FIT3080 Assignment 1
Problem Solving as Search (13%)
Due September 5, 3 pm

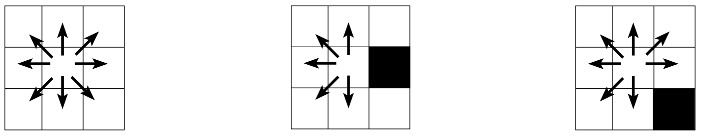
1 The Problem

Consider a grid of size N x N that represents a topographic map. Each tile describes the characteristics of its terrain, which could be of two types: normal or mountainous. ROBBIE the robot must navigate from a starting position (x_s, y_s) to a goal position (x_g, y_g) using several of the learned algorithms (there can only be one start and one goal).

Note: In the explanations below, we assume that tile (1,1) is the top-left tile in the map.

1.1 Transition rules

ROBBIE is allowed to go to one of the eight surrounding tiles (at most), as shown in Figure 1(a). However, it cannot move to a mountainous tile, and it cannot move diagonally if one of the directions composing the diagonal contains a mountainous tile. For instance, the black tile in Figure 1(b) represents a mountain, hence ROBBIE can move only to the five white tiles indicated by the arrows. In contrast, ROBBIE can move to seven tiles in Figure 1(c).



(a) Directions (b) 5 Transitions (c) 7 Transitions

Figure 1: Directions of movement and transition rules

1.2 Path cost

ROBBIE's wheels are built so that a diagonal move is the easiest. Hence, the cost of such a move is 1. Any other move has a cost of 2.

2 The Task

Your task is to write a Java program called `planpath` that plans a path from a given starting tile to a goal tile.[1]

IMPORTANT: Programs that don't conform to the requirements in this section will automatically receive a mark of 0. To assist you in writing a compliant program, we will provide a Java driver and some test files.

[1]Students who don't know Java should make alternative arrangements with the lecturer.

2.1 Input

Your program will receive its input from a file called `input.txt`.

- The first line in the file will state which algorithm your program will use. The options are: K, D, B, U and A for Backtrack, Depth-first search, Breadth-first search, Uniform-cost search and A (or A*) respectively. Any other option should result in an error message.
- The second line will contain the number of iterations for which diagnostic information should be printed. This information is described under Output (Section 2.2). If this number is 0, no diagnostic information is required.
- The third line will contain **one** number that specifies the number of rows and columns in the map.
- Subsequent lines will contain the map, one line per row. The following values will be accepted for each tile:

Tile type	Symbol
Normal	R
Mountain	X
Start	S
Goal	G

The following illustrates a sample input for applying Backtrack to a 3x3 map and printing diagnostic output for 5 iterations:

```
K
5
3
SXR
RRR
XXG
```

2.2 Output

The output of the program will be written to a file called `output.txt`, which will contain the following lines:

- The first line will contain the path found by the chosen algorithm represented as a list of actions separated by dashes followed by a blank space and the cost of the path according to the algorithm. If no path was found, the output should state NO-PATH.

  The actions (in UPPER CASE) are: R (Right), RD (Diagonal Right-Down), D (Down), LD (Diagonal Left-Down), L (Left), LU (Diagonal Left-Up), U (Up), RU (Diagonal Right-Up).

  For example, a path that goes Start, Right, Right-Down, Down, Down, Left-Down and Goal, with a total cost of 8, is represented as follows:
  ```
  S-R-RD-D-D-LD-G 8
  ```

- The next lines should contain diagnostic information for each iteration performed by the algorithm (up to the number of iterations specified in the input). For each iteration, the output should be as follows:

  Backtrack

  Line 1 contains the path to the tile being expanded in the current iteration, followed by a blank and the total cost of the path so far. For example, if tile `S` is $(1,1)$, `S-R 2` designates tile $(1,2)$ reached at a cost of 2.

  Line 2 contains the remaining operators that can be applied from the current tile *sorted in descending order of merit* and *in clockwise order starting from R for operators with the same merit* separated by a blank. For instance, if operators `R`, `RD` and `D` have already been applied at tile $(2,2)$, assuming clockwise ordering only, Line 2 should be `LD L LU U RU`.

  For example, the diagnostic output of two iterations of Backtrack starting in tile `S` and considering the operators clockwise starting from `R` should be

  ```
  S 0
  R RD D
  S-R 2
  R RD D LD L
  ```

  Graphsearch

  Line 1 contains the path to the tile being expanded in the current iteration, followed by a blank and the values of g, h and f separated by blanks. For example, if tile `S` is $(1,1)$, `S-R 2 0 2` designates tile $(1,2)$ reached at a cost of 2, and the value of h is 0.
  IMPORTANT: Note the difference between the cost of the path and the "cost" of a step performed by BFS or DFS, which is 1.

