Single-source shortest paths

October 14, 2014

Content

- Weighted graphs and related notions
  - The relaxation method
- Single-source shortest path algorithms:
  - Dijkstra algorithm
  - Bellman-Ford algorithm

Weighted graphs

A weighted graph is a directed or undirected graph $G = (V, E)$ together with a map $w : E \to \mathbb{R}_+$, where $\mathbb{R}_+$ is the set of positive real numbers.

- If $e \in E$ then $w(e)$ is the weight of edge e.

Weighted graphs

A weighted graph is a directed or undirected graph $G = (V, E)$ together with a map $w : E \to \mathbb{R}_+$, where $\mathbb{R}_+$ is the set of positive real numbers.

- If $e \in E$ then $w(e)$ is the weight of edge e.
- Weights can be used to represent:
  1. lengths of connections between nodes
  2. duration of a flight connection
  3. $\ldots$

Weighted graphs

A weighted graph is a directed or undirected graph $G = (V, E)$ together with a map $w : E \to \mathbb{R}_+$, where $\mathbb{R}_+$ is the set of positive real numbers.

- If $e \in E$ then $w(e)$ is the weight of edge e.
- Weights can be used to represent:
  1. lengths of connections between nodes
  2. duration of a flight connection
  3. $\ldots$
- The weight of a path $\pi = (v_0, v_1, \ldots, v_k)$ is

  $$w(\pi) = w(v_0, v_1) + w(v_1, v_2) + \ldots + w(v_{k-1}, v_k) = \sum_{i=1}^{k} w(v_{i-1}, v_i).$$

  We will write $u \overset{\pi}{\rightsquigarrow} v$ if π is a path from u to v.

Weighted graphs

A weighted graph is a directed or undirected graph $G = (V, E)$ together with a map $w : E \to \mathbb{R}_+$, where $\mathbb{R}_+$ is the set of positive real numbers.

- If $e \in E$ then $w(e)$ is the weight of edge e.
- Weights can be used to represent:
  1. lengths of connections between nodes
  2. duration of a flight connection
  3. ...
- The weight of a path $\pi = (v_0, v_1, \ldots, v_k)$ is

  $$w(\pi) = w(v_0, v_1) + w(v_1, v_2) + \ldots + w(v_{k-1}, v_k) = \sum_{i=1}^{k} w(v_{i-1}, v_i).$$

  We will write $u \overset{\pi}{\rightsquigarrow} v$ if π is a path from u to v.
- The shortest path weight from u to v is

  $$\delta(u, v) = \begin{cases} \min\{w(\pi) \mid u \overset{\pi}{\rightsquigarrow} v\} & \text{if } u \text{ and } v \text{ are connected,} \\ \infty & \text{if } u \text{ and } v \text{ are not connected.} \end{cases}$$

The single-source shortest-path problem

Given a weighted graph $G = (V, E)$ and a source node $s \in V$

Find a shortest path from s to every node $v \in V$.

Theorem (Subpaths of shortest paths are shortest paths)

Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}_+$, let $\pi = (v_1, v_2, \ldots, v_k)$ be a shortest path from v_1 to v_k, and for any i and j such that $1 \leq i \leq j \leq k$, let $\pi_{ij} = (v_i, v_{i+1}, \ldots, v_j)$ be the subpath of π from v_i to v_j. Then π_{ij} is a shortest path from v_i to v_j.

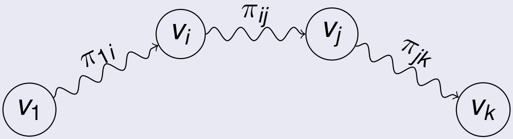
The single-source shortest-path problem

Given a weighted graph $G = (V, E)$ and a source node $s \in V$

Find a shortest path from s to every node $v \in V$.

Theorem (Subpaths of shortest paths are shortest paths)

Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}_+$, let $\pi = (v_1, v_2, \ldots, v_k)$ be a shortest path from v_1 to v_k, and for any i and j such that $1 \leq i \leq j \leq k$, let $\pi_{ij} = (v_i, v_{i+1}, \ldots, v_j)$ be the subpath of π from v_i to v_j. Then π_{ij} is a shortest path from v_i to v_j.



$$w(\pi) = w(\pi_{1i}) + w(\pi_{ij}) + w(\pi_{jk}).$$

Properties of shortest paths

1. If there exists a shortest path from s to v of the form $s \overset{\pi}{\leadsto} u \to v$ then $\delta(s,v) = \delta(s,u) + w(u,v)$.
2. For all edges (u,v) of a directed graph $G = (V,E)$ we have $\delta(s,v) \leq \delta(s,u) + w(u,v)$.

Properties of shortest paths

1. If there exists a shortest path from s to v of the form $s \overset{\pi}{\rightsquigarrow} u \rightarrow v$ then $\delta(s, v) = \delta(s, u) + w(u, v)$.
2. For all edges (u, v) of a directed graph $G = (V, E)$ we have $\delta(s, v) \leq \delta(s, u) + w(u, v)$.

