First Name: Zhicheng
Last Name: Yan
Student ID: 1002643142

First Name: Zhihong
Last Name: Wang
Student ID: 1002095207

First Name: Jialiang
Last Name: Yi
Student ID: 1002286929

We declare that this assignment is solely our own work, and is in accordance with the University of Toronto Code of Behavior on Academic Matters.

This submission has been prepared using LaTeX.

Writer: Zhihong Wang

Reader: Zhicheng Yan, Jialiang Yi

Question 1(12 marks)
Prove that every connected undirected graph G contains at least one vertex whose removal (along with all incident edges) does not disconnect the graph. Give an algorithm that for any connected graph $G = (V, E)$, given by its adjacency list, finds such a vertex in $O(|V| + |E|)$ time in the worst case. Justify your algorithm's correctness and worst-case time complexity.

Solution:

Part 1. Prove that every connected undirected graph G contains at least one vertex whose removal (along with all incident edges) does not disconnect the graph.

Leaf definition: a vertex node who only has one edge.

Note: If a vertex removal does disconnect the graph, then this vertex must be a internal node who has at least 2 edges.

Prove:

Because in BFS tree (more details in part 2), there exists at least one leaf and leaf removal does not disconnect the graph.

Therefore every connected undirected graph G contains at least one vertex whose removal does not disconnect the graph.

Part 2. Give an algorithm that for any connected graph $G = (V, E)$, given by its adjacency list, finds such a vertex in $O(|V| + |E|)$ time in the worst case.

We will use BFS for this question.

Note: $|V|$ = number of vertexes in G, $|E|$ = number of edges in G. "vertex" shares same meaning with "node" in this our discussion. By

lecture and CLRS, we use ”White” represents discovered vertex; ”Grey” represents discovered but unexplored vertex; ”Black” represents discovered and explored vertex. BFS will definetly reach the last leaf at the end of search. Details of BFS, like pseudocode and definitions, are in lecture notes and CLRS.

Our algorithm is designed to find and return a special vertex X which is the last leaf vertex (also the last node BFS need to discover) at the end of BFS for G. X removal does not disconnect the graph because the connected neighbour vertex of X is discovered, which means the connected neighbour vertex has path in G. If X is removed, then the neighbour vertex is still in G, which means G is not disconnect.

In order to get the correct last leaf vertex, we will use a ”counter” to record the number of vertex that our algorithm discovered. (i.e. when meet a ”White” vertex, counter will +1 and the ”White” vertex will become ”Grey”; when meet a ”Grey” or ”Black” vertex, counter will do nothing) When counter is equal to $|V|$, then this vertex is the last leaf vertex X by BFS definition.

Because the worst case running time of BFS is $O(|V| + |E|)$. # By lectures and CLRS

Therefore finds such a vertex in $O(|V| + |E|)$ time in the worst case.

Part 3. Justify your algorithm’s correctness and worst-case time complexity.

Correctness:

Because BFS tree must contains at least one leaf. # By definition from CLRS

Leaf’s connected neighbour vertex must be discovered before BFS reach the leaf, so leaf removal does not disconnect the graph.

Then our algorithm based on BFS must return. # closest linked neighbour vertex of leaf is ”Gray” or ”Black”

Worst-case time complexity:

By lectures, tutorials and textbooks, assume all code lines except recursions or loops only require constant time.

By part 2, the worst-case time complexity of BFS is $O(|V| + |E|)$ and our algorithm will go over the whole G until the last leaf in BST tree.

Therefore, the worst-case time complexity is $O(|V| + |E|)$.

Writer: Zhicheng Yan

Reader: Jialiang Yi, Zhihong Wang

Question 2(24 marks)
Let $G = (V, E)$ be an undirected graph. G is bipartite if the set of nodes V can be partitioned into two subsets V_0 and V_1 (i.e., $V_0 \cup V_1 = V$ and $V_0 \cap V_1 = \emptyset$), so that every edge in E connects a node in V_0 and a node in V_1. For example, the graph shown below is bipartite; this can be seen by taking V_0 to be the nodes on the left and V_1 to be the nodes on the right.

Note: G is bipartite if and only if every connected component of G is bipartite.

