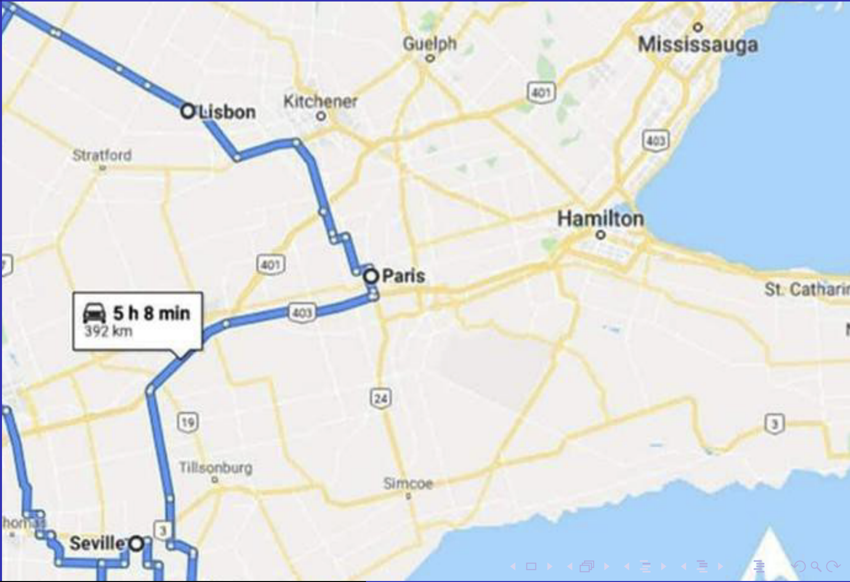
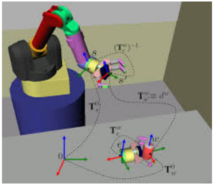
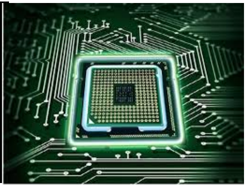
Shortest Paths in Digraphs



Myriad of applications

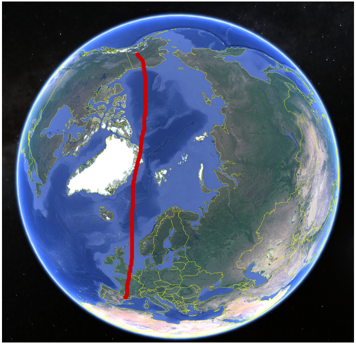
- Finding the shortest paths between 2 locations (Google maps, etc.)
- Internet router protocols: OSPF (Open Shortest Path First) is used to find a shortest path to interchange packages between servers (IP)
- Traffic information systems
- Routing in VSLI
- etc ...





Distance between two points

Distance is usually though as a pure geometric notion, often the Euclidean distance L_2
We use measures of distance that are not geometric: energy consumption, traveling time, payments, costs, etc..



Paths and weights

Given a digraph $G = (V, E)$ with edge's weights $w : E \to \mathbb{R}$.

Paths and weights

Given a digraph $G = (V, E)$ with edge's weights $w : E \to \mathbb{R}$.

- A path is a sequence of vertices $p = (v_0, \ldots, v_k)$ so that $(v_i, v_{i+1}) \in E$, for $0 \leq i < k$.

Paths and weights

Given a digraph $G = (V, E)$ with edge's weights $w : E \to \mathbb{R}$.

- A path is a sequence of vertices $p = (v_0, \ldots, v_k)$ so that $(v_i, v_{i+1}) \in E$, for $0 \leq i < k$.
- A path $p = (v_0, \ldots, v_k)$ has length $\ell(p) = k$ and weight $w(p) = \sum_{i=0}^{k-1} w(v_i, v_{i+1})$.

$$v_0 \xrightarrow{2} v_1 \xrightarrow{-1} v_2 \xrightarrow{-5} v_3 \xrightarrow{3} v_4$$

This path has length 4 and weight -1.

Paths and weights

Given a digraph $G = (V, E)$ with edge's weights $w : E \to \mathbb{R}$.

- A path is a sequence of vertices $p = (v_0, \dots, v_k)$ so that $(v_i, v_{i+1}) \in E$, for $0 \leq i < k$.
- A path $p = (v_0, \dots, v_k)$ has length $\ell(p) = k$ and weight $w(p) = \sum_{i=0}^{k-1} w(v_i, v_{i+1})$.

$$v_0 \xrightarrow{2} v_1 \xrightarrow{-1} v_2 \xrightarrow{-5} v_3 \xrightarrow{3} v_4$$

This path has length 4 and weight -1.

- For a path path $p = \{u, \dots, v\}$, we write $u \rightsquigarrow^p v$ to say that it starts at u and ends at v.

Paths and weights

Given a digraph $G = (V, E)$ with edge's weights $w : E \to \mathbb{R}$.

- A path is a sequence of vertices $p = (v_0, \dots, v_k)$ so that $(v_i, v_{i+1}) \in E$, for $0 \leq i < k$.
- A path $p = (v_0, \dots, v_k)$ has length $\ell(p) = k$ and weight $w(p) = \sum_{i=0}^{k-1} w(v_i, v_{i+1})$.

$$v_0 \xrightarrow{2} v_1 \xrightarrow{-1} v_2 \xrightarrow{-5} v_3 \xrightarrow{3} v_4$$

This path has length 4 and weight -1.

- For a path path $p = \{u, \dots, v\}$, we write $u \overset{p}{\rightsquigarrow} v$ to say that it starts at u and ends at v.
- Note that the definition of path allows repeated vertices

Distance

- We want to associate a distance value $\delta(u, v)$ to each pair of vertices u, v in a weighted digraph (G, w), measuring the minimum weight over the weights of the paths going from u to v.

Distance

- We want to associate a distance value $\delta(u, v)$ to each pair of vertices u, v in a weighted digraph (G, w), measuring the minimum weight over the weights of the paths going from u to v.
- We have two cases:

Distance

- We want to associate a distance value $\delta(u, v)$ to each pair of vertices u, v in a weighted digraph (G, w), measuring the minimum weight over the weights of the paths going from u to v.
- We have two cases:
  - $\{p | u \rightsquigarrow^{p} v\} = \emptyset$, i.e., there is no path from u to v, in such a case we define $\delta(u, v) = +\infty$.