We will present 2 algorithms to find the single-source shortest-path problem a weighted, directed graph $G = (V, E)$:

Properties of shortest paths

1. If there exists a shortest path from s to v of the form $s \overset{\pi}{\rightsquigarrow} u \to v$ then $\delta(s, v) = \delta(s, u) + w(u, v)$.
2. For all edges (u, v) of a directed graph $G = (V, E)$ we have $\delta(s, v) \leq \delta(s, u) + w(u, v)$.

We will present 2 algorithms to find the single-source shortest-path problem a weighted, directed graph $G = (V, E)$:

1. Dijkstra algorithm

Properties of shortest paths

1. If there exists a shortest path from s to v of the form $s \overset{\pi}{\rightsquigarrow} u \to v$ then $\delta(s,v) = \delta(s,u) + w(u,v)$.
2. For all edges (u,v) of a directed graph $G = (V,E)$ we have $\delta(s,v) \leq \delta(s,u) + w(u,v)$.

We will present 2 algorithms to find the single-source shortest-path problem a weighted, directed graph $G = (V,E)$:

1. Dijkstra algorithm
2. Bellman-Ford algorithm

Properties of shortest paths

1. If there exists a shortest path from s to v of the form $s \overset{\pi}{\rightsquigarrow} u \to v$ then $\delta(s, v) = \delta(s, u) + w(u, v)$.
2. For all edges (u, v) of a directed graph $G = (V, E)$ we have $\delta(s, v) \leq \delta(s, u) + w(u, v)$.

We will present 2 algorithms to find the single-source shortest-path problem a weighted, directed graph $G = (V, E)$:

1. Dijkstra algorithm
2. Bellman-Ford algorithm

Both algorithms rely on the relaxation method.

Auxiliary technique for the computation of all shortest paths from a source node s

Auxiliary technique for the computation of all shortest paths from a source node s

- Related data structures:

Auxiliary technique for the computation of all shortest paths from a source node s

- Related data structures:
  1. $d : V \to \mathbb{R}^+$ (*shortest path estimate*)
     $d[v] :=$ upper bound on the weight of a shortest path from s to v.

Auxiliary technique for the computation of all shortest paths from a source node s

- Related data structures:
  1. $d : V \to \mathbb{R}^+$ (*shortest path estimate*)
     $d[v]$:= upper bound on the weight of a shortest path from s to v.
  2. $\pi : V \to C \cup \{NIL\}$
     $\pi[v]$:= predecessor of v on the path from s to v

Auxiliary technique for the computation of all shortest paths from a source node s

- Related data structures:
  1. $d : V \to \mathbb{R}^+$ (*shortest path estimate*)
     $d[v]$:= upper bound on the weight of a shortest path from s to v.
  2. $\pi : V \to C \cup \{NIL\}$
     $\pi[v]$:= predecessor of v on the path from s to v

Initialization step (pseudocode)

INITIALIZESINGLESOURCE(G, s)
  for each $v \in V[G]$
    $d[v] \leftarrow \infty$
    $\pi[v] \leftarrow s$
  $d[s] \leftarrow 0$
  $\pi[s] = NIL$

The relaxation method performs edge relaxation steps.

- Relaxing an edge (u, v) means:
  1. Testing if we can improve the length of $\pi[v]$ by going through u
  2. If so, update $d[v]$ and $\pi[v]$

The relaxation method performs edge relaxation steps.

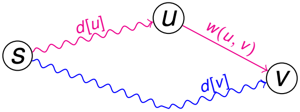
- Relaxing an edge (u, v) means:
  1. Testing if we can improve the length of $\pi[v]$ by going through u
  2. If so, update $d[v]$ and $\pi[v]$
- Pseudocode for relaxation step

RELAX(u, v)
  if $d[v] > d[u] + w(u, v)$
    $d[v] \leftarrow d[u] + w(u, v)$
    $\pi[v] \leftarrow u$

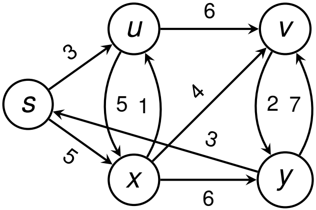
The relaxation method performs edge relaxation steps.

