Single Source Shortest Paths – Dijkstra

Data Structures and Algorithms (094224)

Tutorial 11

Winter 2022/23

The Dijkstra algorithm

- Input:
  - Digraph $G = (V, E)$
  - Weight function $w : E \to \mathbb{R}_{\geq 0}$
    - Note the non-negative edge weights
  - Source vertex $s \in V$
- Computes:
  - $\delta(s, v)$ for each vertex $v \in V$
  - Shortest paths tree rooted at s
- Works also for undirected graphs

Dijkstra – pseudocode

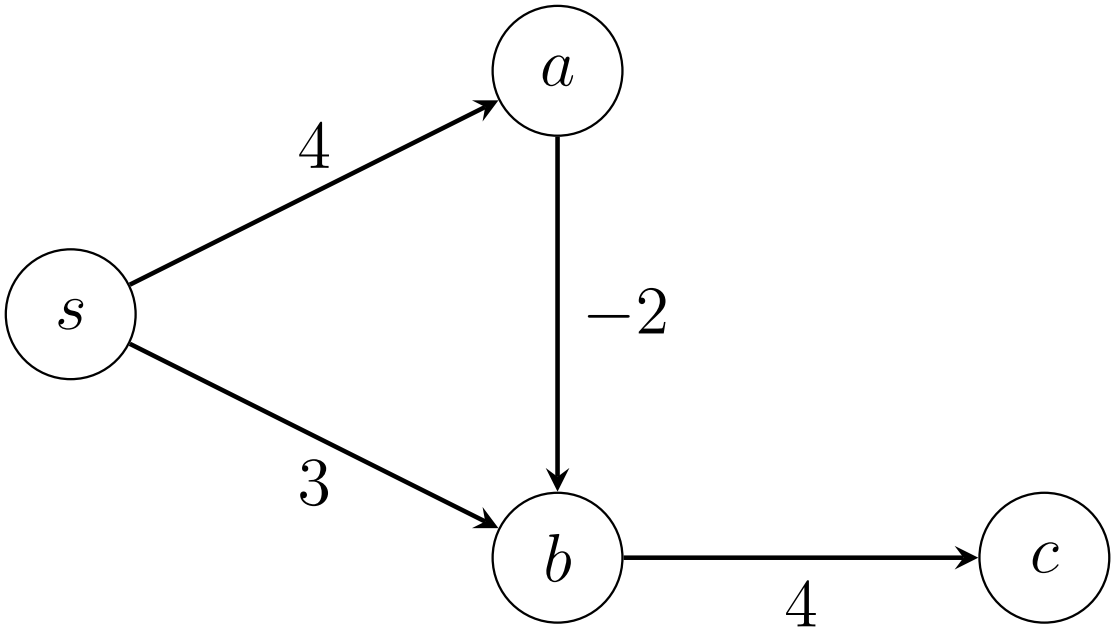
`Dijkstra`(G, w, s)

1: `Initialize_Single_Source`(G, s)
2: $Q = G.V$
3: **while** $Q \neq \emptyset$ **do**
4: $\quad u =$ `Extract_Min`(Q) ▷ minimum w.r.t. d
5: $\quad$ **for all** $v \in G.Adj[u]$ **do**
6: $\quad\quad$ `Relax`(u, v, w)

Question 1

Give a simple example of a directed graph with negative-weight edges for which Dijkstra's algorithm produces an incorrect output even if there is no negative-weight cycle.
Why doesn't the proof of Lemma (in page 43) from lecture go through when negative-weight edges are allowed?

Question 1 — Solution



Solution

	Q	u	$s.d$, $s.\pi$	$a.d$, $a.\pi$	$b.d$, $b.\pi$	$c.d$, $c.\pi$
Init	s, a, b, c	$--$	0, NIL	∞, NIL	∞, NIL	∞, NIL
1	a, b, c	s	0, NIL	4, s	3, s	∞, NIL
2	a, c	b	0, NIL	4, s	3, s	7, b
3	c	a	0, NIL	4, s	2, a	7, b
4	$\emptyset$	c	0, NIL	4, s	2, a	7, b

$c.d = 7$ while $\delta(s, c) = 6$

Solution — cont.