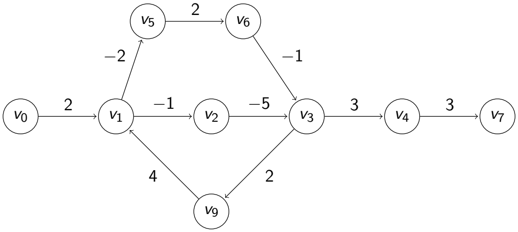
Distance

- We want to associate a distance value $\delta(u, v)$ to each pair of vertices u, v in a weighted digraph (G, w), measuring the minimum weight over the weights of the paths going from u to v.
- We have two cases:
  - $\{p | u \rightsquigarrow^p v\} = \emptyset$, i.e., there is no path from u to v, in such a case we define $\delta(u, v) = +\infty$.
  - $\{p | u \rightsquigarrow^p v\} \neq \emptyset$. In this case, if $\min\{w(p) | u \rightsquigarrow^p v\}$ exists, we define the distance as

$$\delta(u, v) = \min_p \{w(p) | u \rightsquigarrow^p v\}$$

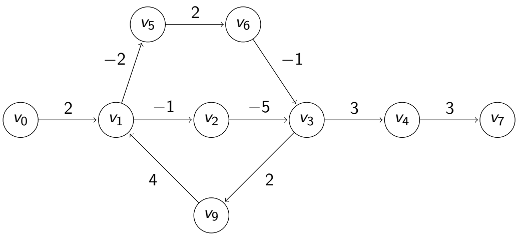
otherwise, the distance cannot be defined.

Distances: examples



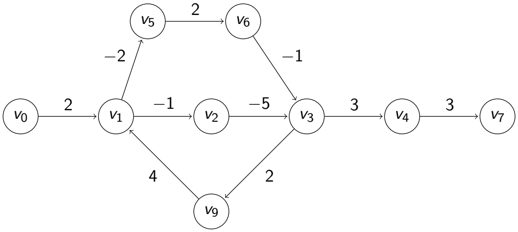
$\delta(v_4, v_7) =$

Distances: examples



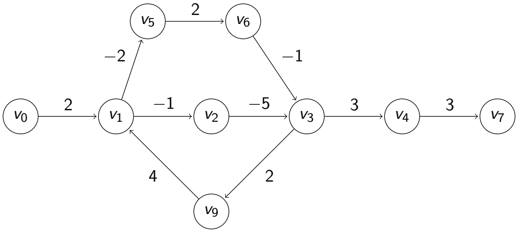
$\delta(v_4, v_7) = 3$

Distances: examples



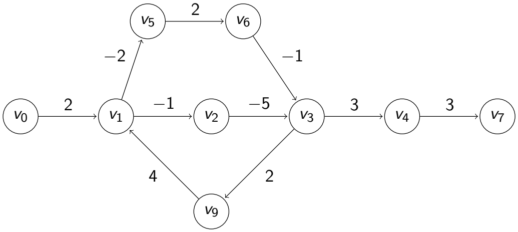
$\delta(v_4, v_7) = 3$ $\delta(v_4, v_3) =$

Distances: examples



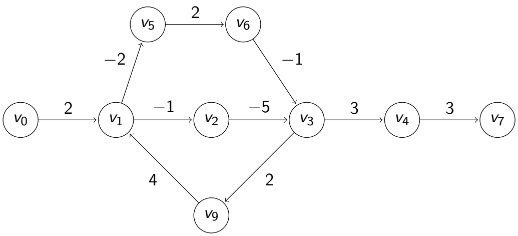
$\delta(v_4, v_7) = 3$ $\delta(v_4, v_3) = +\infty$

Distances: examples



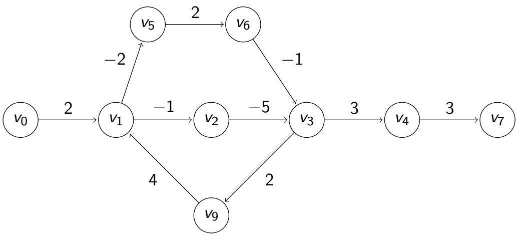
$\delta(v_4, v_7) = 3$ $\delta(v_4, v_3) = +\infty$ $\delta(v_3, v_2) =$

Distances: examples



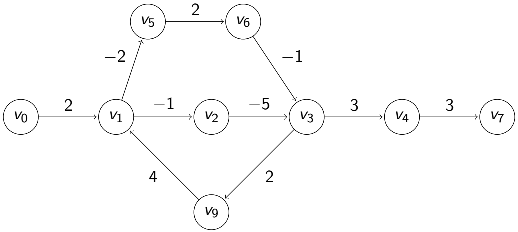
$\delta(v_4, v_7) = 3$ $\delta(v_4, v_3) = +\infty$ $\delta(v_3, v_2) = 5$

Distances: examples



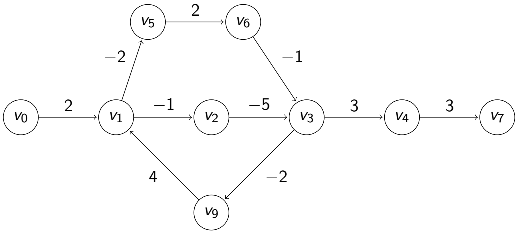
$\delta(v_4, v_7) = 3$ $\delta(v_4, v_3) = +\infty$ $\delta(v_3, v_2) = 5$ $\delta(v_0, v_4) =$

Distances: examples



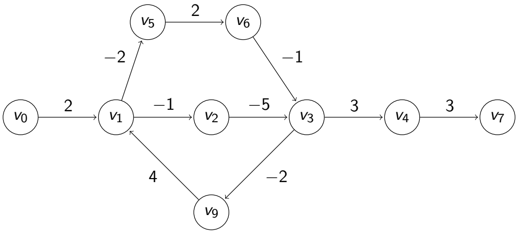
$\delta(v_4, v_7) = 3$ $\delta(v_4, v_3) = +\infty$ $\delta(v_3, v_2) = 5$ $\delta(v_0, v_4) =$ -1

Distances: examples



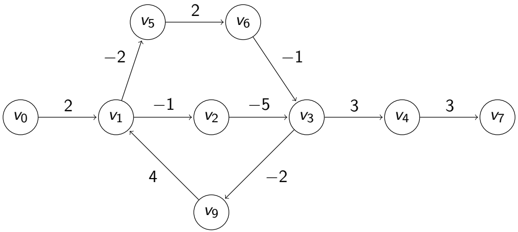
$\delta(v_4, v_7)$

Distances: examples



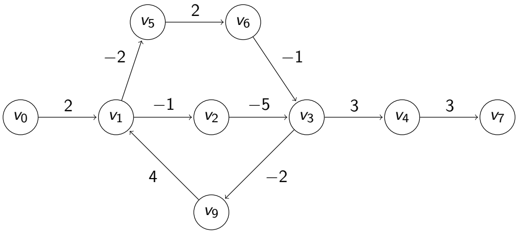
$\delta(v_4, v_7) = 3$

Distances: examples



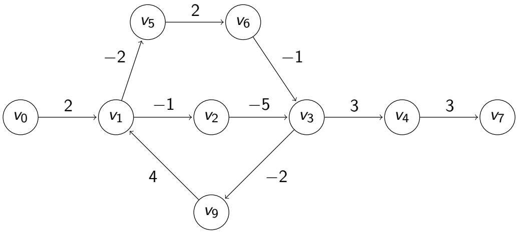
$\delta(v_4, v_7) = 3$ $\delta(v_4, v_3)$

