Introduction to Algorithms

George Voutsadakis[1]

[1]Mathematics and Computer Science
Lake Superior State University

LSSU Math 400

1. Single-Source Shortest Paths
   - The Bellman-Ford Algorithm
   - Single-Source Shortest Paths in Directed Acyclic Graphs
   - Dijkstra's Algorithm
   - Difference Constraints and Shortest Paths
   - Proofs of Shortest-Paths Properties

Shortest Paths and Weights

- In a **shortest-paths problem**, we are given a weighted, directed graph $G = (V, E)$, with weight function $w : E \to \mathbb{R}$ mapping edges to real-valued weights.
- The **weight** $w(p)$ of path $p = \langle v_0, v_1, \ldots, v_k \rangle$ is the sum of the weights of its constituent edges: $w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$.
- We define the **shortest-path weight** $\delta(u, v)$ from u to v by

$$\delta(u, v) = \begin{cases} \min\{w(p) : u \overset{p}{\rightsquigarrow} v\}, & \text{if there is a path from } u \text{ to } v \\ \infty, & \text{otherwise} \end{cases}$$

- A **shortest path** from vertex u to vertex v is defined as any path $u \overset{p}{\rightsquigarrow} v$ with weight $w(p) = \delta(u, v)$.

Variants

- The **single-source shortest-paths problem**:

  Given a graph $G = (V, E)$, we want to find a shortest path from a given source vertex $s \in V$ to each vertex $v \in V$.
- The algorithm for this problem can solve other variants also.
  - **Single-destination shortest-paths problem**:

    Find a shortest path to a given destination vertex t from each vertex v.
  - **Single-pair shortest-path problem**:

    Find a shortest path from u to v for given vertices u and v.
  - **All-pairs shortest-paths problem**:

    Find a shortest path from u to v for every pair of vertices u and v.

  The last variant can be solved by running a single source algorithm once from each vertex, but there is a faster algorithm.

Optimal Substructure of a Shortest Path

- Shortest-paths algorithms typically rely on the property that a shortest path between two vertices contains other shortest paths within it.
- Optimal substructure is one of the key indicators that dynamic programming and the greedy method might apply.

Lemma (Subpaths of Shortest Paths are Shortest Paths)

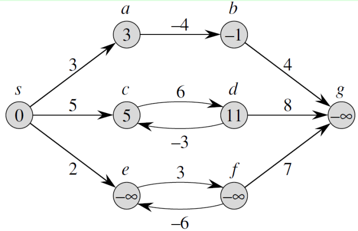
Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$, let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be a shortest path from vertex v_0 to vertex v_k and, for any i and j such that $0 \le i \le j \le k$, let $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ be the subpath of p from vertex v_i to vertex v_j. Then, p_{ij} is a shortest path from v_i to v_j.

- Decompose p into $v_0 \overset{p_{0i}}{\rightsquigarrow} v_i \overset{p_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$. Then $w(p) = w(p_{0i}) + w(p_{ij}) + w(p_{jk})$. Suppose there was a path p'_{ij} from v_i to v_j with weight $w(p'_{ij}) < w(p_{ij})$. Then, $v_0 \overset{p_{0i}}{\rightsquigarrow} v_i \overset{p'_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$ is a path from v_0 to v_k with weight $w(p_{0i}) + w(p'_{ij}) + w(p_{jk}) < w(p)$, a contradiction.

Negative-Weight Edges

- Some instances of the single-source shortest-paths problem may include edges whose weights are negative.
  - If the graph $G = (V.E)$ contains no negative weight cycles reachable from the source s, then, for all $v \in V$, the shortest-path weight $\delta(s, v)$ remains well defined, even if it has a negative value.
  - If the graph contains a negative-weight cycle reachable from s, however, shortest-path weights are not well defined.
    No path from s to a vertex on the cycle can be a shortest path, since we can always find a path with lower weight by following the proposed "shortest" path and then traversing the negative-weight cycle.
- If there is a negative weight cycle on some path from s to v, we define $\delta(s, v) = -\infty$.

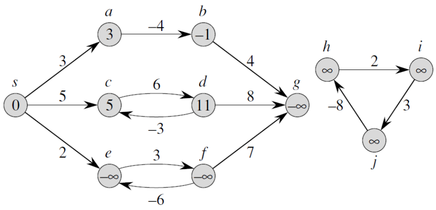
The Effect of Negative-Weight Edges



- There is only one path from s to a. So we have $\delta(s,a) = w(s,a) = 3$.
- There is only one path from s to b. So $\delta(s,b) = w(s,a) + w(a,b)$ $= 3 + (-4) = -1$.
- There are infinitely many paths from s to c : $\langle s,c\rangle$, $\langle s,c,d,c\rangle$, $\langle s,c,d,c,d,c\rangle$, and so on.

  The cycle $\langle c,d,c\rangle$ has weight $6 + (-3) = 3 > 0$. So the shortest path from s to c is $\langle s,c\rangle$, with weight $\delta(s,c) = w(s,c) = 5$.
- Similarly, he shortest path from s to d is $\langle s,c,d\rangle$, with weight $\delta(s,d) = w(s,c) + w(c,d) = 11$.

The Effect of Negative-Weight Edges (Cont'd)



- There are infinitely many paths from s to e: $\langle s, e\rangle$, $\langle s, e, f, e\rangle$, $\langle s, e, f, e, f, e\rangle$, and so on. The cycle $\langle e, f, e\rangle$ has weight $3 + (-6) = -3 < 0$. So there is no shortest path from s to e. By traversing the negative weight cycle $\langle e, f, e\rangle$ arbitrarily many times, we can find paths from s to e with arbitrarily large negative weights. So $\delta(s, e) = -\infty$.
- Similarly, $\delta(s, f) = -\infty$.
- Because g is reachable from f, we can also find paths with arbitrarily large negative weights from s to g. So $\delta(s, g) = -\infty$.
- Vertices h, i and j also form a negative-weight cycle. However, hey are not reachable from s. So $\delta(s, h) = \delta(s, i) = \delta(s, j) = \infty$.

Cycles

- A shortest path cannot contain a negative weight cycle.
- Nor can it contain a positive weight cycle.

  Suppose $p = \langle v_0, v_1, \ldots, v_k \rangle$ is a path and $c = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ is a positive weight cycle on this path (so that $v_i = v_j$ and $w(c) > 0$). Then the path $p' = \langle v_0, v_1, \ldots, v_i, v_{j+1}, v_{j+2}, \ldots, v_k \rangle$ has weight $w(p') = w(p) - w(c) < w(p)$. So p cannot be a shortest path from v_0 to v_k.
- That leaves only 0-weight cycles. We can remove a 0-weight cycle from any path to produce another path whose weight is the same.
- Therefore, without loss of generality we can assume that when we are finding shortest paths, they have no cycles.

