Report for Programming Problem 3 – Bike Lanes

Team:

Student ID: 2018293728 Name: Miguel Rabuge
Student ID: 2018283166 Name: Pedro Rodrigues

1. Algorithm description

This problem consists in answering four questions related to a city that is represented through a graph data structure.The vertices in this graph represent points of interest in this city and the edges represent streets where bike lanes can be built. The challenge consists in finding how many circuits this city has and the size of the largest circuit. After this is done we need to determine the optimal way in which the bike lane should be built inside the circuit such that the bike lane has the minimum length while still connecting all points of interest, in order to find the length of the largest lane and the sum of all of them.The answer to these problems can be reduced to Tarjan's strongly connected components (SCC) and Kruskal's minimum spanning tree (MST) algorithms.

1.1. Circuit identification

Since each circuit is a set of points of interest and connections between those points that allows travelling between any pair of points and it has maximal size, which is the same as saying that a circuit constitutes a strongly connected component in a graph. We need to take into account that to be considered a circuit a SCC must contain at least 2 POIs (points of interest). Applying a modified version of Tarjan's algorithm regarding the restrictions imposed by this problem we are able to answer the questions about the number of circuits and about the size of the largest one. The following pseudo-code shows how this algorithm works.

```
Function Tarjan(v)
        low[v] = dfs[v] = t
        t = t + 1
        push(S, v)
        for each arc (v, w) ∈ G do
                if dfs[w] has no value then
                        Tarjan(w)
                        low[v] = min(low[v], low[w])
                else if w ∈ S then
                        low[v] = min(low[v], dfs[w])
        if low[v] = dfs[v] then
                C = {}
                repeat
                        w = pop(S)
                        push(C, w)
                until w = v
                if |C| >= 2  // circuit size must be >= 2 (problem restriction)
                        push(Scc, C)
```

1.2. Selection the streets for the bike lanes

To determine the streets in which bike lane segments should be built guaranteeing that the sum of all the segments has the minimum length possible is the same as saying that we want to find the MST inside this graph. By applying this algorithm we are able to answer the two last questions about the total sum of all lanes and the length of the largest one. The pseudo-code below shows how this algorithm works.

```
Function Kruskal(G)
        T = {}
        for each vertex v ∈ V do
                make set(v)
        sort edges in E into nondecreasing order by weight
        for each edge {u, v} ∈ E do
                if find_set(u) != find_set(v) then
                        T = T ∪ {u, v}
                        union_set(u, v)
        return T
```

2. Data structures

In terms of data structures used in the implementation of this problem we used a variety of data structures. To store the city graph we used an adjacency list data structure that was implemented using std::vector, and std::pair STL containers (std::vector<std::vector<std::pair<int,int>>>). The adjacency list was used because it allows traversing the neighbors of a given node efficiently while being memory efficient and storing only the necessary information about the connections in the graph.

In the circuit finding algorithm (tarjan) we used as support data structures two std::vector<int> containers to store the “low” and “dfs” values, a std::stack<int> to store the nodes in the current SCC and std::vector to store the multiple circuits (SCCs) found in the city graph (std::vector<std::vector<int>>).

In the lane construction algorithm (Kruskal) we used an edge list as a support data structure. This edge list was implemented using a std::priority_queue which allows us to keep the edges added to it sorted by increasing weight as required for the algorithm (std::priority_queue<int, std::vector<int>, EdgeComparator>) to run. We also used the union-find disjoint set data structure (using the path compression technique) in order to make this algorithm run efficiently. This data structure works on top of standard STL containers too, in our implementation we used std::vector<int> containers to store the “rank” and “pset” values.

3. Correctness

The problem solution was built on top of two well known and proven algorithms, that being said, there aren’t many ways to disprove the correctness of our algorithm. The standard algorithms used (Tarjan and Kruskal) already guarantee some of the restrictions that were imposed in the problem statement and described above. The only restrictions that these algorithms do not cover are related to more problem specific requirements. In our case we only needed to adapt circuit finding algorithm (tarjan) to make sure that the size of a circuit was

greater or equal than two to be added to the list of circuits. This change impacted the correctness of the original algorithm in any way and only filtered the SCCs found by the underlying Tarjan's algorithm.

4. Algorithm Analysis

As our solution is built from two algorithms it is logical that the complexity of answering the four questions asked in the problem statement is the sum of all the parts. The first two questions are answered by applying a modified version of Tarjan's algorithm to the graph and the last two by applying Kruskal's algorithm to each circuit that we found inside our graph.

In case o Tarjan's algorithm the temporal cost of finding all the SCCs on the graph is O(|V| + |E|), which is linear at the worst case. After finding all the SCCs on the graph it becomes really easy to answer the first two questions. The question about how many circuits exist can be computed in constant time after the algorithm stops. The question about the size of the largests circuit can be also obtained in constant time if we update a variable keeping our current largest circuit while we are running the algorithm.

In case of Kruskal's algorithm the temporal cost of finding the minimum spanning tree in a graph, using the union-find disjoint set data structure is O(|E| log |E| + |E| log |V|). The O(|E| log |E|) part comes from the edge sorting that is required before the algorithm can be applied. The O(|E| log |V|) iis an upper bound for the union-find operations. That being said, the worst case complexity of Kruskal's MST is the worst between these two operations. The cost of answering the last two questions of the problem statement can be obtained after K applications of this algorithm to the many circuits in the city graph.

We finally arrive at the overall complexity of computing answering all the questions from the problem statement, that is given for the expression below where |V| is the number of vertices on the graph(POIs), |E| is the number of edges on the same graph (streets in the city) and K is the number of circuits(SCCs) in a given city graph.

Temporal Complexity: $O((|V| + |E|) + K \cdot (|E| \cdot log\ |E| + |E| \cdot log\ |V|))$

In terms of spatial complexity the main things we need to take into account are the cost of the graph (adjacency list) and the auxiliary data structures that we used in our implementation of the algorithm: 1 stack and the 2 auxiliar vectors (with at most |V| size), and the vector of circuits (with at most K x |C| size where |C| is the size of a circuit). In the case of Kruskal's algorithm and the edge list (with size |E|) ,pset and rank vectors (with size |V|) used in the multiple iterations of this algorithm. The following expression gives an upper bound on the overall memory cost for this problem.

Spatial Complexity: $O((|V| + |E|) + (3 \cdot |V| + K \cdot |C|) + K(2 \cdot |V| + |E|) + c)$

5. References

[1] cppreference.com
[2] T. H. Cormen, et a.l Introduction to Algorithms, 3rd ed., 2009.
[3] Week10 and Week11 slides from EA classes.