Lecture 15 — Shortest Paths (Part 1)

COMPSCI 308 — Design and Analysis of Algorithms

Xing Shi Cai https://newptcai.gitlab.io

Duke Kunshan University, Kunshan, China

Jan-Mar 2026

Summary

ITA 24 Single-Source Shortest Paths

ITA 22.1 The Bellman-Ford Algorithm

Assignments[1]

Practice makes perfect!

Introduction to Algorithms (ITA) 📖 Required Readings:

- Section 22.1.

✏️ Required Exercises:

- Exercises 22.1: 1–4.

[1] 💣 Assignments will not be collected; however, quiz problems will be selected from them. (This includes both Readings and Exercises.)

ITA 24 Single-Source Shortest Paths

Applications of Shortest Path Finding

🤔 How do AI find path in complicated environments such as in Real Time Strategy (RTS) games?

Figure 1: Age of Empire IV

Applications of Shortest Path Finding

A 🐰 drives from Kunshan to Chengdu.

How do 🗺️ Apps help the 🐰 find the shortest route?

Figure 1: Driving on the shortest path

Applications of Shortest Path Finding

Network routing involves determining the most efficient data path in internet traffic.

Common routing protocols

- OSPF (Open Shortest Path First)
- BGP (Border Gateway Protocol).

Figure 1: How to find the best route on the internet?

The Weight of a Path

Given a weighted, directed graph $G = (V, E)$ with a weight function $w : E \to \mathbb{R}$, the shortest path problem involves finding the path with minimum weight between two vertices.

The weight of a path $p = \langle v_0, v_1, \ldots, v_k \rangle$ is defined as:

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

Let $p' = \langle 1, 5, 4 \rangle$

$w(p') = 15 + 3 = 18$.

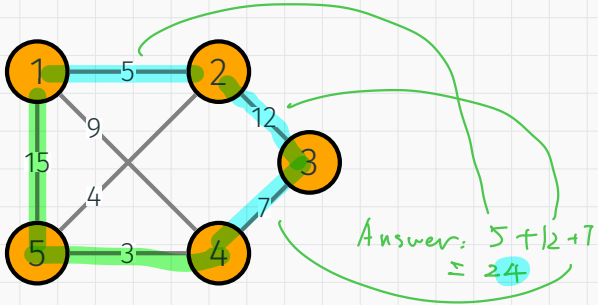


Figure 2: What is the weight of $p = \langle 1, 2, 3, 4 \rangle$?

The shortest-path weight from u to v, denoted $\delta(u, v)$, is:

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

A path p is a shortest path from u to v if $w(p) = \delta(u, v)$.

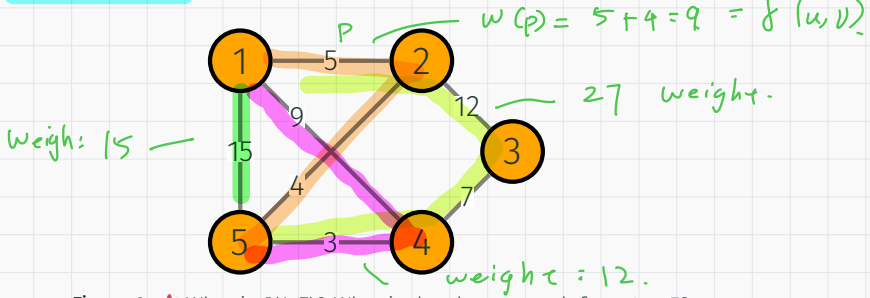


Figure 3: What is $\delta(1,5)$? What is the shortest path from 1 to 5?

This chapter focuses on the single-source shortest-paths problem — find a shortest path from a specified source vertex s to every other vertex $v \in V$.

🍰 Do you see how to use single-source shortest-paths to solve these problems?

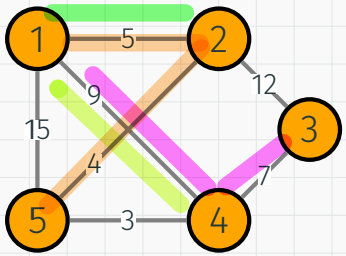


Figure 4: 🍰 What are the shortest paths from 1?

