Algorithms II, CS 502

Single-Source Shortest Paths

Ugur Dogrusoz

Computer Eng Dept, Bilkent Univ

Single-source shortest paths

- Given a road map of some locations on which the distance between each pair of adjacent intersections is marked, how can one determine the shortest route from one location to (an)other(s)?
- Enumerating all the routes from a source to a destination results in examination of an enormous number of possibilities.
  - Besides, when going from Ankara to Istanbul, passing through Izmir is an obviously bad choice.

Single-source shortest paths

- In a shortest-path problem, we are given a weighted directed graph G=(V,E) with a weight function *w*, mapping edges to real valued weights:

$$w : E \to \Re$$

Single-source shortest paths (SP)

- The weight *w(p)* of a path $p=<v_1,v_2,\ldots,v_k>$ is the sum of its constituent edges:

$$w(p)=\sum_{i=0}^{k} w(v_{i-1},v_i)$$

- The shortest path weight from *u* to *v* is defined as:

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

- A shortest path from *u* to *v* is defined as any path *p* with weight:

$$w(p)=\delta(u,v)$$

Shortest paths, variants

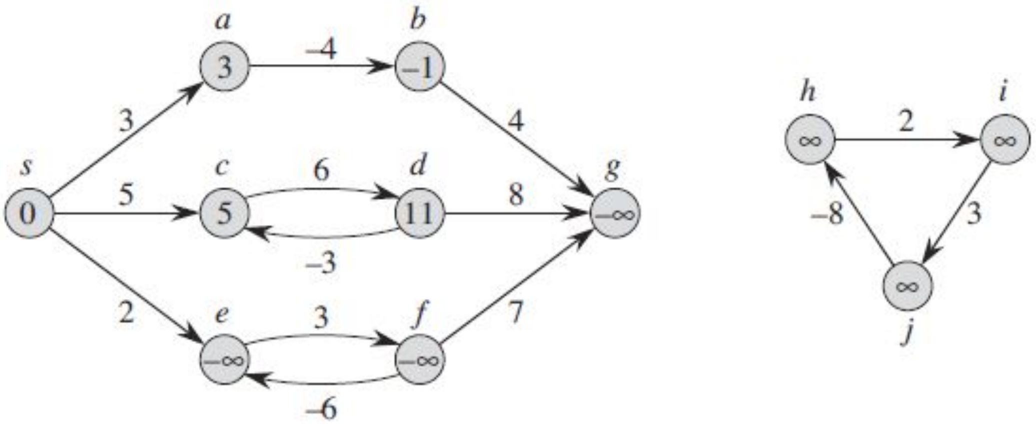
- Single-destination shortest-paths problem
  - Find a shortest path to a given destination vertex *t* from each vertex *v* (same as single-source)
- Single-pair shortest-path problem
  - Find a shortest path from *u* to *v* for given vertices *u* and *v* (same as single-source *asymptotically*)
- All-pairs shortest-paths problem
  - Find a shortest path from u to v for every pair of vertices u and v (running single-source algorithm repeatedly is slower)

Shortest paths, optimal substructure

- **Lemma 24.1 *(Subpaths of shortest paths are shortest paths)*** Given a weighted, directed graph $G=(V,E)$ with weight function $w:E\rightarrow R$, let $p=<v_0,v_1,\ldots,v_k>$ be a shortest path from vertex v_0 to vertex v_k and, for any i and j such that $0 \leq i \leq j \leq k$, let $p_{ij}=<v_i,v_{i+1},\ldots,v_j>$ be the sub-path of p from vertex v_i to vertex v_j. Then, p_{ij} is a shortest path from v_i to v_j.

