The Bellman-Ford Algorithm

Algorithms: Design and Analysis, Part II

Single-Source Shortest Paths Revisited

The Single-Source Shortest Path Problem

Input: Directed graph $G = (V, E)$, edge lengths c_e for each $e \in E$, source vertex $s \in V$. [Can assume no parallel edges.]

Goal: For every destination $v \in V$, compute the length (sum of edge costs) of a shortest s-v path.

On Dijkstra's Algorithm

Good news: $O(m \log n)$ running time using heaps
(n = number of vertices, m = number of edges)

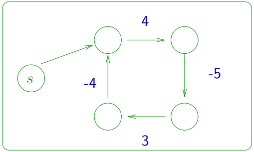
Bad news:

(1) Not always correct with negative edge lengths
[e.g. if edges $\mapsto$ financial transactions]

(2) Not very distributed (relevant for Internet routing)

Solution: The Bellman-Ford algorithm

On Negative Cycles



Question: How to define shortest path when G has a negative cycle?

Solution #1: Compute the shortest s-v path, with cycles allowed.

Problem: Undefined or $-\infty$. [will keep traversing negative cycle]

Solution #2: Compute shortest cycle-free s-v path.

Problem: NP-hard (no polynomial algorithm, unless P=NP)

Solution #3: (For now) Assume input graph has no negative cycles.

Later: Will show how to quickly check this condition.

Quiz

Quiz: Suppose the input graph G has no negative cycles. Which of the following is true? [Pick the strongest true statement.] [$n = \#$ of vertices, $m = \#$ of edges]

A) For every v, there is a shortest s-v path with $\leq n - 1$ edges.

B) For every v, there is a shortest s-v path with $\leq n$ edges.

C) For every v, there is a shortest s-v path with $\leq m$ edges.

D) A shortest path can have an arbitrarily large number of edges in it.

The Bellman-Ford Algorithm

Optimal Substructure

Algorithms: Design and Analysis, Part II

Single-Source Shortest Path Problem, Revisited

Input: Directed graph $G = (V, E)$, edge lengths c_e [possibly negative], source vertex $s \in V$.

Goal: either

(A) For all destinations $v \in V$, compute the length of a shortest s-v path $\rightarrow$ focus of this + next video

OR

(B) Output a negative cycle (excuse for failing to compute shortest paths) $\rightarrow$ later

Optimal Substructure (Informal)

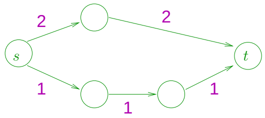
Intuition: Exploit sequential nature of paths. Subpath of a shortest path should itself be shortest.

Issue: Not clear how to define "smaller" & "larger" subproblems.

Key idea: Artificially restrict the number of edges in a path.

Subproblem size $\iff$ Number of permitted edges

Example:



Optimal Substructure (Formal)

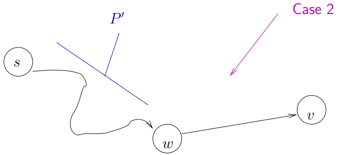
Lemma: Let $G = (V, E)$ be a directed graph with edge lengths c_e and source vertex s.

[G might or might not have a negative cycle]

For every $v \in V$, $i \in \{1, 2, \ldots\}$, let P = shortest s-v path with at most i edges. (Cycles are permitted.)

Case 1: If P has $\leq (i-1)$ edges, it is a shortest s-v path with $\leq (i-1)$ edges.

Case 2: If P has i edges with last hop (w, v), then P' is a shortest s-w path with $\leq (i-1)$ edges.



Proof of Optimal Substructure

Case 1: By (obvious) contradiction.

Case 2: If Q (from s to w, $\leq (i-1)$ edges) is shorter than P' then $Q + (w, v)$ (from s to v, $\leq i$ edges) is shorter than $P' + (w, v)$ $(= P)$ which contradicts the optimality of P. QED!

Quiz

Question: How many candidates are there for an optimal solution to a subproblem involving the destination v?

A) 2

B) $\boxed{1 + \text{in-degree}(v)}$

C) $n - 1$

D) n

1 from Case 1 + 1 from Case 2 for each choice of the final hop (w, c)

The Bellman-Ford Algorithm

The Basic Algorithm

Algorithms: Design and Analysis, Part II

The Recurrence

Notation: Let $L_{i,v}$ = minimum length of a s-v path with $\leq i$ edges.

- With cycles allowed
- Defined as $+\infty$ if no s-v paths with $\leq i$ edges

Recurrence: For every $v \in V$, $i \in \{1, 2, \ldots\}$

$$L_{i,v} = \min \left\{ \begin{array}{ll} L_{(i-1),v} & \text{Case 1} \\ \min_{(u,v)\in E}\{L_{(i-1),w} + c_{wv}\} & \text{Case 2} \end{array} \right\}$$

Correctness: Brute-force search from the only (1+in-deg(v)) candidates (by the optimal substructure lemma).

