FIT3080 Assignment 1, Part 1
Problem Solving as Search (15%)
Due August 30, 12 pm

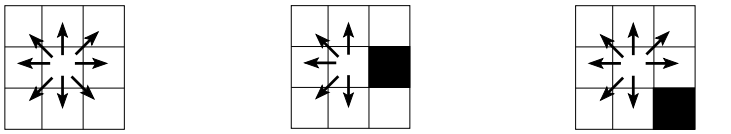
1 The Problem

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 two 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 (at most) eight surrounding tiles, 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 **Python 3** program called `planpath` that plans a path from a given starting tile to a goal tile.

- You should implement **one Graph/Treesearch procedure** that calls different ordering functions to activate different search strategies. Specifically, you need to implement Depth Limited Search (DLS) and A (or A*). **Do not write a different**

Graph/Treesearch procedure for each search strategy. Write one procedure with different ordering options.

Also, note the difference between the cost of the path and the "cost" of a step performed by DLS, which is 1.

- For algorithm A/A*:
  - Propose and implement a heuristic function for solving this problem.
  - Determine whether this function is admissible or monotonic, and **explain why**. Is the resulting algorithm A or A*?
- You will need to implement **tie-breaking rules** when all options have equal merit. Specify clearly in your documentation what are your tie-breaking rules.

2.1 Calling your program

Your program will have three command line arguments as follows:

`python planpath.py INPUT/input`i`.txt OUTPUT/output`i`.txt Flag Algorithm`

where the folders `INPUT` and `OUTPUT` should reside in the same folder as your program.

- `input`i`.txt` – the name of the input file, where i is the number of the input file;
- `output`i`.txt` – the name of the output file, where i is the number of the output file;
- `Flag` – indicates how many node expansions require a diagnostic output (if it is 0, no diagnostic output is required) – see the example at the end of Section 2.4;
- `Algorithm` – indicates which algorithm your program will use. The options are: `D` for Depth-limited search (DLS) and `A` for A or (or A*). Any other option should result in an error message.

IMPORTANT:

- If you don't know Python, please contact the teaching staff to discuss alternatives.
- We plan to run your program with input in the above format, hence programs which deviate from this format **will not be accepted**.
  - To help you ensure that your program complies with the specifications, we have provided the following:
    - A template where you should embed your program (`planpath.py`).
    - A sample input configuration and matching output.
    - Sample input configurations, which we will use to test your program.
  - **Failure to submit a compliant program will be equivalent to submitting a late program – see Late Submission Policy.**

2.2 Input

- The first 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 procedure call and sample input for applying DLS to a 3x3 map and printing diagnostic output for 5 iterations:

```
python planpath.py INPUT/input1.txt OUTPUT/output1.txt 5 D
```

where `input1.txt` is

```
3
SXR
RRR
XXG
```

2.3 Output

Your program should produce a sequence of moves that ROBBIE should perform to get from the start node to the goal, and the **accumulated cost** and ROBBIE's position after each move. If you are programming A*, the sequence of moves should be optimal.

