Assignment #5:

Programming part:

In this part of the assignment you will get practice in recursive backtrack programming and with some of the Java bitwise operators. Your objective is to write a program for solving mazes and for counting the number of solutions to a maze. We supply the code for creating a good-looking random maze.

We will think of the maze as being stored in a two-dimensional array `m[][]` of integers using the explanation given below. Take a look at the output further down for other examples of mazes using this encoding.

```
/*============================================================*/
//       8
//     1 * 4     <-- encodings of various directions around a cell
//       2
//
//       +--+--+     +--+--+
//       |     |     |11 12|     11  12   a maze and its representation
//       +--+  +     +--+  +
//       |     |     |11 06|     11  06
//       +--+--+     +--+--+
//
//      16 16 16 16   initial maze contents returned by constructor
//      16 15 15 16
//      16 15 15 16
//      16 16 16 16
//
/*============================================================*/
```

A template for your program may be found here: [ProgramTemplate.txt](). You should understand the code in this template before writing any code of your own. Some aspects of the code form part of the written questions below. Note that a random number generator is used to produce the maze and knock down walls. A seed is provided to the generator, so that we can reproduce exact mazes over and over.

You should not change any of the code unless there is a comment asking you to. As always, you can add additional methods. You should not need any additional classes.

With the main as provided, `java Maze 3 3` should produce the output shown below. Make sure you understand how the numbers correspond to walls. In the second output one wall has been removed, and there are now two possible solutions. The third output shows the solution to the first maze; each cell along the solution path has 16 added to the initial cell number.

NOTE: Each solution path is between the upper left corner and the lower right corner.

```
 9 10 12
 5 13  5
 3  6  7
Solutions = 1
 9  8 12
 5  5  5
 3  6  7
Solutions = 2
25 26 28
 5 13 21
 3  6 23
```

And `java Maze 12 12` should produce the following output.

```
 9  8 10 10  8 14  9 10 10  8 10 12
 7  5 11 10  6  9  2 14  9  6  9  6
 9  6  9 10 12  3 12  9  6 11  2 14
 1 10  2 14  1 14  5  3 10 12  9 12
 5  9 12  9  6  9  6  9 12  3  6  5
 3  6  7  3 10  2 10  4  3 14  9  4
 9 10 10 10 10 12 13  5  9 10  6  7
 3 10 12  9 12  3  6  5  3 10 10 12
 9 12  5  7  3  8 12  3 10  8 14  5
 5  7  3 12  9  6  3 12 11  6  9  6
 5  9 12  5  3 10 12  3 10 12  3 12
 3  6  3  2 10 10  6 11 10  2 10  6
Solutions = 1
 9  8 10 10  8 14  9 10 10  8 10 12
 7  5 11 10  6  9  2 14  9  6  9  6
 9  6  9 10 12  3  8  8  6 11  2 14
 1 10  2 14  1 14  5  3 10 12  9 12
 5  9 12  9  6  9  4  9 12  3  6  5
 3  6  7  3 10  2  2  4  3 14  9  4
 9 10 10 10 10 12 13  5  9 10  6  7
 3 10 12  9 12  3  2  4  3 10 10 12
 9 12  5  7  3  8  8  2 10  8 14  5
 5  3  2 12  9  6  3 12 11  6  9  6
 5  9 12  5  3 10 12  3 10 12  3 12
 3  2  2  2 10 10  6 11 10  2 10  6
Solutions = 12
25 24 10 10  8 14 25 26 26 24 10 12
 7 21 11 10  6 25 18 14 25 22  9  6
25 22 25 26 28 19 28 25 22 11  2 14
17 26 18 14 17 14 21 19 26 28 25 28
 5  9 12 25 22 25 22  9 12 19 22 21
 3  6  7 19 26 18 10  4  3 14 25 20
 9 10 10 10 10 12 13  5 25 26 22  7
 3 10 12  9 12  3  6  5 19 26 26 28
 9 12  5  7  3  8 12  3 10  8 14 21
 5  7  3 12  9  6  3 12 11  6 25 22
 5  9 12  5  3 10 12  3 10 12 19 28
 3  6  3  2 10 10  6 11 10  2 10 22
```

Written part:

These first problems are related to the code above.

- Explain what `create` is doing. It is guaranteed to generate a maze that always has a unique solution. Why?
- If there are n rows and m columns, then how many times is `create` called when making a maze?
- What is the purpose of the `p^2`?

Additional written part:

- From the backtracking handout: Question 2 about estimating the size of the tree. Compare with the actual size of the tree.
- From the book: Exercise 6.38.
- Find the max-flow and min-cut in the attached network as it would be found by the Ford-Fulkerson algorithm.

Optional written part:

- The linked article recently appeared in Wired. Note the reference to computing the expected number of flips of a fair coin to first get a HT and to first get a HH.

You should have no trouble computing one of these values although the other is more difficult. Do it!