Variants of Shortest-Paths Problems

Variations include:

- Single-destination shortest-paths problem

Already computes this.

🍰 Do you see how to use single-source shortest-paths to solve these problems?

Multiple destination

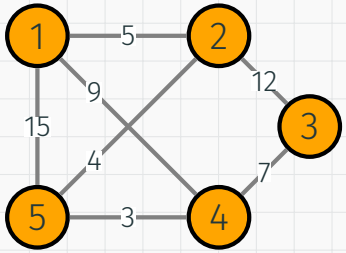


Figure 4: 🍰 What are the shortest paths from 1 to 4?

Variants of Shortest-Paths Problems

Variations include:

- Single-destination shortest-paths problem
- Single-pair shortest-path problem

🍰 Do you see how to use single-source shortest-paths to solve these problems?

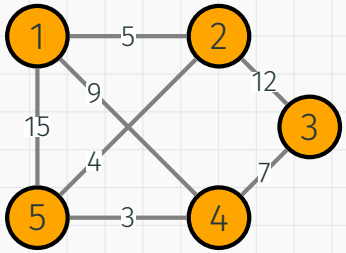


Figure 4: 🍰 What are the shortest paths between every pair of vertices?

Variants of Shortest-Paths Problems

Other problems:
- Reachable vertices given a budget.
- Multi-source and single destination.

Variations include:

- Single-destination shortest-paths problem
- [illegible]
- All-pairs shortest-paths problem

🍰 Do you see how to use single-source shortest-paths to solve these problems?

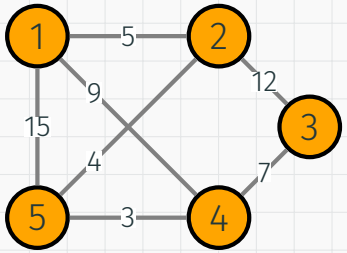


Figure 4:

Run single source from every vertex

Lemma 22.1

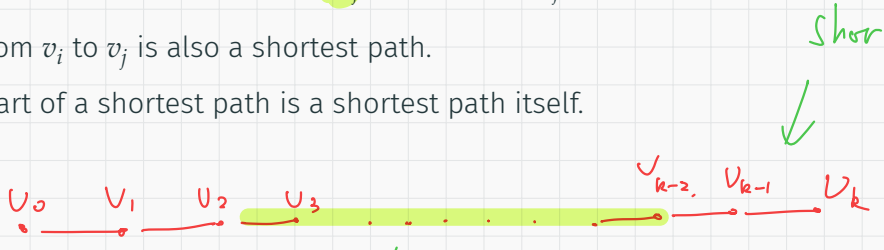
Let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be a shortest path from v_0 to v_k.

For any i and j with $0 \le i \le j \le k$, the subpath

$$p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$$

of p from v_i to v_j is also a shortest path.

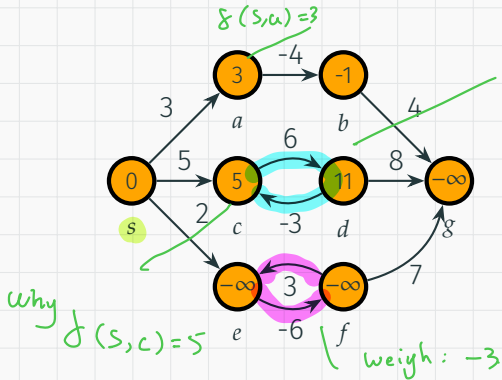
💡 A part of a shortest path is a shortest path itself.



Proof: If p_{ij} is not shortest, replace it by the shortest path, we get a shorter path from v_0 to v_k. → Contradiction.

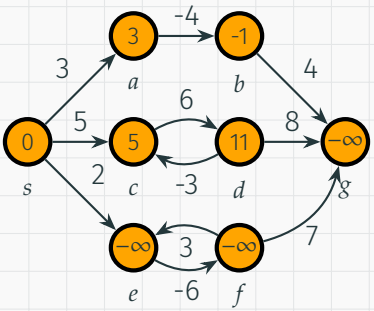
Negative Cycles

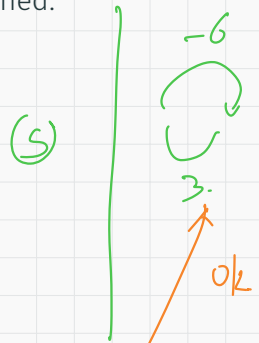
If there is a negative cycle in G, then $\delta(u, v)$ may not be well-defined.



