ALGORITHMS & A.I. FOR GAMES

WEEK 1A MAZE GENERATION AND SOLVING

STUDY

[Data Structures and Problem Solving Using Java] Chapter 24, except 24.2.2, 24.2.3 and 24.6
[Data Structures and Problem Solving Using Java New International Edition (Zebra cover)] Chapter 23 (except 2.2, 2.3 and 6)

WEB

http://en.wikipedia.org/wiki/Maze_solving_algorithm
Try out demo applet: http://www.cs.unm.edu/~rlpm/499/uf.html

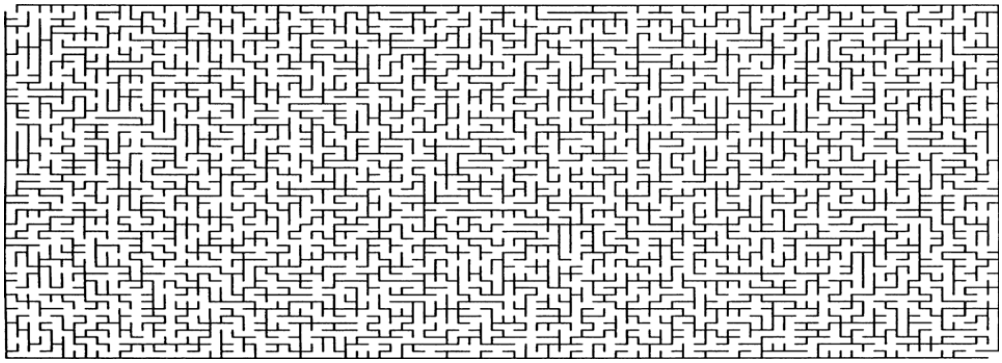
EXERCISE

[Book] 24.3, 24.4
[Book New International Ed.] 24.1, 24.2
Use http://www.cs.usfca.edu/~galles/visualization/DisjointSets.html for visualizing and practicing with disjoint sets.

PROGRAMMING

Choose an object-oriented programming language: C++, C# or Java.

You may use the skeleton code from N@tschool to get a head start.



1. Create a **DisjointSet** class. Make sure this class is a reusable data structure that can be used for other applications as well.
2. Create unit tests to test your data structure. Make sure all operations work correctly before proceeding.
3. Generate a maze using the disjoint set class, with a size of at least 20 rows and 30 columns.
4. Give a graphical presentation of your generated maze.
5. Improve the disjoint set class through implementing find with **path compression** and **union by size**. Make sure the unit tests still pass.

* Extra: Solve the maze (finding a path from start to finish) and show this path in your graphical output, preferably with an animation. Use the left handed walk (or wall follower) algorithm. Note that this is not necessarily the shortest path!

WEEK 1B BACKTRACKING

STUDY From [Data Structures and Problem Solving Using Java]: Chapter 7.7

PROGRAMMING

Exercise 1

In this exercise you are going to find a solution to the 'Family at the bridge' puzzle. You can find the description and play the puzzle online at: `http://www.learn4good.com/games/puzzle/the_bridge.htm`.

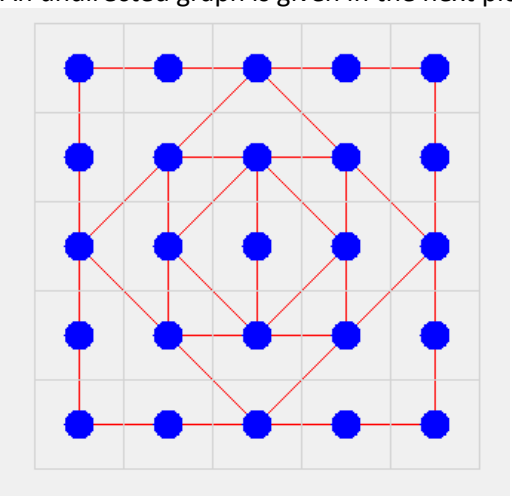
Write a console program that finds a solution using a **backtracking** algorithm. Your program should at least have the following components:

- An efficient data structure to store the game state.
- A method that returns all possible moves at that point in the game.
- The backtracking algorithm to find the solution, including other methods or classes you need.
- A method that shows a solution in the console, like in the following screenshot. The question marks should of course be replaced by real values.

```
<-- person@speed? + person@speed?
--> person@speed?
<-- person@speed? + person@speed?
Solution in ? steps using ?? lamp energy
```

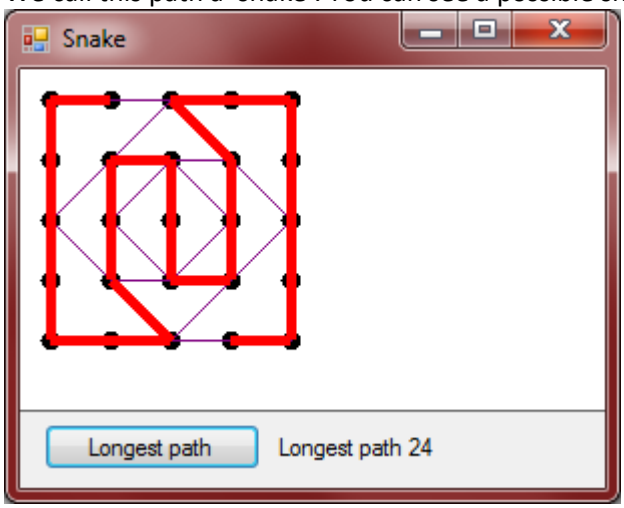
Exercise 2

An undirected graph is given in the next picture:



The blue nodes are connected by red edges. We want to find the **longest path** in this graph, where all nodes are visited only once!

We call this path a ‘snake’. You can see a possible snake in the next picture:



Find a solution for this problem using **recursive backtracking** and show the resulting snake in a graphical way.

Exercise 3
Create a backtracking algorithm to find a path from entrance to exit in the maze you created.

Exercise 4
Write a program to solve exercise 7.38a (see pdf.) in your study book using backtracking.