CS 577: Introduction to Algorithms	Homework 3
Out: 02/16/21	**Due: 02/23/21**

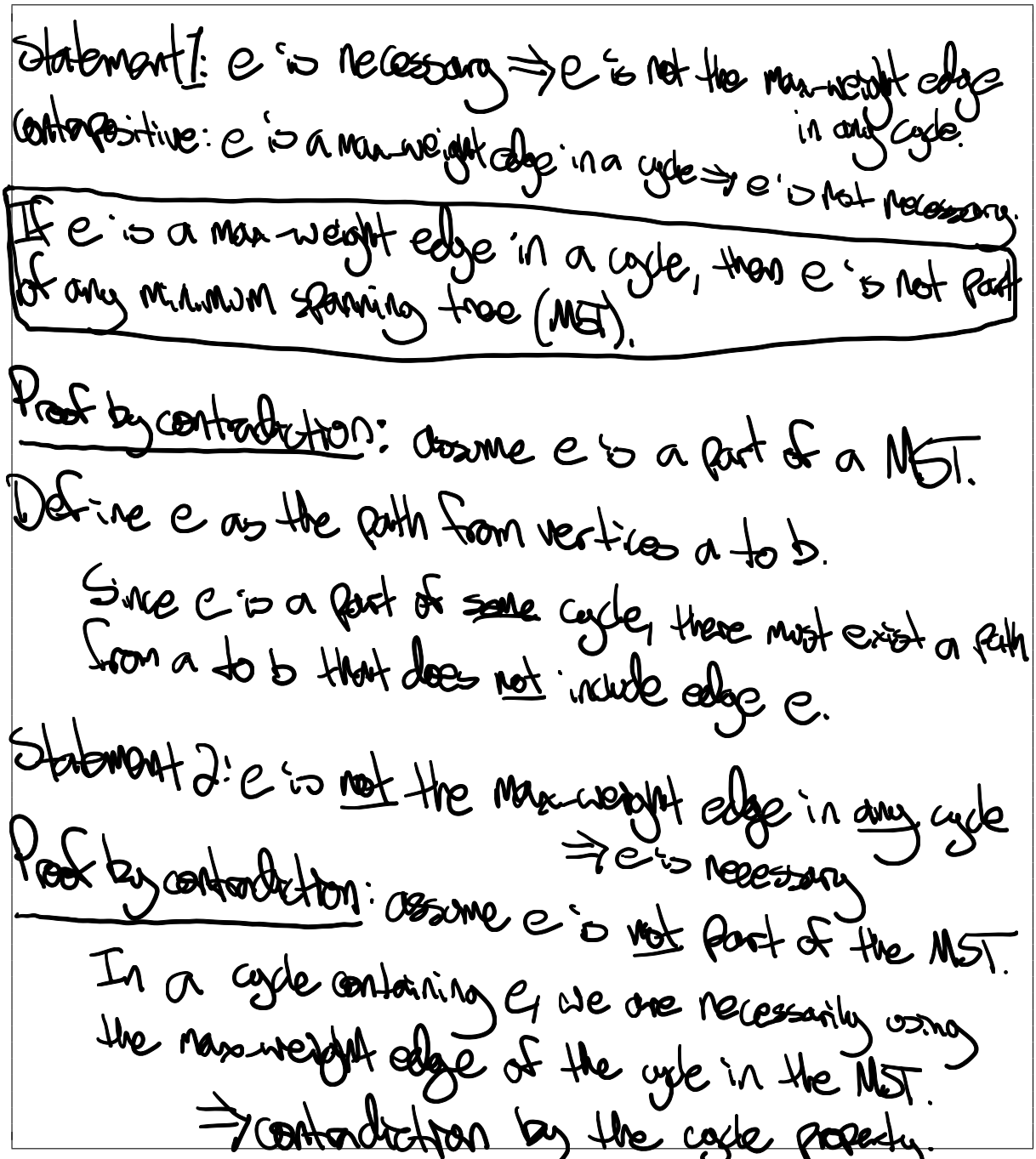
Name: Aiden Tepper Wisc ID: 9081242169

Ground Rules

- Answer the questions in the boxes provided on the question sheets. If you run out of room for an answer, add a page to the end of the document. **Since students have had issues fitting their answers into the boxes, we are increasing the size of the boxes for this problem set. However, do NOT feel obligated to fill the entire solution box. The size of the box does NOT correspond to the intended solution length.**
- The homework is to be done and submitted individually. You may discuss the homework with others in either section but you must write up the solution *on your own*.
- You are not allowed to consult any material outside of assigned textbooks and material the instructors post on the course websites. In particular, consulting the internet will be considered plagiarism and penalized appropriately.
- The homework is due at 11:59 PM CST on the due date. No extensions to the due date will be given under any circumstances.
- Homework must be submitted electronically on Gradescope.