Negative Cycles

If there is a negative cycle in G, then $\delta(u, v)$ may not be well-defined.





Some algorithms assume there is no negative-weight edge.

Others allow such edges, *as long as* the source cannot reach a negative-weight cycle.

Cycles

Can a shortest path contain any of these —

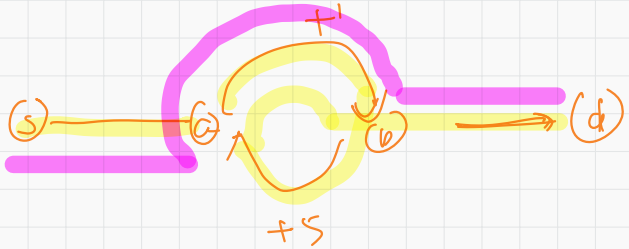
- a negative-weight cycle → No. It makes shortest path not-well defined.

Cycles

Can a shortest path contain any of these —

- a negative-weight cycle – Nope
- or a positive-weight cycle – Nope

Increase a path's weight, can be dropped.

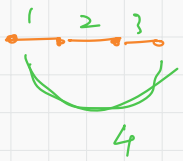


Cycles

Can a shortest path contain any of these —

- a negative-weight cycle
- or a positive-weight cycle
- or a zero-weight cycle — Maybe. But usually not.

Cycles



Can a shortest path contain any of these —

- a negative-weight cycle
- or a positive-weight cycle
- or a zero-weight cycle

usually

Thus, when we talk about shortest paths, we mean the shortest simple path , i.e., path without cycles.

Why does a shortest path in $G = (V, E)$ contain at most $|V| - 1$ edges?

A path with at least $|V|$ vertices has a cycle.

It cannot be a simple path.

The algorithms (in this chapter) will maintain an attribute $v.\pi$ which is the predecessor of v in the shortest path from s to v.

By checking $v.\pi$, we can find the shortest path from s to v.

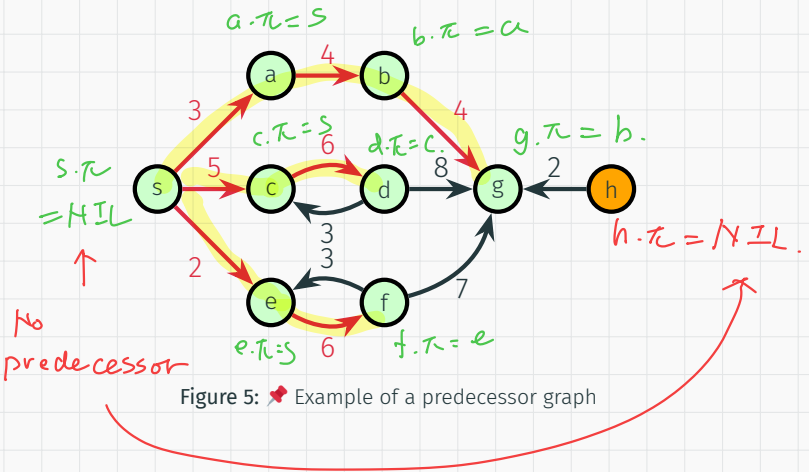


Figure 5: 📌 Example of a predecessor graph

We define the predecessor subgraph G_π as the subgraph with vertex set

$$V_\pi = \{v \in V : v.\pi \neq \texttt{NIL}\} \cup \{s\}$$

and edge set

$$E_\pi = \{(v.\pi, v) : v \in V_\pi - \{s\}\}$$

Call this subgraph G_π

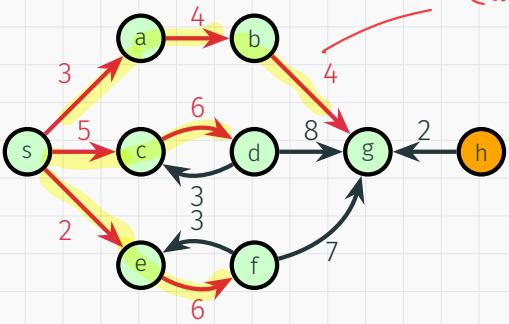


Figure 5: 📌 Example of a predecessor graph

The algorithms which we study will produce G_π with these properties —

- It is a tree with root s.
- It has a vertex set V' containing all vertices reachable from s.
- For all vertex $v \in V'$, the path from s to v in G_π is a shortest-path from v to s.

