Single Source Shortest Paths with Arbitrary Weights

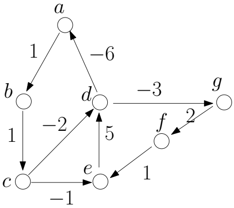
Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong

We will continue our discussion on the single source shortest path (SSSP) problem, but this time we will allow the edges to take **negative** weights.

Dijkstra's algorithm no longer works. We will learn another algorithm — called **the Bellman-Ford algorithm** — to solve the problem.

Let $G = (V, E)$ be a directed graph. Let w be a function that maps each edge in $e \in E$ to an integer $w(e)$, **which can be positive, 0, or negative**.



Shortest Path

Consider a path in G: $(v_1, v_2), (v_2, v_3), ..., (v_\ell, v_{\ell+1})$, for some integer $\ell \geq 1$. We define the path's **length** as

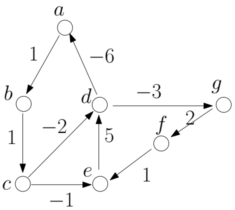
$$\sum_{i=1}^{\ell} w(v_i, v_{i+1}).$$

A **shortest path** from u to v has the minimum length among all the paths from u to v. Denote by $spdist(u, v)$ the length of a shortest path from u to v.

If v is unreachable from u, $spdist(u, v) = \infty$.

New: The length of a path can be negative!

Example



The path $c \to d \to g$ has length -5.

Can you find a shortest path from a to c? Counter-intuitively, it has an **infinite** number of edges such that $spdist(a, c) = -\infty$!

- This is due to the **negative cycle** $a \to b \to c \to d \to a$.

Negative cycle

A path $(v_1, v_2), (v_2, v_3), ..., (v_\ell, v_{\ell+1})$ is a **cycle** if $v_{\ell+1} = v_1$.

It is a **negative cycle** if its length is negative, namely:

$$\sum_{i=1}^{\ell} w(v_i, v_{i+1}) < 0$$

SSSP Problem: Let $G = (V, E)$ be a directed simple graph, where function w maps every edge of E to an arbitrary integer. **It is guaranteed that G has no negative cycles.** Given a **source vertex** s in V, we want to find a shortest path from s to t for every vertex $t \in V$ reachable from s.

The output is a **shortest path tree** T:

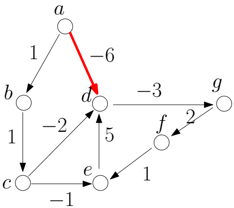
- The vertex set of T is V.
- The root of T is s.
- For each node $u \in V$, the root-to-u path of T is a shortest path from s to u in G.

We will learn an algorithm called **the Bellman-Ford algorithm** that solves both problems in $O(|V||E|)$ time.

We will focus on **computing** $spdist(s, v)$, namely, the shortest path distance from the source vertex s to every vertex $v \in V$.

Constructing the shortest paths is easy and will be left to you.

Example



This graph has no negative cycles.

Lemma: For every vertex $v \in V$, at least one shortest path from s to v is **simple path**, namely, a path where no vertex appears twice.

The proof is left to you — note that you must use the condition that no negative cycles are present.

Corollary: For every vertex $v \in V$, there is a shortest path from s to v having at most $|V| - 1$ edges.

Edge Relaxation

For every vertex $v \in V$, we will — at all times — maintain a value $dist(v)$ equal to the shortest path length from s to v **found so far**.

Relaxing an edge (u, v) means:

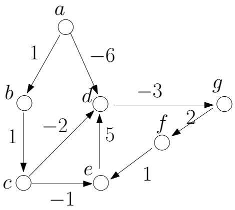
- If $dist(v) \leq dist(u) + w(u, v)$, do nothing;
- Otherwise, reduce $dist(v)$ to $dist(u) + w(u, v)$.

The Bellman-Ford algorithm

1. Set $dist(s) \leftarrow 0$, and $dist(v) \leftarrow \infty$ for all other vertices $v \in V$.
2. Repeat the following $|V| - 1$ times
   - Relax all edges in E (the relaxation order does not matter)

Example

Suppose that the source vertex is a.