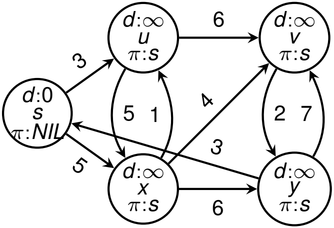
- Relaxing an edge (u, v) means:
  1. Testing if we can improve the length of $\pi[v]$ by going through u
  2. If so, update $d[v]$ and $\pi[v]$
- Pseudocode for relaxation step

RELAX(u, v)
    if $d[v] > d[u] + w(u, v)$
        $d[v] \leftarrow d[u] + w(u, v)$
        $\pi[v] \leftarrow u$



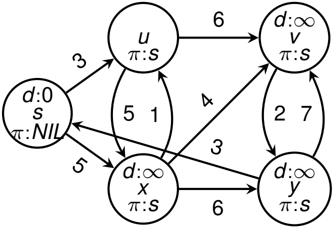
Relaxation method
Example





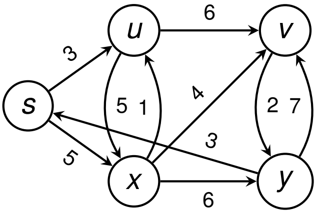
INITIALIZESINGLESOURCE(G, s)

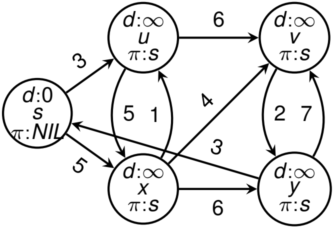
$d[u] = \infty > d[s] + w(s, u) = 0 + 3$



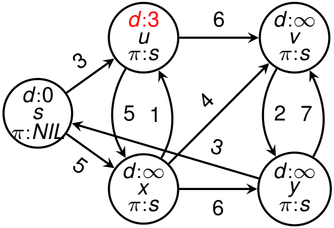
RELAX(s, u)

Relaxation method
Example





INITIALIZESINGLESOURCE(G, s)



RELAX(s, u)

Properties shared by the single-source shortest-path algorithms

- Both algorithms Dijkstra and Bellman-Ford call INITIALIZESINGLESOURCE(G, s) and then repeatedly perform relaxation steps.
- Relaxation is the only means by which the shortest-path estimates $d[v]$ and predecessors $\pi[v]$ can change.
- Dijkstra and Bellman-Ford algorithms differ in
  1. how many times they relax each edge
     - Dijkstra's algorithm relaxes each edge exactly once
     - Bellman-Ford algorithm relaxes edges several times
  2. the order in which they relax edges

Properties of relaxation

ASSUMPTION. $G = (V, E)$ is a weighted digraph.

(P_1) Immediately after the execution of RELAX(u, v), we have $d[v] \leq d[u] + w(u, v)$.

(P_2) $d[v] \geq \delta(s, v)$ for all $v \in V$ holds after INITIALIZESINGLESOURCE(G, s), and is maintained by all relaxation steps produced by executing RELAX(u, v).

(P_3) If v is not connected to s then $d[v] = \delta(s, v) = \infty$ after initialization. Also, this property is maintained by all relaxation steps produced afterwards.

(P_4) Suppose $s \overset{\pi}{\leadsto} u \to v$ is a shortest path from s to v in G, and

- G was initialized with INITIALIZESINGLESOURCE(G, s).
- A sequence of relaxation steps, including RELAX(u, v), was performed afterwards.

If $d[u] = \delta(s, u)$ was true before the relaxation step RELAX(u, v), then $d[v] = \delta(s, v)$ is true at all times afterwards.

Shortest-path trees

- Algorithms based on the relaxation method compute a parent $\pi(v)$ for every node $v \in V$.
  - G_π: the tree with nodes V and edges $E_\pi = \{(\pi(v), v) \mid v \in V \text{ and } \pi(v) \neq NIL\}$.

PROPERTIES OF RELAXATION

Shortest-path trees

- Algorithms based on the relaxation method compute a parent $\pi(v)$ for every node $v \in V$.
  - G_π: the tree with nodes V and edges $E_\pi = \{(\pi(v), v) \mid v \in V \text{ and } \pi(v) \neq NIL\}$.

PROPERTIES OF RELAXATION

- Relaxation steps cause the shortest path estimates to descend monotonically towards the actual shortest-path weight.
- After a sequence of relaxation steps has computed the actual shortest path weights, the graph G_π is a shortest-path tree for G, which means that:
  - The branches from s to any v in the tree G_π are shortest paths from s to v in G.

Dijkstra's algorithm

- Maintains a set S of vertices whose final shortest path weights from source s have already been found. This means that

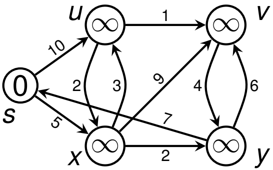
  $$d[v] = \delta(s, v) \text{ for all } v \in S.$$

- The algorithm repeatedly selects $u \in V - S$ with $d[v]$ minimum, inserts u into S, and relaxes all edges leaving u.
- This implementation of the Dijkstra's algorithm assumes that G is represented by adjacency lists:

```
DIJKSTRA(G, w, s)
INITIALIZESINGLESOURCE(G, s)
S ← ∅
Q ← V[G]
while Q − S ≠ ∅
    u ← EXTRACTMIN(Q − S)
    S ← S ∪ {u}
    for each node v ∈ Adj[u]
        RELAX(u, v)
```