  Since any acyclic path in a graph $G = (V, E)$ contains at most $|V|$ distinct vertices, it also contains at most $|V| - 1$ edges.

  We can restrict attention to shortest paths of at most $|V| - 1$ edges.

Representing Shortest Paths

- Given a graph $G = (V, E)$, we maintain for each vertex $v \in V$ a **predecessor** $v.\pi$ that is either another vertex or NIL.
- The shortest paths algorithms we study set the π attributes so that the chain of predecessors originating at a vertex v runs backwards along a shortest path from s to v.
- Thus, given a vertex v for which $v.\pi \neq \text{NIL}$, the procedure $\text{PRINTPATH}(G, s, v)$ will print a shortest path from s to v.
- In the midst of executing a shortest paths algorithm, however, the π-values might not indicate shortest paths.
- As in breadth first search, we shall be interested in the **predecessor subgraph** $G_\pi = (V_\pi, E_\pi)$ induced by the π values:
  - We define the vertex set V_π as the set of vertices of G with non-NIL predecessors, plus s: $V_\pi = \{v \in V : v.\pi \neq \text{NIL}\} \cup \{s\}$.
  - The directed edge set E_π is the set of edges induced by the π values for vertices in V_π: $E_\pi = \{(v.\pi, v) \in E : v \in V_\pi - \{s\}\}$.

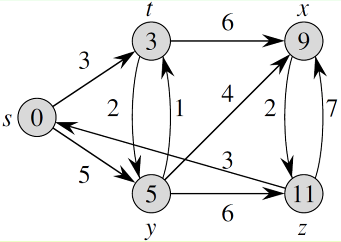
Shortest-Paths Trees

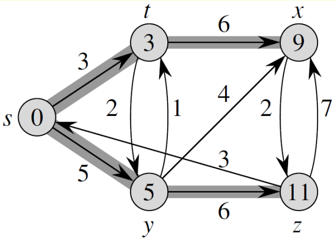
- We prove that the π values produced by our algorithms have the property that at termination G_π is a "shortest-paths tree", a rooted tree containing a shortest path from the source s to every vertex that is reachable from s.
- Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbb{R}$.

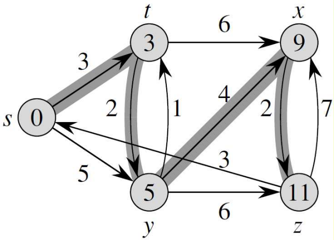
  Assume that G contains no negative weight cycles reachable from the source vertex $s \in V$, so that shortest paths are well defined.
- A **shortest paths tree** rooted at s is a directed subgraph $G' = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$, such that:
  1. V' is the set of vertices reachable from s in G;
  2. G' forms a rooted tree with root s;
  3. For all $v \in V'$, the unique simple path from s to v in G' is a shortest path from s to v in G.

Illustrating Shortest-Paths Trees

- Shortest paths are not necessarily unique, and neither are shortest paths trees.







Relaxation: Initialization

- The algorithms we present use the technique of **relaxation**.
- For each vertex $v \in V$, we maintain an attribute $v.d$, which is an upper bound on the weight of a shortest path from source s to v. We call $v.d$ a **shortest path estimate**.
- We initialize the shortest-path estimates and predecessors by the following $\Theta(V)$-time procedure:

INITIALIZESINGLESOURCE(G, s)

for each vertex $v \in G.V$
    $v.d = \infty$
    $v.\pi = \text{NIL}$
$s.d = 0$

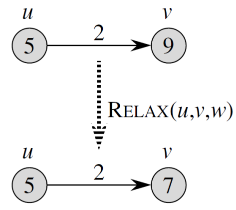
- After initialization, $v.\pi = \text{NIL}$, for all $v \in V$, $s.d = 0$, and $v.d = \infty$, for $v \in V - \{s\}$.

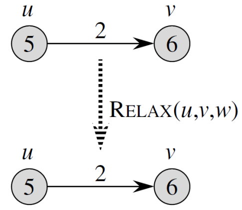
Relaxation: The Evolution

- The process of **relaxing** an edge (u, v) consists of testing whether we can improve the shortest path to v found so far by going through u and, if so, updating $v.d$ and $v.\pi$.
- A relaxation step may decrease the value of the shortest-path estimate $v.d$ and update v's predecessor field $v.\pi$.

Relax(u, v, w)

1. if $v.d > u.d + w(u, v)$
2. $\quad v.d = u.d + w(u, v)$
3. $\quad v.\pi = u$





Properties of Shortest Paths and Relaxation

- Suppose the graph is initialized by $\text{INITIALIZESINGLESOURCE}(G, s)$ and that shortest path estimates and the predecessor subgraph change only due to relaxation steps.
  - **Triangle Inequality**: For all $(u, v) \in E$, $\delta(s, v) \leq \delta(s, u) + w(u, v)$.
  - **Upper-Bound Property**: We always have $v.d \geq \delta(s, v)$, for all vertices $v \in V$, and once $v.d$ achieves the value $\delta(s, v)$, it never changes.
  - **No-Path Property**: If there is no path from s to v, then we always have $v.d = \delta(s, v) = \infty$.
  - **Convergence Property**: If $s \rightsquigarrow u \to v$ is a shortest path in G, for some $u, v \in V$, and if $u.d = \delta(s, u)$ at any time prior to relaxing edge (u, v), then $v.d = \delta(s, v)$ at all times afterward.
  - **Path-Relaxation Property**: If $p = \langle v_0, v_1, \ldots, v_k \rangle$ is a shortest path from $s = v_0$ to v_k, and we relax the edges of p in the order (v_0, v_1), $(v_1, v_2), \ldots, (v_{k-1}, v_k)$, then $v_k.d = \delta(s, v_k)$. This property holds regardless of any other relaxation steps that occur.
  - **Predecessor-Subgraph Property**: Once $v.d = \delta(s, v)$, for all $v \in V$, the predecessor subgraph is a shortest paths tree rooted at s.