If No Negative Cycles

Now: Suppose input graph G has no negative cycles.

$\Rightarrow$ Shortest paths do not have cycles

[removing a cycle only decreases length]

$\Rightarrow$ Have $\leq (n-1)$ edges

Point: If G has no negative cycle, only need to solve subproblems up to $i = n - 1$.

Subproblems: Compute $L_{i,v}$ for all $i \in \{0, 1, \ldots, n-1\}$ and all $v \in V$.

The Bellman-Ford Algorithm

Let A = 2-D array (indexed by i and v)

Base case: $A[0, s] = 0$; $A[0, v] = +\infty$ for all $v \neq s$.

For $i = 1, 2, \ldots, n-1$

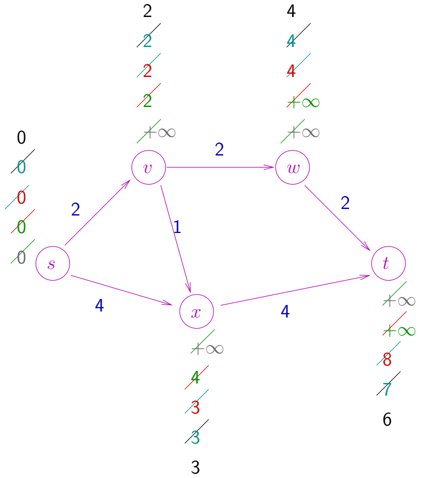
For each $v \in V$

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

As discussed: If G has no negative cycle, then algorithm is correct [with final answers $= A[n-1, v]$'s]

Example

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



Quiz

Question: What is the running time of the Bellman-Ford algorithm? [Pick the strongest true statement.] [$m = \#$ of edges, $n = \#$ of vertices]

A) $O(n^2)$ $\rightarrow$ # of subproblems, but might do $\Theta(n)$ work for one subproblem

B) $\boxed{O(mn)}$

C) $O(n^3)$

D) $O(m^2)$

Reason: Total work is $O(\boxed{n} \boxed{\sum_{v \in V} \text{in-deg}(v)}) = O(mn)$

iterations of outer loop (i.e. choices of i)

work done in one iteration $= m$

Stopping Early

Note: Suppose for some $j < n - 1$, $A[j, v] = A[j - 1, v]$ for all vertices v.

$\Rightarrow$ For all v, all future $A[i, v]$'s will be the same

$\Rightarrow$ Can safely halt (since $A[n - 1, v]$'s $=$ correct shortest-path distances)

The Bellman-Ford Algorithm

Detecting Negative Cycles

Algorithms: Design and Analysis, Part II

Checking for a Negative Cycle

Question: What if the input graph G has a negative cycle?
[Want algorithm to report this fact]

Claim:

G has no negative-cost cycle (that is reachable from s) $\iff$ In the extended Bellman-Ford algorithm, $A[n-1, v] = A[n, v]$ for all $v \in V$.

Consequence: Can check for a negative cycle just by running Bellman-Ford for one extra iteration (running time still $O(mn)$).

Proof of Claim

($\Rightarrow$) Already proved in correctness of Bellman-Ford

($\Leftarrow$) Assume $A[n-1, v] = A[n, v]$ for all $v \in V$. (Assume also these are finite $(< +\infty)$)

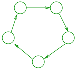
Let $d(v)$ denote the common value of $A[n-1, v]$ and $A[n, v]$.

Recall algorithm: $d(v)$ $d(w)$

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

Thus: $d(v) \leq d(w) + c_{wv}$ for all edges $(w, v) \in E$

Equivalently: $d(v) - d(w) \leq c_{wv}$

Now: Consider an arbitrary cycle C. 

$$\sum_{(w,v)\in C} \geq \sum_{(w,v)\in C} (d(w) - d(v)) = 0$$ QED!

The Bellman-Ford Algorithm

Space Optimization

Algorithms: Design and Analysis, Part II

Quiz

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

A) $\boxed{\Theta(n^2)}$ $\rightarrow \Theta(1)$ for each of n^2 subproblems

B) $\Theta(mn)$

C) $\Theta(n^3)$

D) $\Theta(m^2)$

Predecessor Pointers

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

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

$\Rightarrow$ 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 <u>no negative cycles</u> and we correctly compute the $B[i, v]$'s.

Then: Tracing back predecessor pointers – the $B[n-1, v]$'s (= last hop of a shortest s-v path) – from v to s yields a shortest s-v path.

[Correctness from optimal substructure of shortest paths]

Computing Predecessor Pointers

Recall:

$$A[i, v] = \min \left\{ \begin{array}{ll} (1) & A[i-1, v] \\ (2) & \min_{(w,v) \in E} \{A[i-1, w] + c_{wv}\} \end{array} \right\}$$

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)

The Bellman-Ford Algorithm

Internet Routing

Algorithms: Design and Analysis, Part II

From Bellman-Ford to Internet Routing

Note: The Bellman-Ford algorithm is intuitively "distributed".