Dijsktra's algorithm

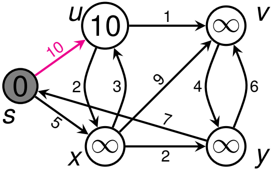
Illustrative example: **while** loop 1



Configuration produced by INITIALIZESINGLESOURCE(G, s)

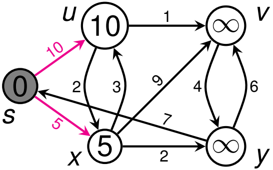
$S = \emptyset$
$Q = \{s, u, v, x, y\}$
select node s

S=$\{s\}$, $Adj[s] = \{u, x\}$
$Q = \{u, v, x, y\}$



RELAX(s, u)
changes: $d[u] := 10$

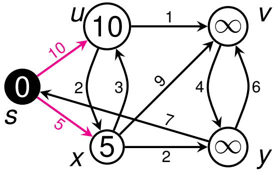
S=$\{s\}$, $Adj[s] = \{u, x\}$
$Q = \{u, v, x, y\}$



RELAX(s, x)
changes $d[x] := 5$

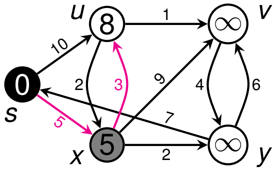
Dijsktra's algorithm

Illustrative example: **while** loop 2



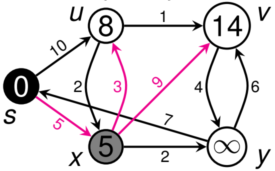
$S = \{s\}$
$Q = \{u, v, x, y\}$
select node x

S=$\{s, x\}$, $Adj[x] = \{u, v, y\}$
$Q = \{u, v, y\}$



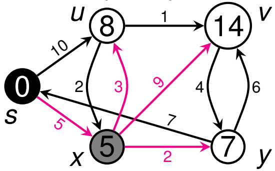
Relax(x, u)
changes: $d[u] := 8, \pi[u] := x$

S=$\{s, x\}$, $Adj[x] = \{u, v, y\}$
$Q = \{u, v, y\}$



Relax(x, v)
changes: $d[v] := 14, \pi[v] := x$

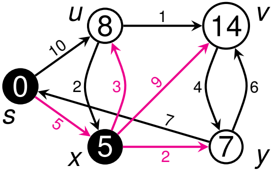
S=$\{s, x\}$, $Adj[x] = \{u, v, y\}$
$Q = \{u, v, y\}$



Relax(x, y)
changes: $d[y] := 7, \pi[y] := x$

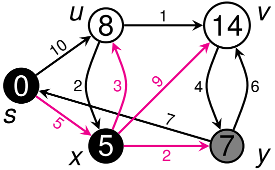
Dijsktra's algorithm

Illustrative example: **while** loop 3



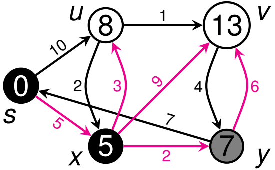
$S = \{s, x\}$
$Q = \{u, v, y\}$
select node y

S=$\{s, x, y\}$, $Adj[y] = \{s, v\}$
$Q = \{u, v\}$

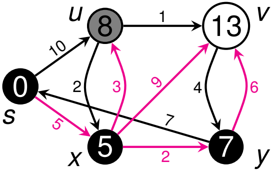


RELAX(y, s)
changes: *none*

S=$\{s, x, y\}$, $Adj[y] = \{s, v\}$
$Q = \{u, v\}$

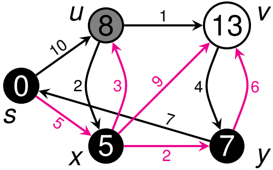


RELAX(y, v)
changes: $d[v] := 13, \pi[v] := y$



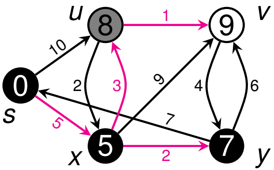
$S = \{s, x, y\}$
$Q = \{u, v\}$
select node u

S=$\{s, x, y, u\}$, $Adj[u] = \{x, v\}$
$Q = \{v\}$

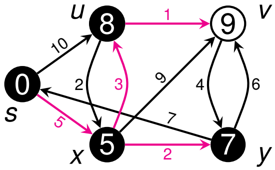


RELAX(u, x)
changes: *none*

S=$\{s, x, y, u\}$, $Adj[u] = \{x, v\}$
$Q = \{v\}$

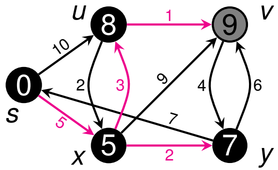


RELAX(u, v)
changes: $d[v] := 9, \pi[v] := u$



$S = \{s, x, y, u\}$
$Q = \{v\}$
select node v

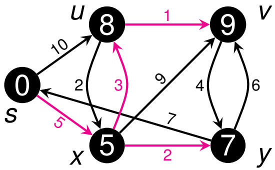
S=$\{s, x, y, u, v\}$, $Adj[v] = \{y\}$
$Q = \emptyset$



