All-Pairs Shortest Paths

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong

In this lecture, we will look at a problem called **all-pairs shortest paths** which is closely related to the SSSP (single-source shortest path) problem discussed in the previous lectures.

We will learn two algorithms: **the Floyd-Warshall algorithm** and **Johnson's algorithm**. The first one is a standard dynamic programming algorithm, while the second is based on a new technique — called **re-weighting** — that removes all negative edges.

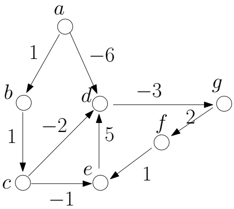
All-Pairs Shortest Paths (APSP)

Input: Let $G = (V, E)$ be a 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: The shortest path (SP) from node s to node t, for every $s \in V$ and every $t \in V$.

We will focus on finding the shortest path distance $\mathit{spdist}(s, t)$ for every $s, t \in V$. Extending the algorithm to report paths is easy and left to you.

Example



We will explain how to compute the following:
$spdist(a, a) = 0$, $spdist(a, b) = 1$, ..., $spdist(a, g) = -9$
$spdist(b, a) = \infty$, $spdist(b, b) = 0$, ..., $spdist(b, g) = -4$
...
$spdist(g, a) = \infty$, $spdist(g, b) = \infty$, ..., $spdist(g, g) = 0$

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

For the general APSP problem (i.e., arbitrary weights), we can run Bellman-Ford's algorithm $|V|$ times. The total running time is $O(|V|^2|E|)$.

At the end of the lecture, we will be able to solve the (general) APSP problem in

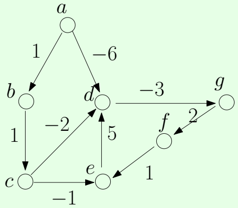
$$O\left(\min\{|V|^3, |V|(|V| + |E|) \log |V|\}\right).$$

Note that the complexity strictly improves that in the second box.

The Floyd-Warshall Algorithm

Set $n = |V|$.
We will assign to every vertex in V a distinct id from 1 to n.

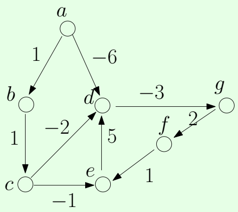
Example:



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

Define $spdist(i, j \mid\le k)$ as the smallest length of all paths from i to j that **pass only vertices with ids $\le k$** (except of course the start vertex i and end vertex j).

Example:



Let us assign 1 to vertex a, 2 to b, ..., 7 to g.
$spdist(1, 5 \mid 1) = \infty$, $spdist(1, 5 \mid 2) = \infty$, $spdist(1, 5 \mid 3) = -1$, $spdist(1, 5 \mid 4) = -1$, $spdist(1, 5 \mid 5) = -1$, $spdist(1, 5 \mid 6) = -1$, $spdist(1, 5 \mid 7) = -6$

Lemma:

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

The proof is simple and left to you.

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 $i, j \in [1, n]$.

This clearly points to a dynamic programming algorithm that finishes in $O(|V|^3)$ time.

Johnson's Algorithm

Recall:

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

But remember we are tackling a graph where edge weights can be negative. Can we **convert all the weights into non-negative values** so that we can apply the above strategy? The challenge is to carry out the conversion **without affecting any shortest paths**.

Re-weighting

Introduce an arbitrary function $h : V \to \mathbb{Z}$, where $\mathbb{Z}$ represents the set of integer values.

For each edge (u, v) in E, redefine its weight as:

$$w'(u, v) = w(u, v) + h(u) - h(v).$$

Denote by G' the graph where

- the set V of vertices and the set E of edges are the same as G;
- the edges are weighted using function w'.

Re-weighting

Lemma: Consider any path $v_1 \to v_2 \to \ldots \to v_x$ in G where $x \geq 1$. If the path has length ℓ, then it has length $\ell + h(v_1) - h(v_x)$ in G'.

Proof: The length of the path in G' is

$$
\begin{aligned}
& \sum_{i=1}^{x-1} w'(v_i, v_{i+1}) \\
= & \sum_{i=1}^{x-1} \left(w(v_i, v_{i+1}) + h(v_i) - h(v_{i+1})\right) \\
= & \left(\sum_{i=1}^{x-1} w(v_i, v_{i+1})\right) + h(v_1) - h(v_x).
\end{aligned}
$$

□

Re-weighting

Corollary: If G has no negative cycles, G' has no negative cycles.

