CSCI3160 Design and Analysis of Algorithms (2025 Fall)

SSSP with Arbitrary Weights

Instructor: Xiao Liang[1]

Department of Computer Science and Engineering
The Chinese University of Hong Kong

[1]These slides are primarily based on materials prepared by Prof. Yufei Tao (please refer to Prof. Tao's version from 2024 Fall for the original content). Some modifications have been made to better align with this year's teaching progress, incorporating student feedback, in-class interactions, and my own teaching style and research perspective.

What we have so far

Shortest path problem:

1. Non-negative weights: Dijkstra's algorithm
2. Negative weights:
   1. negative cycles:
      - Shortest Paths are not well-defined.
      - Shortest Simple Paths are well defined. However, this is a NP-hard problem.
   2. no negative cycles:
      - Shortest Paths are well-defined.
      - But Dijkstra's algorithm does not work.

What should we do with graphs that contain no negative cycles?

- **Bellman-Ford algorithm** (this lecture).

Problem Statement

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 contains all vertices reachable from s.
- 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 the **the Bellman-Ford algorithm** that solves this problem in $O(|V||E|)$ time.

Note:

- 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.

Bellman-Ford Algorithm

Recalling Edge Relaxation

We begin by recalling the Edge Relaxation procedure introduced when studying Dijkstra's algorithm.

Edge Relaxation

Relaxing an edge (u, v) means:

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

Algorithm Description

The Bellman-Ford algorithm

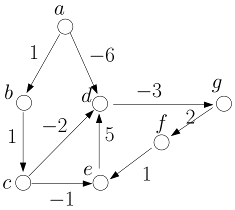
1. Set $dist(s) \leftarrow 0$, and $dist(v) \leftarrow \infty$ for each vertex $v \in V \setminus \{s\}$
2. Repeat the following $|V| - 1$ times
   - Relax all edges in E (the relaxation order does not matter)

Dijkstra's algorithm (for comparison):

1. Set $dist(s) \leftarrow 0$ and $dist(v) \leftarrow \infty$ for each vertex $v \in V \setminus \{s\}$
2. Set $S \leftarrow V$
3. Repeat the following until S is empty:
   - Remove from S the vertex u with the smallest $dist(u)$.
   - Relax every outgoing edge (u, v) of u.

Example

Suppose that the source vertex is a.



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

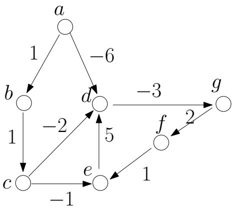
For illustration purposes, we will relax the edges in alphabetic order shown below:

$$(a, b), (a, d), (b, c), (c, d), (c, e), (d, g), (e, d), (f, e), (g, f).$$

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	∞

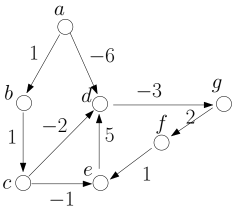
Alphabetic order of the edges in the graph:

$$(a, b), (a, d), (b, c), (c, d), (c, e), (d, g), (e, d), (f, e), (g, f).$$

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	∞

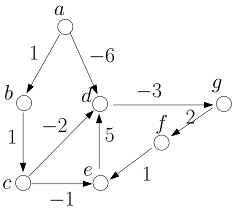
Alphabetic order of the edges in the graph:

$$(a, b), (a, d), (b, c), (c, d), (c, e), (d, g), (e, d), (f, e), (g, f).$$

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	∞

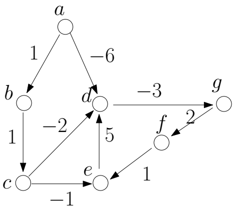
Alphabetic order of the edges in the graph:

$$(a, b), (a, d), (b, c), (c, d), (c, e), (d, g), (e, d), (f, e), (g, f).$$

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	∞

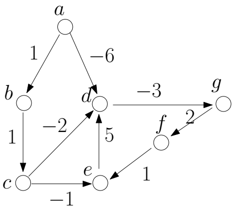
Alphabetic order of the edges in the graph:

$$(a, b), (a, d), (b, c), (c, d), (c, e), (d, g), (e, d), (f, e), (g, f).$$

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	∞

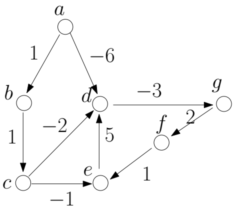
Alphabetic order of the edges in the graph:

$$(a, b), (a, d), (b, c), (c, d), (c, e), (d, g), (e, d), (f, e), (g, f).$$

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

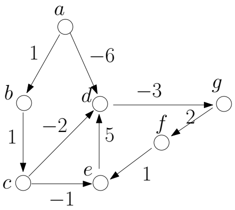
Alphabetic order of the edges in the graph:

$$(a, b), (a, d), (b, c), (c, d), (c, e), (d, g), (e, d), (f, e), (g, f).$$

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

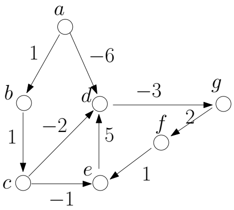
Alphabetic order of the edges in the graph:

$$(a, b), (a, d), (b, c), (c, d), (c, e), (d, g), (e, d), (f, e), (g, f).$$

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

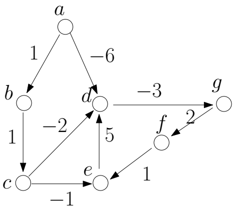
Alphabetic order of the edges in the graph:

$$(a, b), (a, d), (b, c), (c, d), (c, e), (d, g), (e, d), (f, e), (g, f).$$

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

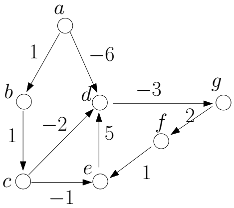
Alphabetic order of the edges in the graph:

$$(a, b), (a, d), (b, c), (c, d), (c, e), (d, g), (e, d), (f, e), (g, f).$$

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:

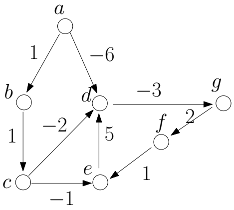


vertex v	$dist(v)$
a	0
b	1
c	2
d	−6
e	−6
f	−7
g	−9

Example

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

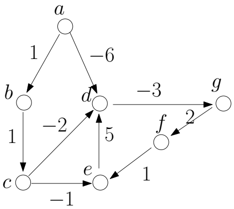
Here is the content of the table at the end of this relaxation round (no changes from the previous round):



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

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 $O(|V||E|)$.

Proof of Correctness

Correctness

Lemma: For every vertex $v \in V$ such that $v \neq s$, at least one shortest path from s to v is a **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.

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 $dist(v) = spdist(v)$.

Convince yourself that this theorem (with the previous corollary) establishes the correctness of Bellman-Ford.

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:

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

In the above:

- The first "$=$": $dist(p)$ is already the shortest path after the $(i-1)$-th round.
    - Note that the shortest path from s to p must coincide with π (Think: why?)
- The second "$=$": by the definition of π and p.

□