Distances: examples



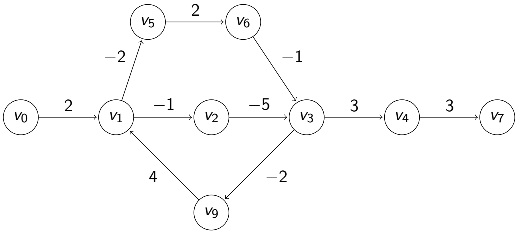
$\delta(v_4, v_7) = 3$ $\delta(v_4, v_3) = +\infty$

Distances: examples



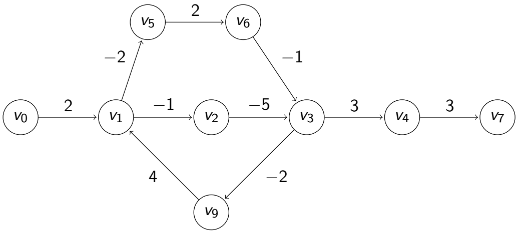
$\delta(v_4, v_7) = 3$ $\delta(v_4, v_3) = +\infty$ $\delta(v_3, v_2)$

Distances: examples



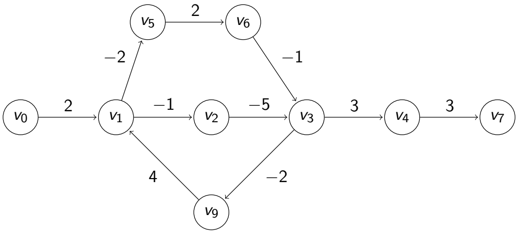
$\delta(v_4, v_7) = 3$ $\delta(v_4, v_3) = +\infty$ $\delta(v_3, v_2)$ cannot be defined

Distances: examples



$\delta(v_4, v_7) = 3$ $\delta(v_4, v_3) = +\infty$ $\delta(v_3, v_2)$ cannot be defined
$w(v_3, v_9, v_1, v_2) = 1$

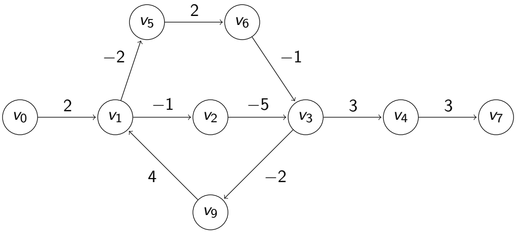
Distances: examples



$\delta(v_4, v_7) = 3$ $\delta(v_4, v_3) = +\infty$ $\delta(v_3, v_2)$ cannot be defined

$w(v_3, v_9, v_1, v_2) = 1$ $w(v_3, v_9, v_1, v_2, v_3, v_9, v_1, v_2) = -3$

Distances: examples

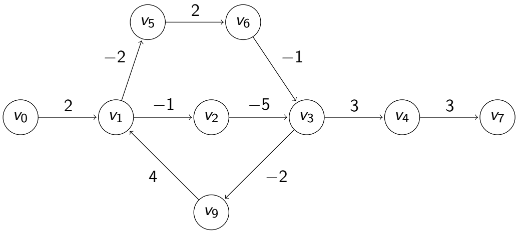


$\delta(v_4, v_7) = 3$ $\delta(v_4, v_3) = +\infty$ $\delta(v_3, v_2)$ cannot be defined

$w(v_3, v_9, v_1, v_2) = 1$ $w(v_3, v_9, v_1, v_2, v_3, v_9, v_1, v_2) = -3$

$w(v_3, v_9, v_1, v_2, v_3, v_9, v_1, v_2, v_3, v_9, v_1, v_2) = -7$

Distances: examples



$\delta(v_4, v_7) = 3$ $\delta(v_4, v_3) = +\infty$ $\delta(v_3, v_2)$ cannot be defined

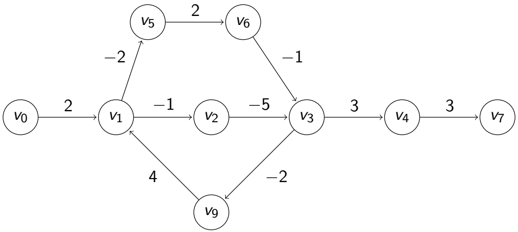
$w(v_3, v_9, v_1, v_2) = 1$ $w(v_3, v_9, v_1, v_2, v_3, v_9, v_1, v_2) = -3$

$w(v_3, v_9, v_1, v_2, v_3, v_9, v_1, v_2, v_3, v_9, v_1, v_2) = -7$

$w(v_3, v_9, v_1, v_2, v_3, v_9, v_1, v_2, v_3, v_9, v_1, v_2, v_3, v_9, v_1, v_2) = -11$

$\ldots$

Distances: examples



$\delta(v_4, v_7) = 3$ $\delta(v_4, v_3) = +\infty$ $\delta(v_3, v_2)$ cannot be defined

$w(v_3, v_9, v_1, v_2) = 1$ $w(v_3, v_9, v_1, v_2, v_3, v_9, v_1, v_2) = -3$

$w(v_3, v_9, v_1, v_2, v_3, v_9, v_1, v_2, v_3, v_9, v_1, v_2) = -7$

$w(v_3, v_9, v_1, v_2, v_3, v_9, v_1, v_2, v_3, v_9, v_1, v_2, v_3, v_9, v_1, v_2) = -11$

...

The cycle v_1, v_2, v_3, v_9, v_1 has weight -4!

When the distance cannot be defined?

A cycle is a path that starts and ends at the same vertex.
A negative weight cycle is a cycle c having $w(c) < 0$

When the distance cannot be defined?

A cycle is a path that starts and ends at the same vertex.
A negative weight cycle is a cycle c having $w(c) < 0$

Theorem

Let $G = (V, E, w)$ be a weighted digraph.
A distance among all pairs of vertices $u, v \in V(G)$ can be defined iff G has no negative weight cycles.

When the distance cannot be defined?

A cycle is a path that starts and ends at the same vertex.
A negative weight cycle is a cycle c having $w(c) < 0$

Theorem

Let $G = (V, E, w)$ be a weighted digraph.
A distance among all pairs of vertices $u, v \in V(G)$ can be defined iff G has no negative weight cycles.

