CMPT 307: Shortest Paths

Week 12-1

Xian Qiu

Simon Fraser University

xianq@sfu.ca

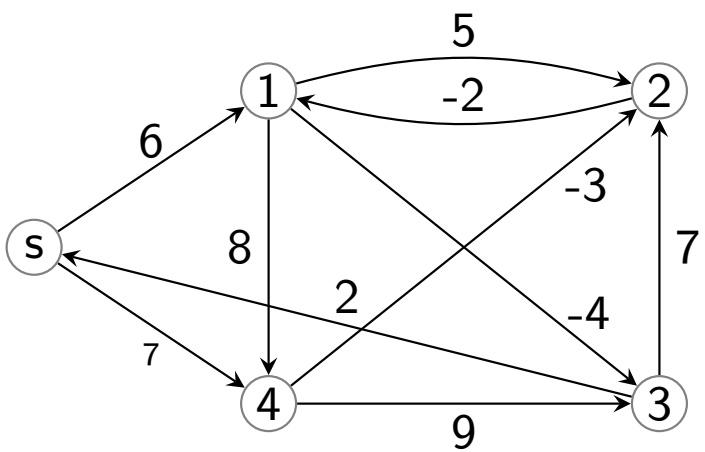
Outline

1. single source shortest paths
2. acyclic graphs
3. nonnegative costs
4. all pairs of shortest paths

Single Source Shortest Paths

Single source shortest path problem

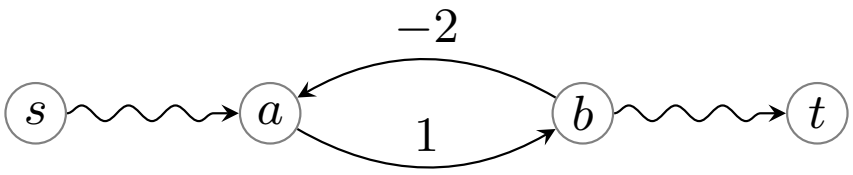
- given: digraph $G = (V, E)$ cost function $c : E \to \mathbb{R}$, and source node s
- goal: compute shortest s-t paths, for all $t \in V$



Distance Function

do shortest s-t paths always exist? exist if

- t is reachable from s
- no s-t paths contain a negative cycle



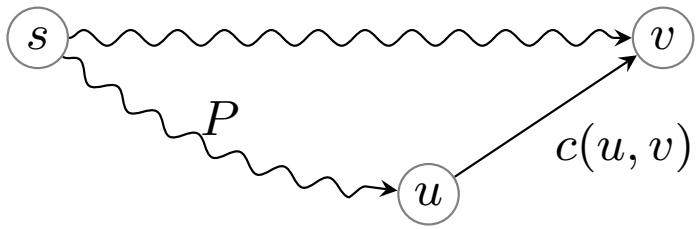
Distance function $\delta : V \to \mathbb{R}$

$$\delta(v) = \min\{c(P) \mid P \text{ is a path from } s \text{ to } v\},\ \forall v \in V$$

- $\delta(v) = \infty \Leftrightarrow$ no path from s to v
- $\delta(v) = -\infty \Leftrightarrow \exists s$-$v$ path containing a negative cycle
- $\delta(v) \in \mathbb{R} \Leftrightarrow \exists s$-$v$ path

Triangle Inequality

Claim 1. $\delta(v) \leq \delta(u) + c(u,v), \forall e = (u,v) \in E.$ (Δ-inequality)



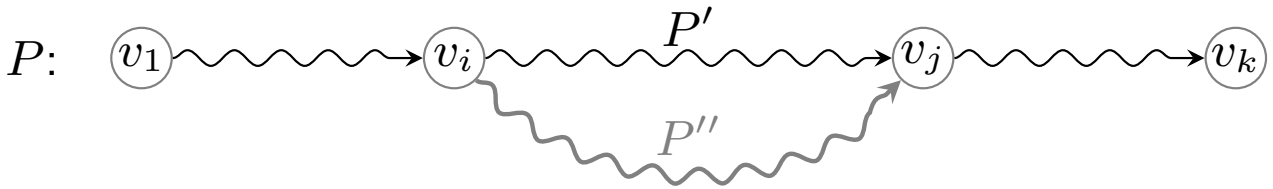
Proof

distinguish three cases $\delta(u) \in \{\mathbb{R}, \infty, -\infty\}$

- $\delta(u) \in \mathbb{R} \Rightarrow \exists$ path P s.t. $c(P) = \delta(u)$
- $\Rightarrow$ $P + e$ is an s-v path of length $\delta(u) + c(u,v) \geq \delta(v)$
- trivial if $\delta(u) = \infty$
- $\delta(v) = -\infty \Rightarrow$ negative cycle $\Rightarrow \delta(u) = -\infty$