Consider the following Lemma from lecture:

Lemma

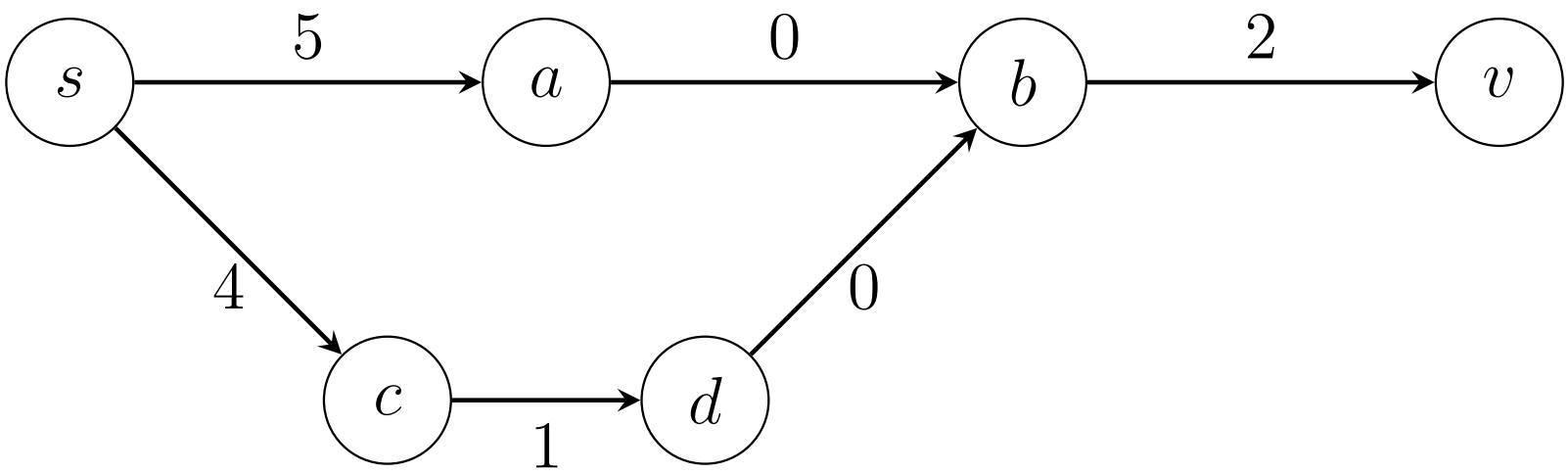
When vertex u is extracted from Q (line 4), it holds that $u.d = \delta(s, u)$.

- Recall the definitions of vertices u and y
  - u - assumed by contradiction that $u.d > \delta(s, u)$ and is the first (w.r.t time) that was extracted from Q with incorrect d value
  - y - the first vertex on a shortest (s, u)-path such that $y \in Q$
- y is on a shortest (s, u)-path $\implies \delta(s, y) \leq \delta(s, u)$
  - Not necessarily true if negative weight edges are allowed
  - Set $u = b$, shortest (s, b)-path is $\langle s, a, b \rangle$ and $y = a$

Question 2

Consider a directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}_{\geq 0}$ and a shortest (s, v)-path $P = \langle s, \ldots, v \rangle$ in G for two nodes $s, v \in V$. Prove/disprove: during `Dijkstra`(G, w, s) the edges of P are relaxed in the order they appear in P.