Subsection 1

The Bellman-Ford Algorithm

Setting Up the Bellman-Ford Algorithm

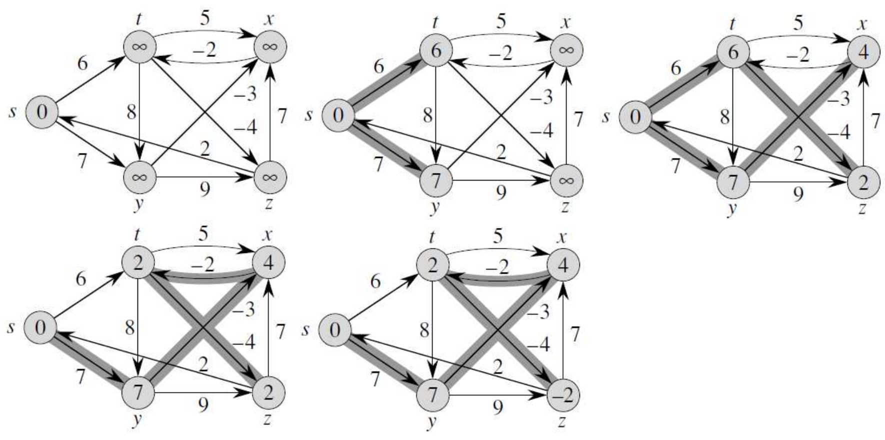
- The **Bellman-Ford algorithm** solves the single-source shortest-paths problem in the general case in which edge weights may be negative.
- Given a weighted, directed graph $G = (V, E)$, with source s and weight function $w : E \to \mathbb{R}$, the Bellman-Ford algorithm returns a boolean value indicating whether or not there is a negative-weight cycle that is reachable from the source.
  - If there is such a cycle, the algorithm indicates that no solution exists.
  - If there is no such cycle, the algorithm produces the shortest paths and their weights.
- The algorithm relaxes edges, progressively decreasing an estimate $v.d$ on the weight of a shortest path from the source s to each vertex $v \in V$, until it achieves the actual shortest path weight $\delta(s, v)$.
- The algorithm returns TRUE if and only if the graph contains no negative-weight cycles that are reachable from the source.

The Bellman-Ford Procedure

BellmanFord(G, w, s)

```
InitializeSingleSource(G, s)
for i = 1 to |G.V| - 1
    for each edge (u, v) ∈ G.E
        Relax(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

Illustrating the Bellman-Ford Procedure



How the Bellman-Ford Procedure Works

- In Line 1, the d and π values of all vertices are initialized.
- Then the algorithm makes $|V|-1$ passes over the edges of the graph.

  Each pass is one iteration of the for loop of Lines 2-4 and consists of relaxing each edge of the graph once.
- After making $|V|-1$ passes, Lines 5-8 check for a negative-weight cycle and return the appropriate boolean value.
- The Bellman-Ford algorithm runs in time $O(|V||E|)$.
  - The initialization in Line 1 takes $\Theta(|V|)$ time;
  - Each of the $|V|-1$ passes over the edges in Lines 2-4 takes $\Theta(|E|)$ time;
  - The for loop of Lines 5-7 takes $O(|E|)$ time.

Correctness without Negative-Weight Cycles

- If there are no negative-weight cycles, the algorithm computes correct shortest-path weights for all vertices reachable from the source.

Lemma

Let $G = (V, E)$ be a weighted, directed graph with source s and weight function $w : E \to \mathbb{R}$, and assume that G contains no negative-weight cycles that are reachable from s. Then, after the $|V| - 1$ iterations of the for loop of Lines 2-4 of BELLMANFORD, we have $v.d = \delta(s, v)$, for all vertices v that are reachable from s.

- Consider a v reachable from s. Let $p = \langle v_0, v_1, \ldots, v_k \rangle$, where $v_0 = s$ and $v_k = v$, be any acyclic shortest path from s to v. Path p has at most $|V| - 1$ edges. So $k \leq |V| - 1$. Each of the $|V| - 1$ iterations of the for loop of Lines 2-4 relaxes all E edges. Among the edges relaxed in the ith iteration, for $i = 1, 2, \ldots, k$, is (v_{i-1}, v_i). By the Path Relaxation Property, therefore, $v.d = v_k.d = \delta(s, v_k) = \delta(s, v)$.

Consequence of Correctness

Corollary

Let $G = (V, E)$ be a weighted, directed graph, with source vertex s and weight function $w : E \to \mathbb{R}$, and assume that G contains no negative weight cycles that are reachable from s. Then, for each vertex $v \in V$, there is a path from s to v if and only if BELLMANFORD terminates with $v.d < \infty$ when it is run on G.

- BELLMANFORD terminates with $v.d < \infty$ when it is run on G

  iff, by the Lemma, BELLMANFORD terminates with $\delta(s, v) < \infty$ when it is run on G

  iff, by definition, there is a path from s to v in G.

Correctness of the Bellman-Ford Algorithm

Theorem (Correctness of the Bellman-Ford algorithm)

Let BellmanFord be run on a weighted, directed graph $G = (V, E)$, with source s and weight function $w : E \to \mathbb{R}$. If G contains no negative weight cycles that are reachable from s, then the algorithm returns TRUE, we have $v.d = \delta(s, v)$, for all vertices $v \in V$, and the predecessor subgraph G_π is a shortest paths tree rooted at s. If G does contain a negative weight cycle reachable from s, then the algorithm returns FALSE.

- Suppose that graph G contains no negative weight cycles that are reachable from the source s.

  Claim: At termination, $v.d = \delta(s, v)$, for all vertices $v \in V$.

  If vertex v is reachable from s, then use the Lemma. If v is not reachable from s, then the claim follows from the No-Path Property.

  The Predecessor-Subgraph Property, along with the Claim, implies that G_π is a shortest paths tree.

Correctness of the Bellman-Ford Algorithm (Cont'd)

We use the claim to show that $\textsc{BellmanFord}$ returns TRUE.

At termination, we have, for all edges $(u, v) \in E$,

$$\begin{array}{rcll} v.d & = & \delta(s, v) & \\ & \leq & \delta(s, u) + w(u, v) & \text{(by the triangle inequality)} \\ & = & u.d + w(u, v). & \end{array}$$

So none of the tests in Line 6 causes $\textsc{BellmanFord}$ to return FALSE. Therefore, it returns TRUE.

Correctness of the Bellman-Ford (Second Case)

- Suppose that G contains a negative weight cycle reachable from s.

  Let this cycle be $c = \langle v_0, v_1, \ldots, v_k \rangle$, where $v_0 = v_k$. Then, $\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$. Assume the algorithm returns TRUE. Thus, $v_i.d \leq v_{i-1}.d + w(v_{i-1}, v_i)$, for $i = 1, 2, \ldots, k$. Summing the inequalities around cycle c gives us

$$\begin{array}{rcl} \sum_{i=1}^{k} v_i.d & \leq & \sum_{i=1}^{k} [v_{i-1}.d + w(v_{i-1}, v_i)] \\ & = & \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} w(v_{i-1}, v_i). \end{array}$$

  Since $v_0 = v_k$, each vertex in c appears exactly once in each of $\sum_{i=1}^{k} v_i.d$ and $\sum_{i=1}^{k} v_{i-1}.d$. So $\sum_{i=1}^{k} v_i.d = \sum_{i=1}^{k} v_{i-1}.d$.

  Moreover, by the Corollary, $v_i.d$ is finite for $i = 1, 2, \ldots, k$.

  Thus, $0 \leq \sum_{i=1}^{k} w(v_{i-1}, v_i)$, a contradiction.

Subsection 2

Single-Source Shortest Paths in Directed Acyclic Graphs

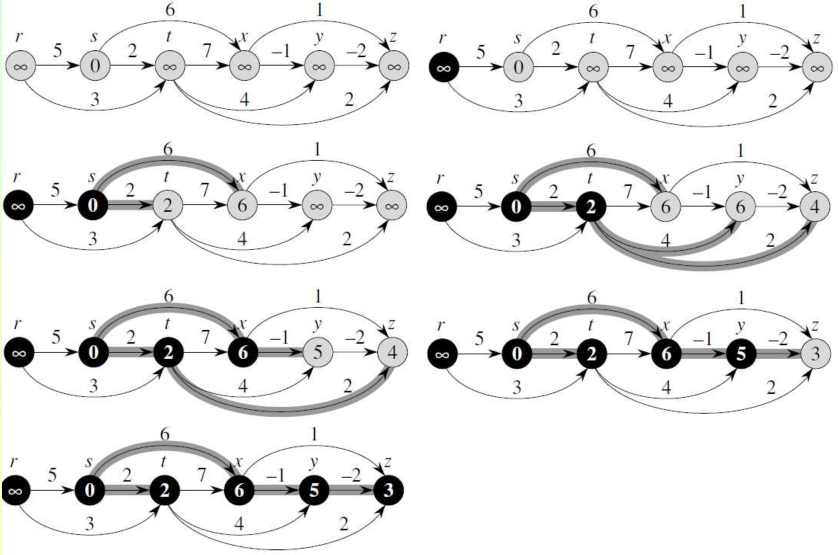
Shortest Paths in DAGs

- By relaxing the edges of a weighted DAG (directed acyclic graph) $G = (V, E)$ according to a topological sort of its vertices, we can compute shortest paths from a single source in $\Theta(|V| + |E|)$ time.
- Shortest paths are always well defined in a DAG, since even if there are negative-weight edges, no negative-weight cycles can exist.
- The algorithm does the following:
  - Topologically sorts the DAG;
  - Makes just one pass over the vertices in the topologically sorted order; As it processes each vertex, it relax each edge that leaves the vertex.

DAGShortestPaths(G, w, s)