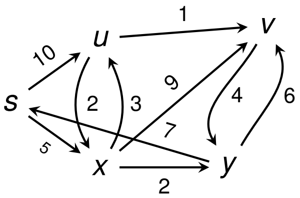
RELAX(v, y)
changes: *none*

$\Rightarrow$

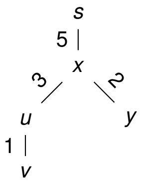
Final configuration:



Dijkstra's algorithm
The shortest-path tree



- $\pi[s] = NIL, \pi[x] = s, \pi[u] = \pi[y] = x, \pi[v] = u$
  $d[s] = 0, d[x] = 5, d[u] = 8, d[y] = 7, d[v] = 9$
- G_π is



NOTE. G_π is shortest-path tree for G.

- Line 1 performs the usual initialization of the values of d and π.
- Line 2 initializes S to the empty set.
- Line 3 initializes the priority queue Q to contain all the vertices in $V - S = V - \emptyset = V$.
- Each time through the **while** loop of lines 4-8, a vertex u is extracted from $Q = V - S$ and inserted into set S. In the first loop, $u = s$.
- Lines 7-8 relax each edge (u, v) leaving u, thus updating $d[v]$ and the predecessor $\pi[v]$ if the shortest path to v can be improved by going through u.
- NOTE. Vertices are never inserted into Q after line 3, and each vertex is extracted from Q and inserted into S exactly once $\Rightarrow$ the while loop of lines 4-8 iterates exactly $|V|$ times.

Dijkstra's algorithm: Properties

greedy: it always chooses the "lightest" node in $V - S$ to insert into S.

correct: Upon termination on a weighted directed graph with weight function $w : E \to \mathbb{R}_{+}$, $d[u] = \delta(s, u)$ for all vertices $u \in V$.

Dijkstra's algorithm: Properties

greedy: it always chooses the "lightest" node in $V - S$ to insert into S.

correct: Upon termination on a weighted directed graph with weight function $w : E \to \mathbb{R}_+$, $d[u] = \delta(s, u)$ for all vertices $u \in V$.

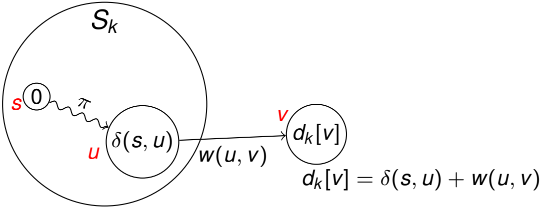
Dijkstra's algorithm: Properties

greedy: it always chooses the "lightest" node in $V - S$ to insert into S.

correct: Upon termination on a weighted directed graph with weight function $w : E \to \mathbb{R}_+$, $d[u] = \delta(s, u)$ for all vertices $u \in V$.

CORRECTNESS PROOF: Let S_k and $d_k[v]$ be the values of S and $d[v]$ before **while** loop k. We prove by induction on k that

1. $d_k[v] = \delta(v)$ for all $v \in S_k$.
2. $d_k[v] = \min\{\delta(u) + w(u, v)| \mid u \in S_k\}$ for all $v \in V - S_k$.



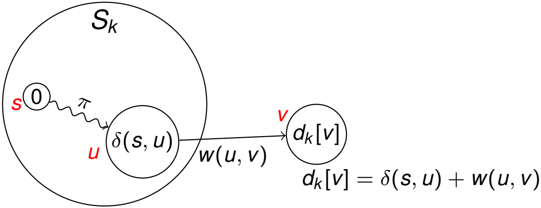
Dijkstra's algorithm: Properties

greedy: it always chooses the "lightest" node in $V - S$ to insert into S.

correct: Upon termination on a weighted directed graph with weight function $w : E \to \mathbb{R}_+$, $d[u] = \delta(s,u)$ for all vertices $u \in V$.

CORRECTNESS PROOF: Let S_k and $d_k[v]$ be the values of S and $d[v]$ before **while** loop k. We prove by induction on k that

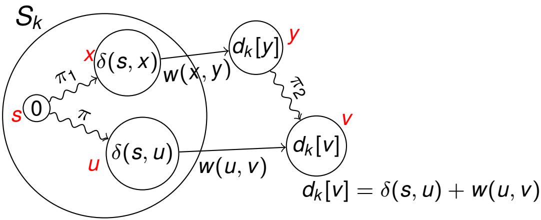
1. $d_k[v] = \delta(v)$ for all $v \in S_k$.
2. $d_k[v] = \min\{\delta(u) + w(u,v)| \mid u \in S_k\}$ for all $v \in V - S_k$.



Assume v is the node added to S_k, that is, $S_{k+1} = S_k \cup \{v\}$ and $d_k[v] = \min\{d_k[x] \mid x \in V - S_k\}$.

