Algorithms
Graph Modeling Project

A detailed description of the maze problem including a figure is provided separately[1]. You may want to look at that first to get a sense of what this project involves! Your assignment is to (1) model the maze problem as a graph (2) use an appropriate graph algorithm that we've encountered in class to solve the problem (3) code the algorithm and (4) submit your program to Gradescope for verification. **Friendly reminder: This is an individual assignment. The departmental collaboration policy (the document you signed at the start of the semester) will be enforced.**

1 Deliverables

The deliverables are (1) a report, submitted to Gradescope, and (2) submission of your program to Gradescope, where it will be verified through several test cases. Your report must include the following items:

1.1 Problem Modeling [50 pts]

- [30] Explain how you modeled the problem as a graph (typically, a few sentences). Draw the entire resulting graph for the following instance (assume the start is A and the finish is G):

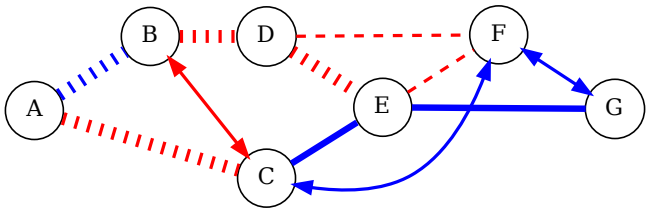

Figure 1: Transit map maze instance. Draw the entire graph model.

- [5] Identify the graph algorithm needed to solve the problem (one sentence). Note: your graph model should encode the complexities of the maze, that is, the algorithm that you have selected should solve the problem if it is run completely unmodified on the graph that you created in the first part.
- [15] Convince us that this algorithm will actually solve the problem. This means that it will find a path if one exists (typically, a few sentences, depending on your approach). Make a convincing and complete argument (similar to a proof) that your algorithm will solve the problem.

[1]The problem was taken from "MAD MAZES: Intriguing Mind Twisters for Puzzle Buffs, Game Nuts and Other Smart People," by Robert Abbott, Bob Adams, Inc. Publishers, 1990.

- Code submission: please include your code in an appendix. It should solve the problem using the algorithm that you have mentioned in the previous section; and should be similar to your last submission on Gradescope. **If these three do not match, we may deduct (some or all) verification points.**
- As mentioned in class, you **must** modify the graph, and not the algorithm that you choose. Modifying the algorithm against instructions will result in a **minimum deduction of 25 points** from your overall score on this project.

1.2 Extra Credit [5 pts]

You will receive a bonus of up to 5 points if you show that you have done something above and beyond what was asked. Possible examples might be (1) using an algorithmic library (e.g., networkx in Python 3) to implement the graph functionality, (2) augmenting your algorithm with a visualization, (3) an experimental comparison between algorithms, etc. *Please explicitly include* a section in your report titled "Extra Credit." Don't just assume the TA will notice your enhancements.

1.3 Results [50 pts]

There are two assignments on Gradescope for verification.

- The Maze Verification Testing assignment is not worth any points on Canvas, but allows infinite submissions. Submit to this assignment first to make sure that you fix any submission-related bugs or typos. It also has some smaller test cases with timing results that you can use to get an idea of your program's run time on Gradescope compared to your computer. While this will have some edge cases, it may not have all edge cases, and the inputs will not be of maximum size.
- The second assignment, Maze Verification, is worth 50 points. Expect extensive testing - edge cases, large maze sizes (up to 50,000 villages and transit lines), etc. There are three submission attempts. Passing 100% of test cases on the first submission will be awarded with 5 extra credit points for a 55/50. Passing 100% of test cases on the second submission will be awarded with 2.5 extra credit points for a 52.5/50. To be clear, this extra credit is only awarded if all tests are passed; failing (or exceeding the time limit) on even only one test = no extra credit. This extra credit is separate from the extra credit given above, meaning a total of 10 extra credit points is possible on this project. The third submission does not have any associated extra credit. Beyond the third submission, a 20% deduction will be applied for each additional submission attempt. **We will not be increasing the number of penalty-free submissions to more than three.**

2 Input

The first line of the input file contains two integers and two strings. The integers are the number of villages $2 \leq V \leq 50000$ and the number of transit lines $1 \leq T \leq 50000$, respectively. The two strings denote the starting village and the ending village (Startsburg and Endenville in the example).

For the example provided from Abbot's Mad Maze book, there are 36 villages labeled $A, B, C, \ldots, X, Y, Z, a, b, \ldots, i, j$ and 70 transit lines. For inputs with more than 52 villages, you can expect the naming convention to begin to repeat alphabetically. For example, z would be followed by $AA, AB, AC, \ldots, AY, AZ, Aa, \ldots, Ay, Az, BA, BB, \ldots, zy, zz, AAA, AAB$, etc. Each subsequent line describes a transit line: it includes the two villages joined by the line, the color of the line (R, G, B) and the *type* of the transit line (H, C, T, B) for Horse, Cable car, Trolley, and Bus, respectively. For example, the second input line indicates that the first transit line joins villages A and B, using the Red company, and that this is a Cable Car. The entire input file is included below.

```
36 70 A j
B A R C
B E B C
B C B T
C D G T
D J R T
B F R T
C G R C
D G B B
D I G B
E F G H
F G G H
G H R C
H I R H
I J B H
E O R C
E K G B
F K R H
G K B B
G L G T
H L R B
I M G T
I N G H
J N R B
L M B B
M N B H
K O B T
K P R C
L Q G T
L R R H
M S R T
N T R B
P Q G C
Q R B C
R S B H
S T B B
O U R B
O V B H
```

```
P V G H
Q W R T
R X R T
S X G B
S d R H
S Y G C
T Y G T
U V G B
V W R C
W X G C
U Z B B
V Z R C
V a G B
V b B T
W b R T
X c R H
Y e G T
Z a R T
a b R C
b c G T
c d G B
d e G H
Z f B H
a f G H
b g R T
b h B B
c h R C
d i R H
e i R B
f g B C
g h G C
h i R B
i j B B
```

3 Output Format

The output should consist of the **shortest** path from Startsburg (Village A) to Endenville (Village j). The path should be presented as a space-delimited sequence of village ids. For example, one possible path may start off as follows:

```
A B F K P Q R
```

If there are multiple shortest paths, output the path that occurs first lexicographically, uppercase before lowercase. For example, if we had two shortest paths `A Z C D` and `A a B D`, Z comes before a, so we would output the path `A Z C D`.
If there is no path from Startsburg to Endenville, then output "NO PATH"