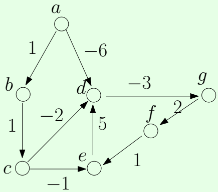
Proof: If $v_1 \to v_2 \to ... \to v_x$ is a cycle, then $v_1 = v_x$. The previous lemma indicates that its length in G is the same as its length in G'. □

Corollary: Let π be a path from vertex u to vertex v in G. If π is a shortest path in G, it is also a shortest path in G'.

Proof: Let π' be any other path from u to v in G'. Denote by ℓ and ℓ' the length of π and π', respectively. It holds that $\ell \leq \ell'$. By the lemma of the previous slide, we know that π and π' have lengths $\ell + h(u) - h(v)$ and $\ell' + h(u) - h(v)$, respectively. □

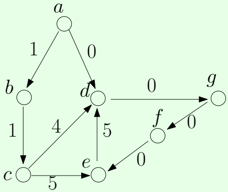
Example

Example:



$h(a) = 0$
$h(b) = 0$
$h(c) = 0$
$h(d) = -6$
$h(e) = -6$
$h(f) = -7$
$h(g) = -9$

After re-weighting:



For our goal (i.e., turning all weights to non-negative), we must ensure:

$$w(u, v) \geq 0$$

for all edges (u, v) in E. Not every function $h(.)$ can fulfill the purpose. In the example of the previous slide, we have provided such a function for illustration purposes.

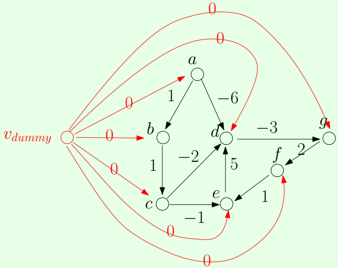
But how to find such a "good" $h(.)$ in general? This calls for a second idea deployed by Johnson's algorithm, which always gives us a good function $h(.)$. In fact, the function $h(.)$ used in the previous slide was obtained using that idea, as we show next.

A "Dummy-Vertex" Trick

From $G = (V, E)$, let us construct a graph $G^{\Delta} = (V^{\Delta}, E^{\Delta})$ where:

- $V^{\Delta} = V \cup \{v_{dummy}\}$;
- E^{Δ} includes all the edges in E, and additionally, a new edge from V^{Δ} to every other vertex in V;
- Each edge inherited from E carries the same weight as in E. Every newly added edge carries the weight 0.

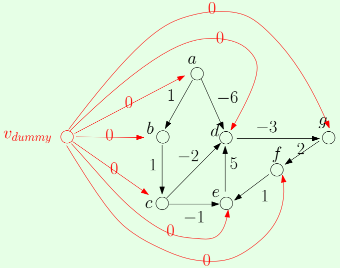
Example:



A "Dummy-Vertex" Trick

In $G^\Delta = (V^\Delta, E^\Delta)$, find the shortest path distance from v_{dummy} to every other vertex. This is an SSSP problem which can be solved by Bellman-Ford's algorithm in $O(|V||E|)$ time.

Example:



$spdist(v_{dummy}, a) = 0$
$spdist(v_{dummy}, b) = 0$
$spdist(v_{dummy}, c) = 0$
$spdist(v_{dummy}, d) = -6$
$spdist(v_{dummy}, e) = -6$
$spdist(v_{dummy}, f) = -7$
$spdist(v_{dummy}, g) = -9$

A "Dummy-Vertex" Trick

Recall that we were looking for a good function $h(.)$ to re-weight the edges of G.

We have just found our function $h(.)$:

$$h(u) = spdist(v_{dummy}, u)$$

for every $u \in V$.

After re-weighting the edges of G with the above $h(.)$, we are guaranteed that all edge weights (in the graph G' obtained after re-weighting) must be non-negative.

Proving the above is easy and will be left as an exercise.

We therefore have obtained an algorithm to solve the APSP problem (with negative weights) in time $O(|V|(|V| + |E|) \log |V|)$.