The University of West Florida
Department of Computer Science
Data Structures and Algorithms II
John W. Coffey

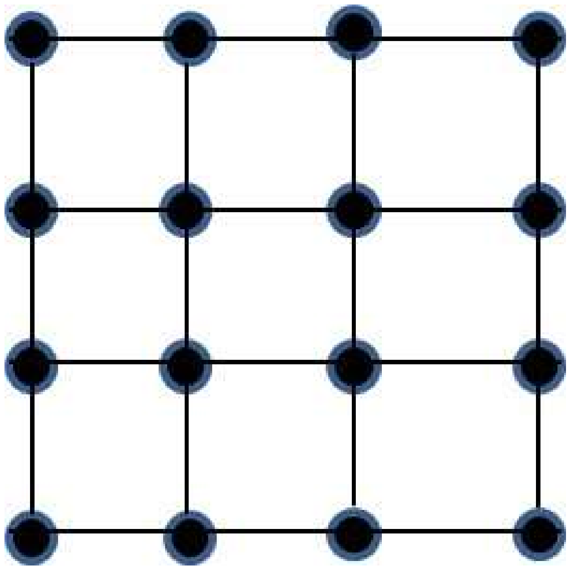
Background

Mazes have a lot of uses. They are fun to program and solve, but they also have mathematical applications. For instance the mathematician Leonhard Euler was one of the first to analyze plane mazes mathematically, and in doing so made the first important contributions to the field of topology. Mazes containing no loops (or cycles) are known as "standard", or "perfect" mazes, and are equivalent to trees in graph theory. If one pulled and stretched out the paths in the maze in the proper way, the result is a spanning tree of the original graph. This fact proves to be very useful for one particular approach to maze creation. Many maze solving algorithms also have a basis in graph theory (adapted from Wikipedia).

Program Description

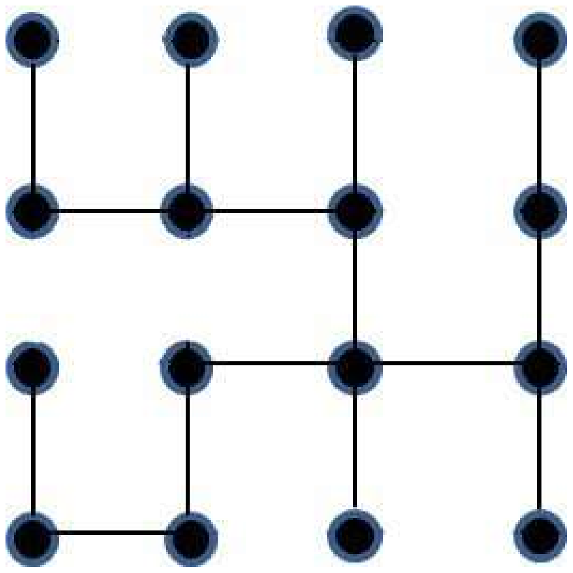
This project will have several parts:

Part 1: you will construct a gridgraph. This gridgraph will have a perfect square number of nodes, eg: 9, 16, 25, 36, etc. The following is a 4x4 gridgraph of 16 nodes.



The reason for this constraint is that you can algorithmically initialize an adjacency matrix with a perfect square size grid quite simply. Draw out the adjacency matrices for a 3x3 and 4x4 gridgraph and you will be able to recognize the pattern.

Part 2: You will construct a minimal spanning tree from the gridgraph, thereby producing a maze. The following is a Spanning tree for the gridgraph above.



We will use the idea of minimal spanning trees to enable us to produce different mazes. When your program runs, the gridgraph will have arbitrarily assigned weights for the edges, different every time. This will ensure that your program creates a different maze every time. You will use Prim's algorithm to produce the MST.

Part 3: From the minimal spanning tree, the actual maze will be produced. We will use asterisks to draw a text-based representation of our maze. The entrance will always be top-left and the exit will always be bottom right, where asterisks will be omitted. The maze for the above MST would look like this:

```
* *******
* * * * *
* * * * *
*     * *
***** * *
* *     *
* * * * *
*   * * *
******* *
```

Running the Program:
When your program runs, it will repeatedly draw new mazes by asking the user for the size of the maze (a perfect square) or -1 to quit. With a proper size, it draws a new maze on the console and writes the maze to a file as specified next. With -1, the program terminates. With any other number, it re-prompts for legal input.

Inputs and outputs:
Input will be the number of nodes in the gridgraph. Output will be a maze represented by asterisks as above. It should be drawn on the console and saved to a file named `maze.txt.` `Maze.txt` should start with the number of rows and columns in the maze (always the same in this case) followed by the asterisks, spaces, and newlines. You should be able to see the shape of the maze (as above) both in the console

output and in the file output. `Maze.txt` will hold the last maze you generated so that it could serve as input to a maze solving program such as the one you might have devedloped or will develop elsewhere.

Important Note: There are many other ways to construct and represent graphs, most of which have easily discoverable solutions on the Internet. In the interest of maximizing the degree to which you create rather than borrow components of the solution, you must implement the graph creation scheme exactly as described here.

Deliverables
You will submit the following for this project:
1. The functional decomposition for your program.
2. A User's manual that describes how to set up and run your program.
3. Your source code in C.

Please review the policy on Academic dishonesty. This is an individual project.

Submission:
1. Compile and run your program one last time before submitting it. Your program must run with gcc in the Linux lab.
2. Create a "zip" file (using WinZip or similar program) to hold your project files.The name of your zip file should be: your first initial, your last name, a dash (-), the letter 'p', the project number, followed by the .zip extension. For example, if your name is John Coffey, and you are submitting project 1, your zip file should be called jcoffey-p1.zip
3. Login to UWF's eLearning system at http://elearning.uwf.edu/. Select our course.
4. Select the Dropbox option. Then select the appropriate project folder.
5. Upload your zip file to that folder. Check the dropbox to insure that the file was uploaded.