All-Pairs Shortest Paths: The Floyd-Warshall algorithm

Yufei Tao's Teaching Team

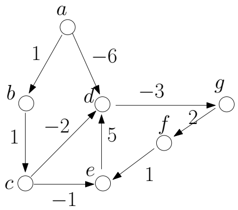
Department of Computer Science and Engineering
Chinese University of Hong Kong

All-Pairs Shortest Paths (APSP)

Input: Let $G = (V, E)$ be a simple directed graph. Let w be a function that maps each edge in E to an integer, **which can be positive, 0, or negative**. It is guaranteed that G has no negative cycles.

Output: We want to find a shortest path (SP) from s to t, for **all** $s, t \in V$. More specifically, the output should be $|V|$ shortest-path trees, each rooted at a distinct vertex in V.

Example



Shortest path distances:
$spdist(a, a) = 0$, $spdist(a, b) = 1$, ..., $spdist(a, g) = -9$
$spdist(b, a) = \infty$, $spdist(b, b) = 0$, ..., $spdist(b, g) = -4$
...
$spdist(g, a) = \infty$, $spdist(g, b) = \infty$, ..., $spdist(g, g) = 0$

We omit the shortest paths in this example.

If all the weights are non-negative, we can run Dijkstra's algorithm $|V|$ times. The total time is $O(|V|(|V|+|E|)\log|V|)$.

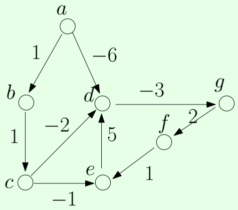
For the general APSP problem (arbitrary weights), we have learned Johnson's algorithm which runs in $O(|V|(|V|+|E|)\log|V|)$ time.

Today we will discuss the **Floyd-Warshall algorithm** that solves the (general) APSP problem in $O(|V|^3)$ time. This improves both Dijkstra's and Johnson's algorithms when $|E|$ is large, e.g., $\Theta(|V|^2)$.

By discussing the Floyd-Warshall algorithm, we will see how dynamic programming can be deployed to find shortest paths.

Set $n = |V|$.
Assign each vertex in V a distinct id from 1 to n.

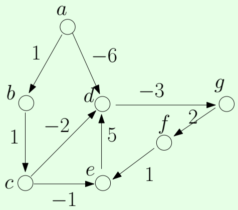
Example:



Let us assign to 1 vertex a, 2 to vertex b, ..., 7 to vertex g.

Define $spdist(i, j \mid\leq k)$ as the smallest length of all paths from the vertex with id i to the vertex with id j that pass only **intermediate** vertices with **ids** $\leq k$.

Example: Vertex ids: 1 for a, 2 for b, ..., 7 for g.



$spdist(1, 5 \mid\leq 0) = \infty$, $spdist(1, 5 \mid\leq 1) = \infty$, $spdist(1, 5 \mid\leq 2) = \infty$, $spdist(1, 5 \mid\leq 3) = 1$, $spdist(1, 5 \mid\leq 4) = 1$, $spdist(1, 5 \mid\leq 5) = 1$, $spdist(1, 5 \mid\leq 6) = 1$, $spdist(1, 5 \mid\leq 7) = -6$

$spdist(3, 4 \mid\leq 0) = -2$, $spdist(3, 5 \mid\leq 0) = \infty$, $spdist(3, 5 \mid\leq 4) = -5$

$spdist(i, j \mid\leq 0)$ equals

- 0, if $i = j$;
- $w(i, j)$, if $(i, j) \in E$;
- ∞, otherwise.

Lemma: It holds for all $i, j, k \in [1, n]$ that

$$spdist(i, j \mid\leq k) = \min \begin{cases} spdist(i, j \mid\leq k-1) \\ spdist(i, k \mid\leq k-1) + spdist(k, j \mid\leq k-1) \end{cases}$$

Observe that $spdist(i, j \mid\leq n) = spdist(i, j)$.
Our goal is therefore to compute $spdist(i, j \mid\leq n)$ for all $i, j \in [1, n]$.

Proof of the lemma. Let π be an arbitrary path that "realizes" $spdist(i, j \mid\leq k)$, namely

- π starts from i and ends at j;
- π uses only intermediate vertices with IDs at most k;
- π has distance $spdist(i, j \mid\leq k)$.

We distinguish two cases.

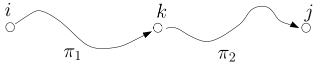
Case 1: k is not on π.

This means that all the intermediate vertices of π have IDs at most $k-1$. Therefore, the length of π must be exactly $spdist(i, j \mid\leq k-1)$.