Solution

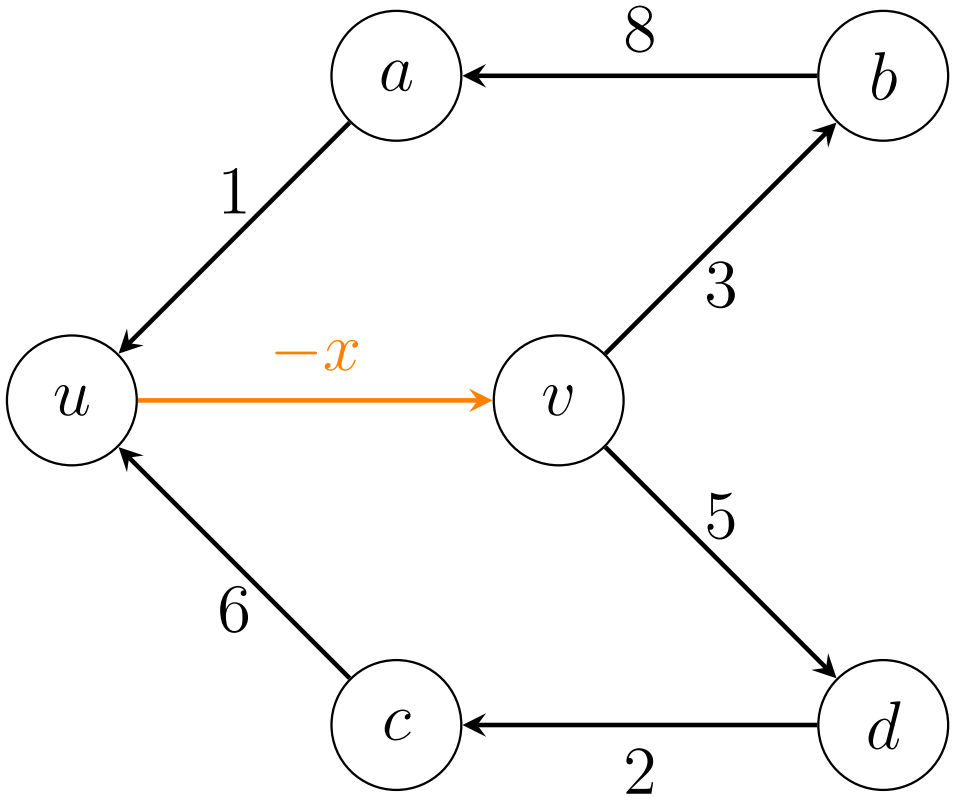


- Let $P = \langle s, a, b, v \rangle$
- P is a shortest (s, v)-path
- After $Q = \{a, d, b, v\}$, it may happen that vertex d is removed from Q
  - $a.d = d.d$
- After $Q = \{a, b, v\}$, it may happen that vertex b is removed from Q
  - $a.d = b.d$
- It may happen that edge (b, v) is relaxed before edge (a, b) in P

Question 3

Given a weighted directed graph $G = (V, E)$ with a weight function $w : E \to \mathbb{R}_{\geq 0}$ and an edge $(u, v) \in E$, describe an efficient algorithm that finds the maximum value $x \in \mathbb{R}_{\geq 0}$ such that if we update the weight of edge (u, v) to $(-x)$, no negative-weight cycles will be formed in G.

Example



Solution

Lemma

For every edge $(u, v) \in E$ the maximum value x such that no negative-weight cycles will be formed in G is $\delta(v, u)$

Proof.

- Updating $w(u, v)$ to $-\delta(v, u) \Longrightarrow$ no negative-weight cycle:
- Assume by contradiction there exists a negative-weight cycle in G after weight update
- Edge (u, v) is part of a negative weight cycle
  - Before weight change no negative-weight cycle exists
- If a negative-weight cycle exists in G then a simple negative-weight cycle exists
- Let $C = \langle v, \ldots, u, v \rangle$ be a simple negative-weight cycle
- Let P be the (v, u)-path such that $C = \langle P, v \rangle$
- $w(C) = w(P) - \delta(v, u) \geq \delta(v, u) - \delta(v, u) \geq 0$ $_{(\rightarrow\leftarrow)}$

Solution

- $\delta(v, u)$ is the maximum value:
- Assume by contradiction there exists $y > \delta(v, u)$ such that updating the weight of edge (u, v) to $-y$ does not form a negative-weight cycle in G
- $\delta(v, u) < \infty$
  - Otherwise, y does not exists
- Let P be a shortest (v, u)-path and $C = \langle P, v \rangle$ a cycle
- $w(C) = w(P) - y = \delta(v, u) - y < 0$ $_{(\rightarrow\leftarrow)}$

Solution — cont.

