Single Source Shortest Paths – Bellman Ford

Data Structures and Algorithms (094224)

Tutorial 10

Winter 2022/23

1. From Lecture

2. Questions

Distances revisited

- Digraph $G = (V, E)$
- Edge *weight* ⟨משקל⟩ function $w : E \to \mathbb{R}$
  - Associates a real weight $w(e)$ with each edge $e \in E$
  - G is called a *weighted* ⟨ממושקל⟩ digraph
- Weight of path $P = \langle v_0, v_1, \ldots, v_k \rangle$ is $w(P) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$
  - A.k.a. (weighted) length of P
- Redefine the distance from $u \in V$ to $v \in V$ w.r.t. w:

$$\delta(u, v) = \begin{cases} \min\left\{ w(P) : u \overset{P}{\rightsquigarrow} v \right\}, & u \rightsquigarrow v \\ \infty, & u \not\rightsquigarrow v \end{cases}$$

  - $u \rightsquigarrow v$ denotes v reachable from u
  - $u \overset{P}{\rightsquigarrow} v$ denotes path P from u to v
- P is a shortest path from u to v if $w(P) = \delta(u, v)$

Distances revisited — cont.

- What if G admits a negative weight cycle C?
  - $\delta(u, v)$ is unbounded (from below) for every $u \rightsquigarrow C \rightsquigarrow v$
    - Can make the path arbitrarily short by including more traversals of C
  - Define $\delta(u, v) = -\infty$
- If G doesn't admit negative weight cycles, then we can restrict our attention to simple shortest paths

Lemma (subpaths of shortest paths)

Consider a shortest path $P = \langle v_0, v_1, \ldots, v_k \rangle$. Then, $P_{i,j} = \langle v_i, \ldots, v_j \rangle$ is a shortest path for every $0 \leq i \leq j \leq k$.

The SSSP problem — Representing shortest paths

- The Single Source Shortest Path (SSSP) problem
- Designated source vertex $s \in V$
- Goal: compute the distances and shortest paths from s to all vertices reachable from s
  - BFS does so for unweighted (di)graphs
- Constructs a shortest paths tree
  - Tree rooted at s
  - Contains all vertices reachable from s
  - Unique simple (s, v)-path in the tree is a shortest (s, v)-path in G

The structure of the algorithms

- The SSSP algorithms we will see have many similarities
- Additional attributes for each vertex $v \in V$:
  - $v.d$ = the distance from s to v in G
  - $v.\pi$ = the parent of v in the shortest paths tree
- The (directed) predecessor subgraph $G_\pi = (V_\pi, E_\pi)$
  - $V_\pi = \{v \in V \mid v.\pi \neq NIL\} \cup \{s\}$
  - $E_\pi = \{(v.\pi, v) \in E \mid v \in V_\pi - \{s\}\}$
  - At termination: (its undirected version is) a shortest paths tree for s
- Algorithm's structure:
  1. Initialize the d and π fields by calling procedure `Initialize_Single_Source`
  2. Update the d and π fields through a sequence of calls to procedure `Relax`

The initialization and relax procedures

Initialize the d and π fields in G

`Initialize_Single_Source`(G, s)

for all $v \in G.V$ **do**
  $v.d = \infty$
  $v.\pi = NIL$
$s.d = 0$

Try to improve the shortest path to v through the edge (u, v)

`Relax`(u, v, w)

if $v.d > u.d + w(u, v)$ **then**
  $v.d = u.d + w(u, v)$
  $v.\pi = u$

Lemmas

Lemma (triangle inequality)

For every edge $(u, v) \in E$, we have $\delta(s, v) \leq \delta(s, u) + w(u, v)$.

Lemma (monotonicity)

The value of $v.d$ is non-increasing over time.

Lemma (upper bound)

$v.d \geq \delta(s, v)$ at all times.

Lemma (no path)

If v is not reachable from s, then $v.d = \infty$ at all times.

Lemmas — cont.