Proof

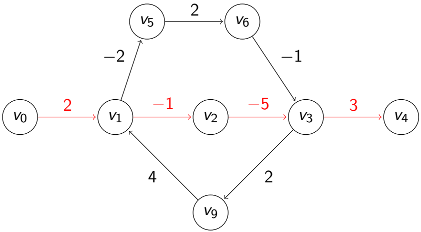
- If $\delta(u, v)$ can be defined, for every $u \in V$, $\delta(u, u) \geq 0$, so any cycle has non negative weight.
- If G has a negative weight cycle C, the distance among pairs of vertices in C cannot be defined.

When the distance cannot be defined?

- The previous theorem states conditions under which a distance measure for all pairs cannot be defined.
- It might be possible to have a digraph with a negative weight cycle, but that distances among some pairs of vertices can be defined, even if not for all pairs.

Shortest paths

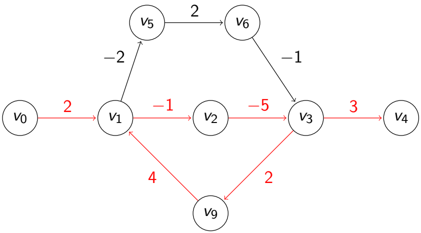
- For $u, v \in V$, such that $\delta(u, v)$ is defined and $\delta(u, v) < +\infty$,
- a shortest path from u to v is a path p, starting at u and ending at v, having $w(p) = \delta(u, v)$.



$$\delta(v_0, v_4) = -1$$

Shortest paths

- For $u, v \in V$, such that $\delta(u, v)$ is defined and $\delta(u, v) < +\infty$,
- a shortest path from u to v is a path p, starting at u and ending at v, having $w(p) = \delta(u, v)$.



$$\delta(v_0, v_4) = -1$$

There are infinite shortest paths $v_0 \rightsquigarrow v_4$

Undirected graphs and unweighted graphs

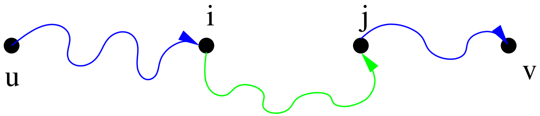
- If G is undirected, we consider every edge as doubly directed and assign the same weight to both directions.

Undirected graphs and unweighted graphs

- If G is undirected, we consider every edge as doubly directed and assign the same weight to both directions.
- If the graph or digraph is unweighted, we assign to each edge a weight of 1.
  In this case the weight of a path coincides with its length.

Optimal substructure of shortest path

Given $G = (V, E, w)$, for any shortest path $p : u \rightsquigarrow v$ and any pair of vertices i, j in p, the sub-path $p' = i \rightsquigarrow j$ of p is a shortest path, i.e., $w(p') = \delta(i, j)$.



Triangle Inequality

$\delta(u, v)$ is the shortest distance from u to v, i.e., the shortest path $u \rightsquigarrow v$ has weight $\leq$ that the weight of any other path from u and v.,

Triangle Inequality

$\delta(u, v)$ is the shortest distance from u to v, i.e., the shortest path $u \rightsquigarrow v$ has weight $\leq$ that the weight of any other path from u and v.,

Theorem

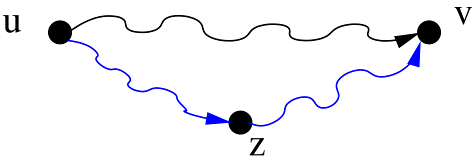
Let $G = (V, E, w)$ be such that, for each $u, v \in V$, $\delta(u, v)$ can be defined. For $u, v, z \in V(G)$, $\delta(u, v) \leq \delta(u, z) + \delta(z, v)$.

Triangle Inequality

$\delta(u, v)$ is the shortest distance from u to v, i.e., the shortest path $u \rightsquigarrow v$ has weight $\leq$ that the weight of any other path from u and v.,

Theorem

Let $G = (V, E, w)$ be such that, for each $u, v \in V$, $\delta(u, v)$ can be defined. For $u, v, z \in V(G)$, $\delta(u, v) \leq \delta(u, z) + \delta(z, v)$.

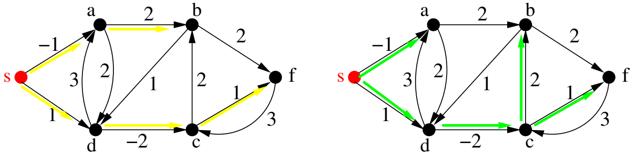


$u \rightsquigarrow z \rightsquigarrow v$ is a path from u to v.

Shortest Path Tree

Given $G = (V, E, w)$ and a distinguished $s \in V$, a shortest path tree is a directed sub-tree, $T_s = (V', E')$, of G, s.t.

- T_s is rooted at s,
- V' is the set of vertices in G reachable from s,
- For $v \in V'$ the path $s \rightsquigarrow v$ in T_s has weight $\delta(s, v)$.



Shortest paths problems

Single source shortest path: Given $G = (V, E, w)$ and $s \in V$, find a shortest path from s to each other vertex in G, if it exists.

To solve this problem we present two algorithms strategies,

- Dijkstra's algorithm: a very efficient greedy algorithm which only works for positive weights. You should know it.
- Bellman-Ford algorithm, devised by several independent teams Bellman, Ford, Moore, Shimbel. It works for general weights and detects whether the distance can be defined.

Both algorithms assume that the input graph is given by adjacency lists.

Shortest paths problems

All pairs shortest paths: Given $G = (V, E, w)$ without negative weight cycles, for each $u, v \in V(G)$, find a shortest path from u to v if it exists.

To solve this problem we present two algorithms strategies,

- Floyd-Warshall algorithm, devised by several independent teams Roy, Floyd, Warshall. Uses dynamic programming and takes as input the weighted adjacency matrix of G.
- Johnson's algorithm: an efficient algorithm for sparse graphs.

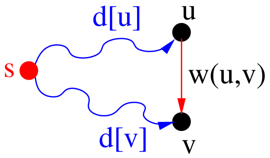
Single source shortest path (SSSP)

Given $G = (V, E, w)$ and $s \in V$, compute $\delta(s, v)$, for $v \in V - \{s\}$.

- The algorithms maintains, for $v \in V$, an overestimate $d[v]$ of $\delta(s, v)$ and a candidate predecessor $p[v]$ on a shortest path from s to v.
- Initially, $d[v] = +\infty$, for $v \in V - \{s\}$, $d[s] = 0$ and $p[v] = v$.
- Repeatedly improve estimates towards the goal $d[v] = \delta(s, v)$
- On selected $(u, v) \in E$ apply the Relax operation

Relaxing and edge

Relax(u, v)
if $d[v] > d[u] + w(u, v)$
then
$d[v] = d[u] + w(u, v)$
$p[v] = u$



Relaxing and edge

Relax: invariant

$d[v] \geq \delta(s, v)$ and, if $d[v] < +\infty$, $p[v]$ is the predecessor of v in a path from s to v with weight $d[v]$, .

