Homework 4
COMS 311 Points: 250
Due: Oct 6, 11:59PM
Late Submission Oct 7, 11:59PM (20% penalty)

Learning outcomes.

Design, implement and evaluate algorithms following specifications.

0 Preamble

Description of a programming assignment is not a linear narrative and may require multiple readings before things start to click. Y   or any
questions/clarificatio   n of)
Java.

1 Problem Description

You are given a 2-D rectangular grid in which a robot is supposed to move fr   another. The grid contains n rows and m columns, with row values in the range $[0, n-1]$ and column values in the range $[0, m-1]$. Each location in the grid is defined by the value of row and column. For instance, the row-column pair $(5, 2)$ is the address of the location with row value 5 and with column value 2. The top-left corner of the grid has the location $(0, 0)$. Locations outside the range of the row and column values do not exist.

The grid includes certain locations that are marked to include obstacles.

The robot at location (i, j) can make *direct* moves in 4 possible directions: north (n) to a location $(i-1, j)$ (if the location exists and does not contain obstacle); south (s) to a location $(i+1, j)$ (if the location exists and does not contain obstacle); west (w) to a location $(i, j-1)$ (if the location exists and does not contain obstacle); east (e) to a location $(i, j+1)$ (if the location exists and does not contain obstacle). The robot can make one or more direct moves to go from one location to another.

You are also given two specific locations: the *start location S* and the *destination location D*. An illustration of the grid is as follows (locations with obstacles are presented as *, start location is S, destination location is D, all other locations are marked 0):

```
0   0   0   *   0   0   0   0   0   0
0   0   S   *   0   0   0   0   0   0
0   *   *   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   *   0   0   0
0   0   0   0   0   0   *   0   0   0
0   0   0   0   0   0   *   0   0   0
0   0   0   0   0   0   *   0   0   0
0   0   0   0   0   0   0   D   0   0
0   0   0   0   0   0   0   0   0   0
```

A plan is a sequence of moves, which can be used to move the robot from S to D (avoiding the obstacles). Your objective is to identify such a plan or plans. For instance, for the above grid, a possible plan can be illustrated as:

```
0   0   0   *   0   0   0   0   0   0
s   w   s   *   0   0   0   0   0   0
s   *   *   0   0   0   0   0   0   0
e   e   e   e   e   e   e   s   0   0
0   0   0   0   0   0   *   s   0   0
0   0   0   0   0   0   *   s   0   0
0   0   0   0   0   0   *   s   0   0
0   0   0   0   0   0   *   s   0   0
0   0   0   0   0   0   0   D   0   0
0   0   0   0   0   0   0   0   0   0
```

It states that starting from S, the robot can move to a location with non-zero value (other than obstacle) and at each location, it can follow the direction of move to eventually reach D. In the above illustration, from S (location $(1,2)$), the robot moves to location $(1,1)$, from where it needs to move *west* (w). By moving west, the robot comes to location $(1,0)$, from where it needs to move *south* (s).

There are many different plans that can move the robot from S to D.

2 Encoding Description

You are required to write a Java program that takes as input the 2-D grid information, as presented above, and outputs plan(s), if one exists. The input will be provided in a file; the file will be present in the same folder (working directory if you use eclipse IDE) as your program's executable and the name of the file will be an input argument for your executable. The file will contain information about the number of rows and columns in the grid, the start and the destination locations of the robot and the obstacle locations. For instance, the above grid will be encoded as

```
nrows 10 ncols 10
start 1 2
dest 8 7
obstacles
2 2
1 3
0 3
2 1
4 6
6 6
5 6
7 6
```

(Each element in the text file is separated by a single space.)

You are required to write the following:

1. A class `RobotPath`. You can decide what attributes/properties should be part of the class.

2. A method in `RobotPath` with signature

   `public void readInput(String FileName) throws IOException`

   The objective of th   , start/destination
   and obstacle infor

3. A method in `Ro`

   `public void planShortest()`

   The objective of this method is to use the input grid and identify a
   The length of a plan is defined by the number of moves made the robot. The plan with
   minimal length is a shortest plan. Note that, there may be more than one shortest plan.

4. A method in `RobotPath` with signature

   `public void quickPlan()`

   The objective of this method is to use the input grid and identify a plan using the following
   strategy:

   (a) *No Back-paddling.* the plan must not include directives that will make the robot to go
       through the same location more than ones.

   (b) *Predictive Selection.* for any location, the plan will consider the moves in terms of how
       the close to D the robot is estimated to be after the move. The estimation is based on
       the *straight-line* or Euclidean distance from a location that is the destination of a move.
       The distance between two locations (x_i, y_i) and (x_j, y_j) is computed by

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

If there are two locations at same distance from D, then location with smaller value for the row will be considered to be closer D; otherwise, the location with smaller value for the column will be considered to be closer to D.

For instance, in our running example, if for the location $(3,3)$, there are 4 possible moves (e, w, s and n). These moves will be considered for exploration in the following order: s, e, w, n; as the location to the south of $(3,3)$ is closest to D, and then the location to the east of $(3,3)$, and so on. In other words, the algorithm will first consider the move to south from $(3,3)$ to look for a plan; if no plan is found using that choice, then the algorithm will consider the move to east from $(3,3)$ to look for a plan; and after than will consider the move to west and finally, the move to north.