Lemma (path relaxation)

Consider some path $P = \langle s = v_0, v_1, \ldots, v_k \rangle$*. If the sequence of calls to* `Relax` *includes as a subsequence*[a] *the calls* $\texttt{Relax}(v_0, v_1, w), \texttt{Relax}(v_1, v_2, w), \ldots, \texttt{Relax}(v_{k-1}, v_k, w)$*, then* $v_k.d \leq w(P)$ *at all times after this subsequence.*

[a] Not necessarily contiguous.

Lemma (shortest path relaxation)

Consider some shortest path $P = \langle s = v_0, v_1, \ldots, v_k \rangle$*. If the sequence of calls to* `Relax` *includes as a subsequence*[a] *the calls* $\texttt{Relax}(v_0, v_1, w), \texttt{Relax}(v_1, v_2, w), \ldots, \texttt{Relax}(v_{k-1}, v_k, w)$*, then* $v_k.d = \delta(s, v_k)$ *at all times after this subsequence.*

[a] Not necessarily contiguous.

The Bellman-Ford algorithm

- Input:
  - Digraph $G = (V, E)$
  - Weight function $w : E \to \mathbb{R}$
  - Source vertex $s \in V$
- Outputs an error message if G admits a negative weight cycle reachable from s
- Computes (if no error):
  - $\delta(s, v)$ for each vertex $v \in V$
  - Shortest paths tree rooted at s

Pseudocode

```
Bellman_Ford(G, w, s)
Initialize_Single_Source(G, s)
for i = 1, ..., |G.V| − 1 do
    for all (u, v) ∈ G.E do
        Relax(u, v, w)
for all (u, v) ∈ G.E do
    if v.d > u.d + w(u, v) then
        error: "negative weight cycle"
```

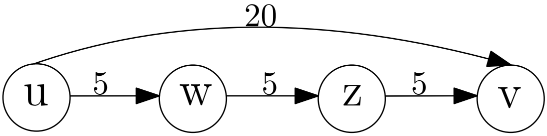
Question 1

Let $G = (V, E)$ be a weighted directed graph with weight function $w : E \to \mathbb{R}$. Let $\alpha \in \mathbb{R}_{\geq 0}$ and define two new weight functions on the edges $c_1 : E \to \mathbb{R}$ and $c_2 : E \to \mathbb{R}$ such that for every $e \in E$, $c_1(e) = w(e) + \alpha$ and $c_2(e) = \alpha w(e)$.
Let $u, v \in V$ and P a (u, v)-path.
Prove/Disprove:

1. If P is a shortest (u, v)-path w.r.t. w, then P is a shortest (u, v)-path w.r.t. c_1
2. If P is a shortest (u, v)-path w.r.t. w, then P is a shortest (u, v)-path w.r.t. c_2

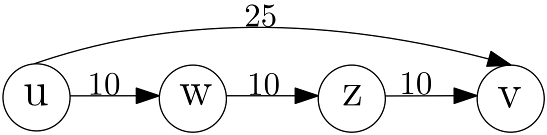
1.1) the claim is false

- The graph G with weight function w



  - $P = \langle u, w, z, v\rangle$ is a shortest (u, v)-path w.r.t. w

- The graph G with weight function c_1 for $\alpha = 5$



  - $P = \langle u, w, z, v\rangle$ is not a shortest (u, v)-path w.r.t. c_1

Solution — cont.

1.2) the claim is true

- Let $P = \langle u = z_0, \ldots, z_k = v \rangle$ be a shortest (u, v)-path w.r.t. w
- If $\alpha = 0$, the claim holds so assume $\alpha > 0$
- Assume by contradiction that P is not a shortest (u, v)-path w.r.t. c_2
- In that case, there exists a (u, v)-path $P' = \langle u = t_0, \ldots, t_{k'} = v \rangle$ such that $c_2(P) > c_2(P')$.
  - P is a shortest (u, v)-path w.r.t. w, thus a negative weight C such that $u \rightsquigarrow C \rightsquigarrow v$ does not exist
  - Since $\alpha \geq 0$, a negative weight cycle w.r.t. c_2 on some path from u to v does not exist
- $c_2(P) = \sum_{i=0}^{k-1} c_2(z_i, z_{i+1}) = \sum_{i=0}^{k-1} \alpha w(z_i, z_{i+1}) = \alpha w(P)$
- $c_2(P') = \sum_{i=0}^{k'-1} c_2(t_i, t_{i+1}) = \sum_{i=0}^{k'-1} \alpha w(t_i, t_{i+1}) = \alpha w(P')$
- Thus, $\alpha w(P) > \alpha w(P') \implies w(P) > w(P')$ $_{(\rightarrow\leftarrow)}$
  - P is a shortest (u, v)-path w.r.t. w