```
1. topologically sort the vertices of G
2. InitializeSingleSource(G, s)
3. for each vertex u, taken in topologically sorted order
4.    for each vertex v ∈ G.Adj[u]
5.       Relax(u, v, w)
```

Illustrating the DAG Shortest Paths Procedure



Running Time of DAGShortestPaths

- The topological sort of Line 1 takes $\Theta(|V| + |E|)$ time.
- The call of InitializeSingleSource in Line 2 takes $\Theta(|V|)$ time.
- The for loop of Lines 3-5 makes one iteration per vertex.
- Altogether, the for loop of Lines 4-5 relaxes each edge exactly once (aggregate analysis).

  Each iteration of the inner for loop takes $\Theta(1)$ time.
- It follows that the total running time is $\Theta(|V| + |E|)$, which is linear in the size of an adjacency-list representation of the graph.

Correctness of DAGShortestPaths

Theorem

If a weighted, directed graph $G = (V, E)$ has source vertex s and no cycles, then at the termination of the DAGShortestPaths procedure, $v.d = \delta(s, v)$, for all vertices $v \in V$, and the predecessor subgraph G_π is a shortest-paths tree.

- We first show that, at termination, $v.d = \delta(s, v)$, for all $v \in V$.
  - If v is not reachable from s, then $v.d = \delta(s, v) = \infty$ by the No-Path Property.
  - If v is reachable from s, there is a shortest path $p = \langle v_0, v_1, \ldots, v_k \rangle$, where $v_0 = s$ and $v_k = v$. Because we process the vertices in topologically sorted order, the edges on p are relaxed in the order $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$. The Path-Relaxation Property implies that $v_i.d = \delta(s, v_i)$ at termination, for $i = 0, 1, \ldots, k$.

By the Predecessor Subgraph Property, G_π is a shortest-paths tree.

Subsection 3

Dijkstra's Algorithm

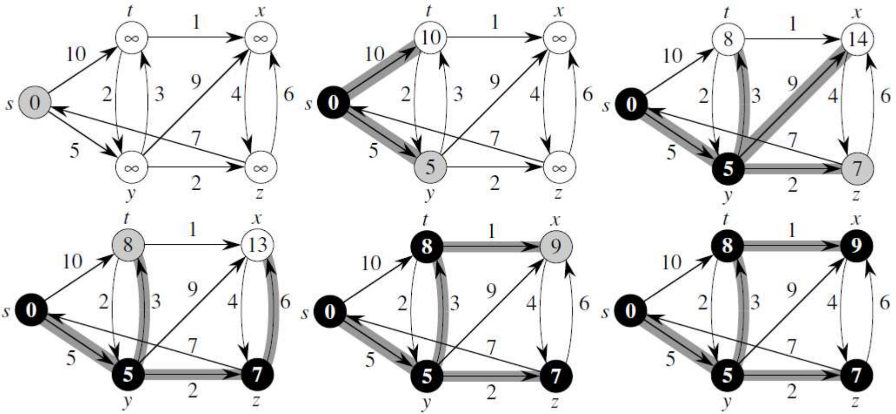
Dijkstra's Algorithm

- Solves the single-source shortest-paths problem on a weighted, directed graph $G = (V, E)$ with all edge weights nonnegative.
- The algorithm maintains a set S of vertices whose final shortest path weights from the source s have already been determined.
  - Repeatedly select $u \in V - S$ with the minimum shortest-path estimate: add u to S, and relax all edges leaving u.
- We use a min-priority queue Q of vertices, keyed by their d values.

$\textsc{Dijkstra}(G, w, s)$

