CSCI 311 - Algorithms and Data Structures
Project 3

April 17, 2022

In this project, we will be implementing Dijkstra's algorithm and using it to solve a small variation on the shortest path problem. As in the first two projects, there is very little direction regarding how to design or structure your code. These decisions are, mostly, left to you. Make sure to **start early** and to **stay organized**. This project is a little smaller than previous projects and is worth a total of 100 points.

C++ code for this project should be submitted on Canvas and turnin. The project is due before **May 7th at 11:59 pm**. All code and tests should be included in a **.zip file** called `Project3`. Your main should be written in a file called `project_3.cpp`. Remember, coding style and comments matter. Poor style or a lack of comments may cost you points!

There will be time in lab to discuss these problems in small groups and I highly encourage you to collaborate with one another outside of class. However, you must write up your own solutions **independently** of one another. Feel free to communicate via Discord and to post questions on the appropriate forum on Canvas. Do not post solutions. Also, please include a list of the people you work with in a comment at the top of your submission.

Good luck!

1 The Problem

You are a software engineer at a nameless electric vehicle company working as part of a team on a tool for helping customers plan road trips. Charging stations are not yet as widely available as gas stations so, when determining a route from the starting point to the destination, the range of the vehicle must be taken into account. With summer approaching, your team is scrambling for a solution to this problem. Using your knowledge of graphs and shortest path algorithms, propose and implement an efficient algorithm for determining the shortest path between two cities paying attention to vehicle range and available charging stations.

2 Input and Output

- Input will come from cin.
  - The first line will contain four space separated integers, n, m, c, and i.

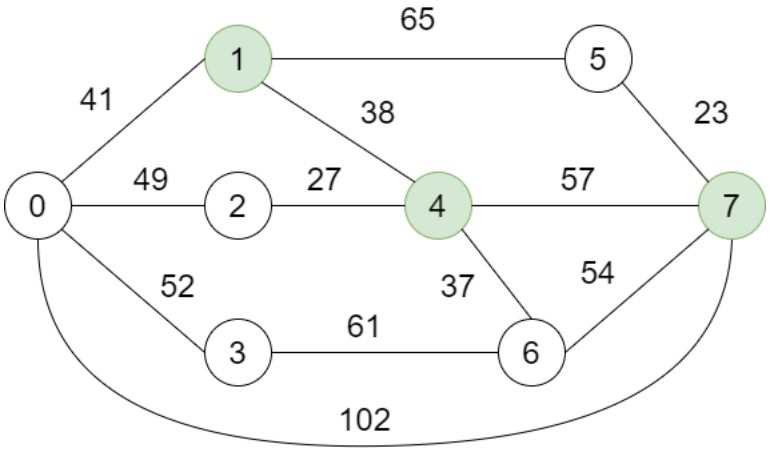
    - n is the number of nodes in the graph.
    - m is the number of edges in the graph.
    - c is the maximum distance in miles the vehicle can travel on a full charge.
    - i is the range of the vehicle with its initial charge.
  - The second line contains a pair of integers, *start* and *end*, indicating the start and end nodes for the trip.
  - The following n lines contain pairs of space separated integers, v and s, representing nodes.
    - v is the name of a node. This will be an integer between 0 and $n - 1$.
    - s is 1 if the node v has a charging station and 0 otherwise.
  - The following m lines each contain three space separated integers, u, v, and d, representing edges.
    - u and v are edge endpoints.
    - d is the distance in miles between nodes u and v.

- Print output to cout.
  - The output will have two lines.
  - The first line should be "Verify Path: 1" when a suitable path exists.
  - The second line should be the length of the trip followed by a colon, the *start* state, the sequence of charging stations used on the trip, and the *end* state.
    - This sequence should be space separated, should start with *start*, and should end with *end*.
    - This sequence should represent a shortest path given the constraints.
  - If no suitable path exists, print "No suitable path from *start* to *end* exists" where *start* and *end* are node ids.

For example, the graph below corresponds to the input:

```
8 12 80 80
0 7
0 0
1 1
2 0
3 0
4 1
5 0
6 0
7 1
0 1 41
0 2 49
0 3 52
0 7 102
1 5 65
1 4 38
2 4 27
3 6 61
```

```
4 6 37
5 7 23
4 7 57
6 7 54
```



Output in this case should be:

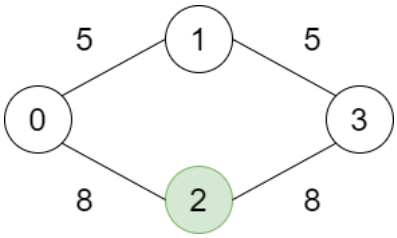
```
Verify Path: 1
133: 0 4 7
```

3 Testing

Similar to project 2, you will need to test your code to make sure that it is working as expected. Along with `.cpp` and `.h` files, you should submit at least 5 tests with expected outputs. Include pictures of these graphs along with short descriptions of what they test in a PDF. For example, you might include a test file "test_1.in" that looks like this:

```
4 4 9 9
0 3
0 0
1 0
2 1
3 0
0 1 5
0 2 8
1 3 5
2 3 8
```

corresponding to the graph



Expected output in "test_1.out" would then be:

```
Verify Path: 1
16: 0 2 3
```

A potential description of this test might be "Verifying that a longer path is chosen when the vehicle would run out of charge on a shorter path".

Think carefully about important scenarios that are worth testing! You may wish to include more than five test cases.

4 turnin

Note that turnin expects a single file called `project_3.cpp` to be submitted. Any classes you implement, including graph, node, and priority queue classes, should be included in this single file. This is to accommodate submissions with different class structures. Your code submitted to Canvas **should not** be organized in this way. Separate classes should have their own files.

5 Requirements and Grading

Your solution must include an implementation of Dijkstra's algorithm and you must implement your own graph class. This can be the same as the class used in labs 7 and 8. You are free to use other data structures available in the C++ standard library like queue and priority_queue. However, since Dijkstra's algorithm requires that the priority queue be updated when distances change, it may be easiest to use your own priority queue class.

Grading for this project will be broken down as follows:

- (45 pts) An implementation of Dijkstra's algorithm.
- (20 pts) A solution to the given problem.
- (10 pts) A function `bool verifyPath(Graph G, vector<int> path, int i, int c)` which, given a graph, a vector of node ids representing a path from the start node to the destination, the initial charge of the vehicle, and the maximum charge of the vehicle, returns `true` if the vehicle can make the trip following the given path and `false` otherwise.
- (15 pts) Well thought out test cases designed to assess specific aspects of your code.
- (5 pts) Comments.
- (5 pts) Organization.