Shortest Paths

In the past, we were able to use breadth-first search to find the shortest paths between a source vertex to all other vertices in some graph G. We weighed each edges equally so the shortest path between two vertices was the one that contained the fewest edges. Now, we introduce edge weights so the cost of traveling through edges can differ from edge to edge. The shortest path between two vertices is defined to be the path whose sum of edge weights is the least. BFS will not work on weighted graphs since the path with the fewest edges may not be the shortest if the edges it contains are expensive.

There are several variants on the shortest paths problem and the algorithms that we will go over that correspond to solving each problem are in parentheses:

- **Single-source shortest-paths problem:** Find a shortest path from a source vertex to each other vertex in the graph (Bellman-Ford, Dijkstra)
- **Single-destination shortest-paths problem:** Find a shortest path to a destination vertex from each other vertex in the graph (Bellman-Ford/Dijkstra on reversed graph)
- **Single-pair shortest-path problem:** Find a shortest path between a vertex u and a vertex v in a graph (Bellman-Ford, Dijkstra)
- **All-pairs shortest-paths problem:** Find a shortest path between every two vertices in a graph (Bellman-Ford/Dijkstra V times, Floyd-Warshall)

Before getting into Bellman-Ford and Dijkstra, we will go over the principles the algorithms are built upon.

Notation

Bellman-Ford and Dijkstra both solve the problem of finding a shortest path from a source vertex to each other vertex in the graph. We will designate the source vertex as s. Every vertex v in the graph is augmented with the following parameters:

- **v.d** - The weight of the current shortest path from s to v. This is initialized to be ∞ for all vertices besides the source vertex but decreases as paths are found and shorter paths are discovered. At the end of the algorithms, this will be the weight of the shortest path. s.d is initialized to be 0.
- **v.π** - The parent vertex of v in the current shortest path. This is initialized to be NIL but gets set to a vertex once a path is discovered from s to v. As shorter paths to v are discovered, the parent updates to reflect the change. At the end of the algorithms, this will be the parent of v in the shortest path to v. $s.\pi$ will always be NIL.

Also,

- **w(u, v)** is the weight of the edge from vertex u to vertex v
- δ**(u, v)** is the weight of the shortest path from vertex u to vertex v

Relaxation

Initializing the algorithms involves setting v.d to ∞ and $v.\pi$ to NIL. Throughout the course of the algorithms, we will need to update these values to find shortest paths.

The idea is that if we found a path costing u.d from s to u and there is an edge from u to v, then the upper bound on the weight of a shortest path from s to v is u.d plus the weight of the edge between u and v. We can thus compare u.d + $w(u, v)$ to v.d and update v.d if u.d + $w(u, v)$ is smaller than the current v.d. In pseudocode, relaxing the edge (u, v) is:

```
RELAX(u, v):
  if v.d > u.d + w(u, v) ## if we find a shorter path to v through u
    v.d = u.d + w(u, v) ## update current shortest path weight to v
    v.pi = u ## update parent of v in current shortest path to v
```

Both Bellman-Ford and Dijkstra use relaxation to discover shortest paths. The difference between the two is the order in which edges are relaxed.

Properties of Shortest Paths

Using our definitions of shortest paths and relaxations, we can come up with several properties. These can all be found in CLRS in chapter 24

Triangle inequality: For any edge (u, v), we have $\delta(s, v) \leq \delta(s, u) + w(u, v)$. In english, the weight of the shortest path from s to v is no greater than the weight of the shortest path from s to u plus the weight of the edge from u to v.

Optimal substructure: Let $\{v_1, v_2, v_3, ..., v_k\}$ be a shortest path that goes from v_1 to v_k through the vertices v_2 through v_{k-1}. Any subpath $\{v_i, v_{i+1}, ..., v_{j-1}, v_j\}$ must be a shortest path from v_i to v_j. That is, a shortest path is constructed of shortest paths between any two vertices in the path.

Upper-bound property: We always have v.d $\geq \delta(s, v)$ for all vertices v. Once v.d = $\delta(s, v)$, it never changes.

No-path property: If there exists no path from s to v, v.d will always be ∞.

Convergence property: If a shortest path from s to v contains the edge (u, v) and u.d = $\delta(s, u)$ before relaxing edge (u, v), then v.d = $\delta(s, v)$ at all times after relaxing edge (u, v).

Path-relaxation property: Let $\{v_1, v_2, v_3, ..., v_k\}$ be a shortest path that goes from v_1 to v_k. If the edges are relaxed in the order (v_1, v_2), (v_2, v_3), etc., then v_k.d = $\delta(s, v_k)$ once the whole path is relaxed.

Predecessor-subgraph property: Once v.d = $\delta(s, v)$ for all vertices v, the predecessor subgraph is a shortest-paths tree rooted at s. The predecessor subgraph is the subgraph of G that contains all the vertices with a finite distance from s (i.e. reachable from s) and only the edges that connect v to $v.\pi$.