```
InitializeSingleSource(G, s)
S = ∅
Q = G.V
while Q ≠ ∅
    u = ExtractMin(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        Relax(u, v, w)
```

Illustrating Dijkstra's Algorithm



How Dijkstra's Algorithm Works

- Line 1 initializes the d and π values.
- Line 2 initializes the set S to the empty set.
- The algorithm maintains the invariant that $Q = V - S$ at the start of each iteration of the while loop of Lines 4-8.
  - Line 3 initializes the min-priority queue Q to contain all vertices in V. Since $S = \emptyset$, the invariant is true after Line 3.
  - Each time through the while loop of Lines 4-8, Line 5 extracts a vertex u from $Q = V - S$ and Line 6 adds it to S, maintaining the invariant.

  Vertex u, therefore, has the smallest shortest-path estimate of any vertex in $V - S$.
- Then, Lines 7-8 relax each edge (u, v) leaving u, thus updating the estimate $v.d$ and the predecessor $v.\pi$ if we can improve the shortest path to v found so far by going through u.
- Observe that the algorithm never inserts vertices into Q after Line 3 and that each vertex is extracted from Q and added to S exactly once, so that the while loop of Lines 4-8 iterates exactly $|V|$ times.

Correctness of Dijkstra's Algorithm

Theorem (Correctness of Dijkstra's Algorithm)

Dijkstra's algorithm, run on a weighted, directed graph $G = (V, E)$ with nonnegative weight function w and source s, terminates with $u.d = \delta(s, u)$, for all vertices $u \in V$.

- We use the following loop invariant:

  > At the start of each iteration of the while loop of Lines 4-8, $v.d = \delta(s, v)$, for each vertex $v \in S$.

  It suffices to show for each vertex $u \in V$, we have $u.d = \delta(s, u)$ at the time when u is added to set S.

  Once we show that $u.d = \delta(s, u)$, we rely on the Upper-Bound Property to show that the equality holds at all times thereafter.
- **Initialization**: Initially, $S = \emptyset$. So the invariant is trivially true.

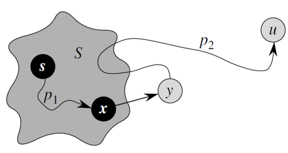
Maintenance

- **Maintenance**: We wish to show that in each iteration, $u.d = \delta(s, u)$ for the vertex added to set S. For the purpose of contradiction, let u be the first vertex for which $u.d \neq \delta(s, u)$ when it is added to set S. We look at the beginning of the iteration of the while loop in which u is added to S. We derive the contradiction that $u.d = \delta(s, u)$ at that time by examining a shortest path from s to u.

  We must have $u \neq s$ because s is the first vertex added to set S and $s.d = \delta(s, s) = 0$ at that time. Because $u \neq s$, we also have that $S \neq \emptyset$ just before u is added to S. There must be some path from s to u, for otherwise $u.d = \delta(s, u) = \infty$ by the No-Path Property, which would violate our assumption that $u.d \neq \delta(s, u)$. Because there is at least one path, there is a shortest path p from s to u. Prior to adding u to S, path p connects a vertex in S, namely s, to a vertex in $V - S$, namely u. Let us consider the first vertex y along p, such that $y \in V - S$, and let $x \in S$ be y's predecessor.

Maintenance (Cont'd)

- p can be decomposed as $s \overset{p_1}{\rightsquigarrow} x \to y \overset{p_2}{\rightsquigarrow} u$.
  Claim: $y.d = \delta(s,y)$ when u is added to S. To prove this, observe that $x \in S$. Then, because u is chosen as the first vertex for which $u.d \neq \delta(s,u)$ when it is added to S, we had $x.d = \delta(s,x)$ when x was added to S. Edge (x,y) was relaxed at that time, so the claim follows from the Convergence Property.



  We can now obtain a contradiction to prove that $u.d = \delta(s,u)$.

  Because y appears before u on a shortest path from s to u and all edge weights are nonnegative, we have $\delta(s,y) \leq \delta(s,u)$. Thus, $y.d = \delta(s,y) \leq \delta(s,u) \leq u.d$ (by the Upper-Bound Property). But because both u and y were in $V - S$ when u was chosen in Line 5, we have $u.d \leq y.d$. Hence, $y.d = \delta(s,y) = \delta(s,u) = u.d$. Consequently, $u.d = \delta(s,u)$. This contradicts our choice of u.

Termination

- We conclude that $u.d = \delta(s, u)$ when u is added to S, and that this equality is maintained at all times thereafter.
- **Termination**: At termination, $Q = \emptyset$. Along with our earlier invariant that $Q = V - S$, implies that $S = V$. Thus, $u.d = \delta(s, u)$, for all vertices $u \in V$.

Corollary

If we run Dijkstra's algorithm on a weighted, directed graph $G = (V, E)$ with nonnegative weight function w and source s, then at termination, the predecessor subgraph G_π is a shortest-paths tree rooted at s.

Aggregate Analysis Based on Operations

- Dijkstra's algorithm maintains the min-priority queue Q by calling three priority-queue operations.
  - INSERT (implicit in Line 3);
  - EXTRACTMIN (Line 5);
  - DECREASEKEY (implicit in RELAX, which is called in Line 8).

  The algorithm calls both INSERT and EXTRACTMIN once per vertex.

  Each vertex $u \in V$ is added to set S exactly once.

  Thus, each edge in the adjacency list Adj[u] is examined in the for loop of Lines 7-8 exactly once during the course of the algorithm.

  Since the total number of edges in all the adjacency lists is $|E|$, this for loop iterates a total of $|E|$ times.

  Thus, the algorithm calls DECREASEKEY at most $|E|$ times overall.

Analysis and Implementation

- The running time of Dijkstra's algorithm depends on how we implement the min-priority queue:
- Suppose we maintain the min-priority queue by taking advantage of the vertices being numbered 1 to $|V|$.

  We simply store $v.d$ in the vth entry of an array.
- Each INSERT and DECREASEKEY operation takes $O(1)$ time.
- Each EXTRACTMIN operation takes $O(|V|)$ time (since we have to search through the entire array).
- Thus, total time is $O\left(|V|^2 + |E|\right) = O\left(|V|^2\right)$.

Subsection 4

Difference Constraints and Shortest Paths

Linear Programming

- The general **linear programming problem**:

