

## CS2420 Assignment 9

This assignment is due **Monday, March 25 at 5:00pm**. You must work in pairs for this assignment.

### Graph Pathfinding

We have been asked to create a tool to help Pacman solve mazes. More generally, we must create a tool that can find the *shortest* path from one location to another in any enclosed field of obstacles. We are given the field as input, represented as a simple text file. The start-point and end point are indicated in this text file, as well as the layout of the field (the location of walls). We must produce a similar text file, but with the shortest path from the start-point to the end-point added to the field. If there is no path at all, the path indicators will simply not exist in the output.

To solve this problem, you must represent the maze as a graph, and perform a breadth-first search from Pacman's starting location. See lecture 17, slides 39-40 for a suggestion on implementing a graph for this assignment (use a 2D array of nodes).

### Requirements

`PathFinder` class:

Create a class `PathFinder` in package `assignment9`, with the following method:

```
public static void solveMaze(String inputFile, String outputFile)
```

This method will read a maze from a file with the given input name, and output the solved maze to a file with the given output name. You must use the filenames exactly as is (do not change the directory or path). We will provide the full path to files we want to read/write in our tests. See required specifications below. **This method must use a graph and graph pathfinding to solve the problem.**

Your program may contain any other methods or classes necessary for your solution (it is recommended you create a separate `Graph` and `Node` class). See lecture 17, slides 39-40 for a suggestion on implementing a graph for this assignment (use a 2D array of

#### NAVIGATION

Home

■ My home

Site pages

My profile

My courses

Computer Science

Previous Semester

[CS 1410-1-S13](#)

[CS\\_2100\\_S\\_13](#)

[CS2420-S13](#)

[Participants](#)

General

Getting started;  
Java review

Generic  
programming;  
Object Oriented  
Programming

Algorithm  
analysis; Data  
Structures

Basic Sorting  
Algorithms

Recursive Sorting  
Algorithms

Linked Lists

Stacks and  
Queues

nodes). You will need a queue for breadth-first search. Implement one using your assignment 6 linked list, or feel free to use Java's [LinkedList](#) which implements [Queue](#). **Make sure you submit all files necessary to run your program.**

## Input

The input files are in the following form:

```
5 5
XXXXX
XS  X
X   X
X  GX
XXXXX
```

Where the first line contains two numbers, separated by a space. The first number is the height of the field, and the second is the width. The rest of the lines contain the layout of the field. The characters have the following meaning:

X - A single wall segment. Pacman can not travel on to or through walls.

S - The starting point of the path that we are trying to find (where Pacman starts). This is an open space (no wall).

G - The ending point of the path we are trying to find (where Pacman wants to be). This is an open space (no wall).

(space) - An open space on the field. Pacman is free to travel in any open space, assuming he can get there.

Pacman is free to move from his current location to any adjacent open space. This includes the space directly above, below, left and right of where he is. It does not include diagonally adjacent spaces. If any of the adjacent spaces are a wall, Pacman can not travel in that direction. The path that your maze solver finds must be a connected path (it can't skip spaces and have Pacman "jumping" over walls or empty spaces).

You can assume that all input mazes will be rectangular (or square). All of the border positions around the perimeter of the field will be walls (the field is fully enclosed).

An example of reading numbers (height and width) from a file, assuming you're using a `BufferedReader` named `input` to read the file:

Trees

Graphs

 Lab 6


 Lab Files

 Slides

 Slides

 **Assignment 9**

 Assignment 9  
files

 Analysis  
Document

Spring Break!

Hash Tables

Binary Heaps

File Compression

Comprehensive  
Project;  
Multithreading

Wrap Up

Final Exam and  
Review

29 April - 5 May

## SETTINGS

Assignment administration  
Submission

Course administration

My profile settings

```
String[] dimensions = input.readLine().split(" ");
```

```
height = Integer.parseInt(dimensions[0]);
```

```
width = Integer.parseInt(dimensions[1]);
```

---

## Output

The output of your program is a similar file. Output the height and width at the top, just like the input. It should not change the layout of the walls, or the start or end points. It will simply replace some of the open spaces (space characters) with dot characters ('.').

The spaces it replaces with dots are the spaces on the shortest path from the start point to the goal point. For example, given the above input, the following output may be produced:

```
5 5
XXXXX
XS..X
X  .X
X  GX
XXXXX
```

Some of the empty spaces are now replaced with dots, connecting a path from 'S' to 'G'.

There is a single newline character after the last 'X' in the output.

**There are multiple shortest path solutions to the above example. Any connected path from 'S' to 'G' with length of 4 is a correct solution.**

For example, another acceptable solution to the above maze is:

```
5 5
XXXXX
XS  X
X.  X
X..GX
XXXXX
```

If there is **no** path from 'S' to 'G', simply do not place any dots in the output file; print the original maze.

Some more interesting examples follow:

(Note: If using Windows, these files will not display properly in notepad. Import them in to your Eclipse project and open them in Eclipse, or use wordpad)

`tinyOpen` : `tinyOpenSol`  
`tinyMaze` : `tinyMazeSol`  
`straight` : `straightSol`  
`turn` : `turnSol`  
`demoMaze` : `demoMazeSol`  
`mediumMaze` : `mediumMazeSol`  
`classic` : `classicSol`  
`bigMaze` : `bigMazeSol`  
`unsolvable` : `unsolvableSol`  
`randomMaze` : `randomMazeSol`

Download all mazes zipped [here](#).

