Coursework Description: Sliding puzzles

In this coursework, you are supposed to use path finding to solve a type of puzzle that occurs in many video games. The basic version that we will be dealing with is this:

```
. . . . . 0. . . S
. . . . 0. . . . .
0. . . . . 0. . 0
. . . 0. . . . 0.
. F. . . . . . 0.
. 0. . . . . . . .
. . . . . . . 0. .
. 0. 0. . 0. . 0
0. . . . . . . . .
. 00. . . . . 0.
```

The player starts at the location labelled “S” and wants to reach the finish, labelled “F”. Each turn they choose one of the four cardinal directions to move. However, except for S and F the floor is covered in frictionless ice, so they will keep sliding in the chosen direction until they hit the wall surrounding the area, or one of the rocks (labelled “0”). For example, starting in the map given above:

```
. . . . . 0. . . @
. . . . 0. . . . .
0. . . . . 0. . 0
. . . 0. . . . 0.
. F. . . . . . 0.
. 0. . . . . . . .
. . . . . . . 0. .
. 0. 0. . 0. . 0
0. . . . . . . . .
. 00. . . . . 0.
```

the player (“@”) moving left would end up here:

```
. . . . . 0@. . S
. . . . 0. . . . .
0. . . . . 0. . 0
. . . 0. . . . 0.
. F. . . . . . 0.
. 0. . . . . . . .
. . . . . . . 0. .
. 0. 0. . 0. . 0
0. . . . . . . . .
. 00. . . . . 0.
```

So we are dealing with the problem of finding a path from S to F, but the reachability relation between points is not the usual one.

Tasks to be performed:

Task 1 (10 marks). Set up a project (Java or C++) as you did for the tutorial exercises.
Task 2 (20 marks). Choose and implement a data structure which can represent maps such as the one in the example. It must provide the necessary infrastructure for finding a shortest path from the start to the finish.
Task 3 (20 marks). Add a simple parser which can read a map like the one in the example from an input file. It needs to determine the width and height and the locations of the start and finish square as well as the rocks. The structure of the files will look like in the example, i.e., use '.'/'0'/'S'/'F' for empty (ice) squares, rocks, the start and the finish.
Your parser should be able to handle all input files which have this format. We will provide benchmark examples for your performance analysis, but you may also want to create some yourself to test your implementation.
Task 4 (20 marks). Choose and implement an algorithm which finds a shortest path from the start to the finish in any given map, if one exists (all the benchmarks we provide will have a solution). It should output all the steps of the solution it found, e.g., for the example above:

1. Start at (10,1)
2. Move left to (7,1)
3. Move down to (7,2)
4. Move left to (6,2)
5. Move down to (6,10)
6. Move right to (8,10)
7. Move up to (8,8)
8. Move right to (9,8)
9. Move up to (9,6)
10. Move left to (3,6)
11. Move up to (3,1)
12. Move left to (1,1)
13. Move down to (1,2)
14. Move right to (4,2)
15. Move down to (4,3)
16. Move left to (2,3)
17. Move down to (2,5)
18. Done!

Where the squares are numbered left to right, top to bottom.
Task 5 (30 marks). Write a brief report (no more than 3 A4 pages) containing the following:

a) A short explanation of your choice of data structure and algorithm.
b) A run of your algorithm on a small benchmark example. This should include the supporting information as described in Task 4.
c) A performance analysis of your algorithmic design and implementation. This can be based either on an empirical study, e.g., doubling hypothesis, or on purely theoretical considerations, as discussed in the lectures and tutorials. It should include a suggested order-of-growth classification (Big-O notation).

To be submitted:

- Your zipped source code (for Tasks 1 to 4) in Java or C++. Your source code shall include header comments with your student ID and name.
- The report about the algorithmic performance analysis (Task 5).

Coursework marking scheme:

Criterion and range	Indicative mark	Comments
Task 1 (0-10 marks)	8-10	A compilable and executable project has been created and follows guidelines for writing clear code such as https://introcs.cs.princeton.edu/java/11style
	4-7	A compilable and executable project has been created but does not follow programming guidelines such as of the *Java* related example above.
	0-3	No project has been created, or it is prone to compilation or runtime errors.
Task 2 (0-20 marks)	16-20	A data structure has been implemented, which satisfies the following principles of abstraction: - Builds on top of already existing programming language specific data structures, e.g., array. - Allows maps as given by the input to be represented - Fits the purpose of the intended algorithm and nature of the problem.
	5-15	A working data structure has been implemented but is limited in functionality
	0-4	A data structure has not been implemented or does not work properly.
Task 3 (0-20 marks)	16-20	A parser has been implemented and is able to initialise the data structure from a given input file. It can handle any input file which has the format given in the problem description.
	5-15	A parser has been implemented and is able to initialise the data structure from some input files, but not others.
	0-4	A parser has not been implemented or does not work properly.
Task 4 (0-20 marks)	16-20	The correct solution, i.e., a shortest path from start to finish, can be found for any solvable problem instance. The implementation outputs the steps of the solution in enough detail that it can be checked independently.
	5-15	The correct solution can be calculated, but the implementation does not work for all possible inputs or does not provide enough information to justify its result.
	0-4	Not done or does not work properly.
Task 5 (0-30 marks)	25-30	The student has submitted a full report explaining their solution and its algorithmic performance based on sufficient empirical data or a formal analysis.
	6-24	The student has submitted a report, but some of the contents (explanation of the data structure, explanation of the algorithm, discussion of algorithmic performance) are lacking.
	0-5	Not done, or no relevant contents.