  Given an $m \times n$ matrix A, an m-vector b and an n-vector c, find a vector x of n elements that maximizes the **objective function** $\sum_{i=1}^{n} c_i x_i$ subject to the m **constraints** given by $Ax \leq b$.
- Importance of understanding the setup of linear-programming problems:
  - If we know that we can cast a given problem as a polynomial-sized linear-programming problem, then we immediately have a polynomial time algorithm to solve the problem.
  - Faster algorithms exist for many special cases of linear programming, e.g., the single-pair shortest-path problem and the maximum-flow problem.
- In a **feasibility problem**, we only wish to find any **feasible solution**, i.e., any vector x that satisfies $Ax \leq b$, or to determine that no feasible solution exists.

Systems of Difference Constraints

- In a **system of difference constraints**, each row of the linear programming matrix A contains one 1 and one -1, and all other entries of A are 0.
- Thus, the constraints given by $Ax \leq b$ are a set of m **difference constraints** involving n unknowns, in which each constraint is a simple linear inequality of the form $x_j - x_i \leq b_k$, where $1 \leq i, j \leq n$, $i \neq j$ and $1 \leq k \leq m$.

  Example: The problem of finding a 5-vector $x = (x_i)$ that satisfies

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} \leq \begin{pmatrix} 0 \\ -1 \\ 1 \\ 5 \\ 4 \\ -1 \\ -3 \\ -3 \end{pmatrix}.$$

Example (Cont'd)

- This problem is equivalent to finding values for the unknowns x_1, x_2, x_3, x_4, x_5, satisfying the following 8 difference constraints:

$$\begin{array}{rcr} x_1 - x_2 & \leq & 0 \\ x_1 - x_5 & \leq & -1 \\ x_2 - x_5 & \leq & 1 \\ x_3 - x_1 & \leq & 5 \\ x_4 - x_1 & \leq & 4 \\ x_4 - x_3 & \leq & -1 \\ x_5 - x_3 & \leq & -3 \\ x_5 - x_4 & \leq & -3 \end{array}$$

  One solution to this problem is $x = (-5, -3, 0, -1, -4)$, which you can verify directly by checking each inequality. In fact, this problem has more than one solution. Another is $x' = (0, 2, 5, 4, 1)$. These two solutions are related: each component of x' is 5 larger than the corresponding component of x. This fact is not mere coincidence.

Adding Constants to Solutions

Lemma

Let $x = \langle x_1, x_2, \ldots, x_n \rangle$ be a solution to a system $Ax \leq b$ of difference constraints, and let d be any constant. Then $x + d = \langle x_1 + d, x_2 + d, \ldots, x_n + d \rangle$ is a solution to $Ax \leq b$ as well.

- For each x_i and x_j, we have

$$(x_j + d) - (x_i + d) = x_j - x_i.$$

Thus, if x satisfies $Ax \leq b$, so does $x + d$.

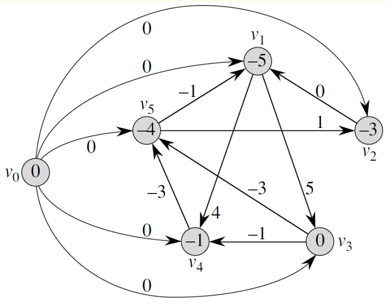
Constraint Graphs

- We can interpret systems of difference constraints from a graph theoretic point of view.
- In a system $Ax \leq b$ of difference constraints, we view the $m \times n$ linear programming matrix A as the transpose of an incidence matrix for a graph with n vertices and m edges.
  - Each vertex v_i in the graph, for $i = 1, 2, \ldots, n$, corresponds to one of the n unknown variables x_i.
  - Each directed edge in the graph corresponds to one of the m inequalities involving two unknowns.
- More formally, given a system $Ax \leq b$ of difference constraints, the corresponding **constraint graph** is a weighted, directed graph $G = (V, E)$, where:
  - $V = \{v_0, v_1, \ldots, v_n\}$;
  - $E = \{(v_i, v_j) : x_j - x_i \leq b_k \text{ is a constraint}\} \cup \{(v_0, v_1), (v_0, v_2), \ldots, (v_0, v_n)\}$.

An Example

- The vertex set V consists of a vertex v_i for each unknown x_i, plus an additional vertex v_0.
- The edge set E contains an edge for each difference constraint, plus an edge (v_0, v_i) for each unknown x_i.
  - If $x_j - x_i \leq b_k$ is a difference constraint, then the weight of edge (v_i, v_j) is $w(v_i, v_j) = b_k$.
  - The weight of each edge leaving v_0 is 0.

Example:



Feasible Solutions and the Constraint Graph

Theorem

Given a system $Ax \leq b$ of difference constraints, let $G = (V, E)$ be the corresponding constraint graph. If G contains no negative-weight cycles, then $x = (\delta(v_0, v_1), \delta(v_0, v_2), \delta(v_0, v_3), \ldots, \delta(v_0, v_n))$ is a feasible solution for the system. If G contains a negative-weight cycle, then there is no feasible solution for the system.

Claim: If the constraint graph contains no negative-weight cycles, then $x = (\delta(v_0, v_1), \delta(v_0, v_2), \delta(v_0, v_3), \ldots, \delta(v_0, v_n))$ is a feasible solution.

Consider any edge $(v_i, v_j) \in E$. By the triangle inequality, $\delta(v_0, v_j) \leq \delta(v_0, v_i) + w(v_i, v_j)$, i.e., $\delta(v_0, v_j) - \delta(v_0, v_i) \leq w(v_i, v_j)$. Thus, the values $x_i = \delta(v_0, v_i)$, $x_j = \delta(v_0, v_j)$ satisfy the difference constraint $x_j - x_i \leq w(v_i, v_j)$ that corresponds to edge (v_i, v_j).

Feasible Solutions and the Constraint Graph (Cont'd)

- We show that if the constraint graph contains a negative-weight cycle, then the system has no feasible solution.
  Without loss of generality, let the negative-weight cycle be $c = \langle v_1, v_2, \ldots, v_k \rangle$, where $v_1 = v_k$. c corresponds to:

$$\begin{array}{rcl} x_2 - x_1 & \leq & w(v_1, v_2) \\ x_3 - x_2 & \leq & w(v_2, v_3) \\ & \vdots & \\ x_{k-1} - x_{k-2} & \leq & w(v_{k-2}, v_{k-1}) \\ x_k - x_{k-1} & \leq & w(v_{k-1}, v_k) \end{array}$$

  We assume that x has a solution satisfying each of these k inequalities and derive a contradiction. The solution must also satisfy the inequality that results when we sum the k inequalities. The left-hand side of the sum is 0. The right-hand side sums to $w(c)$. Thus, $0 \leq w(c)$. Since c is a negative-weight cycle, $w(c) < 0$.

Solving Systems of Difference Constraints

- The Theorem tells us that we can use the Bellman-Ford algorithm to solve a system of difference constraints.
- Because the constraint graph contains edges from the source vertex v_0 to all other vertices, any negative-weight cycle in the constraint graph is reachable from v_0.
  - If the Bellman-Ford algorithm returns TRUE, then the shortest-path weights give a feasible solution to the system.