Shortest paths, negative-weight edges

- Negative weights OK but not negative cycles
  - Some algorithms handle negative weights
  - Others don't



Shortest paths, cycles, length

- A shortest path never has to include a cycle
  - Negative cycle makes shortest path undefined
  - Positive cycles are never on shortest path
  - Zero-weight cycles can be removed
- Thus shortest paths are simple paths
  - Length of a shortest path is at most |V|-1

Shortest paths, weight vs actual path

- Maintain a predecessor v.π for each vertex v to record shortest path
- Predecessor subgraph $G_\pi=(V_\pi,E_\pi)$ will have correct values at the end of calculation
  - In fact, a shortest path tree
  - Shortest paths and shortest path trees are not necessarily unique

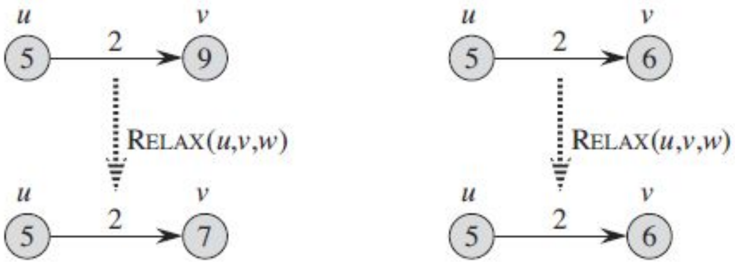
Shortest paths, relaxation

- Use the technique of relaxation
- For each vertex v, maintain an attribute v.*d*
  - an upper bound on the weight of a shortest path from source s to v; a shortest path estimate
- 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*.

Shortest paths, relaxation

```
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex v ∈ G.V
    v.d = ∞
    v.π = NIL
s.d = 0
```

```
RELAX(u, v, w)
if v.d > u.d + w(u, v)
    v.d = u.d + w(u, v)
    v.π = u
```



Shortest paths properties

Triangle inequality (Lemma 24.10)
For any edge $(u, v) \in E$, we have $\delta(s, v) \le \delta(s, u) + w(u, v)$.

Upper-bound property (Lemma 24.11)
We always have $v.d \ge \delta(s, v)$ for all vertices $v \in V$, and once $v.d$ achieves the value $\delta(s, v)$, it never changes.

No-path property (Corollary 24.12)
If there is no path from s to v, then we always have $v.d = \delta(s, v) = \infty$.

Convergence property (Lemma 24.14)
If $s \leadsto u \to v$ 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 (Lemma 24.15)
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, even if they are intermixed with relaxations of the edges of p.

Predecessor-subgraph property (Lemma 24.17)
Once $v.d = \delta(s, v)$ for all $v \in V$, the predecessor subgraph is a shortest-paths tree rooted at s.

Shortest paths, algorithms

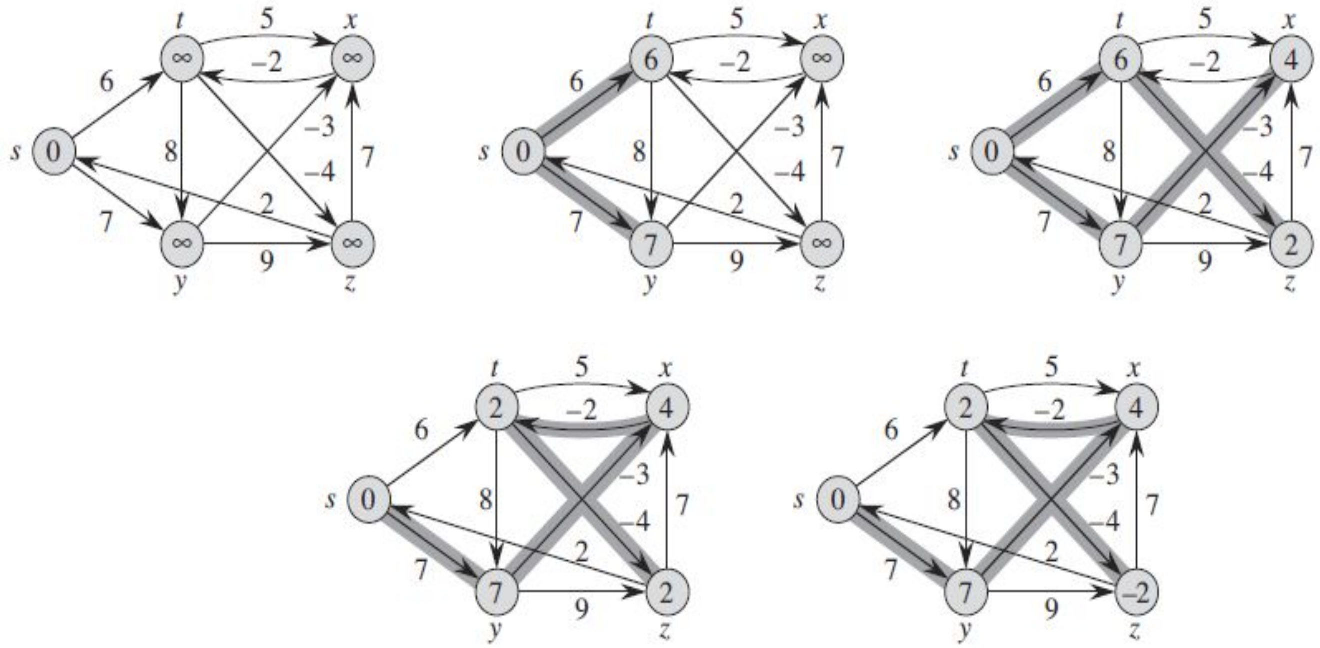
- The algorithms described from here on differ in
  - how many times they relax each edge and
  - the order in which they relax edges.
- Dijkstra’s algorithm and the shortest-paths algorithm for directed acyclic graphs relax each edge exactly once.
- The Bellman-Ford algorithm relaxes each edge |V|-1 times.