`T11_Q3`(G, w, u, v)

1: `Dijkstra`(G, w, v)
2: return $u.d$ ▷ $u.d = \delta(v, u)$

Correctness:

- Follows directly from Lemma and the correctness of `Dijkstra`(G, w, v)

Run time analysis:

- `Dijkstra`(G, w, v) and constant time operation
- In total – $O(V \log V + E)$

Question 4 – Moed B Winter20-21

Given a directed acyclic graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$ and source node $s \in V$, describe an efficient algorithm that computes $\delta(s, v)$ for each node $v \in V$.

Solution

- Naive solution: run `Bellman_Ford`(G, w, s)
  - Runtime is $\Theta(nm)$
  - Didn't use the fact that G is a DAG
- Can we use `Dijkstra`(G, w, s)?
  - Dijkstra doesn't work with negative edge-weights even when G is DAG
    - Observe: the Question 1 example for which Dijkstra fails is a DAG
  - In fact, we show a more efficient algorithm than Dijkstra

Solution – cont.

Reminder: topological sort of DAG $G = (V, E)$ is an ordering $u_1, \ldots, u_n$ of the nodes such that if $(u_i, u_j) \in E$, then $i < j$

- Can be computed in time $O(n + m)$ using DFS

`DAG_distances`(G, w, s)

1: `Initialize_Single_Source`(G, s)
2: Compute a topological sort $u_1, \ldots, u_n$ of G
3: **for all** $i = 1, \ldots, n$ **do**
4: **for all** $v \in G.Adj[u_i]$ **do**
5: `Relax`(u_i, v, w)

Run time analysis:

- `Initialize_Single_Source`(G, s) – $O(n)$ time
- Computing topological sort of G – $O(n + m)$
- Nested for loop – $O(n + m)$
- Total runtime – $O(n + m)$

Solution – cont.

Correctness: recall the following lemma from lecture

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.

- Consider some path $P = \langle s, v_1 \ldots, v_k \rangle$ in G
- By structure of topological sort and the construction of the algorithm, the subsequence $\texttt{Relax}(s, v_1, w), \ldots, \texttt{Relax}(v_{k-1}, v_k, w)$ occurs during the execution
- The subsequence occurs for any path P and thus for any shortest path
- $\implies$ $v.d = \delta(s, v)$ for all $v \in V$

Question 5 — All-Pairs Shortest Paths

Given a directed graph $G = (V, E)$ with a weight function $w : E \to \mathbb{R}$, describe an efficient algorithm that finds $\delta(u, v)$ for each pair of vertices $u, v \in V$.

All-Pairs Shortest Paths

	Eilat	Be'er-Sheba	Ashkelon	Jerusalem	Massada	Rehovot	Ben-Gurion Airport	Tel-Aviv	Netanya	Zikhron-Ya'akov	Afula	Nazareth	Tiberias	Haifa	Safed	Metula
Eilat		241	307	307	214	320	351	354	385	403	432	444	473	427	439	471
Be'er-Sheba	241		66	81	105	85	110	115	144	176	191	217	246	200	269	293
Ashkelon	307	66		71	166	48	49	49	78	107	139	151	180	134	200	255
Jerusalem	307	81	71		90	55	45	59	93	110	120	131	176	131	194	214
Massada	214	105	166	90		145	141	153	171	190	220	230	242	213	225	257
Rehovot	320	85	48	55	145		16	22	56	88	114	127	156	120	191	221
Ben-Gurion Airport	351	110	49	45	141	16		15	44	62	85	109	137	115	157	199
Tel-Aviv	354	115	49	59	153	22	15		29	61	90	102	131	95	162	195
Netanya	385	144	78	93	171	56	44	29		34	59	72	100	58	135	166
Zikhron-Ya'akov	403	176	107	110	190	88	62	61	34		44	40	75	21	106	140
Afula	432	191	139	120	220	114	85	90	59	44		12	41	48	72	103
Nazareth	444	217	151	131	230	127	109	102	72	40	12		29	37	70	91
Tiberias	473	246	180	176	242	156	137	131	100	75	41	29		57	35	62
Haifa	427	200	134	131	213	120	115	95	58	21	48	37	57		74	115
Safed	439	269	200	194	225	191	157	162	135	106	72	70	35	74		46
Metula	471	293	255	214	257	221	199	195	166	140	103	91	62	115	46	

Road Distances (In KM)

1 MILE=1.61; 1 KM=0.62 MILES

Solution

- We can run a single-source shortest-paths algorithm $|V|$ times, once for each vertex as the source
- If all edge weights are nonnegative, we can use Dijkstra's algorithm with a running time: $n \cdot O(n \log n + m) = O(n^2 \log n + nm)$
- Otherwise (negative weights are possible), we need to use Bellman-Ford with a running time: $n \cdot O(nm) = O(n^2 m)$

Can we do any better for real valued weight function?

