CSCI 2300
Introduction to Algorithms: HW3

Sean Hyde
RIN: 662096071

Due: September 19th, 2024

1 3.15

1.1 (A)

This problem can be expressed as a directed graph where each intersection is represented by a node, and each edge is represented as a street, forming a directed graph. To test the Mayor's hypothesis that each intersection is able to be reached from any other intersection, we can run Kosaraju's Algorithm to find the Strongly Connected Components (SCCs) of the graph. If the entire graph consists of only one SCC, then we know that every node (intersection) is reachable from every other node, and the Mayor's claim is correct. However, if there is more than one SCC in the graph, then we can conclude that the Mayor's claim is false.

1.2 (A) Runtime

Kosaraju's Algorithm runs in linear time $O(V + E)$, where V is the number of vertices (intersections), and E is the number of edges (streets).

1.3 (B)

This problem can also be expressed as a directed graph where each intersection is represented by a node, and each edge is modeled as a street. To test the mayor's weaker claim, we can run BFS starting from the Town Hall node, finding all connected nodes from the Town Hall, and compiling a list $L1$ of those nodes (we have just found all nodes reachable **FROM** the Town Hall).

We can then inverse the graph, by reversing the direction of every edge in the graph and run BFS again and we will then find all nodes within the graph which have a path TO the Town Hall and compile them into a list L2. We then check if all nodes in $L1$ are present in $L2$. If all nodes in $L1$ are present in $L2$, then the Mayor's claim is valid. $L2$ may include additional nodes, but these can be ignored for the comparison. If any node from $L1$ is missing from $L2$, the Mayor's claim is false.

2 3.22

This problem can be solved by using Kosaraju's algorithm on graph G, which will output a DAG of its SCCs. We will treat all outputted SCCs as nodes of the DAG. We can then perform a topological sort on the DAG, which will sort the SCCs in linear time. From here, we check if the first SCC in the topological order can reach every other SCC by running BFS on the source node.

The algorithm will return **true** if the first SCC in the topological order can reach all other SCCs, meaning there exists a vertex $s \in V$ that can reach all other vertices in V. The algorithm will return **false** if the first SCC cannot reach every other SCC in the topological sort.

3 4.11

To solve this problem under the constraints of the $O(|V|^3)$ runtime, for each node $u \in V$ in the graph G, we can run Dijkstra's algorithm. This will find the shortest path to every other node in the graph.

If we encounter an edge that goes back to node u, we check if a cycle exists. The total cycle length will be the shortest path from u to some node v plus the length of the edge from v back to u. We have to keep track of this cycle length and store it as the current minimum cycle length. If we find a shorter cycle, we update the minimum value.

After running the algorithm for all nodes, if any cycles are found, we return the length of the shortest cycle. If no cycles are found, the graph is acyclic.

4 4.13

4.1 (A)

Iterate through all edges $e \in E$, and if the edge length L_e is greater than L, we can remove it from the graph since it does not satisfy the condition $L_e \leq L$. After removing all edges that do not satisfy the condition $L_e \leq L$, we can run BFS starting from node S (representing the city) to find a path to node T if it exists.

4.2 (B)

We must modify Dijkstra's algorithm to accomplish this task. Instead of choosing the shortest path to each node, we must choose the smallest of the maximum edge length required path and keep track of it. So, the algorithm will keep track of the largest edge length required to reach a specific node, then select the minimum largest edge length required to keep the gas tank required to a minimum.