Bellman-Ford algorithm

- Solves general case (negative weights are ok)
- Returns FALSE if no solution exists (i.e. negative cycle)
- Runs in O(V E) time

```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(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
```

Bellman-Ford algorithm



Bellman-Ford algorithm

- **Lemma 24.2** Let $G=(V,E)$ be a weighted, directed graph with source s and weight function $w:E\rightarrow R$, 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 Bellman-Ford, we have $v.d=\delta(s,v)$ for all vertices v that are reachable from s.

Bellman-Ford algorithm

- **Proof** Consider any vertex v reachable from s, and let $p=\langle v_0,v_1,\ldots,v_k\rangle$, where $v_0=s$ and $v_k=v$, be any shortest path from s to v. Since $|p|=k\leq|V|-1$, each of the $|V|-1$ iterations of the for loop of lines 2–4 relaxes all $|E|$ edges. Among the edges relaxed in the i^{th} iteration, for $i=1,2,\ldots,k$, is (v_{i-1},v_i). By the path-relaxation property, therefore, $v.d=v_k.d=\delta(s,v_k)=\delta(s,v)$.

Bellman-Ford algorithm

- **Corollary 24.3** Let $G=(V,E)$ be a weighted, directed graph with source vertex s and weight function $w:E\rightarrow R$, 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 Bellman-Ford terminates with $v.d < \infty$ when it is run on G.

Bellman-Ford algorithm

- **Theorem 24.4 (Correctness)** Let Bellman-Ford be run on a weighted, directed graph $G=(V,E)$ with source s and weight function $w:E\rightarrow R$. 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.

Bellman-Ford algorithm

- **Proof**
  - **v.*d*=δ(s,v) for all *v*:**
    - if *v* is reachable from *s*, true by Lemma 24.2
    - if *v* is not reachable from *s*, true by no-path property

Bellman-Ford algorithm

- **Proof cntd**
  - G_π **is a shortest-paths tree rooted at** s**:** true by the predecessor-subgraph property

Bellman-Ford algorithm

- **Proof cntd**
  - **G contains no negative cycles:** we know $v.d = \delta(s,v)$ whether v is reachable from s or not and the predecessor subgraph property implies G_π is a shortest paths tree. The algorithm returns true since at termination for each edge (u,v), we have $v.d = \delta(s,v) \le \delta(s,u) + w(u,v) = u.d + w(u,v)$, so none of the tests in line 6 causes the algorithm to return FALSE.