First, we show that $d_{k+1}[v] = \delta(s, v)$.
Proof by contradiction: If $d_{k+1}[v] \neq \delta(s, v)$ then there is a path π' shorter than $s \overset{\pi}{\rightsquigarrow} u \rightarrow v$.

- π' must have a node not in S_k before v.
- Let y be the first node of π' not in S_k. Path π' is depicted below:



Then $w(\pi') = w(\pi_1) + w(x, y) + w(\pi_2) < d_k[v]$.
But then $d_k[w] > w(\pi') > w(\pi_1) + w(x, y) = d_k[y]$, which contradicts the fact that $d_k[v] =| \{d_k[x] \mid x \in V - S_k\}$.
Next, we can show that $d_{k+1}[y] = \min\{\delta(u) + w(u, y) \mid u \in S_{k+1}\}$ for all $y \in V - S_{k+1}$

...Easy proof by contradiction.

Dijkstra's algorithm: Complexity analysis

ASSUMPTION. $Q = V - S$ is implemented as a linear array.

- The extraction of a certain element from Q takes time proportional with the length of Q, which is $\leq |V|$. Thus, every operation $\text{EXTRACTMIN}(Q)$ takes time $O(|V|)$.
- There are $|V|$ such extractions $\Rightarrow$ total time for EXTRACTMIN is $O(|V|^2)$.
- Each node $v \in V$ is inserted into S exactly once, so each edge in $Adj[v]$ is examined in the **for** loop of lines 4-8 exactly once. Since the total number of edges in all the adjacency lists is $|E|$, there is a total of $|E|$ iterations of this **for** loop, with each iteration taking constant time, $O(1)$.

  The total running time of Dijkstra's algorithm is $O(|V|^2 + |E|) = O(|V|^2)$.

The Bellman-Ford algorithm

- Dijkstra's algorithm was designed to work for positive weight $w : E \to \mathbb{R}_+$
- Bellman-Ford algorithm is designed to work also with negative weights; In general, we assume $w : E \to \mathbb{R}$

Given a weighted, directed graph $G = (V, E)$, a source s, and a weight function $w : E \to \mathbb{R}$

Return a boolean value indicating whether or not there is a negative-weight cycle that is reachable from the source. If there is no such cycle, produce the shortest paths and their weights.

The algorithm returns `true` if and only if G has no negative weight cycles that are reachable from the source node s.

The Bellman-Ford algorithm

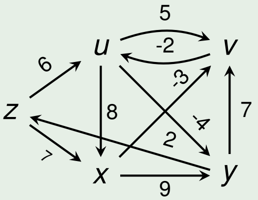
```
BellmanFord(G, w, s)
InitializeSingleSource(G, s)
for i ← 1 to |V[G]| − 1
    for each edge (u, v) ∈ E[G]
        Relax(u, v)
for each edge (u, v) ∈ E[G]
    if d[v] > d[u] + w(u, v)
        return false
return true
```

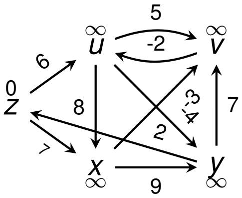
The Bellman-Ford algorithm

```
BELLMANFORD(G, w, s)
INITIALIZESINGLESOURCE(G, s)
for i ← 1 to |V[G]| − 1
    for each edge (u, v) ∈ E[G]
        RELAX(u, v)
for each edge (u, v) ∈ E[G]
    if d[v] > d[u] + w(u, v)
        return false
return true
```

Example

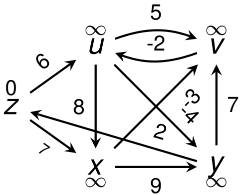
Suppose $s = z$ and G is the graph

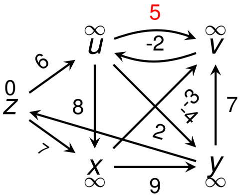




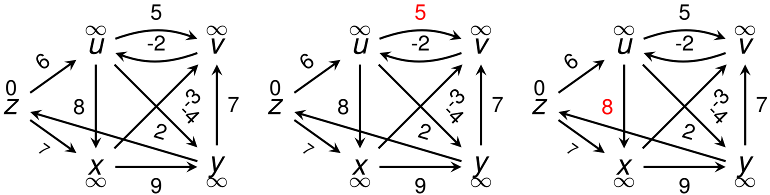
5
-2
6
8
7
2
-3
-4
7
9
0
z
∞
u
∞
v
∞
x
∞
y

Bellman-Ford algorithm
Execution of first `for` loop





RELAX(u, v)

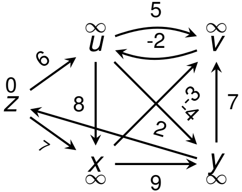


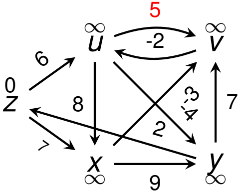
RELAX(u, v)

RELAX(u, x)