Solution — cont.

Johnson's algorithm: general idea

- Check for negative weight cycle. If one exists output an error message.
- Compute a new set of non-negative edge weights, and run Dijkstra's algorithm once from each vertex.

Remark:

- The process of computing a new set of edge weights is called reweighing.

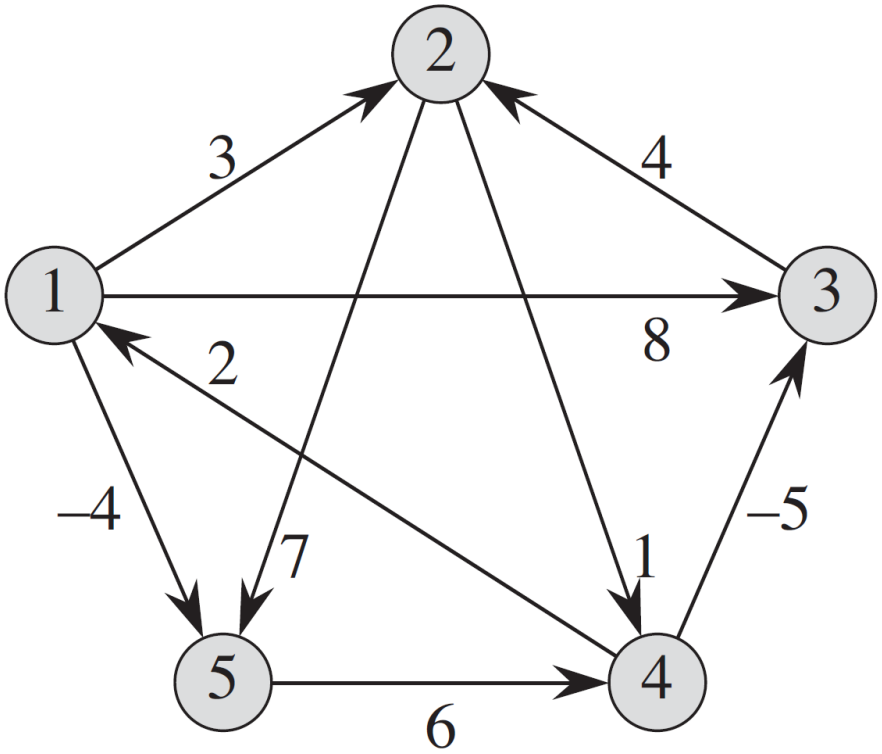
Solution — cont.

How can we check for a negative weight cycle in a graph?