  - If the Bellman-Ford algorithm returns FALSE, there is no feasible solution to the system of difference constraints.
- A system of difference constraints with m constraints on n unknowns produces a graph with $n+1$ vertices and $n+m$ edges.

  Using the Bellman-Ford algorithm, we can solve the system in $\mathrm{O}((n+1)(n+m)) = \mathrm{O}\left(n^2+nm\right)$ time.

Subsection 5

Proofs of Shortest-Paths Properties

The Triangle Inequality

Lemma (Triangle Inequality)

Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbb{R}$ and source vertex s. Then, for all edges $(u, v) \in E$, we have $\delta(s, v) \leq \delta(s, u) + w(u, v)$.

- Suppose that p is a shortest path from source s to vertex v. Then p has no more weight than any other path from s to v. Specifically, path p has no more weight than the particular path that takes a shortest path from source s to vertex u and then takes edge (u, v).

  The case in which there is no shortest path from s to v can be easily handled.

Effects of Relaxation: The Upper Bound Property

Lemma (Upper-Bound Property)

Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbb{R}$. Let $s \in V$ be the source vertex, and let the graph be initialized by $\text{INITIALIZESINGLESOURCE}(G, s)$. Then, $v.d \geq \delta(s, v)$, for all $v \in V$, and this invariant is maintained over any sequence of relaxation steps on the edges of G. Moreover, once $v.d$ achieves its lower bound $\delta(s, v)$, it never changes.

- We prove the invariant $v.d \geq \delta(s, v)$, for all vertices $v \in V$, by induction over the number of relaxation steps.
  - For the basis, $v.d \geq \delta(s, v)$ is certainly true after initialization:
    - $v.d = \infty$ implies $v.d \geq \delta(s, v)$, for all $v \in V - \{s\}$;
    - $s.d = 0 \geq \delta(s, s)$ (note that $\delta(s, s) = -\infty$, if s is on a negative-weight cycle, and 0, otherwise).

Effects of Relaxation: The Upper Bound Property (Cont'd)

- For the inductive step, consider the relaxation of an edge (u, v).
  By the inductive hypothesis, $x.d \geq \delta(s, x)$, for all $x \in V$, prior to the relaxation. The only d value that may change is $v.d$.
  If it changes, we have

$$\begin{array}{rcll} v.d & = & u.d + w(u, v) & \\ & \geq & \delta(s, u) + w(u, v) & \text{(by inductive hypothesis)} \\ & \geq & \delta(s, v). & \text{(by triangle inequality)} \end{array}$$

  So the invariant is maintained.

To see that the value of $v.d$ never changes once $v.d = \delta(s, v)$, note that:

- $v.d$ cannot decrease because we have just shown that $v.d \geq \delta(s, v)$;
- It cannot increase because relaxation steps do not increase d values.

Effects of Relaxation: No-Path Property

Corollary (No-Path Property)

Suppose that in a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$, no path connects a source vertex $s \in V$ to a given $v \in V$. Then, after initialization by $\textsc{InitializeSingleSource}(G, s)$, we have $v.d = \delta(s, v) = \infty$, and this equality is maintained as an invariant over any sequence of relaxation steps on the edges of G.

- By the Upper-Bound Property, $\infty = \delta(s, v) \leq v.d$.
  It follows that $v.d = \infty = \delta(s, v)$.

The Convergence Property: A Lemma

Lemma

Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbb{R}$, and let $(u, v) \in E$. Then, immediately after relaxing edge (u, v) by executing $\text{Relax}(u, v, w)$, we have $v.d \leq u.d + w(u, v)$.

- If, just prior to relaxing edge (u, v), we have
  - $v.d > u.d + w(u, v)$, then $v.d = u.d + w(u, v)$ afterward.
  - $v.d \leq u.d + w(u, v)$, then neither $u.d$ nor $v.d$ changes.
    So $v.d \leq u.d + w(u, v)$ afterward.

The Convergence Property

Lemma (Convergence Property)

Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbb{R}$, $s \in V$ a source vertex and $s \leadsto u \to v$ a shortest path in G for some vertices $u, v \in V$. Suppose that G is initialized by $\textsc{InitializeSingleSource}(G, s)$ and then a sequence of relaxation steps that includes the call $\textsc{Relax}(u, v, w)$ is executed on the edges of G. If $u.d = \delta(s, u)$ at any time prior to the call, then $v.d = \delta(s, v)$ at all times after the call.

- By the Upper-Bound Property, if $u.d = \delta(s, u)$ at some point prior to relaxing edge (u, v), then this equality holds thereafter. In particular, after relaxing edge (u, v), we have $v.d \leq u.d + w(u, v)$ (by the Lemma) $= \delta(s, u) + w(u, v) = \delta(s, v)$ (by the Subpaths Lemma). By the Upper-Bound Property, $v.d \geq \delta(s, v)$. Therefore, $v.d = \delta(s, v)$, and this equality is maintained thereafter.

The Path-Relaxation Property

Lemma (Path-Relaxation Property)

Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbb{R}$, and let $s \in V$ be a source vertex. Consider any shortest path $p = \langle v_0, v_1, \ldots, v_k \rangle$ from $s = v_0$ to v_k. If G is initialized by $\text{Initialize SingleSource}(G, s)$ and then a sequence of relaxation steps occurs that includes, in order, relaxing the edges $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, then $v_k.d = \delta(s, v_k)$ after these relaxations and at all times afterward. This property holds no matter what other edge relaxations occur, including relaxations that are intermixed with relaxations of the edges of p.

- We show, by induction, that after the i-th edge of path p is relaxed, we have $v_i.d = \delta(s, v_i)$.

The Path-Relaxation Property (Cont'd)

- For the basis, $i = 0$, and before any edges of p have been relaxed, we have from the initialization that
$$v_0.d = s.d = 0 = \delta(s, s).$$
By the Upper-Bound Property, the value of $s.d$ never changes after initialization.
- For the inductive step, we assume that $v_{i-1}.d = \delta(s, v_{i-1})$, and we examine what happens when we relax edge (v_{i-1}, v_i).
By the Convergence Property, after relaxing this edge, we have $v_i.d = \delta(s, v_i)$, and this equality is maintained at all times thereafter.

Relaxation and Shortest-Paths Trees I

