CS 420
Section 001
Winter 2019
Mazen Alotaibi *(alotaima)* HW 2

Problem 1. Reachability. Let $G = (V, E)$...

Solution: We can use BFS algorithm to determine the min(u). However, we will need to sort the vertices in creasing order first before applying BFS. In order to make the sorting algorithm faster, we will need to use a linear sorting algorithm, such as Pigeonhole Sort or Counting Sort. BFS will find the lowest unvisited node by previous BFS and start BFS from that node. Please check ***Algorithm 1*** *Problem 1.* Time Complexity:

1. Sorting: $O(V)$
2. BFS: $O(V + E)$
3. Total: $O(V + V + E) = O(V)$

□

Problem 2. The police department in the city of Computopia...

(a) Formulate this problem graph-theoretically, and explain why it can indeed be solved in linear time.

(b) Suppose it now turns out that the mayors original claim is false. She next claims something weaker: if you start driving from town hall, navigating one-way streets, then no matter where you reach, there is always way to drive legally back to the town hall. Formulate this weaker property as a graph-theoretical problem and carefully show how it too can be checked in linear time.

Solution:

(a) In order for the mayor to check whether their statement is correct, that all streets are one-way streets. We can check this by treating the graph as a Strongly-Connected Component (SCC) graph, which means we can use BFS to prove whether the graph is SCC graph or not, which takes a linear time to prove that. $O(V + E)$.

(b) We can prove that the graph is still SCC, one-way streets, and can reach back to Town Hall when starting from Town Hall by using DFS. Whenever apply to DFS and couldn't reach the end point as Town Hall then the Mayor's statement is wrong, which it could be checked in a linear time. $O(V + E)$. a graph-theoretical problem and carefully show how it too can be checked in linear time.

□

Problem 3. Let $G = (V, E)$ be a connected...

Solution: There are three cases of decrease the weight of an edge, if the T was the MST tree of G before the update:

1. If we decreased all the edges in G, then T is the MST tree of G.

2. If we decreased one or multiple edges in T, then T is the MST tree of G.
3. If we decreased one or multiple edges in G but not in T, then we need to apply this algorithm:
   (a) Add the updated edge to MST, then we will have a cycle.
   (b) Remove the highest edge on the cycle to turn it to MST. (Cycle Property)
   (c) Time Complexity: $O(V)$

□

Problem 4. Euclidean MST Implementation ...

(a) A verbal describe of your algorithm and data structures

(b) The pseudo-code.

(c) Theoretical running time.

Solution:

(a) From my knowledge, I have founded this problem is similar to what we learned in week 3, MST slides. The only difference is the weight calculation and estimation, so I have solved the problem by doing the following:
   (a) Reading the file line by line and save the vertices (x, y) positions into a list and storing every list of vertices to a list of test cases.
   (b) Mapping the vertices position in every test case to a list of possible edges with calculated weights from the Euclidean distance function.
   (c) Apply Kruskal's algorithm the same way as we have learned in class.
   (d) Get the cumulative value of the suggested MST to printed.

(b) Please check ***Algorithm 2*** *Problem 4.b.*

(c) Time Complexity:
   (a) Sorting: $O(E\ log(E))$
   (b) Finding Union: $O(E\ log(V))$
   (c) Overall: $O((E\ log(E))\ +\ (E\ log(V))) = O(E\ log(E)) = O(E\ log(V))$

□

Algorithm 1 Problem 1

```
procedure REACHABILITY
    $sortedList = Sort(v)$
    $visited =$
    for $v_i \in sortedList$ do
        if $v_i \notin visited$ then
            $min = BFS(v)$
            $visited.append(BFS(v))$
```

Algorithm 2 Problem 4.b

```
procedure KRUSKAL
    $Sort(listEdges)$
    for $i = 1$ to $n$ do
        $MakeSet(i)$
    $count = 0$
    $i = 1$
    $sum = 0$
    while $count < n - 1$ do
        if $FindSet(v_{i_1}) \neq FindSet(v_{i_2})$ then
            $sum = sum + GetWeight(i)$
            $count = count + 1$
            $Unoin(v_1, v_2)$
        $count = count + 1$
    return sum
```