Question 2

Consider a weighted, directed graph $G = (V, E, w)$ with no negative-weight cycles and a source node $s \in V$. Let $R(s) = \{v \in V \mid s \leadsto v\}$ be the set of nodes reachable from s in G, and let $\mu(v)$ be the minimum number of edges in a shortest path (w.r.t. weight) from s to v for each $v \in R(s)$.
Define $\gamma = \max_{v \in R(s)} \mu(v)$. Suggest a simple change to the Bellman-Ford algorithm that allows it to terminate in $\gamma + 1$ passes on the edges, even if γ is not known in advance.

Solution:

- There exists a shortest path from the source s to every $v \in R(s)$ with at most γ edges
  - G has no negative-weight cycles
- By shortest path relaxation property, after at most γ iterations of relaxing the edges of G, it holds that $v.d = \delta(s, v)$ for all $v \in V$, and $v.d$ does not change afterwards

Solution — cont.

Bellman_Ford_Gamma(G, w, s)

```
1: Initialize_Single_Source(G, s)
2: change = true
3: while change == true do
4:     change = false
5:     for all (u, v) ∈ G.E do
6:         v.d_old = v.d
7:         Relax(u, v, w)
8:         if v.d ≠ v.d_old then
9:             change = true
```

Run time analysis:

- Initialization takes $O(V)$ time
- The while loop in lines 3-9 will end after $\gamma + 1$ iterations – based on previous discussion
- In total – $O(V + \gamma \cdot E)$

Question 3

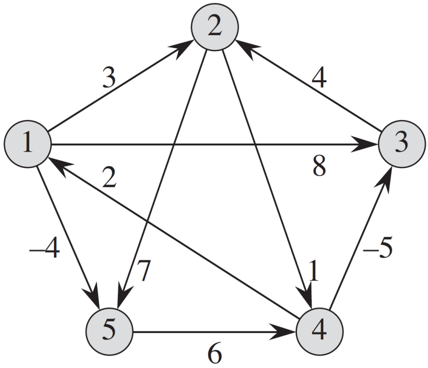
Let $G = (V, E)$ be a weighted directed graph with weight function $w : E \to \mathbb{R}$. Assume that G has no negative weight cycles. Give an $O(VE)$-time algorithm to find, for each vertex $v \in V$, the value $\delta^*(v) = \min_{u \in V}\{\delta(u, v)\}$.

Solution

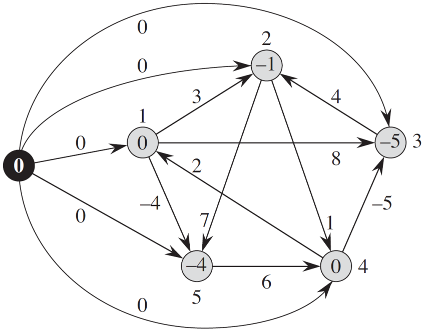
- The solution that will be presented will use a reduction
- Notice, $\delta^*(v) \leq 0$ for all $v \in V$
  - Since $\delta(v, v) = 0$ for all $v \in V$
- Define $G' = (V', E', w')$ where
  - $V' = V \cup \{s\}$ for some new vertex $s \notin V$
  - $E' = E \cup \{(s, v) \mid v \in V\}$
  - $w'(s, v) = 0$ for all $v \in V$
  - $w'(u, v) = w(u, v)$ for all $(u, v) \in E$

Solution — cont.

$G = (V, E)$:



$G' = (V', E')$: vertex s is black



Solution — cont.