Bellman-Ford algorithm

- **Proof cntd**
  - **G contains a negative cycle c=<v_0,v_1,…,v_k> with v_0=v_k**: Assume it returns TRUE. Thus, $v_i.d \le v_{i-1}.d + w(v_{i-1}, v_i)$ for i=1,2,…,k (line 6). Summing the inequalities around cycle c and the fact that $v_i.d$ is finite (Cor. 24.3) gives us the following contradiction to our assumption:

$$\sum_{i=1}^{k} v_i.d \le \sum_{i=1}^{k} (v_{i-1}.d + w(v_{i-1}, v_i))$$

$$= \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

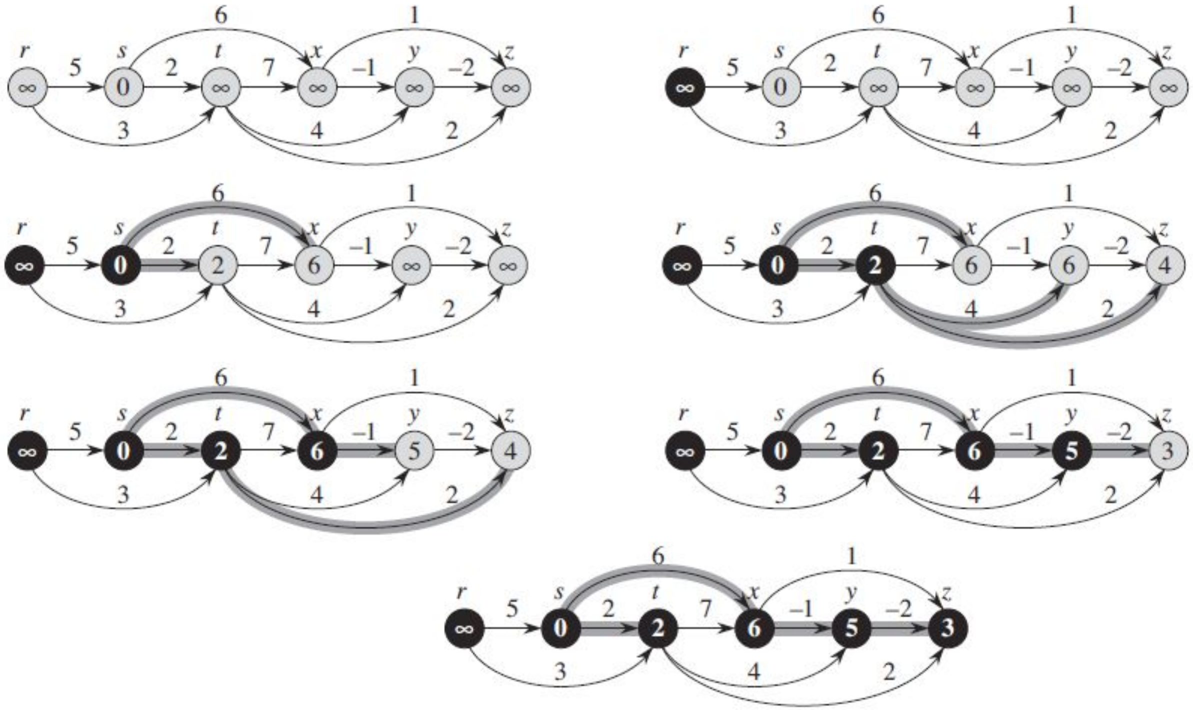
$$\Rightarrow 0 \le \sum_{i=1}^{k} w(v_{i-1}, v_i) \text{ since } v_0 = v_k$$