  Line 2 contains the word `OPEN` followed by a blank and the list OPEN *sorted in descending order of merit and according to the tie breaking rules described below*; and Line 3 contains the word `CLOSED` followed by a blank and the list CLOSED sorted in First-In-First-Out order (to facilitate marking). Nodes in OPEN and CLOSED are separated by a blank, and are identified by the path that led to them. For example, after expanding the node for tile `S` = $(1,1)$, Lines 2 and 3 should be

  ```
  OPEN S-R S-RD S-D
  CLOSED S
  ```

  Tie breaking rules:

  - According to generation time: earliest node first.
  - Nodes that were generated at the same time (i.e., with the same parent) according to the operator that produced them in the following order: `R`, `RD`, `D`, `LD`, `L`, `LU`, `U`, `RU` (i.e., descending preference clockwise starting from `R`).

  In summary, the nodes in OPEN will be sorted according to: (1) the requirements of the algorithm, (2) the time of generation (First-In-First-Out), and (3) the operator that generated them.

For example, the diagnostic output of two iterations of BFS starting in tile S $(1,1)$ and considering the operators clockwise starting from R should be

```
S 0 0 0
OPEN S-R S-RD S-D
CLOSED S
S-R 2
OPEN S-RD S-D S-R-R S-R-RD S-R-D S-R-LD S-R-L
CLOSED S S-R
```

In summary, if the input requires diagnostics for M iterations, Backtrack should produce $2M+1$ output lines in total (2 diagnostic lines for each iteration and 1 line for the path), and Graphsearch (DFS, BFS, UCS and A) should produce a total of $3M+1$ output lines (3 diagnostic lines for each iteration and 1 line for the path).

3 Programming Requirements

Your program should call **two** main modules: one that implements the Backtrack algorithm and **one** that implements the Graphsearch algorithm. That is, for Graphsearch, you should implement **one procedure** that employs an ordering function to distinguish between Depth First Search (DFS), Breadth First Search (BFS), Uniform Cost Search (UCS), and A (or A*). You may implement Graphsearch or Treesearch.

- For Backtrack, you are advised to use Backtrack1 (from the lectures), and you should devise a scheme for ordering the operators depending on their merit.
- For algorithm A/A*:
  - Propose and implement a heuristic function h for solving this problem.
  - Determine whether this function is admissible or monotonic. Is the resulting algorithm A or A*?

IMPORTANT: You should implement only **one** Graphsearch procedure, where the only difference between the options is how the nodes in OPEN are ordered. **Implementing different procedures for DFS, BFS, UCS and Algorithm A will incur loss of marks.**

4 Submission Requirements

- You are allowed to work with **one** other student if you wish. Beyond that, make sure you haven't shared your work with other students.
- Your code should have adequate in-line documentation.
- Demonstrate that you have adequately tested your program on at least five maps of different sizes and terrain configurations.
  Important: Inadequate program testing will incur loss of marks.

- Prepare a brief report (2-3 pages) that includes
  - A justification for your heuristic function for algorithm A and for the operator-ordering heuristic for Backtrack, and a discussion of any implementation issues that are not straightforward, e.g., the value of *BOUND* for Backtrack1 and DFS, and efficiency measures. Indicate whether the h function is admissible or monotonic. Is the resulting Graphsearch algorithm A or A*?
  - An analysis of the results obtained by the various algorithms on your sample maps, e.g., run time, quality of the results (did they find the optimal path?).
- Prepare your submission as follows:
  - Submit on moodle a zip file that contains (1) a `jar` file with all the relevant class files and source code, (2) your test maps, and (3) your report. The zip file should be named `A1`*StudentID*`.zip`, where *StudentID* is your Student ID number.

    If you have done the work in pairs, **use only the ID of one student** in your submission, but **put both names** in the report. In addition, **both students must complete the electronic Assignment Cover Sheet**.
  - Hand in the report in hard copy (in the assignment box).

Assessment

- Backtrack: 3 marks.
- Graphsearch: 8 marks.
- Report, including appropriate demonstration of performance with evidence of testing: 3 marks.
  Note: We are not marking the quality of your code, but marks will be deducted for inadequate code and inadequate comments.
- **Late submission policy:**
  10% of the maximum mark will be deducted for every work day a submission is late.