Shortest Path and BFS

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. The reason it worked is that each edge had equal weight (e.g., 1) 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.

However, if all the weights are intergers and they are bounded by a small number, say k, we can still use BFS. To do this, for each edge (u, v), we split it into $w(u, v)$ edges with weight 1 connecting u to v through some dummy vertices. Then we do BFS on the new graph to find the shortest path. The time complexity will be $O(kE)$ if the original graph has $|E|$ edges.

Graph Transformation

Shortest path with even or odd length

Given a weighted graph $G = (V, E, w)$, suppose we only want to find a shortest path with odd number of edges from s to t. To do this, we can make a new graph G'. For every vertex u in G, there are two vertices u_E and u_O in G': these represent reaching the vertex u through even and odd number of edges respectively. For every edge (u, v) in G, there are two edges in G': (u_E, v_O) and (u_O, v_E). Both of these edges have the same weight as the original. Constructing this graph takes linear time $O(V + E)$. Then we can run shortest path algorithms from s_E to t_O.

Optimization in two dimensions

Suppose you want to drive from city s to city d, and want to reach d as early as possible while minimizing the gas cost. Function $g(u, v)$ represents the gas cost for travelling from city u to city v. Function $f(u, v, t)$ represnets the time it takes to drive from u to v starting at time t because the traffic is different throught out the day. All times are integers in mininutes. Suppose the upper bound of the time it takes from s to d is T. Find the best route to do this. Note that you can stop at some cities to wait for a better traffic.

For each city u, we create $T + 1$ copies of vertices u_t for $t = 0, 1, \ldots, T$ representing reaching u at time $0, 1, \ldots, T$. We also add an edge (u_t, u_{t+1}) for $t = 0, 1, \ldots, T - 1$ with weight 0 to represent waiting at city u. For every route between u and v, we add an edge $(u_t, v_{t+f(u,v,t)})$ for $t = 0, 1, \ldots, T$ if $t + f(u, v, t) \leq T$ with weight equal to $g(u, v)$.

Then we can run Dijkstra's algorithm starting at s_0 and check from $d_0, d_1, \ldots, d_T$ to find the first d_t that is not infinity. Then t is the earliest time we can reach d and the path costs minimum gas.