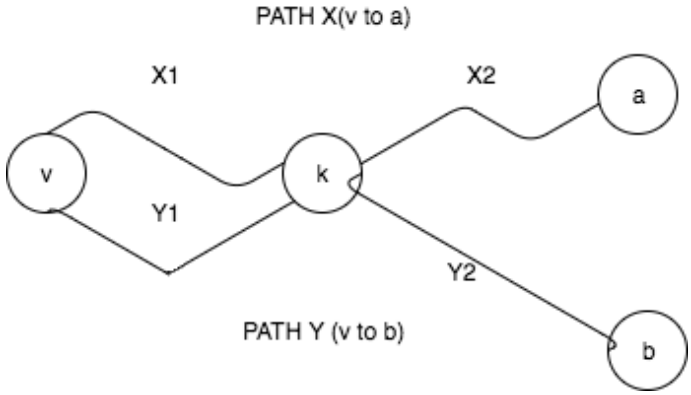
a. (8 marks) Prove that if G is bipartite then it has no *simple cycle* of odd length. Hint: Give a proof by contradiction.

Solution:
Assume there exists a bipartite undirected graph G such that it has simple cycle of odd length.
Let $\{v_1, v_2, ..., v_{2k+1}\}$ be vertexes of the simple cycle component of this graph G with odd length where v_i is connected with v_{i+1} by an edge. Also, v_{2k+1} is connected to v_1
Let V be set of vertexes of the graph and V_1, V_2 be the partition of V_0(i.e. $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \emptyset$)
Let $v_1 \subseteq V_1$ Then by the property of bipartite graph, $v_2 \subseteq V_2$, $v_3 \subseteq V_1$, ..., $V_2k \subseteq V_2$, $V_{2k+1} \subseteq V_1$
Since we get that $v_{2k+1} \subseteq V_1$ and v_{2k+1} is connected to v_1
. Also, v_{2k+1} and v_0 should not be contained by the same subset because they are connected by an edge, but both of them are element of the V_1. (i.e. v_1 and v_{2k+1} are connected by an edge and they are in the same subset of V), which contradicts that G is bipartite.
Therefore, if G is bipartite then it has no simple cycle of odd length.

b. (8 marks) Prove that if G has no simple cycle of odd length (i.e. every simple cycle of G has even length) then G is bipartite. (Hint: Suppose every simple cycle of G has even length. Perform a BFS starting at any node s. Assume for now that G is connected, so that the BFS reaches all nodes;

you can remove this assumption later on. Use the distance of each node from s to partition the set of nodes into two sets, and prove that no edge connects two nodes placed in the same set.)

Solution:
We will show that if G has no simple cycle of odd length then G is bipartite for both cases that G is a connected graph and G is not a connected graph.
Case 1: G is a connected graph
Let v be an aribtrary vertex in the graph G.
Let V_0 be the set that for all vertexes v_0 in V_0 where $d(v, v_0)$ is even.
Let V_1 be the set that for all vertexes v_1 in V_1 where $d(v, v_1)$ is odd.
Prove by contradiction:
Let a and b be two vertices such that both of them are from V_0 (both $d(v, a)$ and $d(v, b)$ are even) or both of them are from V_1 (both $d(v, a)$ and $d(v, b)$ are odd).
Let X be the shortest path from v to a , let Y be the shortest path from v to b and let k be the last vertex that X and Y intersects. if v is the last point they intersects, then v is k.
Since X is the shortest path from v to a and Y is the shortest distance from v to b, therefore, $X1 = Y1$ where $X1$ and $Y1$ are the distance form v to k. Consequently, $X2$ and $Y2$ have the same parity (both odd or both even) where $X1$ is $d(k, a)$ and $X2$ is $d(k, b)$. Thus, distance from a to b is $d(k, a) + d(k, b)$. Since the sum of two odd numbers or two even numbers is even, a to b is even.
Assume the edge ab exists, that a and b is directly connected, the length is 1. Then the cycle has length $1 + d(k, a) + d(k, b)$, which is odd, contradicts with the assumption. So, no such edge (a, b) may exist. Then, we can conclude that if G is a connected graph has no simple cycle of odd length, G is bipartite.