We call such a subgraph of G a shortest-paths tree.

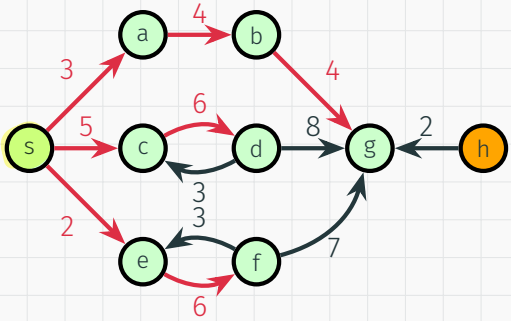


Figure 6: 📌 Example of a shortest-paths tree

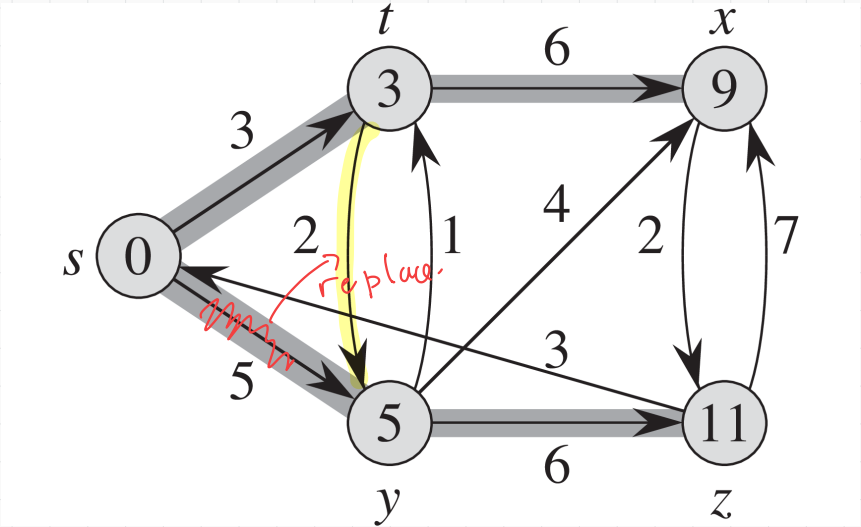


Figure 7: A shortest-paths tree

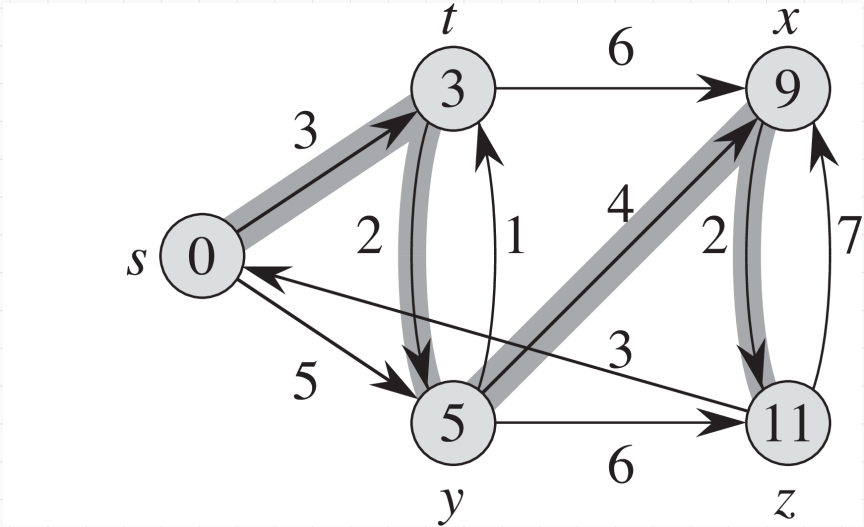


Figure 7: Another shortest-paths tree

Shortest-path Estimate

The algorithms which we study maintain an attribute $v.d$ which is the upper bound of the weight of the shortest path from s to v.
We call $v.d$ the shortest-path estimate of v.