Relaxing and edge

Relax: invariant

$d[v] \geq \delta(s, v)$ and, if $d[v] < +\infty$, $p[v]$ is the predecessor of v in a path from s to v with weight $d[v]$, .

Let d be the values before executing Relax and d' the ones after executing it.

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

$d'[v] = \min\{d[v], d[u] + w(u, v)\} \geq \delta(s, v)$.
The second part also follows from this formula.

SSSP: Dijkstra

Edsger .W.Dijkstra, *"A note on two problems in connexion with graphs"*. Num. Mathematik 1, (1959)

- Works only when $w(e) \geq 0$.
- Greedy algorithm, at each step for a vertex v, $d[v]$ becames $\delta(s, v)$ with correct distance
- Relax edges out of the actual vertex.
- Uses a priority queue Q

Recall: Dijkstra SSSP

Dijkstra(G, w, s)
Set $d[u] = +\infty$ and $p[u] = u$, $u \in V$.
$d[s] = 0$
$S = \emptyset$, Insert all the vertices in Q with key d
while $Q \neq \emptyset$ **do**
  $u =$EXT-MIN(Q)
  $S = S \cup \{u\}$
  for all $v \in Adj[u]$ and $v \notin S$ **do**
    Relax(u, v)
    change, if needed, the key of v in Q

Recall: Dijkstra SSSP

Theorem

Consider the set S at any point in the algorithm execution. For each $u \in S$, $d[u] = \delta(s, u)$

Proof

The proof is by induction on the size of $|S|$.

Recall: Dijkstra SSSP (correctness)

- For $|S| = 1$, $S = \{s\}$ and $d[s] = 0 = \delta(s, s)$.
- Assume that the statement is true for $|S| = k$ and that the next vertex selected by the algorithm in the ExtractMin is v.
  - Consider a s, v shortest path P, let y be the first vertex in P that does not belong to S and let $x \in S$ be the node just before y in P.
  - By induction hypothesis $d[x] = \delta(s, x)$
  - As P is a shortest path, the edge (x, y) has been relaxed with $d[x] = \delta(s, x)$, and $w \geq 0$, we get $\delta(s, y) = d[y] = d[x] + w(x, y) \leq \delta(s, v)$.
  - As the algorithm selected v, $d[v] \leq d[y]$, therefore $d[v] = \delta(s, v)$.

Recall: Dijkstra SSSP

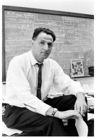
Theorem

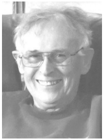
Using a priority queue Dijkstra's algorithm can be implemented on a graph with n nodes and m edges to run in $O(m)$ time plus the time for n ExtractMin and m ChangeKey operations.

Q implementation	Worst-time complexity
Heap	$O(m \lg n)$
Fibonacci heap	$O(m + n \lg n)$

SSSP: Bellman-Ford

Richard E. Bellman (1958)
Lester R. Ford Jr. (1956)
Edward F. Moore (1957)
Alfonso Shimbel (1955) (Shimbel matrices)



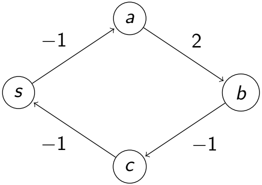



- The BF algorithm works for graphs with general weights.
- It detects the existence of negative cycles.

Bellman Ford Algorithm (BF)

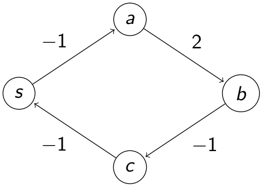
BF (G, w, s)
For $v \in V$, $d[v] = +\infty$, $p[v] = v$
$d[s] = 0$
for $i = 1$ to $n - 1$ **do**
  for all $(u, v) \in E$ **do**
    Relax(u, v)
for all $(u, v) \in E$ **do**
  if $d[v] > d[u] + w(u, v)$ **then**
    return Negative-weight cycle
return d, p

BF Algorithm: Example



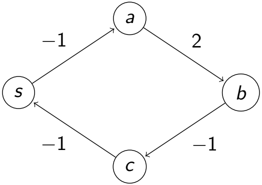
	s	a	b	c
0	0	$+\infty$	$+\infty$	$+\infty$

BF Algorithm: Example



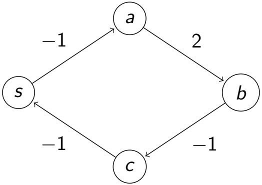
	s	a	b	c
0	0	$+\infty$	$+\infty$	$+\infty$
1	0	-1	$+\infty$	$+\infty$

BF Algorithm: Example



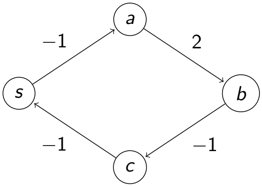
	s	a	b	c
0	0	$+\infty$	$+\infty$	$+\infty$
1	0	-1	$+\infty$	$+\infty$
2	0	-1	1	$+\infty$

BF Algorithm: Example



	s	a	b	c
0	0	$+\infty$	$+\infty$	$+\infty$
1	0	-1	$+\infty$	$+\infty$
2	0	-1	1	$+\infty$
3	0	-1	1	0

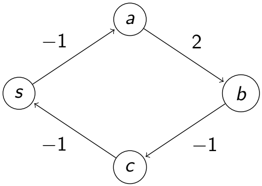
BF Algorithm: Example



	s	a	b	c
0	0	$+\infty$	$+\infty$	$+\infty$
1	0	-1	$+\infty$	$+\infty$
2	0	-1	1	$+\infty$
3	0	-1	1	0

$d[s] = 0$

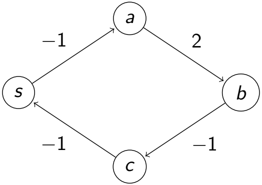
BF Algorithm: Example



	s	a	b	c
0	0	$+\infty$	$+\infty$	$+\infty$
1	0	-1	$+\infty$	$+\infty$
2	0	-1	1	$+\infty$
3	0	-1	1	0

$d[s] = 0$ but $d[c] + w(c, s) = -1$

BF Algorithm: Example

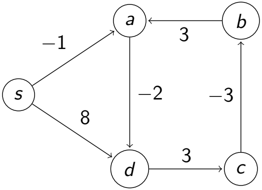


	s	a	b	c
0	0	$+\infty$	$+\infty$	$+\infty$
1	0	-1	$+\infty$	$+\infty$
2	0	-1	1	$+\infty$
3	0	-1	1	0