Output format
The sequence of moves should be formatted 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
```

2.4 Programming Requirements

- Your program should be written in Python 3.
- Your program receives an input parameter called `Algorithm`, which has the possible values DLS or A. This results in a call to a Graph/Treesearch procedure with the appropriate search strategy.

  IMPORTANT: As mentioned above, you should implement only **one** Graph/Treesearch procedure, where the `Algorithm` parameter (`DLS` or `A`) determines how the nodes in

OPEN are ordered (DLS requires a bound L — this is a program variable that is not relevant to A). **If you implement more than one Graph/Treesearch procedure, you will lose half the marks assigned to the program.**

- Your program should create a unique identifier for every generated node. The identifier of the start node should be **N0**, and other nodes should be identified as **N1, N2,** ... according to the order in which they are **generated**. If you wish, you can include the actions used to reach a node in its identifier, e.g., N0-R-D.

  If you implement Graphsearch (as opposed to Treesearch), you need to identify when a repeated instance of a node is reached, and use only the first identifier created for this node.

- Your program should work on different maps, including those provided in `moodle`.

- The Graph/Treesearch procedure should build a *search graph/tree* respectively. Each node in the search graph/tree should have the following components:

  (1) an identifier;
  (2) the operator that generated it;
  (3) order of expansion (if in *CLOSED*) otherwise 0;
  (4) the cost g of reaching that node;
  (5) a heuristic value h (0 for DLS);
  (6) a value f, where $f = g + h$;
  (7) a pointer to its children; and
  (8) a pointer to its parent (if you implement Treesearch, and your action sequence is included in a node identifier, then this pointer is not necessary).

- In diagnostic mode (`Flag` ≥ 1), each time a node is **expanded**, your program should output the following components:

  (1) node identifier;
  (3) order of expansion;
  (4) the cost g of reaching that node;
  (5) a heuristic value h (0 for DLS);
  (6) a value f, where $f = g + h$;
  (7) the resolution of pointer to its children, i.e., the identifier of **each child** of the expanded node and the action that generated it;
  (9) the lists *OPEN* (**sorted in descending order of merit**) and *CLOSED*. The value of `Flag` will be small, so don't worry about *OPEN* and *CLOSED* being too long. **Failure to print *OPEN* and *CLOSED* will result in loss of marks.**

  - For each node in *OPEN*, you should include components 1-2, 4-6 above, and the parent of the node (only for Graphsearch).
  - For each node in *CLOSED* you should include components 1, 3-6 above and the parent of the node.

2.5 Output example

Say node `N0:S` at position $(1, 1)$ is the first node to be expanded by DLS. The output for this step should be something like the following:

```
N0:S        1 0 0 0    # ID expansion-order g, h, f
Children:   {N1:S-R, N2:S-RD, N3:S-D }
OPEN:       {(N1:S-R 2 0 2), (N2:S-RD 1 0 1), (N3:S-D 2 0 2) }
CLOSED:     {(N0:S 1 0 0 0)}
```

3 Submission Requirements

- **IMPORTANT:** We use a program that reliably detects cheating. Make sure that your work is your own, and that you have not shared your work with other people. If cheating is detected, both the giver and the receiver are deemed to be at fault.
- Your code should have adequate in-line documentation.
- Demonstrate that you have adequately tested your program on at least seven maps of different sizes and terrain configurations (including those provided in moodle). This means you should characterize your maps in terms of size and obstacles between start and goal, e.g., no obstacles, some obstacles, and all paths are blocked. **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, your tie-breaking rules, and a discussion of any implementation issues that are not straightforward. Indicate whether the h function is admissible or monotonic. Is the resulting algorithm A or A*?
  - Screenshots of your test maps, and of the output of your program for each map.
  - Output in diagnostic mode (three examples only is fine). In this mode, you need to output items 1, 3-7 and 9 specified in Section 2.4.
  - An analysis of the results obtained by the two algorithms for **all your test maps**, specifically run time and quality of the results (did they find the optimal path?).
- Prepare your submission as follows:
  - Submit on moodle your source code, test maps and report in a zip file named FIT3080-*StudentID*-Assignment1-Part1.zip, where *StudentID* is your Student ID number. **Assignments that don't follow this naming convention will be rejected.**
  - Make sure to include all the source files along with the executable file of your assignment. All the code and executables should be **in the top directory** (no sub-directories). The input maps should be in a directory called `INPUT` and the outputs should be in a directory called `OUTPUT`.

Assessment

- Graph/Treesearch: basic algorithm 8 marks, DLS procedure 1.5 marks, A/A* with heuristic 2.5 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:** 1.5 marks (10% of maximum) will be deducted for every work day a submission is late, and every time we need to contact you because we can't run your submission **for any reason**, e.g., wrong directories, wrong input format, wrong version of Python. **To avoid any penalties, make sure your submission runs in the University labs before the due date.**