The algorithms which we study maintain an attribute $v.d$ which is the upper bound of the weight of the shortest path from s to v.
We call $v.d$ the shortest-path estimate of v.
All algorithms call the following procedure —

Init-Single-Source(G, s)
1: **for** each vertex $v \in G.V$ **do**
2: $v.d = \infty$
3: $v.\pi = \texttt{NIL}$
4: $s.d = 0$

What is the complexity?

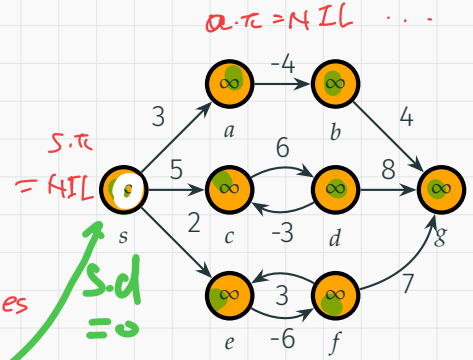


Figure 8: After **Init-Single-Source** is called

Relax

Relaxing an edge (u, v) means —

- Test whether we can find a shorter path to v by going through u.
- If so, update $v.d$ and $v.\pi$.

↑ decrease estimate

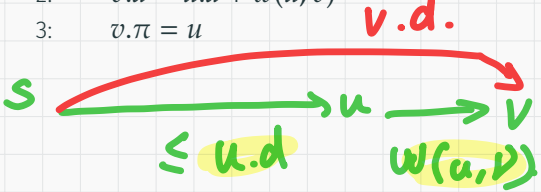
Relax

Relaxing an edge (u, v) means —

- Test whether we can find a shorter path to v by going through u.
- If so, update $v.d$ and $v.\pi$.

Relax(u, v, w)

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



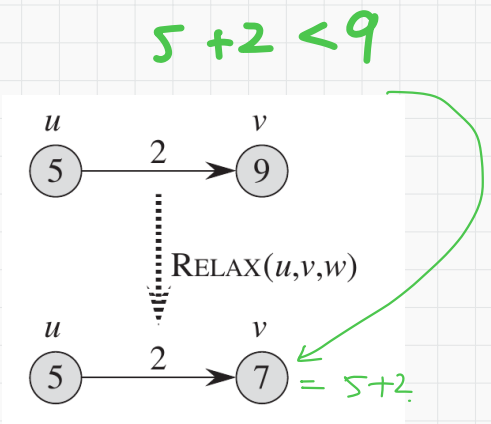


Figure 9: A "successful 😄" relaxation of an edge (u, v)

Relax

Relaxing an edge (u, v) means —

- Test whether we can find a shorter path to v by going through u.
- If so, update $v.d$ and $v.\pi$.

Relax(u, v, w)

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

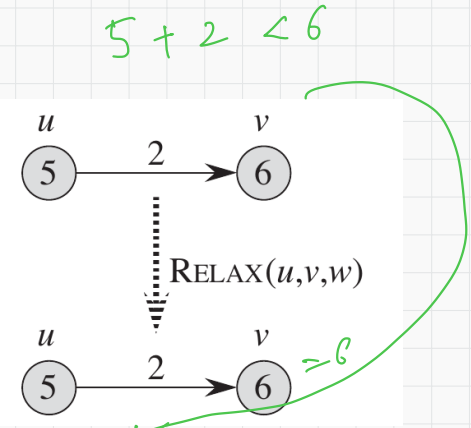


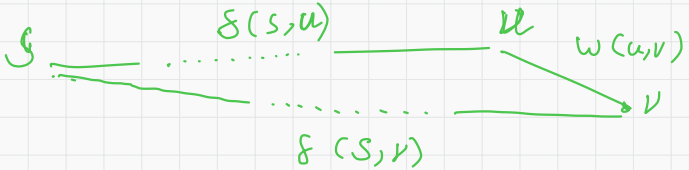
Figure 9: A "failed 🙁" relaxation of an edge (u, v)

Properties of Shortest Paths

Properties that is true for all graph —

Triangle Inequality (Lemma 22.10): For any edge (u, v),

$$\delta(s, v) \leq \delta(s, u) + w(u, v)$$



Properties of Shortest Paths

Properties that is true for all graph —

Triangle Inequality (Lemma 22.10): For any edge (u, v),

$$\delta(s, v) \leq \delta(s, u) + w(u, v)$$

💻 Properties that is maintained by the algorithms —

Upper-bound Property (Lemma 22.11): 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.