Case 2: G is not a connected graph.
If G is not connected, and it has no simple cycle of odd length. Then each connected component of G has no simple cycle of odd length, since each cycle is in one of the connected components. By the proof of the case 1, we can get that each connected component is bipartite. So, G is bipartite since the graph is bipartite if and only if every connected component of the graph is bipartite.

c. (8 marks) Describe an algorithm that takes as input an undirected graph $G = (V, E)$ in adjacency list form. If G is bipartite, then the algorithm returns a pair of sets of nodes (V_0, V_1) so that $V_0 \cup V_1 = V$, and $V_0 \cap V_1 = \emptyset$, and every edge in E connects a node in V_0 and a node in V_1; if G is is not bipartite, then the algorithm returns the string not bipartite. Your algorithm should run in $O(n + m)$ time, where $n = |V|$ and $m = |E|$. Explain why your algorithm is correct and justify its running time. (Hint: Use parts (a) and (b).)

Solution:

(i) Algorithm Description The way to solve this problem is applying the BFS. The input (precondition) of this algorithm is the Graph, the output (postcondition) of this algorithm is two seperated sets or the string ”not bipartite”. For each connected component of the graph, will add attribute ”1” to represent the subset for the first vertex we discovered and explored(i.e. assign it to group 1) in that connected component of the graph. For each vertex we explored, we get the extensions by the edge and check whether the vertex from the extension is discovered. If it has not been dis-

covered before, we will add the attribute ”1”(group 1 is assigned) if the recent vertex is in group 2, ”2” will be added(group 2 is assigned) if the recent vertex is in group 1. If the vertex of the extension has already been discovered from another vertex, we will check whether it is in the different group with the recent vertex. If both of them are in the same group, the string ”not bipartite” will be returned. Otherwise, it will assign a group for another extension. After the BFS has finished. Loop over all the vertexes. Put the vertex of group 1 and group 2 to the sets V_0 and V_1 respectively. Finally, return these two sets V_0 and V_1.

(ii) Show correctness
The input of the graph is bipartite or not bipartite(precondition).
Case 1: If the input is a bipartite graph, for each connected component, the first vertex of the graph will be assigned to group 1. The vertexes that discovered from the first vertex will be assigned to group 2. According to the condition of this case, the vertexes that discovered from the explored vertexes, will be assigned or has already been assigned a different group. After the BFS has finished, it will loop over all of the vertexes and put into two different set V_0 and V_1. These two sets will be returned at the end, which meets the postcondition.
Case 2: If the input is not a bipartite graph, when doing the BFS, there must exists a vertex that when it is explored, at least one of the vertexes that discovered from the recent vertex has already been assigned the same group as the current vertex, the process will be stoped and return the string ”not bipartite”, which meets the postconditon
Therefore, since the postcondition meets from the both cases of the precondition, the algorithm is correct.

(iii) Show Complexity
The complexity of the algorithm is O(n+m). Since the algorithm is based on the BFS, for each iteration we explore a vertex, it has additional steps, which is checking whether the group is assigned, and then assigned a group if it has not been assigned a group, or check whether they are in different group or not. They all take constant time, so it will not more complex than $O(n + m)$. Also, if the graph is bipartite, after the BFS it will put all the vertexes to two different sets, which take linear time. So, totally it is $O(n + m + n)$, which is same as $O(n + m)$.

Writer: Jialiang Yi

Reader: Zhihong Wang, Zhicheng Yan

Question 3(24 marks)
You are riding your bike and you have an aerial map of the routes you could take. This map contains the (x, y)-coordinates of n bike pump stations. There is a straight bikeway that goes directly between any two bike pump stations. You want to travel from station s to station t. Since your tires aren't very good, the best route from s to t is one that minimizes the distance that you have to travel between any two successive stations on this route. Your task is to design an $O(n^2 logn)$-time algorithm for finding a best route.

Note: Your algorithm can use any algorithms that we studied in the course as "black-boxes," and you can refer to their worst-case time complexity as stated in lecture or in the textbook.

a. Restate this task as a graph problem. To do so: (i) describe the graph: its vertices, edges, and edge weights, and (ii) restate the task as a graph problem on this graph.

Solution:

i) describe the graph: the graph is an aerial map of the routes you could take. Vertices are (x,y)-coordinates of n bike pump stations. Edges are straight bikeway that goes directly between two bike pump stations. Edge weights are representation of distance between successive stations.

ii) restate the task in term of a graph problem: Given an connected, weighted and undirected graph, find a path between station s and t, such that the distance between any two successive stations on this path is minimized.(NOTE: Not the total travel distance that is minimized)

b. We learned several algorithms that construct trees from an input graph. One these trees can be used to find the desired path efficiently. Explain how and prove it.

