Programming Assignment

Complexity and Correctness Proof

Student Full Name

Institutional Affiliation

Course Full Title

Instructor Full Name

Due date

Proof Activity

Claim 1.

The Bellman-Ford shortest path algorithm takes time $\Theta(E \cdot V)$ where V is the number of vertices and E is the number of edges.

Proof

This proof is adapted from Cormen, Leiserson, Rivest, and Stein, Introduction to Algorithms [3].

The Bellman-Ford algorithm finds the shortest path between any two vertices in a weighted graph, even in the presence of negative weights. The algorithm works by iteratively relaxing all the edges in the graph. Let G = (V, E) be a weighted directed graph with vertices V = v1, v2, ... vn and edges E = (u,v,w) | u,v V, and w is the weight of the edge from u to v. Let dist[v] be the shortest distance from the source vertex s to vertex v in the ith iteration of the algorithm.

The Bellman-Ford algorithm relaxes all the edges (u, v) in the graph by comparing the distance from s to v through u with the current distance estimate for v. If the distance estimate for v can be improved by going through u, the algorithm updates dist[v] with the new distance estimate, i.e., dist[v] = dist[u] + w(u,v).

The algorithm repeats the relaxation process for V-1 iterations since any shortest path between two vertices can have at most V-1 edges. At the end of each iteration, dist[v] is the shortest path distance from s to v that uses at least i edges.

If there is a negative weight cycle in the graph, the algorithm will continue to find shorter and shorter paths through the cycle, and the distances will become arbitrarily small. Therefore, the Bellman-Ford algorithm can detect the presence of negative weight cycles by checking if there are any improvements in the distances after the V-1st iteration. If there are, then there is a negative weight cycle in the graph. The Bellman-Ford algorithm has a time complexity of O(|V||E|), which is worse than Dijkstra's algorithm. However, Bellman-Ford can handle negative weight edges, while Dijkstra's algorithm cannot.

Proof Summary

We debate how long each of the algorithm's three main parts runs. As we visit each vertex exactly once, initializing the graph's vertices takes linear time. The primary loop repeatedly cycles through the number of vertices and all the edges in the examine whether using that edge results in a shorter cost path on the graph. Keep in mind that we cycle through the vertices because the longest, non-cyclic path can only visit a certain number of them in the graph. Then, to look for a cycle, we iterate over the edges once more. As a result, the iteration that repeats for the number of vertices and iterates through all the edges each time dominates the overall run time.

Algorithm

```
public class BellmanFord{


    /**
     * Utility class. Don't use.
     */
    public class BellmanFordException extends Exception{
        private static final long serialVersionUID = -4302041380938489291L;
        public BellmanFordException() {super();}
        public BellmanFordException(String message) {
            super(message);
        }
    }

    /**
     * Custom exception class for BellmanFord algorithm
     *
     * Use this to specify a negative cycle has been found
     */
    public class NegativeWeightException extends BellmanFordException{
        private static final long serialVersionUID = -7144618211100573822L;
        public NegativeWeightException() {super();}
        public NegativeWeightException(String message) {
            super(message);
        }
    }
```

Figure 1: Bellman-Ford Algorithm

Real World Application

One real-world application of the Bellman-Ford algorithm is in the field of network routing protocols. Routing protocols are used to determine the best path for data to travel through a network, and the Bellman-Ford algorithm can be used to calculate the shortest path between two nodes in the network.

For example, the Routing Information Protocol (RIP) is a distance-vector routing protocol that uses the Bellman-Ford algorithm to calculate the best path for data to travel through a network. Each router in the network maintains a routing table that contains the distance to each destination network and the next hop router that should be used to reach that destination. The routers exchange routing updates with each other to keep their routing tables up to date.

The Bellman-Ford algorithm is used by RIP to calculate the shortest path between two routers in the network. Each router uses the distance and next hop information in its routing table to determine the shortest path to each destination network. The routers exchange routing updates with each other to ensure that they all have the same information about the network topology.

References

Tanenbaum, A. S., & Wetherall, D. (2011). Computer networks. Pearson.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms. MIT Press.