An example of creating a file and writing to it in Java is below (where `outputFile` is a string). It will create the specified file if it does not exist. Use the exact string `outputFile` passed in to your `solveMaze` method (do not modify the path of the file). See [lecture 17](#) (slide 48) for more info on writing files in Java.

```
try
{
    PrintWriter output = new PrintWriter(new FileWriter(outputFile));

    output.println(height + " " + width);

    // write more data here

    output.close()
}
```

---

## Rules

- The path can not go through or on top of walls
- The path must be connected (no skips or jumps)
- Diagonally-adjacent spaces are not connected

- Only up, left, down, right
- If no path exists, the output file will have no dots
- If multiple shortest paths exist, any of them are valid
- Must produce output in exact format specified (for grading purposes)
- Your program must run in 10 seconds or less on all mazes we test. This includes file input, graph construction, pathfinding, and file output. The biggest test maze is 100x100 (see [randomMaze](#)). There will be other test mazes of similar size.

---

## Program guidelines

Other than the required `PathFinder` class with method `solveMaze` (see above), the design of your program is completely up to you.

- See lecture 17 for suggestions, and examples of writing files in Java.

Download [TestPathFinder.java](#) as an example of running your program. Read the comments in the code.

---

## Pacman tool

**The pacman tool is not necessary for the successful completion of this assignment. However, it will help you visualize your mazes and verify that your solutions are correct.**

Download [pacman.zip](#) to help you visualize your text-form mazes.

Unzip the compressed "pacman" folder. From the command line, change directories to that folder. On the CADE windows machines, I recommend putting it in C:\users\<your user name>

To run pacman, type the command:

**On a Linux machine:**

```
/usr/bin/python pacman.py -l <mazefile>
```

**On a Windows machine:**

```
python pacman.py -l <mazefile>
```

Where <mazefile> is the path to a file containing a maze, for example:

```
python pacman.py -l demoMaze.txt
```

Where "demoMaze.txt" is a file containing a maze layout (such as the one above). This assumes that "demoMaze.txt" is in your pacman directory (it is included in the zip file). To test your own maze solutions, I recomend putting the solution maze file in the pacman directory.

If your maze files are all in a different directory, you can provide the full path to the file.

### Auto pacman

The above command will display the maze and the path (if there is one), but pacman will not move. This is designed so that you can examine the path without pacman eating it. You can control pacman with the arrow keys if you wish. If you want to see pacman follow the path automatically, add some additional command-line arguments:

```
python pacman.py -l demoMazeSol.txt -p auto
```

The `-p auto` tells pacman to follow the path laid out in the maze.

### Zoom

If your maze is too small or too large, you can change the size of the pacman window with the `-z` command. For example:

```
python pacman.py -l demoMazeSol.txt -p auto -z 0.5
```

The `-z 0.5` makes the window half as big

```
python pacman.py -l demoMazeSol.txt -p auto -z 2
```

The `-z 2` makes the window twice as big

You can use any number after `-z` to increase or decrease the size of the window.

Note that I only provide instructions for using the pacman tool on the CADE machines. If you can install python on your own machine, it should work without any trouble.

---

### Contest:

Just as with assignment 5. The team with the fastest solution will be awarded 25 extra credit assignment points. We will award points for up to a 2-way tie. If more than that are tied, no extra points will be given.

Just as with assignment 5, please don't use multi-threading in your contest solution, since it is not fair to those who are unfamiliar with it.

All timing will be done on the CADE lab1 machines.

As a baseline for comparison, my unoptimized solution performs as follows:

Maze	Time (ms)
tinyOpen	1.7
tinyMaze	1.9
straight	2.9
turn	5.6
demoMaze	2.8
mediumMaze	4.4
classic	4.5
bigMaze	9.5
unsolvable	4.4
randomMaze	12

We will also time your program on many more mazes of all varieties and sizes (including bigger than 100x100). Use [this program](#) to generate your own random mazes. See the comments and example usage in the code for instructions.

---

## Commenting

Since we give you no code to start with, you must be more thorough in commenting your classes and methods' purposes than with other assignments.

---

## Testing

The mazes I have provided will not guarantee that your algorithm works. Develop your

own test mazes, or move the start and goal points in the provided mazes to convince yourself that your algorithm can find any path.

---

## Analysis

[Analysis document](#) must be completed by both programming partners and submitted here by 5pm on March 25.

---

## Handing in

Hand in all files necessary to run your program, zipped. Only one partner needs to hand in the code. Both partners must hand in an analysis.

---

<b>Due date:</b>	Monday, 25 March 2013, 5:10 PM
------------------	--------------------------------

## Submission feedback



Sarah Hong

Saturday, 30 March 2013, 2:36 PM

Grade: 100.00 / 100.00

aaronsmithjustinbarketies

===PathFinder Grader Output===

There is no solution

There is no solution

There is no solution

There is no solution

Passed 32/32 tests

Test maze score: 80/80

Quality of student's code and style: 5/5

Students tests: 5/5

Analysis document: 10/10

Total: 100/100



-----  
TA coments:

## Submission

 Analysis8.pdf

---

**No further submissions are allowed.**