Solution: For this question, Prim's algorithm is used to find a minimal spanning tree of such graph, such that a desired path can be found efficiently. According to the definition of MST, a MST "spans" the graph without creating any cycle and whose weight between any successive vertices is minimized.

```
PRIM_MST(G=(V, E));
1.    A = {}
2.    for all v in V
3.        key[v] = infinity //smallest cost of v to enter tree
4.        p[v] = NIL //parent of v
5.    Initialize priority queue Q with all v in V
6.    Pick any arbitrary vertex r as root (binary min_heap)
7.    key[r] = o
8.    while Q is no empty:
9.        u = EXTRACT_MIN(Q)
10.       if p[u] != NIL:
11.               A = A + {(p[u], u)}
12.               for each neighbor v of u
13.                     if v in Q and w(u, v) < key[v]:
14.                        DECREASE_KEY(Q,V,W(u,v))
15.                        p[v] = u
16.   return A
```

According to CLRS, Prim's algorithm starts at some arbitrary vertex r and grows until the tree spans all the vertices in Graph V. By a property of Prim's algorithm, edges in the set A always form a single tree. At each stage, an edge of minimum weight among the edges joining nodes in A to nodes not in A. This algorithm qualifies as greedy since each step it adds to the tree an edge that contributes the minimum amount possible to the tree's weight. Consequently, a mst can be found such that distance of any two successive stations is minimized.(Not minimum in terms of total distance)

Claim: Prim's algorithm constructs a single tree where distance between two successive vertices are minimal. Prove by induction:

Base Case: Assume there are only two vertices in the graph, by the fact

that Prim's Algorithm satisfies greedy, the algorithm picks the minimum weight among the edges joining these two vertices together. It's obvious that the path between this two vertices are minimized. P(T) holds.

Inductive Step: Assume P(k) holds where a mst tree with k nodes are constructed using Prim's Algorithm such that distance between any two successive vertices is minimized. (IH). WTS P(k+1) holds. By IH, distance between two successive vertices in those k vertices are minimized. Now, joining a new vertex that is not in the tree, the algorithm will greedily chooses an edge of minimum distance among the edges joining the tree and the new vertex. Then a mst with $k+1$ vertices is constructed and every successive vertices have minimized distance, P(k+1) holds.

Therefore, Prim's algorithm can be use to find the desired path efficiently.

c. Describe a corresponding algorithm for solving this problem in plain and clear English.

Solution: The algorithm firstly uses Prim's algorithm to find a mst of the graph from station s to station t , referring to CLRS, Prim's algorithm has the property that the edges in the set always form a single tree; The mst construction starts from station s and begin to grow until station t is connected to the tree, stop immediately. In this mst, there exists a path from s to t such that the distance between every successive node on this path is minimized. Since Prim's algorithm satistises greedy algorith, such time when it is expanding, it will chooses the edge with shortest distance each time. Consequently, when it connects the last station t, each successive station has minimized distance.(Not overall distance) At the time the mst is growing, an array uses to store previous visited vertices will be argumented on each visited or visiting vertices. The array in current vertex includes all the previous added vertices and itself following time order. By the time the algorithm reaches t, all the vertices from s to t is stored in the array in order, indicating the path. Thus, the desired path is found and return.

d. Explain why the worst-case running time of your algorithm is $O(n^2 logn)$.

Solution: The algorithm firstly uses Prim's algorithm to construct a

mst of the graph; examine the psedocode of Prim's algorithm, a min-priority queue Q is implemented as a binary min-heap. We can use the BUILD-MIN-HEAP procedure to perform it in $O(V)$ times. The body of the while loop executes $|V|$ times and each EXTRACT-MIN operation takes $O(logV)$ time, which gives us the total time for all calls to EXTRACT-MIN is $O(VlgV)$ times. The for loop in $Line12$ executes $O(E)$ times in total. Within the for loop, all operations can be done in constant time, except DECREASE-KEY operation,which is can be done in $O(logV)$ time. Thus, the total time for Prim's algorithm is $O(VlogV + ElogV) = O(ElogV)$ times. Let $|V| = n$ and $|E| = m$, we have runtime $O(mlogn)$. As we argument an array on each vertex to record the visited vertices, the path is done in constant time. After all, the dominate pieces is Prim's algorithm, which is $O(mlogn)$. According to lectures, tutorial and CLRS, $m \leq n^2$, therefore, the worst-case running time of this algorithm is $O(n^2logn)$.