Properties of Shortest Paths

Properties that is true for all graph —

Triangle Inequality (Lemma 22.10): For any edge (u, v),

$$\delta(s, v) \leq \delta(s, u) + w(u, v)$$

💻 Properties that is maintained by the algorithms —

Upper-bound Property (Lemma 22.11): 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 (Lemma 22.12): If no path exists from s to v, the shortest path estimate remains infinite.

$v.d = \infty$ forever

Properties of Shortest Paths

Properties that is true for all graph —

Triangle Inequality (Lemma 22.10): For any edge (u, v),

$$\delta(s, v) \leq \delta(s, u) + w(u, v)$$

💻 Properties that is maintained by the algorithms —

Upper-bound Property (Lemma 22.11): 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 (Lemma 22.12): If no path exists from s to v, the shortest path estimate remains infinite.

Convergence Property (Lemma 22.14): If $s \rightsquigarrow u \rightarrow v$ 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 afterwards.

Properties of Shortest Paths

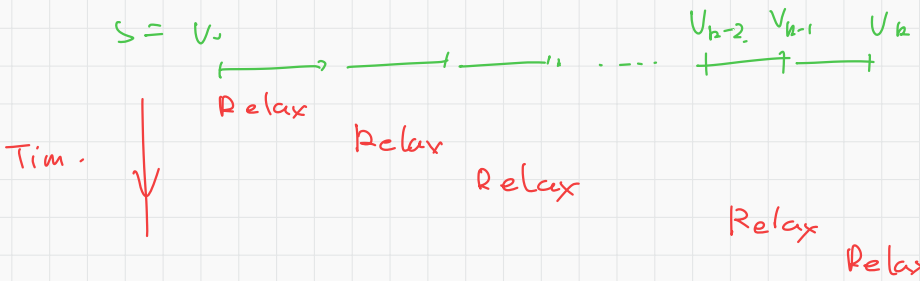
💻 Properties that is maintained by the algorithms —

★ Path-relaxation Property (Lemma 22.15): If $p = \langle v_0, v_1, \ldots v_k \rangle$ is the shortest path from $v_0 = s$ to v_k, then relaxing the edges in the order

$$(v_0, v_1), \ldots, (v_{k-1}, v_k)$$

will make $v_k.d = \delta(s, v_k)$.

This hold *regardless* of any other relaxation steps that occur.



Properties of Shortest Paths

💻 Properties that is maintained by the algorithms —

⭐ Path-relaxation Property (Lemma 22.15): If $p = \langle v_0, v_1, \dots v_k \rangle$ is the shortest path from $v_0 = s$ to v_k, then relaxing the edges in the order

$$(v_0, v_1), \dots, (v_{k-1}, v_k)$$

will make $v_k.d = \delta(s, v_k)$.

This hold *regardless* of any other relaxation steps that occur.
Predecessor-subgraph Property (Lemma 22.17): After all vertices are processed, the predecessor subgraph forms a shortest-paths tree rooted at the source s.

🤪 Proof for these lemmas are not required.

ITA 22.1 The Bellman-Ford Algorithm

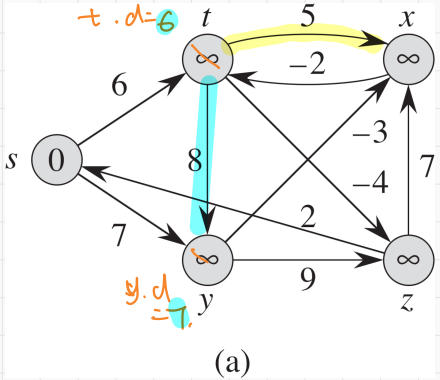
Example Run of Bellman-Ford's Algorithm

In the example, if (u, v) is shaded, then $v.\pi = u$.