Toward a routing protocol:
(1) Switch from source-driven to destination driven

[Just reverse all directions in the Bellman-Ford algorithm]

- Every vertex v stores shortest-path distance from v to destination t and the first hop of a shortest path

[For <u>all</u> relevant destinations t]

("Distance vector protocols")

Handling Asynchrony

(2) Can't assume all $A[i, v]$'s get computed before all $A[i-1, v]$'s

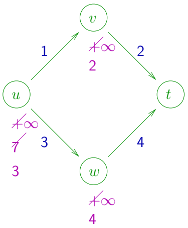
Fix: Switch from "pull-based" to "push-based": As soon as $A[i, v] < A[i-1, v]$, v notifies all of its neighbors.

Fact: Algorithm guaranteed to converge eventually. (Assuming no negative cycles)

[Reason: Updates strictly decrease sum of shortest-path estimates]

$\Rightarrow$ RIP, RIP2 Internet routing protocols very close to this algorithm [see RFC 1058]

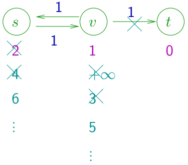
Example:



Handling Failures

Problem: Convergence guaranteed only for static networks (not true in practice).

Counting to Infinity:



Fix: Each V maintains <u>entire shortest path</u> to t, not just the next hop.

"Path vector protocol" "Border Gateway Protocol (BGP)"

Con: More space required.

Pro#1: More robust to failures.

Pro#2: Permits more sophisticated route selection (e.g., if you care about intermediate stops).

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 $u \to v$ path for all 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?

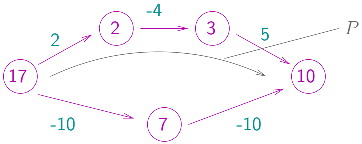
Optimal Substructure

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 <u>all internal nodes in $V^{(k)}$</u>.

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



Optimal Substructure (con'd)

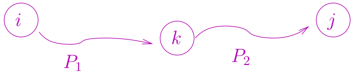
Optimal substructure lemma: Suppose G has no negative cost cycle. Let P be a shortest (cycle-free) i-j path with all internal nodes in $V^{(k)}$. Then:

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

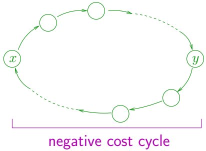
$$A[i,j,k] = \min \left\{ \begin{array}{ll} A[i,j,k-1] & \text{Case 1} \\ A[i,k,k-1] + A[k,j,k-1] & \text{Case 2} \end{array} \right\}$$

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: $O(mn \log n)$ 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

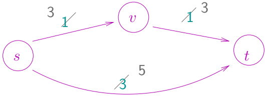
Suppose: $G = (V, E)$ directed graph with edge lengths. Obtain G' from G by adding a constant M to every edge's length. When is the shortest path between a source s and a destination t guaranteed to be the same in G and G'?

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

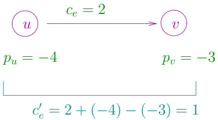


Quiz

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

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!

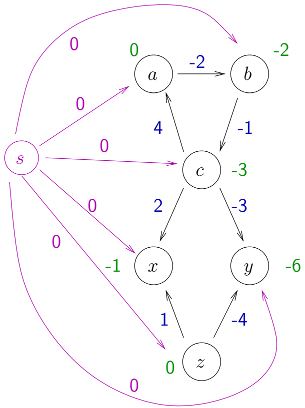
Requires Bellman-Ford, enables Dijkstra!

All-Pairs Shortest Paths (APSP)

Johnson's Algorithm

Algorithms: Design and Analysis, Part II

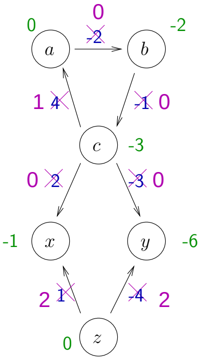
Example



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 $c'_e = c_e + p_u - p_v$.

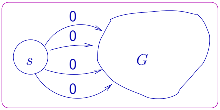
Note: After reweighting, all edge lengths nonnegative! $\Rightarrow$ Can compute all (reweighted) shortest paths via n Dijkstra computations! [No need for Bellman-Ford]

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

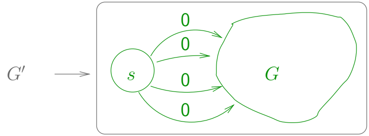
$= O(mn \log n)$. [Much better than Floyd-Warshall for sparse graphs!]

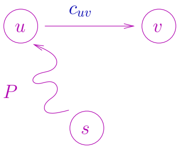
Correctness: Assuming $c'_e \geq 0$ for all edges e (see next slide for proof), correctness follows from last video's quiz.

[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!