Optimal Substructure

Claim 2. Let $P = \langle v_1, v_2, \ldots, v_k \rangle$ be a shortest v_1-v_k path, then every subpath $P' = \langle v_i, v_{i+1} \ldots, v_j \rangle$ is a shortest v_i-v_j path.



Claim 3. Let $P = \langle s, \ldots, u, v \rangle$ be a shortest s-v path. It holds that $\delta(v) = \delta(u) + c(u, v)$.

Proof

$\delta(v) = c(P) = c(\langle s, \ldots, u \rangle) + c(u, v) = \delta(u) + c(u, v)$

- $\delta(s) = 0$, if the successor x of s is known, so is $\delta(x)$
- x or $\delta(x)$ is found by brute force

Computing $\delta(v)$

- define tentative distance function $d : V \to \mathbb{R}$
- the idea is to use d to compute a more and more refined approximation of δ until $d(v) = \delta(v)$ for all $v \in V$
- initialization: $d(s) = 0, d(v) = \infty, \forall v \in V \setminus \{s\}$
- improve $d(v)$ by operation: $\textsc{Relax}(u, v, c)$

$\textsc{Relax}(u, v, c)$

```
1 if d(v) > d(u) + c(u, v) then
2     d(v) = d(u) + c(u, v);
3     v.p = u;                          // record predecessor
```

Properties of $d(v)$

Claim 4. $d(v) \geq \delta(v)$, for all $v \in V$. $d(v)$ is feasible

Claim 5. Let P be a shortest s-v path and $e = (u, v) \in P$. If $d(u) = \delta(u)$, then $d(v) = \delta(v)$ after relaxing e.

Proof

$\textsc{Relax}(u, v) \Rightarrow d(v) := d(u) + c(u, v) = \delta(u) + c(u, v) = \delta(v)$ (Claim 3)

- if predecessor u is computed, then its successor can be found after $\textsc{Relax}(u, v, c)$ for all v with $(u, v) \in \delta(u)$

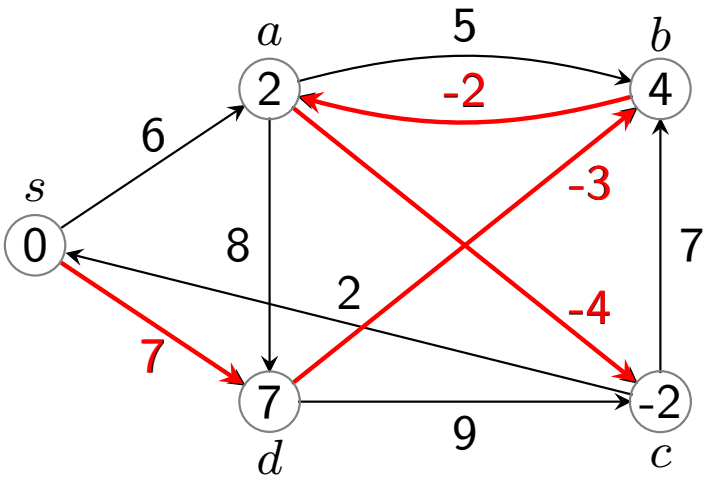
Bellman-Ford Algorithm

- relaxing each edge $n-1$ times can compute $\delta(v),\ \forall v \in V$ if no negative cycle in G
- how to identify negative cycles?