$d[s] = 0$ but $d[c] + w(c, s) = -1$

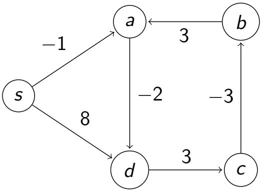
BF reports Negative cycle

BF Algorithm: Example



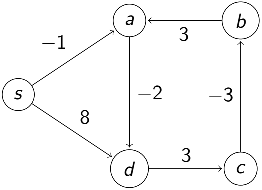
	s	a	b	c	d
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$

BF Algorithm: Example



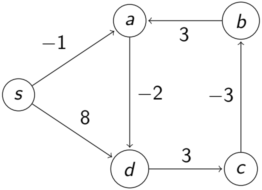
	s	a	b	c	d
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	0	-1	$+\infty$	$+\infty$	8

BF Algorithm: Example



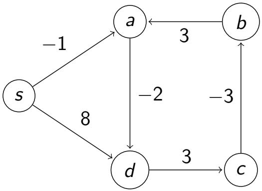
	s	a	b	c	d
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	0	-1	$+\infty$	$+\infty$	8
2	0	-1	$+\infty$	11	-3

BF Algorithm: Example



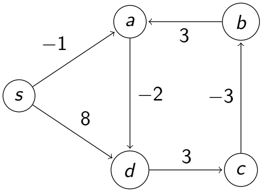
	s	a	b	c	d
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	0	-1	$+\infty$	$+\infty$	8
2	0	-1	$+\infty$	11	-3
3	0	-1	8	0	-3

BF Algorithm: Example



	s	a	b	c	d
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	0	-1	$+\infty$	$+\infty$	8
2	0	-1	$+\infty$	11	-3
3	0	-1	8	0	-3
4	0	-1	-3	0	-3

BF Algorithm: Example



	s	a	b	c	d
0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$
1	0	-1	$+\infty$	$+\infty$	8
2	0	-1	$+\infty$	11	-3
3	0	-1	8	0	-3
4	0	-1	-3	0	-3

d verifies the triangle inequality

Complexity of BF

BF (G, w, s)
Initialize $\forall v \neq s, d[v] = \infty,\ p[v] = v$
Initialize $d[s] = 0$
for $i = 1$ to $n - 1$ **do**
  for all $(u, v) \in E$ **do**
    Relax(u, v)
for all $(u, v) \in E$ **do**
  if $d[v] > d[u] + w(u, v)$ **then**
    return Negative-weight cycle
return $d,\ p$

$O(nm)$

Correctness of BF

Lemma

Consider the vector d *computed by BF at the end of the* i*-th iteration. For* $v \in V$, $d[v] \leq w(P)$ *for every path* P *such that* $s \leadsto^{P} v$ *and* $\ell(P) \leq i$.

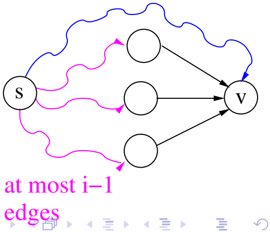
Correctness of BF

Lemma

Consider the vector d computed by BF at the end of the i-th iteration. For $v \in V$, $d[v] \leq w(P)$ for every path P such that $s \leadsto^P v$ and $\ell(P) \leq i$.

Proof (Induction on i)
Before the i-th iteration, $d[v] \leq \min\{w(p)\}$ over all paths p with at most $i - 1$ edges.

The i-th iteration considers all paths with $\leq i$ edges reaching v, when relaxing the last edge in such paths. $\square$



Correctness of BF

Theorem

If (G, w) has no negative weight cycles, BF computes correctly $\delta(s, v)$.

Correctness of BF

Theorem

If (G, w) has no negative weight cycles, BF computes correctly $\delta(s, v)$.

Proof

- Without negative-weight cycles, shortest paths are always simple (no repeated vertices), i.e., at most n vertices and $n-1$ edges.
- By the previous lemma, the $n-1$ iterations yield $d[v] \leq \delta(s, v)$.
- By the invariant of the relaxation algorithm $d[v] \geq \delta(s, v)$.

Correctness of BF

Theorem

BF reports "negative-weight cycle" iff there exists a negative weight cycle in G reachable from s.

Correctness of BF

Theorem

BF reports "negative-weight cycle" iff there exists a negative weight cycle in G reachable from s.

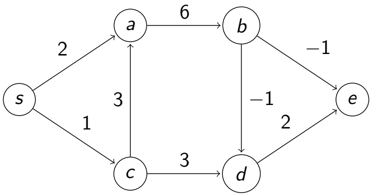
Proof

- Without negative-weight cycles in G, the previous theorem implies $d[v] = \delta(s, v)$, and by triangle inequality $d[v] \leq \delta(s, u) + w(u, v)$, so BF won't report a negative cycle if it doesn't exists.
- If there is a negative-weight cycle, then one of its edges can be relaxed, so BF will report correctly.

SSSP in a direct acyclic graphs (dags).

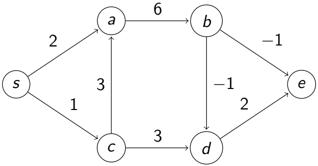
SSSP in DAG

Given an edge weighted dag $G = (V, E, w)$ and $s \in V$, find a shortest path from s to each other vertex in G, if it exists.



SSPP in a direct acyclic graphs (dags).

- A DAG has no cycles, so a distance can be defined among any pair of vertices.
- In particular there are shortest paths from s to any vertex v reachable from s.
- To obtain a faster algorithm we look for a good ordering of the edges: topological sort.



s, c, a, b, d, e

SSSP in a direct acyclic graphs (dags).

Let $Pre(v) = \{u \in V \mid (u, v) \in E\}$

SSSP-DAG(G, w)
Sort V in topologica order
For $v \in V$ set $d[v] = \infty$ and $p[v] = v$
$d[s] = 0.$
for all $v \in V - \{s\}$ in order **do**
$d[v] = \min_{u \in Pre(v)} \{d[u] + w_{uv}\}$
$p[v] = \arg\min_{u \in Pre[v]} \{d[u] + w_{uv}\}$

SSSP in a direct acyclic graphs (dags).

Let $Pre(v) = \{u \in V \mid (u, v) \in E\}$

SSSP-DAG(G, w)
Sort V in topologica order
For $v \in V$ set $d[v] = \infty$ and $p[v] = v$
$d[s] = 0$.
for all $v \in V - \{s\}$ in order **do**
$d[v] = \min_{u \in Pre(v)}\{d[u] + w_{uv}\}$
$p[v] = \arg\min_{u \in Pre[v]}\{d[u] + w_{uv}\}$

Complexity?