Problem 1:

1. Let $G = (V, E)$ be an undirected weighted graph with distinct weights w_e for every edge $e \in E$. Let $T(V, E)$ denote the edge set of the unique MST of (V, E). Let $W(T) = \sum_{e \in T} w_e$ be the weight of the edge set T. We say an edge e is "necessary" for the Minimum Spanning Tree $T(G, E)$ if when we delete it, the weight of the MST increases. So we have that $W(T(V, E \setminus \{e\})) > W(T(V, E))$). Show that an edge e is necessary if and only if for every cycle C in G that contains it, e is not the maximum weight edge in this cycle (so there exists $e' \in C$ such that $w_{e'} > w_e$).



Problem 2:

2. You're trying to run through an obstacle course, but there's a twist. Each step on the course is marked either "left" or "right". You're only allowed to step on squares labeled "left" with your left foot, and likewise for your right foot. Of course, you can decide which foot to take your first step with. Is it possible for you to run from your starting location to a target location while obeying the course constraints?

   You are given an undirected graph $G = (V, E)$ in the form of an adjacency list, a start node s, a target node t, and a partition of E into two sets L and R. Call a path "feasible" if it alternates between edges in L and edges in R.

   (a) Give a linear time (i.e. $O(n + m)$) algorithm to compute the shortest feasible path from s to t or determine that no such path exists. Show your algorithm is correct and achieves the desired runtime.

input: undirected graph G, start node s, target node t, sets L and R

output: the shortest feasible path, or indication that no such path exists

```
find_path:
    queue := []
    visited := array of False * length of G
    for node in G:
      dist[node] := ∞
      prev[node] := -1
    visited[s] := True
    dist[s] := 0
    queue.append(s)
    while length of queue != 0:
        v = queue[0]
        queue.pop(0)
        feat := L if v in L set, R if in R set
        for node in adjacency list for v:
            if visited[node] == False and feat != feat of node:
                dist[node] := dist[v] + 1
                prev[node] := v
                queue.append(node)
                if (node == t):
                    return prev
    return -1
```

This algorithm is correct and has runtime of O(n+m) because it is a basic and slightly modified version of a BFS algorithm (only difference is that it checks that the feat alternate). BFS has runtime O(n+m) for an adjacency list, and the algorithm returns the correct path and terminates if one is found, otherwise returning -1.

(b) You've decided that you'd be willing to hop, stepping with the same foot twice in a row, if it shortens your path. However, you aren't very coordinated so you can only hop once during the run. Call a path "1-feasible" if it alternates between edges in L and R, with the possible exception of a single pair of consecutive edges in either L or R. Note that feasible paths are necessarily 1-feasible. Give a linear time (i.e. $O(n+m)$) algorithm to compute the shortest 1-feasible $s-t$ path in G or determine that none exists. Show your algorithm is correct and achieves the desired runtime.

input: undirected graph G, start node s, target node t, sets L and R

output: the shortest feasible path, or indication that no such path exists

```
find_path:
    queue := []
    visited := array of False of length of G
    for node in G:
        dist[node] := ∞
        prev[node] := -1
    visited[s] := True
    dist[s] := 0
    queue.append(s)
    hopped := False
    while length of queue != 0:
        v = queue[0]
        queue.pop(0)
        feet := L if v in L set, R if in R set
        for node in adjacency list for v:
            if visited[node] == False and feet != feet of node:
                dist[node] := dist[v]+1
                prev[node] := v
                queue.append(node)
                if (node == t):
                    return prev
            one: if visited[node] == False and hopped == False:
                hopped := True
    return -1
```

```
dist[node] := dist[v]+1
prev[node] := v
queue.append(node)
if (node == t):
    return prev
```

Same as problem 2a

This algorithm is correct and has runtime of $O(n+m)$ because it is a basic and slightly modified version of a BFS algorithm (only difference is that it checks that the feet alternate). BFS has runtime $O(n+m)$ for an adjacency list, and the algorithm returns the correct path and terminates if one is found, otherwise returning -1.

(c) Suppose there are also some edges that you are allowed to step on with either foot. Give a one sentence explanation of how to modify your algorithms to work with these new edges.

I would create a separate list for edges that can be stepped on by either foot, then in the first if statement in the while loop add the code "or if node is in both feet list" to make sure they're accounted for.