BELLMAN-FORD(G, c, s)

```
1 initialization: s.d = 0 and v.d = ∞, ∀v ∈ V \ {s};
2 for i = 1 to n − 1 do
3     for each edge (u, v) ∈ E do
4         RELAX(u, v, c);
5 for each edge (u, v) ∈ E do
6     if v.d > u.d + c(u, v) then
7         return false;                          // negative cycle
8 return true;                                   // no negative cycle
```

Example

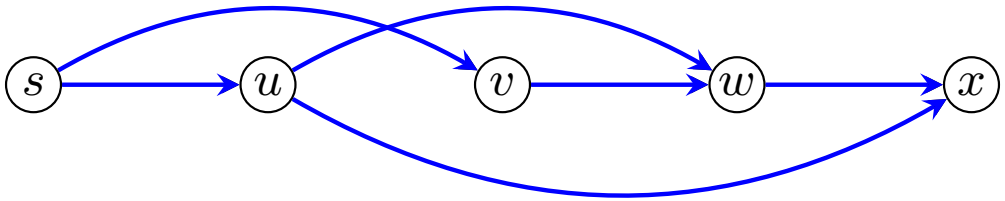


	s	a	b	c	d
initial	0	∞	∞	∞	∞
iter. 1	0	6	11	2	7
iter. 2	0	6	4	2	7
iter. 3	0	2	4	-2	7
iter. 4	0	2	4	-2	7

shortest path tree: $T = (V_p, E_p)$, where

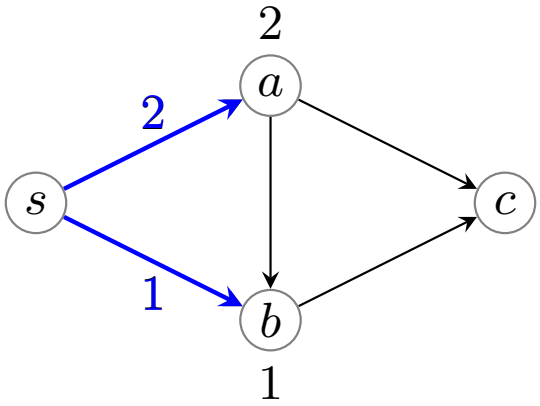
- $V_p = \{v \in V \mid v.\mathsf{p} \neq \mathsf{nil}\}$
- $E_p = \{(v.\mathsf{p}, v) \mid v \in V_p - s\}$

Acyclic Graphs



1. topologically sort vertices of G
2. relax $\delta(v)$, for all v "from left to right"

Nonnegative Costs



- relax $\delta(s)$ and find $v_1 = \arg\min d(v)$
- $d(v_1) = \delta(v_1)$
- relax $\delta(v_1)$ and find $v_2 = \arg\min d(v)$ ...

Dijkstra's Algorithm

DIJKSTRA(G, c, s)

```
1 s.d = 0; v.d = ∞, ∀v ∈ V − s;
2 Q = G.V;
3 while Q ≠ ∅ do
4     u = EXTRACT-MIN(Q);
5     for each vertex v ∈ δ(u) do
6         RELAX(u, v, c);                    // call DECREASE-KEY implicitly
```

- running time?
- if min-priority queue is implemented by Fibonacci heap, the running time can be improved to $O(m + n \log n)$

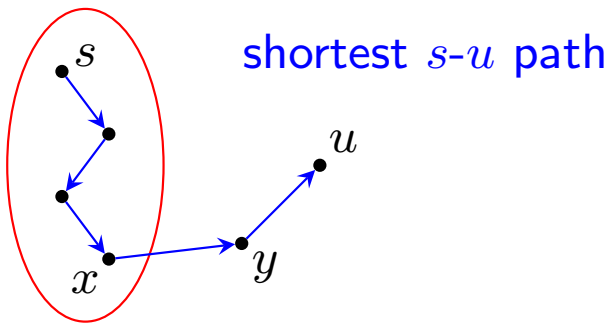
Correctness

Theorem. Whenever u is removed from V^*, we have $d(u) = \delta(u)$.

Proof

assume u is removed from V^* with $d(u) > \delta(u)$ for the first time

- $\delta(u) < d(u)$ implies u is reachable from s



$$A = \{v \in V \mid d(v) = \delta(v)\}$$

- $d(u) > \delta(u) \geq \delta(x) = d(x)$ (nonnegative costs) $\Rightarrow$ x is removed before u
  the algorithm does not remove wrong nodes before removing u
- but $d(y) = \delta(y)$ after relaxing edge (x, y), contradicting with $y \notin A$