SSSP in a direct acyclic graphs (dags).

Let $Pre(v) = \{u \in V \mid (u, v) \in E\}$

SSSP-DAG(G, w)
Sort V in topologica order
For $v \in V$ set $d[v] = \infty$ and $p[v] = v$
$d[s] = 0.$
for all $v \in V - \{s\}$ in order **do**
$d[v] = \min_{u \in Pre(v)}\{d[u] + w_{uv}\}$
$p[v] = \arg\min_{u \in Pre[v]}\{d[u] + w_{uv}\}$

Complexity? $T(n) = O(n + m)$

SSSP in a direct acyclic graphs (dags).

Let $Pre(v) = \{u \in V \mid (u, v) \in E\}$

SSSP-DAG(G, w)
Sort V in topologica order
For $v \in V$ set $d[v] = \infty$ and $p[v] = v$
$d[s] = 0$.
for all $v \in V - \{s\}$ in order **do**
$d[v] = \min_{u \in Pre(v)}\{d[u] + w_{uv}\}$
$p[v] = \arg\min_{u \in Pre[v]}\{d[u] + w_{uv}\}$

Complexity? $T(n) = O(n + m)$

Correctness?

SSSP in a direct acyclic graphs (dags).

Let $Pre(v) = \{u \in V \mid (u,v) \in E\}$

SSSP-DAG(G, w)
Sort V in topologica order
For $v \in V$ set $d[v] = \infty$ and $p[v] = v$
$d[s] = 0$.
for all $v \in V - \{s\}$ in order **do**
$d[v] = \min_{u \in Pre(v)}\{d[u] + w_{uv}\}$
$p[v] = \arg\min_{u \in Pre[v]}\{d[u] + w_{uv}\}$

Complexity? $T(n) = O(n+m)$

Correctness? $d[u] = \delta(s,u)$, for $u \in Pre(v)$

All pairs shortest paths (APSP)

- Given $G = (V, E, w)$, $|V| = n, |E| = m$, we want to determine $\forall u, v \in V$, $\delta(u, v)$.
- We assume that G does not contain negative cycles.
- Naive idea: We apply n times BF or Dijkstra (if there are not negative weights)
- Repetition of BF: $O(n^2 m)$
- Repetition of Dijkstra: $O(nm \lg n)$ (if Q is implemented by a heap)

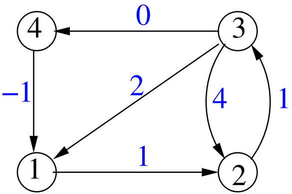
All pairs shortest paths: APSP

- Unlike in the SSSP algorithm that assumed adjacency-list representation of G, for the APSP algorithm we consider the adjacency matrix representation of G.
- For convenience $V = \{1, 2, \ldots n\}$. The $n \times n$ adjacency matrix $W = (w(i,j))$ of G, w:

$$w_{ij} = \begin{cases} 0 & \text{if } i = j \\ w_{ij} & \text{if } (i,j) \in E \\ +\infty & \text{if } i \neq j \text{ and } (i,j) \notin E \end{cases}$$

All pairs shortest paths: APSP

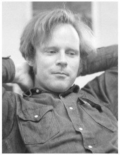
- The input is a $n \times n$ adjacency matrix $W = (w_{ij})$



$$W = \begin{pmatrix} 0 & 1 & \infty & \infty \\ \infty & 0 & 1 & \infty \\ 2 & 4 & 0 & 0 \\ -1 & \infty & \infty & 0 \end{pmatrix}$$

- The output is a $n \times n$ matrix D, where $D[i,j] = \delta(i,j)$ and a $n \times n$ matrix P where $P[i,j]$ is the predecessor of j in a shortest path from i to j

Floyd-Warshall Algorithm

Bernard Roy: *Transitivité et connexité* C.R.Aca. Sci. 1959
Robert Floyd: *Algorithm 97: Shortest Path.* CACM 1962
Stephen Warshall: *A theorem on Boolean matrices.* JACM, 1962

The FW Algorithm is a dynamic programming algorithm that exploits the recursive structure of shortest paths.

Optimal substructure of APSP

- Recall: any subpath of a shortest path is a shortest path
- Let $p = p_1, \underbrace{p_2, \ldots, p_{r-1}}_{\text{intermediate v.}}, p_r$ and
- Let $d_{ij}^{(k)}$ be the minimum weight of a path $i \rightsquigarrow j$ s.t. the intermediate vertices are in $\{1, \ldots, k\}$.
- When $k = 0$, $d_{ij}^{(0)} = w_{ij}$ (no intermediate vertices).

The recurrence

Let p a path $i \rightsquigarrow j$ with intermediate vertices in $\{1, \ldots, k\}$ and weight $d_{ij}^{(k)}$

- If k is not an intermediate vertex of p, then $d_{ij}^{(k)} = d_{ij}^{(k-1)}$.
- If k is an intermediate vertex of p, then $p = i \rightsquigarrow^{p_1} k \rightsquigarrow^{p_2} j$
- p_1 and p_2 are shortest paths with intermediate vertices in $\{1, \ldots, k-1\}$.

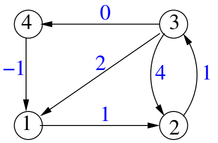
Therefore $d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 \\ \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\} & \text{if } k \geq 1 \end{cases}$

FW-algorithm

BFW (W)
$d^{(0)} = W$
for $k = 1$ to n **do**
  for $i = 1$ to n **do**
    for $j = 1$ to n **do**
      $d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$
return $d^{(n)}$

- Time complexity: $T(n) = O(n^3), S(n) = O(n^3)$
- Correctness follows from the recurrence argument.

