Homework Set 3

Miles Van de Wetering

February 7, 2017

Problem 1 - a

Modify Shimbel's algorithm to detect a negative cycle in a directed weighted graph G = (v, E). Your algorithm should return a cycle of negative weight if one exists in G. otherwise, it whould return "No Negative Cycle."

This may be most easily solved by simply running Shimbel's algorithm for V + 1 steps, instead of V steps. If, on the V+1th step, there exists one or more tense edges, we simply make note of which vertices were involved and by how much the edge was 'loosened'. The edge which was used in the most extreme loosening is necessarily a member edge of the most negative cycle. The entire negative cycle may be found by tracing back from the vertex using the predecessor function (if we have marked that) until re-arriving at the same vertex. The amount by which the path was shortened corresponds to the magnitude of the negative cycle in question (though this could also be calculated during the cycle traversal).

Problem 1 - b

Arbitrage: Suppose that we are given n currencies $c_1, ...c_n$, and an nxn table R of exchange rates, such that one unit of currency c_i buys $R(i, j)$ units of currency c_j. Give an efficient algorithm to find out a sequence of currencies $c_{i1}, ..., c_{ik}$ such that:

$R[i_1, i_2]xR[i_2, i_3]x...xR[r_{k-1}, r_1] > 1$,

if such a sequence exists. Otherwise, your algorithm must return "No Arbitrage."

This algorithm is a very straightforward modification of the algorithm given in Problem 1 - a. The only change is that before running our modified version of Shimbel's algorithm, we first replace every edge weight w with $-log(w)$, since by using logs we may turn the multiplication problem into an addition problem which is then solveable using a usual shortest path algorithm (we reverse the sign because a cycle yielding a product like 1.1 would be better than one which yields .9).

Problem 2

Let G = (V, E) be a connected directed graph with non-negative edge weights, let s and t be vertices of G, and let H be a subgraph of G obtained by deleting some edges. Suppose we want to reinsert exactly one edge from G back into H, so that the shortest path from s to t in the resulting graph is as short as possible. Describe an analyse an algorithm that chooses the best edge to reinsert, in O(ElogV) time.

My algorithm will be dominated by the runtime of Dijkstra's algorithm, which will run in O(E * log(V)) time.

First, run Dijkstra's algorithm from s. This will give us the shortest path in H from s to t, along with the shortest path from s to every vertex along that path, and of course the shortest distance from s to every other node as well. Second, run Dijkstra's algorithm again from t, this time with the direction of every edge reverse. This will give us the shortest distance from every node to t. Then observe the following procedure:

```
min <- (length of current shortest path, NULL) // the NULL here will later refer to the edge which we plan to insert

for each edge e = (u->v) in E, s.t. E = E_g - E_h:
    if dist[s, u] + w(e) + dist[v, t] < min: min <- (dist[s, u] + w(e) + dist[v, t], e)
```

Upon completion, min will contain both the length of the shortest path as well as the edge we plan to insert to accomplish this.

Observe that the inner conditional runs in constant time, and the for each loop will occur O(E) times. Each run of Dijkstra, on the other hand, is O(E log(V)) so O(E log(V)) will be our runtime.