Single-source SP in dags

- Relax edges of a weighted dag G=(V,E) according to a topological sort of its vertices in O(V+E) time.

```
DAG-SHORTEST-PATHS(G, w, s)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, taken in topologically sorted order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

Single-source SP in dags



Single-source SP in dags

- **Theorem 24.5** If a weighted, directed graph $G=(V,E)$ has source vertex s and no cycles, then at the termination of the Dag-Shortest-Paths procedure, $v.d=\delta(s,v)$ for all vertices $v \in V$, and the predecessor subgraph G is a shortest-paths tree.

Single-source SP in dags

- **Proof**
  - **If v is not reachable from s:** $v.d=\delta(s,v)=\infty$ by the no-path property
  - **If v is reachable from s:** let $p=\langle v_0,v_1,\ldots,v_k\rangle$ be a shortest path from s to v, where $v_0=s$ and $v_k=v$. Since we process edges in topologically sorted order, we relax edges in the order: (v_0,v_1), (v_1,v_2), … (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 path tree.

Single-source SP in dags, critical paths

- Critical (longest) paths in PERT charts is an application
  - negate edge weights and run Dag-Shortest-Paths, or
  - modify Dag-Shortest-Paths:
    - replace ∞ with -∞,
    - replace > with < in line 2 of Relax

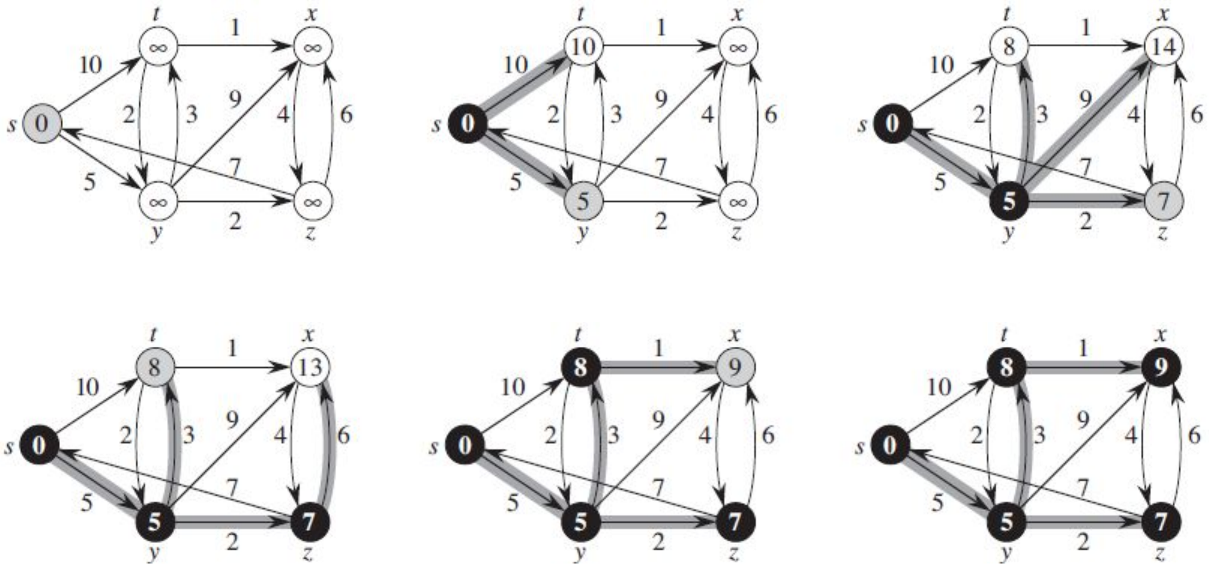
Dijkstra's algorithm

- Solves single-source shortest-paths problem on a weighted, directed graph with nonnegative edge weights
- A greedy algorithm that maintains a set of vertices whose final shortest-path weights from source s have already been determined
  - Selects vertex u ϵ V-S with minimum shortest-path estimate, adds u to S, and relaxes all edges leaving u

Dijkstra's algorithm

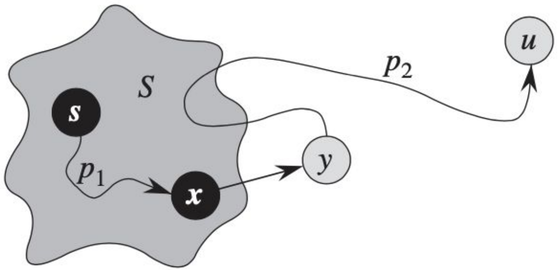
```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

Dijkstra's algorithm



Dijkstra's algorithm

- **Theorem 24.6 *(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 $u \in V$.
- **Proof** Use the following loop invariant:
  - At the start of each iteration of the while loop of $v.d = \delta(s,v)$ for each vertex $v \in S$
  - Show $u.d = \delta(s,u)$ at the time when u is added to set S



Dijkstra’s algorithm

- **Corollary 24.7** 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.
- **Proof** Immediate from Theorem 24.6 and the predecessor-subgraph property.

Dijkstra's algorithm, analysis

- Store v.*d* in vth entry of an array
  - Insert and Decrease-Key in $O(1)$, Extract-Min in $O(V)$ time, results in $O(V^2 + E) = O(V^2)$ time
- Use a binary min-heap
  - $O((V + E) \lg V) = O(E \lg V)$ [assuming all vertices reachable]
  - Better than array implementation if $E = o(V^2 / \lg V)$.
- Use a Fibonacci heap
  - $O(V \lg V + E)$

Dijkstra's algorithm

- Resembles
  - BFS in that set S corresponds to set of black vertices in a BFS; just as vertices in S have their final shortest-path weights, so do black vertices in a breadth-first search have their correct breadth-first distances,
  - Prim's algorithm in that both algorithms use a min-priority queue to find the "lightest" vertex outside a given set, add this vertex into the set, and adjust weights of remaining vertices outside the set accordingly.