Think: It must hold that $spdist(i, j \mid\leq k-1) \leq spdist(i, k \mid\leq k-1) + spdist(k, j \mid\leq k-1)$ in this case. Why?

Case 2: k is on π.

It suffices to consider that k appears on π only once (**think:** if k appears on π twice, what would you do?).



Length of π_1 must be exactly $spdist(i, k \mid\leq k-1)$ (**think:** why?).
Length of π_2 must be exactly $spdist(k, j \mid\leq k-1)$.

Therefore, in this case, the length of π must be $spdist(i, k \mid\leq k-1) + spdist(k, j \mid\leq k-1)$.

Think: It must hold that $spdist(i, k \mid\leq k-1) + spdist(k, j \mid\leq k-1) \leq spdist(i, j \mid\leq k-1)$ in this case. Why? □

Lemma: It holds for all $i, j, k \in [1, n]$ that

$$spdist(i, j \mid\leq k) = \min \begin{cases} spdist(i, j \mid\leq k-1) \\ spdist(i, k \mid\leq k-1) + spdist(k, j \mid\leq k-1) \end{cases}$$

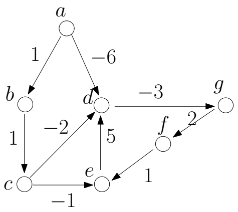
Goal: Compute $spdist(i, j \mid\leq n)$ for all $i, j \in [1, n]$.

The lemma suggests a dynamic programming algorithm that computes $spdist(i, j| \leq n)$ for all $i, j \in [1, n]$ in $O(|V|^3)$ total time.

Sub-problems: $spdist(i, j \mid\leq k)$ for all $i, j \in [1, n]$ and $k \in [0, n]$.
Think: Dependency graph for the sub-problems?

Example

First, decide $spdist(i, j \mid \leq 0)$ for all $i, j \in [1, 7]$.

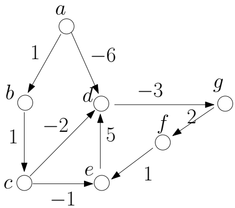


vertex v	a	b	c	d	e	f	g
a	0	1	∞	-6	∞	∞	∞
b	∞	0	1	∞	∞	∞	∞
c	∞	∞	0	-2	-1	∞	∞
d	∞	∞	∞	0	∞	∞	-3
e	∞	∞	∞	5	0	∞	∞
f	∞	∞	∞	∞	1	0	∞
g	∞	∞	∞	∞	∞	2	0

Example

$$spdist(i,j \mid\leq k) = \min \begin{cases} spdist(i,j \mid\leq k-1) \\ spdist(i,k \mid\leq k-1) + spdist(k,j \mid\leq k-1) \end{cases}$$

Then, compute $spdist(i,j \mid \leq 1)$ for all $i,j \in [1,7]$. No changes.



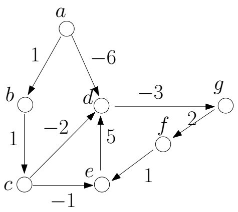
vertex v	a	b	c	d	e	f	g
a	0	1	∞	-6	∞	∞	∞
b	∞	0	1	∞	∞	∞	∞
c	∞	∞	0	-2	-1	∞	∞
d	∞	∞	∞	0	∞	∞	-3
e	∞	∞	∞	5	0	∞	∞
f	∞	∞	∞	∞	1	0	∞
g	∞	∞	∞	∞	∞	2	0

Example

$$spdist(i,j \mid\leq k) =$$

$$\min \begin{cases} spdist(i,j \mid\leq k-1) \\ spdist(i,k \mid\leq k-1) + spdist(k,j \mid\leq k-1) \end{cases}$$

Compute $spdist(i,j \mid \leq 2)$ for all $i,j \in [1,7]$.

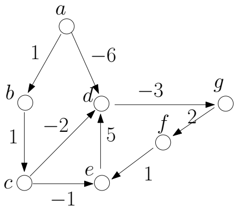


vertex v	a	b	c	d	e	f	g
a	0	1	2	−6	∞	∞	∞
b	∞	0	1	∞	∞	∞	∞
c	∞	∞	0	−2	−1	∞	∞
d	∞	∞	∞	0	∞	∞	−3
e	∞	∞	∞	5	0	∞	∞
f	∞	∞	∞	∞	1	0	∞
g	∞	∞	∞	∞	∞	2	0

Example

$$spdist(i,j \mid\leq k) = \min \begin{cases} spdist(i,j \mid\leq k-1) \\ spdist(i,k \mid\leq k-1) + spdist(k,j \mid\leq k-1) \end{cases}$$

Compute $spdist(i,j \mid \leq 3)$ for all $i,j \in [1,7]$.