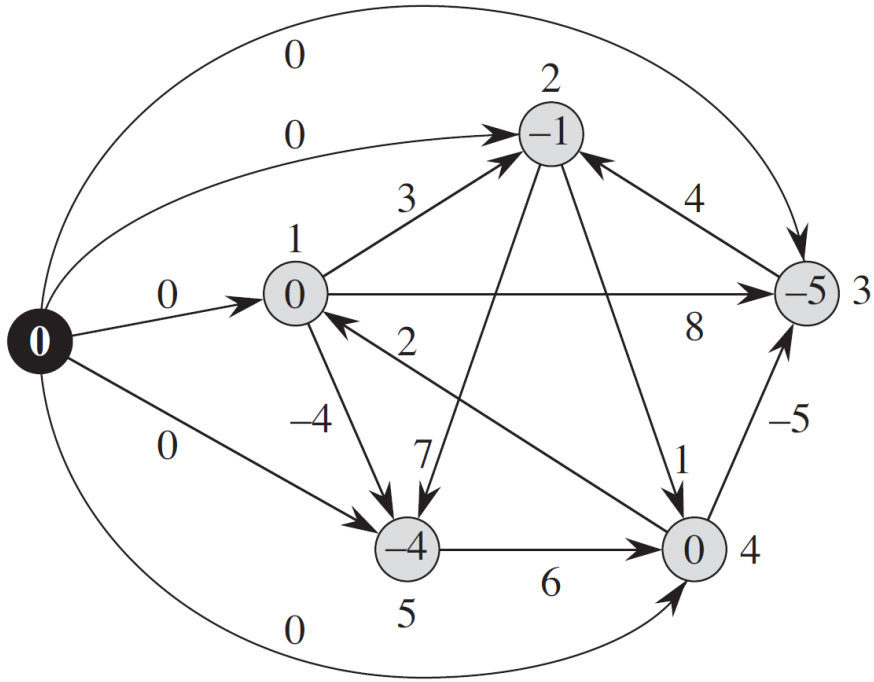
- We cannot pick some arbitrary vertex $s \in V$, since a negative weight cycle might not be reachable from s
- Build $G' = (V', E')$ where
  - $V' = V \cup \{s\}$ for some new vertex $s \notin V$
  - $E' = E \cup \{(s, v) \mid v \in V\}$
  - $w'(s, v) = 0$ for all $v \in V$
  - $w'(u, v) = w(u, v)$ for all $(u, v) \in E$
- Run `Bellman_Ford`(G', w', s)
- G has negative weight cycle iff G' has a negative weight cycle

Solution — cont.

$G = (V, E)$:



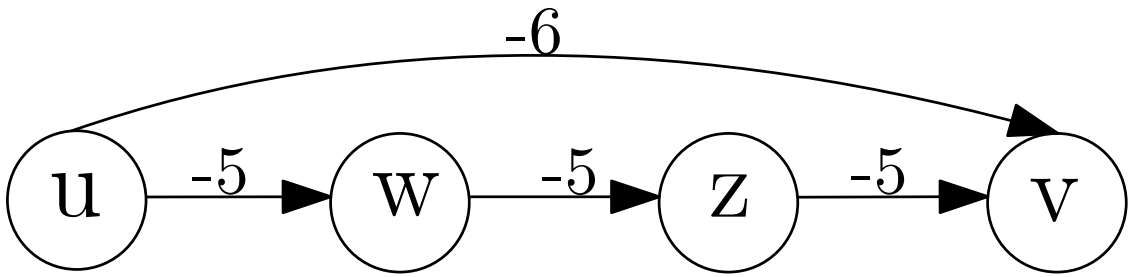
$G' = (V', E')$: vertex s is black



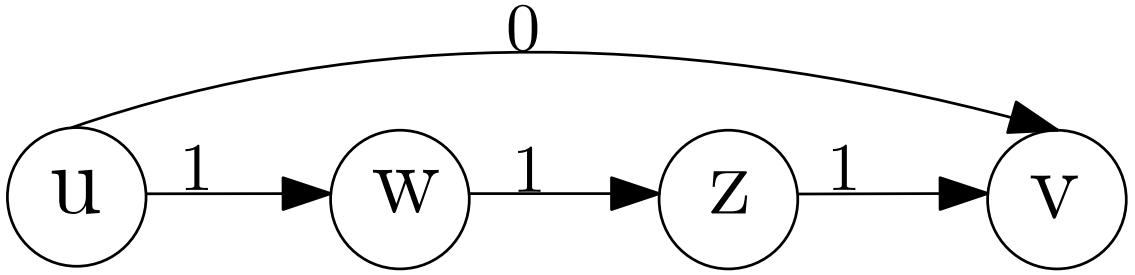
Solution — cont.

Reweighing

- Assuming no negative weight cycle, can't we simply add a value to each edge weight thus insuring no negative edge weights?
  - The graph G



    - $P = \langle u, w, z, v \rangle$ is a shortest (u, v)-path w.r.t. w
  - The graph G after $+6$ to every edge



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

Solution — cont.

What are the requirement of a transformed weight function, $\hat{w}$?

1. For all pairs of vertices $u, v \in V$, a path P is a shortest path from u to v using weight function w if and only if P is also a shortest path from u to v using weight function $\hat{w}$
2. For all edges $(u, v) \in E$ the new weight $\hat{w}(u, v)$ is non-negative

Solution — cont.