FW: Example



$$D^{(0)} = \begin{pmatrix} 0 & 1 & \infty & \infty \\ \infty & 0 & 1 & \infty \\ 2 & 4 & 0 & 0 \\ -1 & \infty & \infty & 0 \end{pmatrix} \qquad D^{(1)} = \begin{pmatrix} 0 & 1 & \infty & \infty \\ \infty & 0 & 1 & \infty \\ 2 & 3 & 0 & 0 \\ -1 & 0 & \infty & 0 \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 1 & 2 & \infty \\ \infty & 0 & 1 & \infty \\ 2 & 3 & 0 & 0 \\ -1 & 0 & 1 & 0 \end{pmatrix} \qquad D^{(3)} = \begin{pmatrix} 0 & 1 & 2 & 2 \\ 3 & 0 & 1 & 1 \\ 2 & 3 & 0 & 0 \\ -1 & 0 & 1 & 0 \end{pmatrix} \qquad D^{(4)} = \begin{pmatrix} 0 & 1 & 2 & 2 \\ 0 & 0 & 1 & 1 \\ -1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \end{pmatrix}$$

$d^{2}_{3,2} = 3$, $3 \to 1 \to 2$ (interm vertices in $\{1, 2\}$)
$d^{4}_{3,2} = 0$, $3 \to 4 \to 1 \to 2$ (interm vertices in $\{1, 2, 3, 4\}$)

FW: Constructing shortest paths

- To construct the matrix P, where
  $p_{i,j}$ is the predecessor of j in a shortest path $i \rightsquigarrow j$,
- we define a sequence of matrices $P^{(0)}, \ldots, P^{(n)}$.
  $p^k_{i,j}$ is the predecessor in a shortest path $i \rightsquigarrow j$, which uses only vertices in $\{1, \ldots, k\}$.
- $p^{(0)}_{i,j} = \begin{cases} NIL & \text{if } i = j \text{ or } w_{ij} = +\infty, \\ i & \text{if } i \neq j \text{ and } w_{ij} \neq +\infty. \end{cases}$
- For $k \geq 1$ we get the recurrence:

$$p^{(k)}_{i,j} = \begin{cases} p^{(k-1)}_{i,j} & \text{if } d^{(k-1)}_{ij} \leq d^{(k-1)}_{ik} + d^{(k-1)}_{kj}, \\ p^{(k-1)}_{k,j} & \text{otherwise.} \end{cases}$$

BFW with paths

BFW W
$d^{(0)} = W$
Initialize $p^{(0)}$
for $k = 1$ to n **do**
  for $i = 1$ to n **do**
    for $j = 1$ to n **do**
      if $d_{ij}^{(k)} \leq d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$ **then**
        $d_{ij}^{(k)} = d_{ij}^{(k-1)}$
        $p_{ij}^{(k)} = p_{ij}^{(k-1)}$
      else
        $d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$
        $p_{ij}^{(k)} = p_{kj}^{(k-1)}$
return $d^{(n)}$

Complexity: $T(n) = O(n^3)$

APSP: Johnson's algorithm

- A faster algorithm for sparse graphs, i.e., $m = o(n^2)$
- The graph is given by adjacency list and we assume that it has no negative weight cycles. In fact the algorithm detects its existence.

Johnson's algorithm

Donald B. Johnson: *Efficient algorithms for shortest paths in sparse networks*, JACM 1977

- The algorithm uses BF to reduce the problems to one with positive weights.
- Then it runs n times Dijkstra's algorithm.

Weight modification that preserve path weight

Lemma

Let $G = (V, E, w)$ be a weighted digraph. Let $f : V \to \mathbb{R}$ and, for $(u, v) \in E$, let $w'(u, v) = w(u, v) + f(u) - f(v)$. Let p be a path $u \leadsto^p v$ in G. Then $w'(p) = w(p) + f(u) - f(v)$.

Proof

As an intermediate vertex w in the path is the end of one edge and the start of another the contribution of $f(w)$ cancels.

The weight modification

- Let $G = (V, E, w)$ be a weighted digraph with no negative weight cycle.
- Construct a graph $G' = (V', E', w')$ by adding to G a new vertex s and edges (s, u), for $u \in V$. Define $w'(e) = w(e)$ if $e \in E' \cap E$ and 0 otherwise.
- Let d be the output of the BF algorithm on input (V', E', w', s).
- As G has no negative weight cycles, G' has no negative weight cycles, so BF computes $d : V \to \mathbb{R}$. Furthermore, for $u \in V$, $d(u) = \delta_{G'}(s, u)$.

The weight modification

Lemma

Let $G = (V, E, w)$ be a weighted digraph with no negative weight cycles. Let $d : V \to \mathbb{R}$ be the function computed by the BF algorithm on G' described before. Let $G_d = (V, E, w')$ where $w'(u, v) = w(u, v) + d(u) - d(v)$.
If p is a shortest path $u \leadsto^p v$ in G, p is a shortest path in G_d. Furthermore, $\delta_{G_d}(u, v) = \delta_G(u, v) + d(u) - d(v)$.

Proof

For any path p, $u \leadsto^p v$, $w'(p) = w(p) + d(u) - d(v)$. As the last term depends only on u and v, the claim follows.

The weight modification

Lemma

Let $G = (V, E, w)$ be a weighted digraph with no negative weight cycles. Let $d : V \to \mathbb{R}$ be the function computed by the BF algorithm on G' described before. Let $G_d = (V, E, w')$ where $w'(u, v) = w(u, v) + d(u) - d(v)$.
For $(u, v) \in E$, $w'(u, v) \geq 0$.

Proof

- By triangle inequality, for a path p, $u \rightsquigarrow^p v$, $\delta_{G'}(s, v) \leq \delta_{G'}(s, u) + w(p)$,
- i.e., $0 \leq w(p) + \delta_{G'}(s, u) - \delta_{G'}(s, v)$
- Therefore $w'(p) = w(p) + d(u) - d(v) \geq 0$.

Johnson's algorithm

Johnson (V, E, W)
Compute G'
$f = BF(G', s)$
Compute G_f
for all $v \in V$ **do**
  $d[v] = \textbf{Dijkstra}(G_f, v)$
for all $u, v \in V$ **do**
  $d[u][v] = d[u][v] + f[v] - f[u]$
return d

- Time complexity: $O(nm)+$ the cost of n calls to Dijkstra
- Correctness follows from the previous lemmas.

Conclusions

SSSP no negative weight cycles accessible form s.

	Dijkstra	BF
$w \geq 0$	$O(m + n \lg n)$	$O(nm)$
$w \in \mathbb{Z}$	NO	$O(nm)$

APSP no negative weight cycles.

	Dijkstra	BF	FW	Johnson
$w \geq 0$	$O(nm + n^2 \lg n)$	$O(n^2 m)$	$O(n^3)$	$O(nm + n^2 \lg n)$
$w \in \mathbb{R}$	NO	$O(n^2 m)$	$O(n^3)$	$O(nm + n^2 \lg n)$

Conclusions: Remarks for APSP algorithms

- For sparse graphs with $m = \omega(n)$ $m = o(n^2)$, Johnson is the most efficient.
- For dense graphs with $m = \Theta(n^2)$, FW has the best complexity.
- For unweighted and undirected graphs, there is an algorithm by R.Seidel that works in $O(n^\omega \lg n)$, where n^ω is the complexity of multiplying two $n \times n$ matrices, which of as today is $\omega \sim 2.3$.
- For further reading on shortest paths, see chapters 24 and 25 of CLRS or 4.4 and 6.8–6.10 of KT.