odds and Ends

Question #1: What if input graph G has a negative cycle?



Answer: Will have $A[i, i, n] < 0$ for at least one $i \in V$ at end of algorithm.

Question #2: How to reconstruct a shortest i-j path?

Answer: In addition to A, have Floyd-Warshall compute $B[i, j] =$ max label of an internal node on a shortest i-j path for all $i, j \in V$.

[Reset $B[i, j] = k$ if 2nd case of recurrence used to compute $A[i, j, k]$]

$\Rightarrow$ Can use the $B[i, j]$'s to recursively reconstruct shortest paths!

All-Pairs Shortest Paths (APSP)

A Reweighting Technique

Algorithms: Design and Analysis, Part II

Motivation

Recall: APSP reduces to n invocations of SSSP.

- Nonnegative edge lengths: $\boxed{O(mn \log n) \text{ via Dijkstra}}$

- General edge lengths: $O(mn^2)$ via Bellman-Ford

Johnson's algorithm: Reduces APSP to

- 1 invocation of Bellman-Ford ($O(mn)$)

- n invocations of Dijkstra ($O(nm \log n)$)

Running time: $O(mn) + O(mn \log n) = \boxed{O(mn \log n)}$

As good as with nonnegative edge lengths!

Quiz

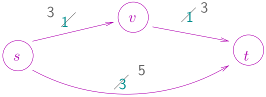
Suppose: $G = (V, E)$ directed graph with edge lengths. Obtain G' from G by adding a constant M to every edge's length. When is the shortest path between a source s and a destination t guaranteed to be the same in G and G'?

A) When G has no negative-cost cycle

B) When all edge costs of G are nonnegative

C) When all s-t paths in G have the same number of edges

D) Always

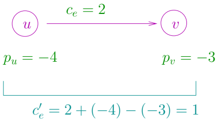


Quiz

Setup: $G = (V, E)$ is a directed graph with general edge lengths c_e. Fix a real number p_v for each vertex $v \in V$.

Definition: For every edge $e = (u, v)$ of G, $c'_e := c_e + p_u - p_v$

Question: If the s-t path P has length L with the original edge lengths $\{c_e\}$, what is P's length with the new edge length $\{c'_e\}$?



A) L

B) $L + p_s + p_t$

C) $\boxed{L + p_s - p_t}$

D) $L - p_S + p_t$

New length $= \sum_{e \in P} c'_e = \sum_{e=(u,v) \in P} [c_e + p_u - p_v] = (\sum_{e \in P} c_e) + p_s - p_t$

Reweighting

Summary: Reweighting using vertex weights $\{p_v\}$ adds the same amount (namely, $p_s - p_t$) to every s-t path.

Consequence: Reweighting always leaves the shortest path unchanged.

Why useful? What if:

(1) G has some negative edge lengths

(2) After reweighting by some $\{p_v\}$, all edge lengths become nonnegative!

Question: Do such weights always exist?

Yes, and can be computed using the Bellman-Ford algorithm!

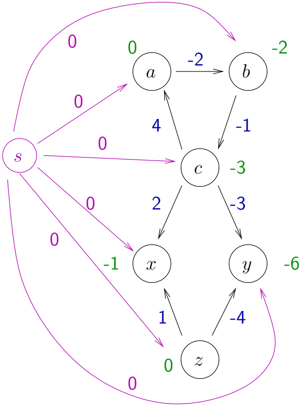
Requires Bellman-Ford, enables Dijkstra!

All-Pairs Shortest Paths (APSP)

Johnson's Algorithm

Algorithms: Design and Analysis, Part II

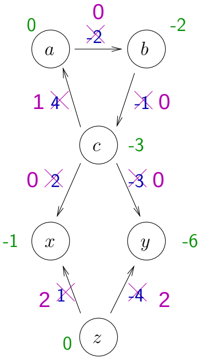
Example



Note: Adding s does not add any new u-v paths for any $u, v \in G$.

Key insight: Define vertex weight $p_v :=$ length of a shortest s-v path.

Example (con'd)



Recall: For each edge $e = (u, v)$, define $c'_e = c_e + p_u - p_v$.

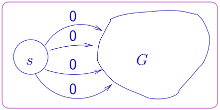
Note: After reweighting, all edge lengths nonnegative! $\Rightarrow$ Can compute all (reweighted) shortest paths via n Dijkstra computations! [No need for Bellman-Ford]

Johnson's Algorithm

Input: Directed graph $G = (V, E)$, general edge lengths c_e.

(1) Form G' by adding a new vertex s and a new edge (s, v) with length 0 for each $v \in G$.

G' →



(2) Run Bellman-Ford on G' with source vertex s. [If B-F detects a negative-cost cycle in G' (which must lie in G), halt + report this.]

(3) For each $v \in G$, define p_v = length of a shortest $s \to v$ path in G'. For each edge $e = (u, v) \in G$, define $c'_e = c_e + p_u - p_v$.

(4) For each vertex u of G: Run Dijkstra's algorithm in G, with edge lengths $\{c'_e\}$, with source vertex u, to compute the shortest-path distance $d'(u, v)$ for each $v \in G$.

(5) For each pair $u, v \in G$, return the shortest-path distance $d(u, v) := d'(u, v) - p_u + p_v$

Analysis of Johnson's Algorithm

Running time: $O(n) + O(mn) + O(m) + O(nm \log n) + O(n^2)$

Step (1), form G' — Step (2), run BF — Step (3), form c' — Step (4), n Dijkstra — Step (5), $O(1)$ work per u-v pair

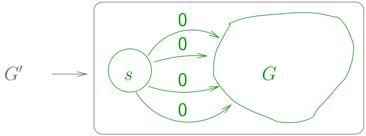
$= O(mn \log n)$. [Much better than Floyd-Warshall for sparse graphs!]

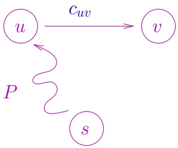
Correctness: Assuming $c'_e \geq 0$ for all edges e (see next slide for proof), correctness follows from last video's quiz.

[Reweighting doesn't change the shortest u-v path, it just adds $(p_u - p_v)$ to its length]

Correctness of Johnson's Algorithm

Claim: For every edge $e = (u, v)$ of G, the reweighted length $c'_e = c_e + p_u - p_v$ is nonnegative.





Proof: Fix an edge (u, v). By construction,
p_u = length of a shortest s-u path in G'
p_v = length of a shortest s-v path in G'
Let P = a shortest s-u path in G' (with length p_u - exists, by construction of G')
$\Rightarrow P + (u, v)$ = an s-v path with length $p_u + c_{uv}$
$\Rightarrow$ Shortest s-v path only shorter, so $p_v \leq p_u + c_{uv}$
$\Rightarrow c'_{uv} = c_{uv} + p_u - p_v \geq 0$. QED!