ASSIGNMENT 5: 8-PUZZLE

Goal: The goal of this assignment is to get some practice with shortest path algorithms.

Problem Statement: Most of you may have played an 8-puzzle game in childhood. The game consists of a 3x3 grid with 8 tiles numbered 1 to 8. There is one gap in the puzzle that allows movement of tiles. Tiles can move horizontally or vertically. Here may be one starting configuration of an 8-puzzle:

1	2	3
4	6	8
7	5	

We will use the word "state" to refer to each configuration of the puzzle. We can call the configuration above as the "start state". Typically, our goal is to reach this configuration (goal state)

1	2	3
4	5	6
7	8	

One possible solution to reach the target in this example is by moving 8 down, 6 right, 5 up and then 8 left. The task in this assignment is to find this shortest path using Djikstra's algorithm.

To make a twist in the assignment, we will play a cost-version of the game. Here you will be given a cost function described by 8 integers, d_1 … d_8, such that moving the tile numbered *i* will cost d_i units. If $d_1 = 1$ and $d_2 = 7$, then you can move 1 seven times to cost equal to moving 2 once. You will be given several test cases, and each test case will consist of a start state, a goal state, and a cost function. Your task is to find the cheapest path to the goal. If multiple paths exist with the same minimum cost, print the path with fewest moves. If multiple optimal paths to the goal exist which take the fewest moves, then you can print any of them.

We can easily represent an 8-Puzzle configuration using a 9-character string if we call the GAP G. For example, the first configuration above will look like 12346875G.

Input File: The input filename will be given as a system argument when the code is run. The first line will consist of an integer T denoting the number of testcases in the file. T will be <= 200.

The first line of each test case consists of two space separated strings representing the start state and the goal state. The second line of a test case consists of 8 non-negative integers, d_1 … d_8 representing the cost function ($0 \le d_i \le 1000$). It is guaranteed that both the start state and the goal state are permutations of "12345678G".

1	2	3
4	5	6
	7	8

A sample input file is as follows:

```
4

12346875G 12345678G

1 2 3 4 5 6 7 8

12346875G 123456G78

0 0 0 0 0 0 0 0

12346875G 1b2346875G

1 2 3 4 4 3 2 1

12346875G 12346857G

1 2 3 4 5 6 7 8
```

Output File: Your code should generate an output file, whose name will be given as a system argument during runtime. For each test case, write two lines. In the first line you should write a line with two integers `*n*` and `*d*`, the number of moves in the optimal path to reach the goal state from the start state, and the cost of the optimal path. If goal state is not reachable then print "-1 -1" (without quotes).

In the next line, if a path exists, then write `n` space separated tokens to describe the moves in your optimal path. Each token consists of an integer from 1 to 8 and a character in {U, D, L, R} which describes which number was moved in the position of the gap, and the direction it moved. For example, suppose your code computes solution "8 down, 6 right, 5 up, 8 left" (4 moves, cost 27) for first test case and "8 down, 6 right, 5 up, 7 right" (4 moves, cost 0) for the second test case (and so on), you will output the following output file. If the path is of length 0, or it doesn't exist, leave this line blank.

```
4 27

8D 6R 5U 8L

4 0

8D 6R 5U 7R

0 0

<blank line>

-1 -1

<blank line>
```

Method: First, compute the graph of the 8-Puzzle game. The set of vertices will be all states and edges will denote all possible valid moves. Then use Djikstra's to find the cheapest path.

Code: Your code must compile and run on GCL machines. Your code will be run using the following command:

```
javac Puzzle.java

java Puzzle someinputfilename.txt someoutputfilename.txt
```

What is being provided?

We will be providing you the following tools to help check your code.

1. formatChecker.py: This takes an input file and an output file and verifies that all the paths from the start state to goal state in the output file are correct, and the cost is calculated correctly. It does NOT check if the paths are optimal and if the path exists. (Usage: python formatChecker.py input.txt output.txt)
2. randomMoves.py: This takes an integer n as argument. It randomly selects a board, and performs n random moves on it. It will print the start state, the end state and the moves that it followed. You may use this to generate testcases for your self.
3. uniform_cost_optimal.txt: This is the optimal cost to all the states from [[1, 2, 3], [4, 5, 6], [7, 8, _]] assuming that each step had cost 1, i.e. the cost function is uniform. More testcases will be posted on piazza soon.

Note that the python code is written so that it is compliant with both python2 and python3. It has been tested on Linux and Mac OS X, but not on Windows.

What to submit?

1. Submit your code in a .zip file named in the format **<EntryNo>.zip.** Make sure that when we run "unzip yourfile.zip" in addition to your code "writeup.txt" should be produced in the working directory.

   You will be penalized for any submissions that do not conform to this requirement.
2. The writeup.txt should have a line that lists names of all students you discussed/collaborated with (see guidelines on collaboration vs. cheating on the course home page). If you never discussed the assignment with anyone say None.

   After this line, you are welcome to write something about your code, though this is not necessary.

Evaluation Criteria

This assignment is worth 4.5 points. Your code will be autograded BEFORE the demo against a series of tests, so please correctly follow the input-output format. Out of this, 2 marks are for passing the tests, 1.5 marks for runtime efficiency, and the remaining 1 mark is for the demo.

What is allowed? What is not?

1. This is an individual assignment.

2. Your code must be your own. You are not to take guidance from any general purpose code or problem specific code meant to solve these or related problems.

3. You are not allowed to use built-in (or anyone else's) implementations of heaps or graphs or Djikstra's. A key aspect of the assignment is to have you learn how to implement graphs and heaps (if you need them).

4. You are allowed to use built-in Java String functions, hash functions, queues, stacks, vectors, and linked lists. If you wish to use another built-in method, please ask on Piazza, but the key insight is that everything we have tested you before on can be used built-in.

5. There are many (more efficient) solutions to this assignment than Djikstra's. But, you are not allowed to use any other algorithm.

6. You should not Google search for direct solutions to this assignment. However, you are welcome to Google search for generic Java-related syntax.

7. You must not discuss this assignment with anyone outside the class. **Make sure you mention the names in your write-up in case you discuss with anyone from within the class.** Please read academic integrity guidelines on the course home page and follow them carefully.

8. Your submitted code will be automatically evaluated against another set of benchmark problems. You get significant penalty if your output is not automatically parsable and does not follow input-guidelines. Note that given the end of semester, we won't have enough time to deal with everyone's individual issues, so please be careful in submitting your code.

9. We will run plagiarism detection software. Anyone found guilty will be awarded a suitable penalty as per IIT rules.