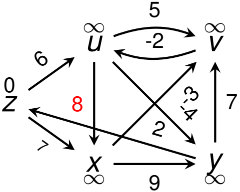
Bellman-Ford algorithm

Execution of first `for` loop

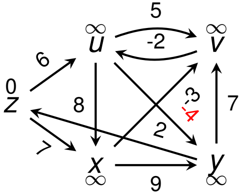




RELAX(u, v)



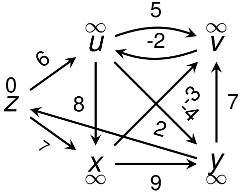
RELAX(u, x)

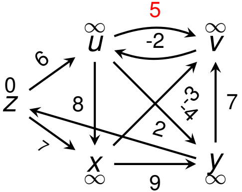


RELAX(u, y)

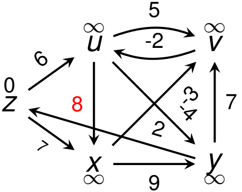
Bellman-Ford algorithm

Execution of first `for` loop

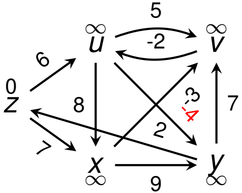




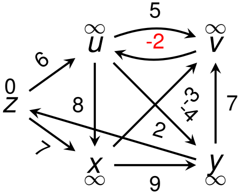
RELAX(u, v)



RELAX(u, x)



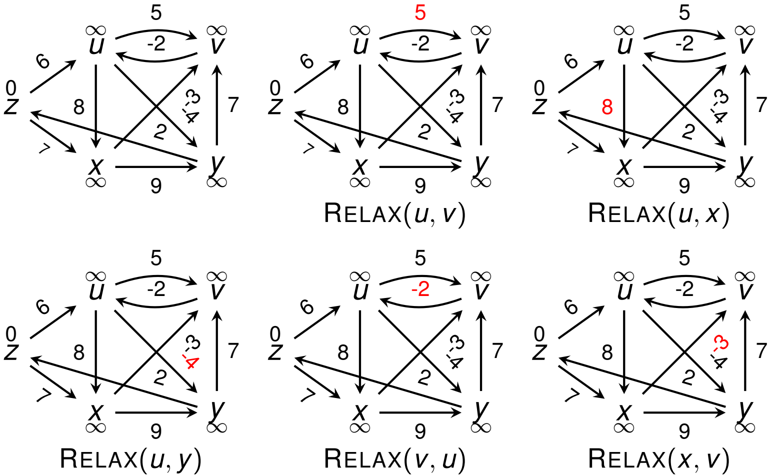
RELAX(u, y)

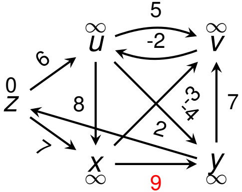


RELAX(v, u)

Bellman-Ford algorithm

Execution of first `for` loop

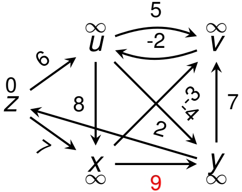




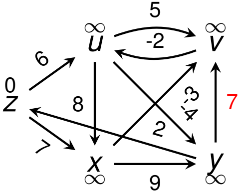
RELAX(x, y)

Bellman-Ford algorithm

Execution of first `for` loop (continued)



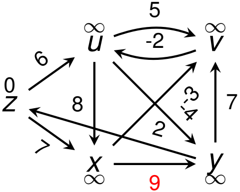
RELAX(x, y)



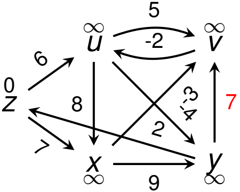
RELAX(y, v)

Bellman-Ford algorithm

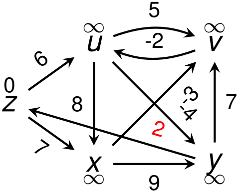
Execution of first `for` loop (continued)



$\textsc{Relax}(x, y)$



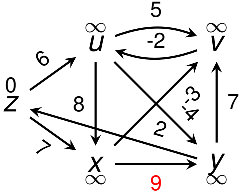
$\textsc{Relax}(y, v)$



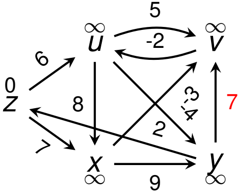
$\textsc{Relax}(y, z)$

Bellman-Ford algorithm

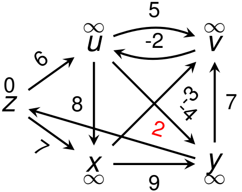
Execution of first `for` loop (continued)



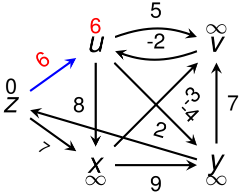
RELAX(x, y)



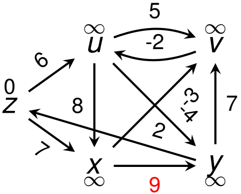
RELAX(y, v)



