CSE250 Assignment A3 – Maze

Due: 08/06/2018, 1:59PM

Last updated: 2019-07-26 11:01

Objectives

- Practice design and implementation of recursion- (or iteration-) based algorithms.
- Implement a component to work with larger C++ code.
- Solve a practical problem.

Introduction

The shortest path problem is one of the most practical and popular problems in computer science (think GPS navigation, robot routing, etc.). We are interested in the instance where a robot must find a shortest path in a maze. Maze is represented by a two-dimensional binary array (or grid) $n \times m$, where the top-left corner has coordinates (0,0) and the bottom-right corner has coordinates (n-1,m-1). 0 at position (x,y) in the array indicates open space (or passage) to which our robot can move freely. 1 indicates a wall that blocks the robot, that is, the robot cannot move to a position occupied by a wall. The array below is an example of a 4×5 maze:

```
00000
01010
01010
00000
```

The robot starts at the position (sx,sy) and must find a shortest path to the position (fx,fy). In a single step the robot can move in one of four directions (up, down, left or right) but only to a position that is open. The shortest path is a path that requires the least number of moves from the robot to reach (fx,fy) starting at (sx,sy). For instance, if we consider our example maze, and (sx=0,sy=0) and (fx=3,fy=2), then there are two possible shortest paths of length 5, i.e. the robot must make 5 moves to reach the finishing position (3,2) from the starting position (0,0).

Your task is to implement a function that will find **the length** of a shortest path of our robot for a given maze, starting position (sx,sy) and finishing position (fx,fy). You can assume that such path always exists. To implement the function consider the following observation. After making a move from (sx,sy) to a new position, the length of the best path the robot can take to (fx,fy), which potentially can be the shortest, is equal 1 + the length of a shortest path from the new position to (fx,fy).

Hints

Read about BFS and DFS. If you plan on implementing this using DFS, read about dynamic programming.

Instructions

1. Create directory `A3` where you will place your code.
2. Download `A3handout.tar` from the piazza resources section.
3. Untar handout, and move `a3.hpp`,`a3.cpp` and `Maze.hpp` to your `A3` directory. These files provide all functionality to: read input maze and input parameters, invoke your distance function and generate output. Additionally, `Maze.hpp` defines the interface to interact with the input maze (see below).
4. Create `Makefile` to automate compilation process of your code. You can directly adopt Makefile from A0 (replace a0 with a3).
5. `a3.hpp` implement function `distance` considering the following:
   (a) Arguments `sx,sy,fx,fy` represent the starting position (sx,sy) and finishing position (fx,fy).
   (b) Argument `maze` of the type `Maze` represents the input maze.
   (c) Class `Maze` provides the following methods, which are the only methods you can use to interact with the maze:
   - `int nrow() const;` — returns the number of rows in the maze.
   - `int ncol() const;` — returns the number of columns in the maze.
   - `bool is_visited(int x, int y) const;` — returns `true` if position (x,y) in the maze has been marked as visited, `false` otherwise. For any position outside the maze, e.g. (-1,0), it returns `true`.
   - `bool is_open(int x, int y) const;` — returns `true` if position (x,y) in the maze is open and the robot can move to it, `false` otherwise. For any position outside the maze, e.g. (-1,0), it returns `false`.
   - `void mark(int x, int y);` — marks position (x,y) as visited, which implies that is_visited(x,y) will return `true`. If (x,y) is outside the maze, or is not open, the method does nothing.
   - `void unmark(int x, int y);` — removes visited mark if position (x,y) was marked as visited otherwise does nothing.

   (d) The return value must be the length of a shortest path between (sx,sy) and (fx,fy) in the maze.
   (e) Arguments passed to distance are always correct and (fx,fy) is always reachable from (sx,sy).
6. You can edit only `a3.hpp` and all your code must be contained in this file only.
7. To test your code you can use the `a3.txt` file provided in the handout. If passed to `a3` like this:
   `./a3 4 5 0 0 3 2 a3.txt`
   it should product the following output:
   `5`
   Here, `4 5` are maze dimensions, `0 0` corresponds to `sx sy`, `3 2` corresponds to `fx fy`, and `5` is the length of the resulting shortest path.

Submission

1. Remove your binary code and other unrelated files (e.g. your test files).
2. Create a tarball with your `A3` folder.
3. Follow to https://autograder.cse.buffalo.edu and submit `A3.tar` for grading.
4. You can make five submissions, and the last submission will be graded.

5. Any submission after the deadline will have 50% points deducted.

Grading

- 5pt: Code builds via `make` and `README` is provided.
- 5pt: The example a3.txt above works.
- 90pt: There will be nine test cases. You will get 10pt for each correctly solved maze.
- If your code is **extremely** inefficient, autograder will terminate your code and you will receive 0pt.

Remarks

- Make sure that all file and directory names are exactly as instructed. Otherwise the grading system will miss your submission and you will get 0pt.
- Make sure that you start working on the assignment early! If you wait to the very last moment, you may expect delays from autograder (overloaded by other students who like you waited with their submission).
- If you are having trouble, have a discussion on piazza! Before posting your question, ask yourself: Am I giving enough information where someone who is not looking at my code can still help? Do not, of course, share your code.