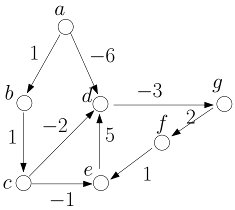
vertex v	$dist(v)$
a	0
b	∞
c	∞
d	∞
e	∞
f	∞
g	∞

For illustration purposes, we will relax the edges in alphabetic order: $(a, b), (a, d), (b, c), (c, d), (c, e), (d, g), (e, d), (f, e), (g, f)$.

Example

Relaxing all edges for the **first time**.

Here is what happens after relaxing (a, b):



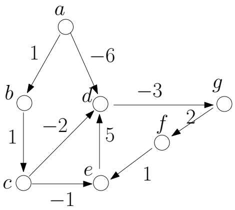
vertex v	$dist(v)$
a	0
b	1
c	∞
d	∞
e	∞
f	∞
g	∞

Alphabetic order of the edges in the graph:
$(a, b), (a, d), (b, c), (c, d), (c, e), (d, g), (e, d), (f, e), (g, f)$.

Example

Relaxing all edges for the **first time**.

Here is what happens after relaxing (a, d):



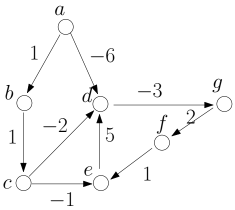
vertex v	$dist(v)$
a	0
b	1
c	∞
d	-6
e	∞
f	∞
g	∞

Alphabetic order of the edges in the graph:
$(a, b), (a, d), (b, c), (c, d), (c, e), (d, g), (e, d), (f, e), (g, f)$.

Example

Relaxing all edges for the **first time**.

Here is what happens after relaxing (b, c):



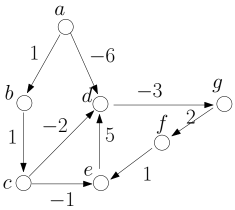
vertex v	$dist(v)$
a	0
b	1
c	2
d	−6
e	∞
f	∞
g	∞

Alphabetic order of the edges in the graph:
$(a, b), (a, d), (b, c), (c, d), (c, e), (d, g), (e, d), (f, e), (g, f)$.

Example

Relaxing all edges for the **first time**.

Here is what happens after relaxing (c, d):



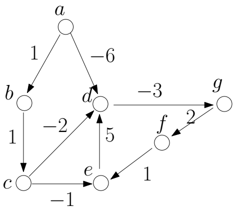
vertex v	$dist(v)$
a	0
b	1
c	2
d	−6
e	∞
f	∞
g	∞

Alphabetic order of the edges in the graph:
$(a, b), (a, d), (b, c), (c, d), (c, e), (d, g), (e, d), (f, e), (g, f)$.

Example

Relaxing all edges for the **first time**.

Here is what happens after relaxing (c, e):



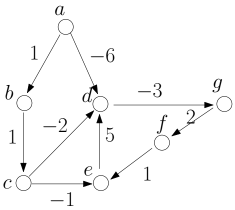
vertex v	$dist(v)$
a	0
b	1
c	2
d	−6
e	1
f	∞
g	∞

Alphabetic order of the edges in the graph:
$(a, b), (a, d), (b, c), (c, d), (c, e), (d, g), (e, d), (f, e), (g, f)$.

Example

Relaxing all edges for the **first time**.

Here is what happens after relaxing (d, g):



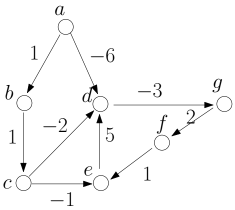
vertex v	$dist(v)$
a	0
b	1
c	2
d	−6
e	1
f	∞
g	−9

Alphabetic order of the edges in the graph:
$(a, b), (a, d), (b, c), (c, d), (c, e), (d, g), (e, d), (f, e), (g, f)$.

Example

Relaxing all edges for the **first time**.