Lemma

Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbb{R}$, let $s \in V$ be a source vertex, and assume that G contains no negative-weight cycles that are reachable from s. Then, after the graph is initialized by InitializeSingleSource(G, s), the predecessor subgraph G_π forms a rooted tree with root s, and any sequence of relaxation steps on edges of G maintains this property as an invariant.

- Initially, the only vertex in G_π is s, and the lemma is trivially true.
- Consider a predecessor subgraph G_π that arises after a sequence of relaxation steps. We shall first prove that G_π is acyclic. Suppose that some relaxation step creates a cycle $c = \langle v_0, v_1, \ldots, v_k \rangle$ in the graph G_π, where $v_k = v_0$. Then, $v_i.\pi = v_{i-1}$, for $i = 1, 2, \ldots, k$. Without loss of generality, assume that relaxing (v_{k-1}, v_k) created the cycle in G_π. We claim that all vertices on c are reachable from s.

Relaxation and Shortest-Paths Trees II

Claim: All vertices on cycle c are reachable from the source s.

Each vertex on c has a non-NIL predecessor. So each vertex on c was assigned a finite shortest path estimate when it was assigned its non-NIL π value. By the Upper-Bound Property, each vertex on cycle c has a finite shortest path weight. This implies that it is reachable from s.

- We examine the shortest path estimates on c just prior to the call $\text{RELAX}(v_{k-1}, v_k, w)$ and show that c is a negative weight cycle, thereby contradicting the assumption that G contains no negative weight cycles that are reachable from the source.

  Just before the call, we have $v_i.\pi = v_{i-1}$, for $i = 1, 2, \ldots, k-1$.

  Thus, for $i = 1, 2, \ldots, k-1$, the last update to $v_i.d$ was by the assignment $v_i.d = v_{i-1}.d + w(v_{i-1}, v_i)$. If $v_{i-1}.d$ changed since then, it decreased. Therefore, just before the call $\text{RELAX}(v_{k-1}, v_k, w)$, we have $v_i.d \geq v_{i-1}.d + w(v_{i-1}, v_i)$, for all $i = 1, 2, \ldots, k-1$.

Relaxation and Shortest-Paths Trees III

- Because $v_k.\pi$ is changed by the call, immediately beforehand we also have the strict inequality $v_k.d > v_{k-1}.d + w(v_{k-1}, v_k)$.

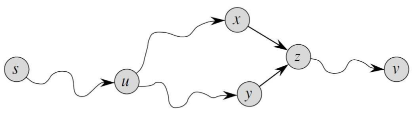
  Summing this strict inequality with the preceding $k-1$ inequalities, we obtain the sum of the shortest path estimates around cycle c:

$$\begin{array}{rcl} \sum_{i=1}^k v_i.d & > & \sum_{i=1}^k (v_{i-1}.d + w(v_{i-1}, v_i)) \\ & = & \sum_{i=1}^k v_{i-1}.d + \sum_{i=1}^k w(v_{i-1}, v_i). \end{array}$$

  But $\sum_{i=1}^k v_i.d = \sum_{i=1}^k v_{i-1}.d$, since each vertex in the cycle c appears exactly once in each summation. This equality implies $0 > \sum_{i=1}^k w(v_{i-1}, v_i)$. Thus, the sum of weights around the cycle c is negative, a contradiction.
- We have now proven that G_π is a directed, acyclic graph.

Relaxation and Shortest-Paths Trees IV

- To show that V_π forms a rooted tree with root s, it suffices to prove that for each vertex $v \in V_\pi$, there is a unique path from s to v in G_π.
  - We first must show that a path from s exists for each vertex in V_π. The vertices in V_π are those with non-NIL π values, plus s. The idea here is to prove by induction that a path exists from s to all vertices in V_π.
  - To complete the proof of the lemma, we must now show that for any vertex $v \in V_\pi$, there is at most one path from s to v in the graph G_π. Suppose there are two simple paths from s to some vertex v:



    - p_1, which can be decomposed into $s \rightsquigarrow u \rightsquigarrow x \to z \rightsquigarrow v$;
    - p_2, which can be decomposed into $s \rightsquigarrow u \rightsquigarrow y \to z \rightsquigarrow v$, where $x \neq y$.

    Then, $z.\pi = x$ and $z.\pi = y$, which implies the contradiction that $x = y$. Hence, there exists a unique simple path in G_π from s to v.

Thus, G_π forms a rooted tree with root s.

The Predecessor-Subgraph Property

Lemma (Predecessor-Subgraph Property)

Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbb{R}$, let $s \in V$ be a source vertex, and assume that G contains no negative-weight cycles that are reachable from s. Let us call INITIALIZE SINGLESOURCE(G, s) and then execute any sequence of relaxation steps on edges of G that produces $v.d = \delta(s, v)$, for all $v \in V$. Then, the predecessor subgraph G_π is a shortest-paths tree rooted at s.

- The three properties of shortest-paths trees hold for G_π.
    - To show the first property, we must show that V_π is the set of vertices reachable from s. By definition, a shortest-path weight $\delta(s, v)$ is finite if and only if v is reachable from s. Thus, the vertices that are reachable from s are exactly those with finite d values. But a vertex $v \in V - \{s\}$ has been assigned a finite value for $v.d$ if and only if $v.\pi \neq \text{NIL}$. Thus, the vertices in V_π are exactly those reachable from s.

The Predecessor-Subgraph Property (Cont'd)

- The second property follows directly from the lemma.
- It remains to prove the last property of shortest-paths trees, i.e., that for each vertex $v \in V_\pi$, the unique simple path $s \overset{p}{\rightsquigarrow} v$ in G_π is a shortest path from s to v in G. Let $p = \langle v_0, v_1, \ldots, v_k \rangle$, where $v_0 = s$ and $v_k = v$. For $i = 1, 2, \ldots, k$, we have both $v_i.d = \delta(s, v_i)$ and $v_i.d \geq v_{i-1}.d + w(v_{i-1}, v_i)$. So $w(v_{i-1}, v_i) \leq \delta(s, v_i) - \delta(s, v_{i-1})$. Summing the weights along path p yields

$$\begin{array}{rcl} w(p) & = & \sum_{i=1}^{k} w(v_{i-1}, v_i) \\ & \leq & \sum_{i=1}^{k} (\delta(s, v_i) - \delta(s, v_{i-1})) \\ & = & \delta(s, v_k) - \delta(s, v_0) \quad \text{(because the sum telescopes)} \\ & = & \delta(s, v_k). \quad \text{(because } \delta(s, v_0) = \delta(s, s) = 0\text{)} \end{array}$$

  Thus, $w(p) \leq \delta(s, v_k)$. Since $\delta(s, v_k)$ is a lower bound on the weight of any path from s to v_k, we conclude that $w(p) = \delta(s, v_k)$. Thus, p is a shortest path from s to $v = v_k$.