The edges are relaxed in the order

$$(t,x), (t,y), (t,z), (x,t), (y,x), (y,z), (z,x), (z,s), (s,t), (s,y)$$

F F F F F F F F S



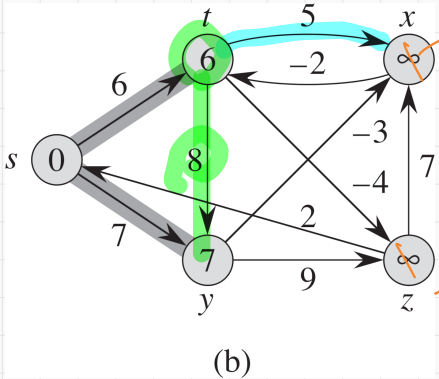
Example Run of Bellman-Ford's Algorithm

In the example, if (u, v) is shaded, then $v.\pi = u$.

The edges are relaxed in the order

$$(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$$

S F S



x.d = 11 = 5 + 6

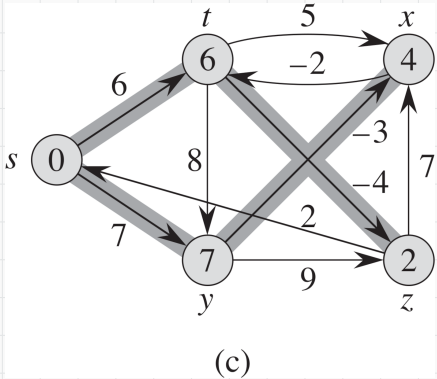
z.d = 2.

Example Run of Bellman-Ford's Algorithm

In the example, if (u, v) is shaded, then $v.\pi = u$.

The edges are relaxed in the order

$$(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$$



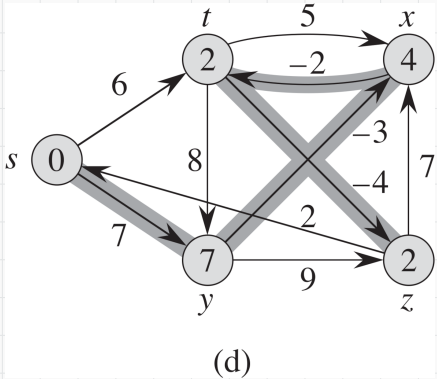
(c)

Example Run of Bellman-Ford's Algorithm

In the example, if (u, v) is shaded, then $v.\pi = u$.

The edges are relaxed in the order

$$(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$$

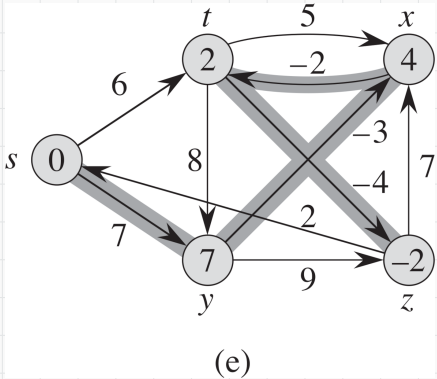


Example Run of Bellman-Ford's Algorithm

In the example, if (u, v) is shaded, then $v.\pi = u$.

The edges are relaxed in the order

$$(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$$



Pseudocode of Bellman-Ford's Algorithm

The Bellman-Ford algorithm allows for some of the edges to have negative weights.

- The algorithm relaxes every edge $|V| - 1$ times.
- It returns `False` if a negative-weight cycle is reachable from the source s.
- Otherwise, it successfully computes the shortest paths from s and their corresponding weights.

Pseudocode of Bellman-Ford's Algorithm

The Bellman-Ford algorithm allows for some of the edges to have negative weights.