Here is what happens after relaxing (e, d):



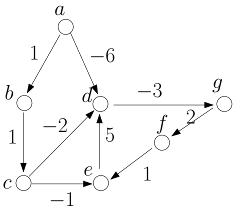
vertex v	$dist(v)$
a	0
b	1
c	2
d	−6
e	1
f	∞
g	−9

Alphabetic order of the edges in the graph:
$(a, b), (a, d), (b, c), (c, d), (c, e), (d, g), (e, d), (f, e), (g, f)$.

Example

Relaxing all edges for the **first time**.

Here is what happens after relaxing (f, e):



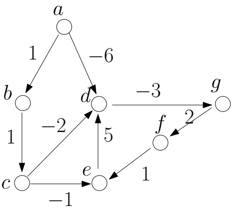
vertex v	$dist(v)$
a	0
b	1
c	2
d	−6
e	1
f	∞
g	−9

Alphabetic order of the edges in the graph:
$(a, b), (a, d), (b, c), (c, d), (c, e), (d, g), (e, d), (f, e), (g, f)$.

Example

Relaxing all edges for the **first time**.

Here is what happens after relaxing (g, f):



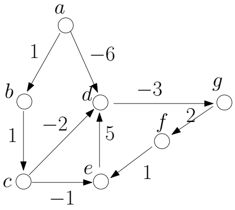
vertex v	$dist(v)$
a	0
b	1
c	2
d	−6
e	1
f	−7
g	−9

Alphabetic order of the edges in the graph:
$(a, b), (a, d), (b, c), (c, d), (c, e), (d, g), (e, d), (f, e), (g, f)$.

Example

In the same fashion, relax all edges for a **second time**.

Here is the content of the table at the end of this relaxation round:



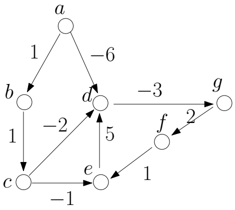
vertex v	$dist(v)$
a	0
b	1
c	2
d	−6
e	−6
f	−7
g	−9

Alphabetic order of the edges in the graph:
$(a, b), (a, d), (b, c), (c, d), (c, e), (d, g), (e, d), (f, e), (g, f)$.

Example

In the same fashion, relax all edges for a **third time**.

Here is the content of the table at the end of this relaxation round (no changes from the previous round):

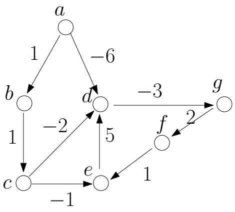


vertex v	$dist(v)$
a	0
b	1
c	2
d	−6
e	−6
f	−7
g	−9

Alphabetic order of the edges in the graph:
$(a, b), (a, d), (b, c), (c, d), (c, e), (d, g), (e, d), (f, e), (g, f)$.

Example

In the same fashion, relax all edges for a **fourth time**, **fifth time**, and then a **sixth time**. No more changes to the table:



vertex v	$dist(v)$
a	0
b	1
c	2
d	−6
e	−6
f	−7
g	−9

The algorithm then terminates here with the above values as the final shortest path distances.

Remark: We did 6 rounds only to follow the algorithm description faithfully. As a heuristic, we can stop as soon as no changes are made to the table after some round.

Time

The running time is clearly $O(|V||E|)$.

Correctness

Theorem: Consider any vertex v; suppose that there is a shortest path from s to v that has ℓ edges. Then, after ℓ rounds of edge relaxations, it must hold that $dist(v) = spdist(v)$.

Proof:

We will prove the theorem by induction on ℓ. If $\ell = 0$, then $v = s$, in which case the theorem is obviously correct. Next, assuming the statement's correctness for $\ell < i$ where i is an integer at least 1, we will prove it holds for $\ell = i$ as well.

Denote by π the shortest path from s to v, namely, π has i edges. Let p be the vertex right before v on π.

By the inductive assumption, we know that $dist(p)$ was already equal to $spdist(p)$ after the $(i-1)$-th round of edge relaxations.

In the i-th round, by relaxing edge (p, v), we make sure:

$$\begin{aligned} dist(v) &\le dist(p) + w(p, v) \\ &= spdist(p) + w(p, v) \\ &= spdist(v). \end{aligned}$$

□