Recitation 14

Single Source Shortest Paths Review

We've learned four algorithms to solve the single source shortest paths (SSSP) problem; they are listed in the table below. Then, to solve shortest paths problems, you must first define or construct a graph related to your problem, and then running an SSSP algorithm on that graph in a way that solves your problem. Generally, you will want to use the fastest SSSP algorithm that solves your problem. Bellman-Ford applies to any weighted graph but is the slowest of the four, so we prefer the other algorithms whenever they are applicable.

Restrictions		SSSP Algorithm	
Graph	Weights	Name	Running Time $O(\cdot)$
General	Unweighted	BFS	$\lvert V\rvert + \lvert E\rvert$
DAG	Any	DAG Relaxation	$\lvert V\rvert + \lvert E\rvert$
General	Non-negative	Dijkstra	$\lvert V\rvert \log \lvert V\rvert + \lvert E\rvert$
General	Any	Bellman-Ford	$\lvert V\rvert \cdot \lvert E\rvert$

We presented these algorithms with respect to the SSSP problem, but along the way, we also showed how to use these algorithms to solve other problems. For example, we can also **count connected components** in a graph using Full-DFS or Full-BFS, **topologically sort** vertices in a DAG using DFS, and **detect negative weight cycles** using Bellman-Ford.

All Pairs Shortest Paths

Given a weighted graph $G = (V, E, w)$, the (weighted) All Pairs Shortest Paths (APSP) problem asks for the minimum weight $\delta(u, v)$ of any path from u to v for every pair of vertices $u, v \in V$. To make the problem a little easier, if there exists a negative weight cycle in G, our algorithm is not required to return any output. A straight-forward way to solve this problem is to reduce to solving an SSSP problem $|V|$ times, once from each vertex in V. This strategy is actually quite good for special types of graphs! For example, suppose we want to solve ASPS on an unweighted graph that is **sparse** (i.e. $|E| = O(|V|)$). Running BFS from each vertex takes $O(|V|^2)$ time. Since we need to return a value $\delta(u, v)$ for each pair of vertices, any ASPS algorithm requires at least $\Omega(V^2)$ time, so this algorithm is optimal for graphs that are **unweighted** and **sparse**. However, for general graphs, possibly containing negative weight edges, running Bellman-Ford $|V|$ times is quite slow, $O(|V|^2|E|)$, a factor of $|E|$ larger than the output. By contrast, if we have a graph that only has non-negative weights, applying Dijkstra $|V|$ times takes $O(|V|^2 \log |V| + |V||E|)$ time. On a sparse graph, running Dijkstra $|V|$ times is only a $\log |V|$ factor larger than the output, while $|V|$ times Bellman-Ford is a linear $|V|$ factor larger. Is it possible to solve the APSP problem on general weighted graphs faster than $O(|V|^2|E|)$?

Johnson's Algorithm

The idea behind Johnson's Algorithm is to reduce the ASPS problem on a graph with **arbitrary edge weights** to the ASPS problem on a graph with **non-negative edge weights**. The algorithm does this by re-weighting the edges in the original graph to non-negative values in such a way so that shortest paths in the re-weighted graph are also shortest paths in the original graph. Then finding shortest paths in the re-weighted graph using $|V|$ times Dijkstra will solve the original problem. How can we re-weight edges in a way that preserves shortest paths? Johnson's clever idea is to assign each vertex v a real number $h(v)$, and change the weight of each edge (a, b) from $w(a, b)$ to $w'(a, b) = w(a, b) + h(a) - h(b)$, to form a new weight graph $G' = (V, E, w')$.

Claim: A shortest path $(v_1, v_2, \ldots, v_k)$ in G' is also a shortest path in G from v_1 to v_k.

Proof. Let $w(\pi) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$ be the weight of path π in G. Then weight of π in G' is:

$$\begin{aligned}\sum_{i=1}^{k-1} w'(v_i, v_{i+1}) &= \sum_{i=1}^{k-1} \Big(w(v_i, v_{i+1}) + h(v_i) - h(v_{i+1}) \Big) \\ &= \Big(\sum_{i=1}^{k-1} w(v_i, v_{i+1}) \Big) + \Big(\sum_{i=1}^{k-1} h(v_i) \Big) - \Big(\sum_{i=1}^{k-1} h(v_{i+1}) \Big) = w(\pi) + h(v_1) - h(v_k).\end{aligned}$$

So, since each path from v_1 to v_k is increased by the same number $h(v_1) - h(v_k)$, shortest paths remain shortest. □

It remains to find a vertex assignment function h, for which all edge weights $w'(a, b)$ in the modified graph are non-negative. Johnson's defines h in the following way: add a new node x to G with a directed edge from x to v for each vertex $v \in V$ to construct graph G^*, letting $h(v) = \delta(x, v)$. This assignment of h ensures that $w'(a, b) \geq 0$ for every edge (a, b).

Claim: If $h(v) = \delta(x, v)$ and $h(v)$ is finite, then $w'(a, b) = w(a, b) + h(a) - h(b) \geq 0$ for every edge $(a, b) \in E$.

Proof. The claim is equivalent to claiming $\delta(x, b) \leq w(a, b) + \delta(x, a)$ for every edge $(a, b) \in E$, i.e. the minimum weight of any path from x to b in G^* is not greater than the minimum weight of any path from x to a than traversing the edge from a to b, which is true by definition of minimum weight. (This is simply a restatement of the triangle inequality.) □

Johnson's algorithm computes $h(v) = \delta(x, v)$, negative minimum weight distances from the added node x, using Bellman-Ford. If $\delta(x, v) = -\infty$ for any vertex v, then there must be a negative weight cycle in the graph, and Johnson's can terminate as no output is required. Otherwise, Johnson's can re-weight the edges of G to $w'(a, b) = w(a, b) + h(a) - h(b) \geq 0$ into G' containing only positive edge weights. Since shortest paths in G' are shortest paths in G, we can run Dijkstra $|V|$ times on G' to find a single source shortest paths distances $\delta'(u, v)$ from each vertex u in G'. Then we can compute each $\delta(u, v)$ by setting it to $\delta'(u, v) - \delta(x, u) + \delta(x, y)$. Johnson's takes $O(|V||E|)$ time to run Bellman-Ford, and $O(|V|(|V| \log |V| + |E|))$ time to run Dijkstra $|V|$ times, so this algorithm runs in $O(|V|^2 \log |V| + |V||E|)$ time, asymptotically better than $O(|V|^2|E|)$.

6.006 Introduction to Algorithms

Spring 2020