Basics, the triangle inequality

- **Lemma 24.10 *(Triangle inequality)*** Let G=(V,E) be a weighted, directed graph with weight function $w: E \rightarrow R$, and source vertex s. Then, for all edges $(u,v) \in E$, we have $\delta(s,v) \leq \delta(s,u) + w(u,v)$.

Basics, upper-bound property

- **Lemma 24.11 *(Upper-bound property)*** Let G=(V,E) be a weighted, directed graph with weight function w:E→R. Let s ϵ V be the source vertex, and let the graph be initialized by Initialize-Single-Source(G,s). Then, v.d ≥ *δ(*s,v*)* for all v ϵ 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 δ(s,v), it never changes.
- **Proof** Prove the invariant v.d ≥ *δ(*s,v*) for all vertices* v ϵ V by induction on the number of relaxation steps

Basics, no-path property

- **Corollary 24.12 *(No-path property)*** Suppose that in a weighted, directed graph G=(V,E) with weight function w $w: E \rightarrow R$, no path connects a source vertex s ϵ V to a given vertex v ϵ V . Then, after the graph is initialized by Initialize-Single-Source(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.
- **Proof** By the upper-bound property, we always have $\infty = \delta(s,v) \leq v.d$, and thus $v.d = \infty = \delta(s,v)$.

Basics

- **Lemma 24.13** Let $G=(V,E)$ be a weighted, directed graph with weight function $w:E\rightarrow R$, and let $(u,v) \in E$. Then, immediately after relaxing edge (u,v) by executing Relax(u,v,w), we have $v.d \leq u.d+w(u,v)$.
- **Proof** 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. If, instead, $v.d\leq u.d+w(u,v)$ just before the relaxation, then neither $u.d$ nor $v.d$ changes, and so $v.d\leq u.d+w(u,v)$ afterward.

Basics, convergence property

- **Lemma 24.14 *(Convergence property)*** Let G=(V,E) be a weighted, directed graph with weight function w:E→R, let s ϵ V be a source vertex, and let s→u→v be a shortest path in G for some vertices u,v ϵ V . Suppose that G is initialized by Initialize-Single-Source(G,s), and then a sequence of relaxation steps that includes the call RELAX(u,v,w) is executed on the edges of G. If u.d=δ(s,u) at any time prior to the call, then v.d=δ(s,v) at all times after the call.

Basics, convergence property

- **Proof** By the upper-bound property, if u.d=δ(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 ≤ u.d + w(u,v) (by Lemma 24.13)
  - = δ(s,u) + w(u,v)
  - = δ(s,v) (by Lemma 24.1) .

  By the upper-bound property, v.d ≥ δ(s,v), from which we conclude that v.d = δ(s,v), and this equality is maintained thereafter.

Basics, path relaxation property

- **Lemma *24.15 (Path-relaxation property)*** Let G=(V,E) be a weighted, directed graph with weight function w:E→R, and let s ϵ V be a source vertex. Consider any shortest path p=<$v_0,v_1,\ldots,v_k$> from s=v_0 to v_k. If G is initialized by Initialize-Single-Source(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.

Basics, path relaxation property

- **Proof** by induction that after the i^{th} edge of path p is relaxed, we have $v_i.d=\delta(s,v_i)$. For the basis, i=0, and 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.

Basics, pred-subgraph property

- **Lemma 24.17 *(Predecessor-subgraph property)*** Let G=(V,E) be a weighted, directed graph with weight function w:E→R, let s ϵ V be a source vertex, and assume that G contains no negative-weight cycles that are reachable from s. Let us call INITIALIZE-SINGLE-SOURCE(G,s) and then execute any sequence of relaxation steps on edges of G that produces v.*d*=δ(v,s) for all v ϵ V. Then, the predecessor subgraph G_π is a shortest-paths tree rooted at s.