- The algorithm relaxes every edge $|V| - 1$ times.
- It returns `False` if a negative-weight cycle is reachable from the source s.
- Otherwise, it successfully computes the shortest paths from s and their corresponding weights.

```
Bellman-Ford(G, w, s)
1: Init-Single-Source(G, s)
2: for i = 1 to |G.V| − 1 do
3:     for each edge (u, v) ∈ G.E do
4:         Relax(u, v, w)
5: for each edge (u, v) ∈ G.E do
6:     if v.d > u.d + w(u, v) then
7:         return False          ▷ There is a negative cycle
8: return True
```

What is the ⏱ complexity?

|V| - 1 Time.

If still succeed in relax.

🍰 Relax the edges in the order

$$(s,a),(s,c),(a,b),(b,c),(c,a)$$

three times. Which edge is the last one to be relaxed successfully?

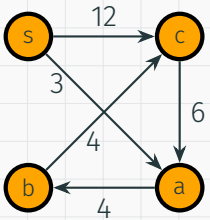


Figure 10: 📌 A weighted digraph without negative-weight cycles

Lemma 22.2

Assume that there is no negative-weight cycle reachable from the source s.

Then after the $|V| - 1$ iterations of the for loop, $v.d = \delta(s, v)$ for all vertices reachable from s.

Lemma 22.2

Assume that there is no negative-weight cycle reachable from the source s.

Then after the $|V|-1$ iterations of the for loop, $s.d = \delta(s,v)$ for all vertices reachable from s.

This follows from the following —

★ Path-relaxation Property (Lemma 22.15)

If $p = \langle v_0, v_1, \ldots v_k \rangle$ is the shortest path from $v_0 = s$ to v_k, then relaxing the edges in the order

$$(v_0, v_1), \ldots, (v_{k-1}, v_k)$$

will make $v_k.d = \delta(s, v_k)$.
This hold *regardless* of any other relaxation steps that occur.

Theorem 22.4

If there is no *negative-weight* cycle reachable from the source s, then after Bellman-Ford terminates,

- 🍰 (a) $v.d = \delta(s, v)$ for all vertices
- 🍰 (b) G_π as a shortest-paths tree rooted at s
- 🤔 (c) and the algorithm returns `True`

Theorem 22.4

If there is no *negative-weight* cycle reachable from the source s, then after Bellman-Ford terminates,

🍰 (a) $v.d = \delta(s, v)$ for all vertices

🤔 (c) and the algorithm returns `True`

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

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

So none of the vertex v satisfies

$$v.d > u.d + w(u, v)$$

and causes the algorithm to return `False`.

Theorem 22.4

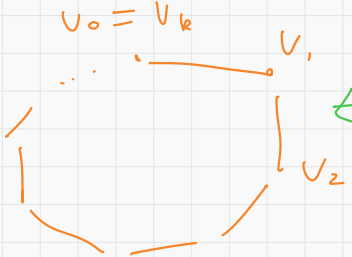
If graph G contains a negative-weight cycle that is reachable from the source s, the `Bellman-Ford` returns `False`.

Proof — Let this cycle be $c = \langle v_0, v_1, \ldots, v_k \rangle$, where $v_0 = v_k$.

Assume for the purpose of contradiction that the Bellman-Ford algorithm returns `True`. Thus,

$$v_i.d \leq v_{i-1}.d + w(v_{i-1}, v_i) \qquad \text{for all } i = 1, 2, \ldots, k$$

BF returns True. All relax fails in last round.



← Negative weight.

Theorem 22.4

If graph G contains a negative-weight cycle that is reachable from the source s, the `Bellman-Ford` returns `False`.

Proof — Summing the inequalities around cycle c gives us

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

The same.

Total weight.

Theorem 22.4

If graph G contains a negative-weight cycle that is reachable from the source s, the `Bellman-Ford` returns `False`.

Proof — Summing the inequalities around cycle c gives us

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

Note that

$$\sum_{i=1}^{k} v_i.d = \sum_{i=1}^{k} v_{i-1}.d$$

Theorem 22.4

If graph G contains a negative-weight cycle that is reachable from the source s, the `Bellman-Ford` returns `False`.

Proof — Summing the inequalities around cycle c gives us

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

Note that

$$\sum_{i=1}^{k} v_i.d = \sum_{i=1}^{k} v_{i-1}.d$$

Therefore,

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

🍰 Do you see why this is a contradiction? Because this is negative cycle.

What are your main takeaways today? Any questions?