Lemma

For every $v \in V$ it holds that $\delta^*(v) = \delta_{G'}(s, v)$

Proof.

- Let $v \in V$ be some node and $u = \arg\min_{u \in V}\{\delta_G(u, v)\}$
  - Exists since G has no negative weight cycles
- Let $P\langle u, \dots, v\rangle$ be a shortest (u, v)-path, thus $w(P) = \delta^*(v)$
- $\delta_{G'}(s, v) \leq \delta^*(v)$
  - Let $P' = s \to u \overset{P}{\rightsquigarrow} v$ a path in G'
  - $\delta_{G'}(s, v) \leq w'(P') = w'(s, u) + w'(P) = w'(P) = w(P) = \delta^*(v)$
    - Since P does not contain s
- $\delta^*(v) \leq \delta_{G'}(s, v)$
  - Let $P' = \langle s, z, \dots, v\rangle$ be a shortest (s, v)-path in G'
  - Let Q be the sub-path of P' from z to v
  - $\delta_{G'}(s, v) = w'(P') = w'(s, z) + w'(Q) = w(Q) \geq \delta^*(v)$
    - Last inequality follows from definition of $\delta^*(v)$

Solution — cont.

Algorithm: `Delta_Star`(G, w)

- Input: Digraph G with no negative-weight cycles and a weight function w
- Output: $\delta^*(v)$ for every $v \in V$
  1. Build G' and run `Bellman_Ford`(G', w', s)
  2. for every $v \in V' - \{s\}$ output $v.d$

Correctness:

- From previous lemma ($\delta_{G'}(s, v) = \delta^*(v)$) and the correctness of `Bellman_Ford`(G', w', s)

Solution — cont.

Run time analysis:

- Let $n = |V|$, $m = |E|$ and notice that $|V'| = n + 1$, $|E'| = m + n$
- Building G' – $O(n + 1 + m + n) = O(m + n)$
- `Bellman_Ford`(G', w', s) – $\Theta(nm + n^2 + m + n) = \Theta(nm + n^2)$
- Problematic in cases where $m = o(n)$
  - Runtime of our algorithm in this case is $\Omega(n^2)$ instead of $O(nm)$
- Easily fixable in various ways
- Examples:
  1. Use `Bellman_Ford_Gamma`(G', w', s) from Question 2 (instead of `Bellman_Ford`(G', w', s))
     - Meets runtime requirements since $\gamma \leq \min\{n - 1, m\}$
  2. Start with preprocessing where we output $\delta^*(v) = 0$ for all isolated nodes $v \in V$ (i.e., nodes with no incident edges) and remove them from G
     - Resulting G' has $O(m)$ edges

Question 4

Modify the Bellman-Ford algorithm so that it sets $v.d$ to $-\infty$ for all vertices v for which there is a negative-weight cycle on some path from the source to v.

Solution

- The vertices for which there is a negative-weight cycle on some path from the source are the vertices reachable from a vertex on a negative weight cycle
- The for loop in line 5 of `Bellman_Ford` checks for the existence of a negative-weight cycle reachable from the source s
- If a negative-weight cycle reachable from s exists in G, then the if condition in line 6 will be satisfied and the vertices v and u are vertices in a negative-weight cycle
- How can we find every vertex that is reachable from v?
    - We can do so using, e.g., $\texttt{BFS}(G, v)$
- `Bellman_Ford` halts when the if condition in line 6 is satisfied
- By modifying it to continue the search for a negative-weight cycle we can find a vertex in **every** negative-weight cycle reachable from s in G

Solution — cont.

- Let C be the set of vertices that are in a negative weight cycle reachable from s
- How can we find all the vertices reachable from the vertices in C by running BFS once?
- Define $G' = (V', E')$ where
  - $V' = V \cup \{z\}$ for some new vertex $z \notin V$
  - $E' = E \cup \{(z, v) \mid v \in C\}$
- Running $\texttt{BFS}(G', z)$ will output all reachable vertices from vertices in C

Solution — cont.