Reweighing

Let $\hat{\delta}(u, v)$ be the distance function w.r.t. weight function $\hat{w}$

Lemma : Reweighing does not change shortest paths

Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$, let $h : V \to \mathbb{R}$ be any function mapping vertices to real numbers. For each edge $(u, v) \in E$ define

$$\hat{w}(u, v) = w(u, v) + h(u) - h(v)$$

Let $P = \langle v_0, v_1, ..., v_k \rangle$ be a path from v_0 to v_k in G. Then $w(P) = \delta(v_0, v_k)$ iff $\hat{w}(P) = \hat{\delta}(v_0, v_k)$. That is, P is a shortest path from v_0 to v_k w.r.t. w iff P it is a shortest path w.r.t. $\hat{w}$.

Solution — cont.

Reweighing

Proof : a shortest path remains a shortest path

- Let $P = \langle v_0, v_1, ..., v_k \rangle$ be any path from v_0 to v_k
  - $\hat{w}(P) = \Sigma_{i=0}^{k-1} \hat{w}(v_i, v_{i+1}) =$
  - $= \Sigma_{i=0}^{k-1} [w(v_i, v_{i+1}) + h(v_i) - h(v_{i+1})] =$
  - $= w(P) + h(v_0) - h(v_k)$ (because the sum is telescope)
- Since $h(v_0) - h(v_k)$ is non-dependent on the path, all paths from v_0 to v_k are added the same constant $h(v_0) - h(v_k)$
- Therefore a shortest path under w is a shortest path under $\hat{w}$
- $w(P) = \delta(v_0, v_k)$ iff $\hat{w}(P) = \hat{\delta}(v_0, v_k)$

Solution — cont.

Reweighing

Constructing the weight function:

- We are left to find a function $h : V \to \mathbb{R}$ such that $\hat{w}(u, v) \geq 0$ for all $(u, v) \in E$
- Consider the graph G' constructed to check the existence of a negative weight cycles in G using Bellman-Ford
- Suppose G' has no negative weight cycle (implies that G has no negative weight cycles)
- Set $h(v) = \delta(s, v)$ for all $v \in V$
  - Recall the definition of $\delta^*(v)$ from Question 3 of Tutorial 10
    - $\delta^*(v) = \min_{u \in V}\{\delta(u, v)\} = \delta_{G'}(s, v)$
- By the triangle inequality, for every $(u, v) \in E$
  $h(v) \leq h(u) + w(u, v) \iff w(u, v) + h(u) - h(v) \geq 0$
- Reweighing $\hat{w}(u, v) = w(u, v) + h(u) - h(v)$ we conclude $\hat{w}(u, v) \geq 0$ for all $(u, v) \in E$

Solution — cont.

Algorithm (Not a proper pseudocode):

- Input: Directed Graph G and a weight function w
- Output: "error" if G has a negative weight cycle otherwise all pairs shortest paths
  1. Build G' and run Bellman–Ford
  2. If Bellman–Ford outputs "error" output "error"
  3. Otherwise, compute $\hat{w}$
  4. For every $u \in V$ run $\texttt{Dijkstra}(G, \hat{w}, u)$ to compute $\hat{\delta}(u, v), \forall v \in V$
  5. Output $\hat{\delta}(u, v) + h(v) - h(u)$ for all $u, v \in V$

Remark:

- Full pseudocode and example can be found in CLRS 3rd edition chapter 25.3

Solution — cont.

Correctness

- Immediate from our discussion

Run Time:

- Recall: the run time function is a function of the input. In our case the input is the graph $G = (V, E)$, $|V| = n$, $|E| = m$
- Build G' – $O(n + m)$
- `Bellman_Ford`(G', w', s) – $O((n + 1)(m + n)) = O(n^2 + nm)$
- Compute $\hat{w}$ – $O(m)$
- Running Dijkstra (on G) n times – $O(n^2 \log n + nm)$
- Compute $\delta(u, v)$ for every $u, v \in V$ – $O(n^2)$
- In total –
  $O(n + m) + O(n^2 + nm) + O(m) + O(n^2 \log n + nm) + O(n^2) =$
  $O(n^2 \log n + nm)$