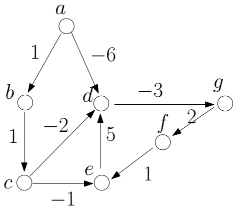


vertex v	a	b	c	d	e	f	g
a	0	1	2	−6	1	∞	∞
b	∞	0	1	−1	0	∞	∞
c	∞	∞	0	−2	−1	∞	∞
d	∞	∞	∞	0	∞	∞	−3
e	∞	∞	∞	5	0	∞	∞
f	∞	∞	∞	∞	1	0	∞
g	∞	∞	∞	∞	∞	2	0

Example

$$spdist(i,j \mid\leq k) = \min \begin{cases} spdist(i,j \mid\leq k-1) \\ spdist(i,k \mid\leq k-1) + spdist(k,j \mid\leq k-1) \end{cases}$$

Compute $spdist(i,j \mid \leq 4)$ for all $i,j \in [1,7]$.

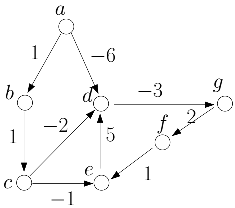


vertex v	a	b	c	d	e	f	g
a	0	1	2	−6	1	∞	−9
b	∞	0	1	−1	0	∞	−4
c	∞	∞	0	−2	−1	∞	−5
d	∞	∞	∞	0	∞	∞	−3
e	∞	∞	∞	5	0	∞	2
f	∞	∞	∞	∞	1	0	∞
g	∞	∞	∞	∞	∞	2	0

Example

$$spdist(i,j \mid\leq k) = \min \begin{cases} spdist(i,j \mid\leq k-1) \\ spdist(i,k \mid\leq k-1) + spdist(k,j \mid\leq k-1) \end{cases}$$

Compute $spdist(i,j \mid \leq 5)$ for all $i,j \in [1,7]$.

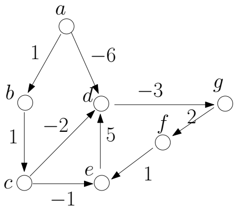


vertex v	a	b	c	d	e	f	g
a	0	1	2	−6	1	∞	−9
b	∞	0	1	−1	0	∞	−4
c	∞	∞	0	−2	−1	∞	−5
d	∞	∞	∞	0	∞	∞	−3
e	∞	∞	∞	5	0	∞	2
f	∞	∞	∞	6	1	0	3
g	∞	∞	∞	∞	∞	2	0

Example

$$spdist(i,j \mid\leq k) = \min \begin{cases} spdist(i,j \mid\leq k-1) \\ spdist(i,k \mid\leq k-1) + spdist(k,j \mid\leq k-1) \end{cases}$$

Compute $spdist(i,j \mid \leq 6)$ for all $i,j \in [1,7]$.

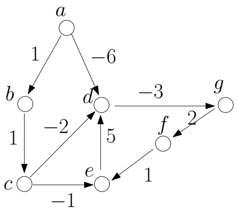


vertex v	a	b	c	d	e	f	g
a	0	1	2	−6	1	∞	−9
b	∞	0	1	−1	0	∞	−4
c	∞	∞	0	−2	−1	∞	−5
d	∞	∞	∞	0	∞	∞	−3
e	∞	∞	∞	5	0	∞	2
f	∞	∞	∞	6	1	0	3
g	∞	∞	∞	8	3	2	0

Example

$$spdist(i,j \mid\leq k) = \min \begin{cases} spdist(i,j \mid\leq k-1) \\ spdist(i,k \mid\leq k-1) + spdist(k,j \mid\leq k-1) \end{cases}$$

Compute $spdist(i,j \mid \leq 7)$ for all $i,j \in [1,7]$.



vertex v	a	b	c	d	e	f	g
a	0	1	2	−6	−6	−7	−9
b	∞	0	1	−1	−1	−2	−4
c	∞	∞	0	−2	−2	−3	−5
d	∞	∞	∞	0	0	−1	−3
e	∞	∞	∞	5	0	4	2
f	∞	∞	∞	6	1	0	3
g	∞	∞	∞	8	3	2	0

Now we are done.

We have focused on computing the shortest path distances *spdist*(s, t) for all $s, t \in V$. How to extend the algorithm to report the shortest path tree rooted at each $s \in V$?

Hint: The piggyback technique.