Programming homework 1

Here is our first programming homework, on implementing a solver for a simplified version of a Flash game named "Bloxorz", using BFS on a graph that is not given explicitely.

Problem description

Bloxorz is a game in Flash, which you can access here. As a first step for this assignment, play it for a few levels.

The objective of Bloxorz is simple; you must navigate your rectangular block to the hole at the end of the board, by rolling it, in the fewest number of moves possible. A block can be moved in four possible directions, left, right, up, down, using the appropriate keys on the keyboard.

Program behavior

Your task is to write a program that can solve a simplified version of this game, with no orange tiles, circles or crosses on the terrain. The input to your program will be a text file describing a terrain with a start position and a goal position, and the output should be the exact sequence of keys to type in order to reach the goal position, using the smallest possible number of moves.

You can use Java, C, C++, Python, or Scala (or ask me for permission to use a different language). Do not use any special libraries. (You can use arrays, lists, sets, maps etc., from the standard libraries, but no special libraries with graph operations.)

As an example, the first level of the Bloxorx game would be represented as a text file level1.txt like this:

```
ooo
oSoooo
oooooooooo
```

```
-oooooooooo
-----ooToo
------ooo
```

S indicates the start position of the block, T is the goal (target) position of the block. A o or . indicates part of the terrain, a space or a dash is not part of the terrain. We use a coordinate system that has *(0,0)* at the top left corner of the input file, the *x*-coordinate increases to the right, the *y*-coordinate to the bottom. In our example file, the start position is at *(1, 1)*, and the goal position is at *(7, 4)*.

Your program should take the filename of the terrain on the command line, and print a sequence of moves on standard output, using the letters LRUD (for left, right, up, and down). For the filelevel1.txt, a possible move sequence would be (this is not the shortest sequence):

```
DRRRRRRD
```

Here are a few example terrains for you to examine and to test your program:

- **level1.txt**
- **level2.txt**
- **level3.txt**
- **level4.txt**
- **level5.txt**
- **level6.txt**
- **level7.txt**
- **level11.txt**
- **level13.txt**

Modelling as a graph and implementation

We model the problem as a graph. The nodes of the graph are the possible configurations of the block. For each configuration, there are at most four outgoing edges, corresponding to the four possible ways of rolling the block.

Your program must run breadth-first search on this graph to find the shortest path to the goal configuration (or to determine that the goal configuration cannot be reached). The procedure should stop as soon as you have found the target—do not explore the entire graph!

Your program must not build the graph, but must represent it implicitely. Find a useful representation of the possible configurations (for instance, use the coordinates of the two parts of the block), and write a function that returns all the neighbors of a given configuration.

We store the visited vertices in a hash table. This allows us to determine efficiently the value of `dist(u)` without needing an array of all the vertices of the graph. (You should not need to implement a hash table yourself—use a data structure from a standard library, such as `HashMap` in Java, `scala.collection.mutable.Map` in Scala, `hash_map` in C++, or a dict in Python).

To display a sequence of operations leading to a solution, you need to remember for each visited configuration the parent of the configuration in the shortest-path tree. Again this can be done using a hash table (or the same hash table can be used for both purposes).

Bonus

Some of the example terrains contain periods `.` to mark a terrain tile instead of a `o`. These correspond to orange tiles in the Flash game: The block cannot stand on these tiles, but it can lie there. For bonus credit, implement

these tiles correctly, so that your program's solution will not contain moves where the block stands on a period tile.

Submitting

Submission rules: (Points are subtracted for submissions that do not follow the rules) You must upload the **sources** of your program. Do not upload any executable files, data files, Eclipse settings files, etc. No *.exe, no *.class files, please!

If your source is a single source file, for instance Bloxorx.java, just upload that file.

If your source consists of several source files, for instance bloxorx.h, bloxorx.cpp, terrain.cpp, then make a zip file and upload the zip file.

Your program must be written so that it can be run from the command line and reads the terrain file given on the command line.

For instance

```
$ bloxorx terrain.txt
$ java Bloxorx terrain.txt
$ python bloxorx.py terrain.txt
```

Points will be subtracted if your program reads a hard-coded filename or if it reads from standard input!

Submit by uploading your file at the **submission server**.

You will need to register on the server before using it for the first time. When you register, you have to choose an alias. This alias will be used when I post your homework and exam scores, so you should pick a name that you can remember, but which other students do not know (everybody who knows your alias will know your scores). Please register an email address that you read

every day, so that I can sent important announcements (for instance, if the class is canceled because I am sick).

You must remember your password. There is currently no way to retrieve or change your password (except for asking me).

The deadline is midnight on Thursday, May 2, sharp.