(c) *Quick Stop.* The search for a plan must terminate as soon as a plan is found.

5. A method in `RobotPath` with signature

```
public void output()
```

The objective of this method is to output the plans generated by either `planShortest` or `quickPlan` methods. In other words, you can assume that this method will be only invoked after `planShortest` or `quickPlan`. For instance, for the above example,

```
planShortest();
output();
```

will produce the following

```
 O    O    O    *    O    O    O    O    O    O
 s    w    s    *    0    0    0    0    0    0
 s    *    *    0    0    0    0    0    0    0
se   se   se   se   se   se    e    s    0    0
se   se   se   se   se    s    *    s    0    0
se   se   se   se   se    s    *    s    0    0
se   se   se   se   se    s    *    s    0    0
se   se   se   se   se    s    *    s    0    0
 e    e    e    e    e    e    e    D    0    0
 0    0    0    0    0    0    0    0    0    0
```

Note that, some of the grid locations have more than one directive for the robot plan. For instance, location $(3,0)$ has the directive that the robot can move south or east—each directive is part of some shortest plan.

On the other hand, in our running example, the output after

```
quickPlan();
output();
```

```
    O     O     O     *     O     O     O     O     O     O
    s     w     S     *     O     O     O     O     O     O
    s     *     *     O     O     O     O     O     O     O
    e     e     e     s     O     O     O     O     O     O
    O     O     O     e     s     O     *     O     O     O
    O     O     O     O     e     s     *     O     O     O
    O     O     O     O     O     s     *     O     O     O
    O     O     O     O     O     s     *     O     O     O
    O     O     O     O     O     e     e     D     O     O
    O     O     O     O     O     O     O     O     O     O
```

The methods `planShortest` or `quickPlan` may not find any plan due to the presence of obstacles in the input grid. In such scenario, the output should just produce the input grid. For instance,

```
    O     *     O     *     O     O     O     O     O     O
    O     *     S     *     O     O     O     O     O     O
    O     O     *     O     O     O     O     O     O     O
    O     O     O     O     O     O     O     O     O     O
    O     O     O     O     O     O     *     O     O     O
    O     O     O     O     O     O     *     O     O     O
    O     O     O     O     O     O     *     O     O     O
    O     O     O     O     O     O     *     O     O     O
    O     O     O     O     O     O     D     O     O     O
    O     O     O     O     O     O     O     O     O     O
```

3 Output Formatting

As the assessment of your submission will be conducted on the output, which has a strict format, it is necessary to adhere to the format. Submissions that produce incorrectly formatted output are incorrect. To match the output correctly, use the following directive for all grid-cell outputs (in addition to appropriate row and column format):

```
System.out.printf("%5s", <string-to-output-in-a-cell>);
```

4 Submission Requirements

1. You shall use default package. Though it is not recommended in practice, organize your classes as follows:

   (a) Prepare a file `RobotPath.java`. In this file you will write the specified class and its methods, and other helper classes. You can have as many helper classes and methods as you want.

   (b) Prepare another file with any name suitable for your setup which only contains the `main` method and declares the object of type `RobotPath` and invokes it methods. You can use the following sample file containing the main method.

```
import java.io.IOException;
public class HW4Test {
    public static void main(String[] args) throws exception {
        RobotPath rPath = new RobotPath();
        rPath.readInput(args[0]);

        System.out.println("\n planShortest:\n");
        rPath.planShortest();
        rPath.output();

        System.out.println("\n quickPlan:\n");
        rPath.quickPlan();
        rPath.output();
    }
}
```

Assume that the running example is present in a file named Grid.txt, then for a correct implementation of `RobotPath`, it (`javac HW4.java; java HW Grid.txt`) should output on the console two grids with plans (a sample file is provided).

You are required to submit the RobotPath.java file and nothing else. (Double check that all necessary helper classes are present in this file.)

5 Postscript

1. You must follow the given specifications. Method names, classna
   types. Any discrepancy may result in lower than expected gr   .

2. In the above problems, there are several data structure/organi
   decide. Do not ask questions related to such data structure/organization. Part of the exercise to understand and assess a good way to organize data that will allow effective application of methods/algorithms.

3. You will have to think about how to model the problem into a graph-based problem and apply your knowledge of graph algorithms to address the original problem. Do not ask questions about how to model the problem as a graph-based problem and/or what graph algorithms to use. Part of the exercise is to understand and assess a good way to represent/reduce a problem to a known problem for which we know an efficient algorithm.

4. Start reading and sketching the strategy for implementation as early as possible. That does not mean starting to "code" without putting much thought on what to code and how to code it. This will also help in resolving all doubts about the assignment before it is too late. Early detection of possible pitfalls and difficulties in the implementation will help in reducing the finishTime-startTime for this assignment.

5. Both correctness and efficiency are important for any algorithm assignment. Writing a highly efficient incorrect solution *will* result in low grade. Writing a highly inefficient correct solution

may result in low grade. In most cases, the brute force algorithm is unlikely to be the most efficient. Use your knowledge from lectures, notes, book-chapters to design and implement algorithms that are correct and efficient.

6. Test your code extensively (starting with individual methods). Your submission will be assessed using test cases that are different from the ones provided as part of this assignment specification. Your grade will primarily depend on the number of these test cases for which your submission produces the correct result.

Assignment Project Exam Help

https://eduassistpro.github.io/

Add WeChat edu_assist_pro