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)

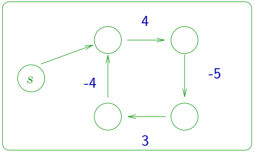
Bad news:

(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)

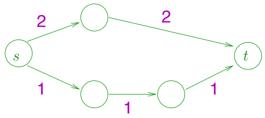
Intuition: Exploit sequential nature of paths. Subpath of a shortest path should itself be shortest.

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

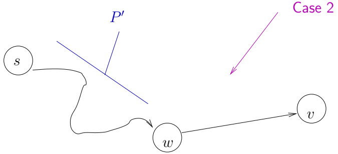
Example:



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 $\underline{\text{for each choice of the final hop } (w, c)}$

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 $i \in \{0, 1, \dots, n-1\}$ 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 $i = 1, 2, \ldots, n-1$

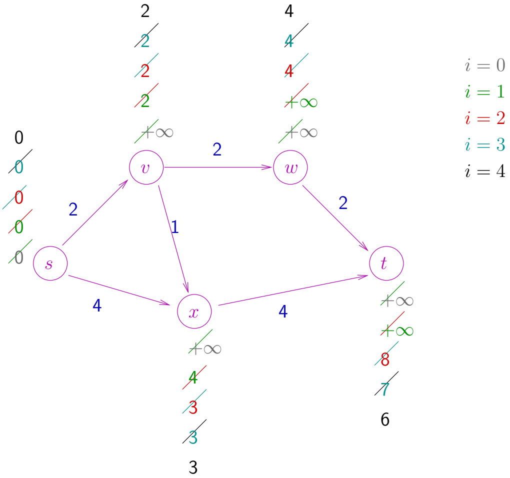
For each $v \in V$

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

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

Example

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



Tim Roughgarden

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

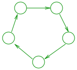
($\Rightarrow$) Already proved in correctness of Bellman-Ford
($\Leftarrow$) Assume $A[n-1,v] = A[n,v]$ for all $v \in V$. (Assume also these are finite ($< +\infty$))
Let $d(v)$ denote the common value of $A[n-1,v]$ and $A[n,v]$.

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

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

Now: Consider an arbitrary cycle C. 

$$\sum_{(w,v)\in C} \ge \sum_{(w,v)\in C}(d(w) - d(v)) = 0$$ 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 $s \to v$ 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+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

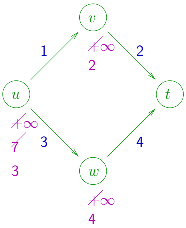
Fix: Switch from "pull-based" to "push-based": As soon as $A[i,v] < A[i-1,v]$, v notifies all of its neighbors.

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]

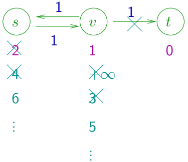
Example:



Handling Failures

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

Counting to Infinity:



Fix: Each V maintains entire shortest path 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).