Algorithm (Not a proper pseudocode):

- Input: Directed Graph G, a weight function w and a source vertex $s \in V$
- Output: $v.d = -\infty$ for all vertices which there is a negative weight cycle on some path from s to v
  1. Run the aforementioned modified version of $\texttt{Bellman\_Ford}(G, w, s)$ and return C (at least one vertex for each negative weight cycle in G)
  2. Build G' and run $\texttt{BFS}(G', z)$
  3. For every $v \in V$ that is reachable from z set $v.d = -\infty$

Correctness:

- Directly from the correctness of $\texttt{Bellman\_Ford}(G, w, s)$ and BFS

Run time analysis:

- $\texttt{Bellman\_Ford}(G, w, s)$ – $O(VE)$
- Building G' – $O(V' + E') = O(V + 1 + E + V) = O(V + E)$
- $\texttt{BFS}(G', z)$ – $O(V' + E') = O(V + E)$
- In total – $O(VE)$

Question 5

- *Arbitrage* is the use of discrepancies in currency exchange to transform one unit of a currency into more than one unit of the same currency
- Suppose:
  - 1\$ buys 0.87 €
  - 1 € buys 5 NIS
  - 1 NIS buys 0.25 \$
- In this example, one can yield a profit by converting currencies
- Start with 1\$ and buy $0.87 \times 5 \times 0.25 = 1.0875\$$

Given n currencies $a_1, \cdots, a_n$ and an $n \times n$ matrix A of exchange rates such that one unit of currency a_i buys $A_{i,j}$ units of currency a_j, propose an efficient algorithm to determine whether or not there exists an arbitrage in A.

Solution

- Main Idea:
  - Solved using reduction to the single source shortest path problem
- In order to find an arbitrage in A we need to find a sequence of currencies $\langle a_{i_1}, \cdots, a_{i_k}\rangle$, $1 \leq k \leq n$
  - $A_{i_1,i_2} \cdot A_{i_2,i_3} \cdots A_{i_{k-1},i_k} \cdot A_{i_k,i_1} > 1$
  - $\iff \log(A_{i_1,i_2} \cdot A_{i_2,i_3} \cdots A_{i_{k-1},i_k} \cdot A_{i_k,i_1}) > 0$
  - $\iff \sum_{j=1}^{k-1} \log(A_{i_j,i_{j+1}}) + \log(A_{i_k,i_1}) > 0$
  - $\iff -\sum_{j=1}^{k-1} \log(A_{i_j,i_{j+1}}) - \log(A_{i_k,i_1}) < 0$
- Reduction:
  - Represent each currency with a vertex. $V = \{a_1, \cdots, a_n\}$, $|V| = n$
  - $E = \{(a_i, a_j) \,|\, 1 \leq i \leq n, 1 \leq j \leq n\}$, $|E| = n^2$
  - $w(a_i, a_j) = -\log(A_{i,j})$
  - From A we can build $G = (V, E)$ and w
- There exist an arbitrage in A iff there exists a negative weight cycle

Solution — cont.

Algorithm (Not a proper pseudocode):

- Input: Exchange rates matrix A of size $n \times n$
- Output: "yes" if an arbitrage exists in A, otherwise "no"
  1. Build G and w (as in previous slide) from A
  2. Pick some arbitrary vertex $s \in V$ and run $\texttt{Bellman\_Ford}(G, w, s)$
  3. If $\texttt{Bellman\_Ford}(G, w, s)$ outputs an error message return "yes" otherwise return "no"

Why can we pick any arbitrary vertex $s \in V$ as the source vertex?

- $\texttt{Bellman\_Ford}(G, w, s)$ returns an error message iff there is a negative weight cycle reachable from s
- Can't we pick the a wrong vertex s and miss a negative weight cycle?
- According to the way G was constructed, if a negative cycle exists in G it is reachable from s
  - Since every vertex $v \in V$ is reachable from s

Solution — cont.

Correctness:

- Immediate from our discussion

Run Time:

- Building G from A – $O(n^2)$
- $\texttt{Bellman\_Ford}(G, w, s)$ – $O(VE) = O(n^3)$
- In total – $O(n^3)$