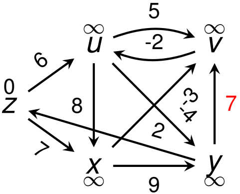
RELAX(y, z)



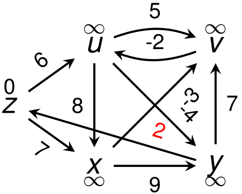
RELAX(z, u)



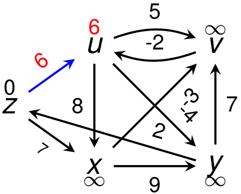
RELAX(x, y)



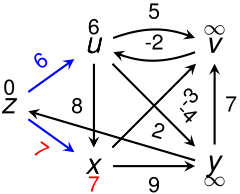
RELAX(y, v)



RELAX(y, z)



RELAX(z, u)



RELAX(z, x)

Overall execution of Bellman-Ford algorithm

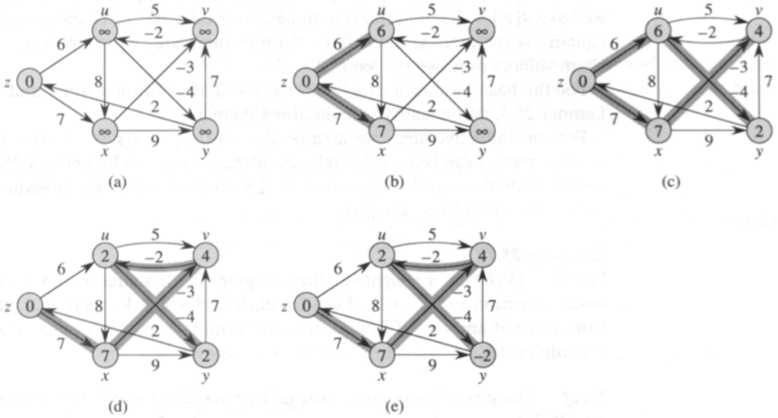
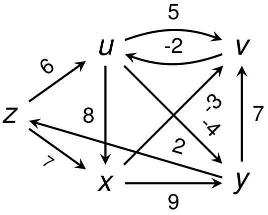


Figure 25.7 The execution of the Bellman-Ford algorithm. The source is vertex z. The d values are shown within the vertices, and shaded edges indicate the π values. In this particular example, each pass relaxes the edges in lexicographic order: $(u, v), (u, x), (u, y), (v, u), (x, v), (x, y), (y, v), (y, z), (z, u), (z, x)$. **(a)** The situation just before the first pass over the edges. **(b)–(e)** The situation after each successive pass over the edges. The d and π values in part (e) are the final values. The Bellman-Ford algorithm returns TRUE in this example.



- $\pi[z] = NIL, \pi[x] = z, \pi[v] = x, \pi[u] = v, \pi[y] = u$
  $d[z] = 0, d[x] = 7, d[v] = 4, d[u] = 2, d[y] = -2$
- G_π is

$$\begin{array}{rc} & z \\ 7 & | \\ & x \\ -3 & | \\ & v \\ -2 & | \\ & u \\ -4 & | \\ & y \end{array}$$

NOTE. G_π is shortest-part tree for G.

- After performing the usual initialization, 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. Figures (b)-(e) show the outcome of the algorithm after each of the four passes of the edges.
- After making $|V| - 1$ passes, lines 5-8 check for a negative-weight cycle and return the appropriate boolean value.

- The initialization in line 1 takes $\Theta(|V|)$ time
- Each of the $|V| - 1$ passes over the edges in lines 2-4 takes $O(|E|)$ time
- The **for** loop of lines 5-7 takes $O(|E|)$ time

$\Rightarrow$ Bellman-Ford algorithm runs in time $O(|V|\,|E|)$.

Bellman-Ford algorithm

Main properties

1. If G contains no negative-weight cycles that are reachable from s, then at the termination of BELLMANFORD(G, w, s), we have $d[v] = \delta(s, v)$ for all nodes v reachable from s
2. For each $v \in V$ there is a path $s \overset{\pi}{\leadsto} v$ if and only if BELLMANFORD(G, w, s) terminates with $d[v] < \infty$. (This is a corollary of the previous property).

Theorem (Correctness of Bellman-Ford algorithm)

Let BELLMANFORD(G, w, s) run on a weighted, directed graph G with source s and weight function $w : E \to \mathbb{R}$.

1. If G contains no negative-weight cycles reachable from s, then the algorithm returns `true`, we have $d[v] = \delta(s, v)$ for all $v \in V$, and G_π is a shortest-path tree rooted at s.
2. If G contains a negative-weight cycle reachable from s, then the algorithm returns `false`.

References

- Chapter 25 of
  - T. H. Cormen, C. E. Leiserson, R. L. Rivest. *Introduction to Algorithms*. MIT Press, 2000.