# MP 7 Mazes

Extra credit: **Tuesday, April 26 at 11:59 PM**
Due: **Tuesday, May 3 at 11:59 PM**

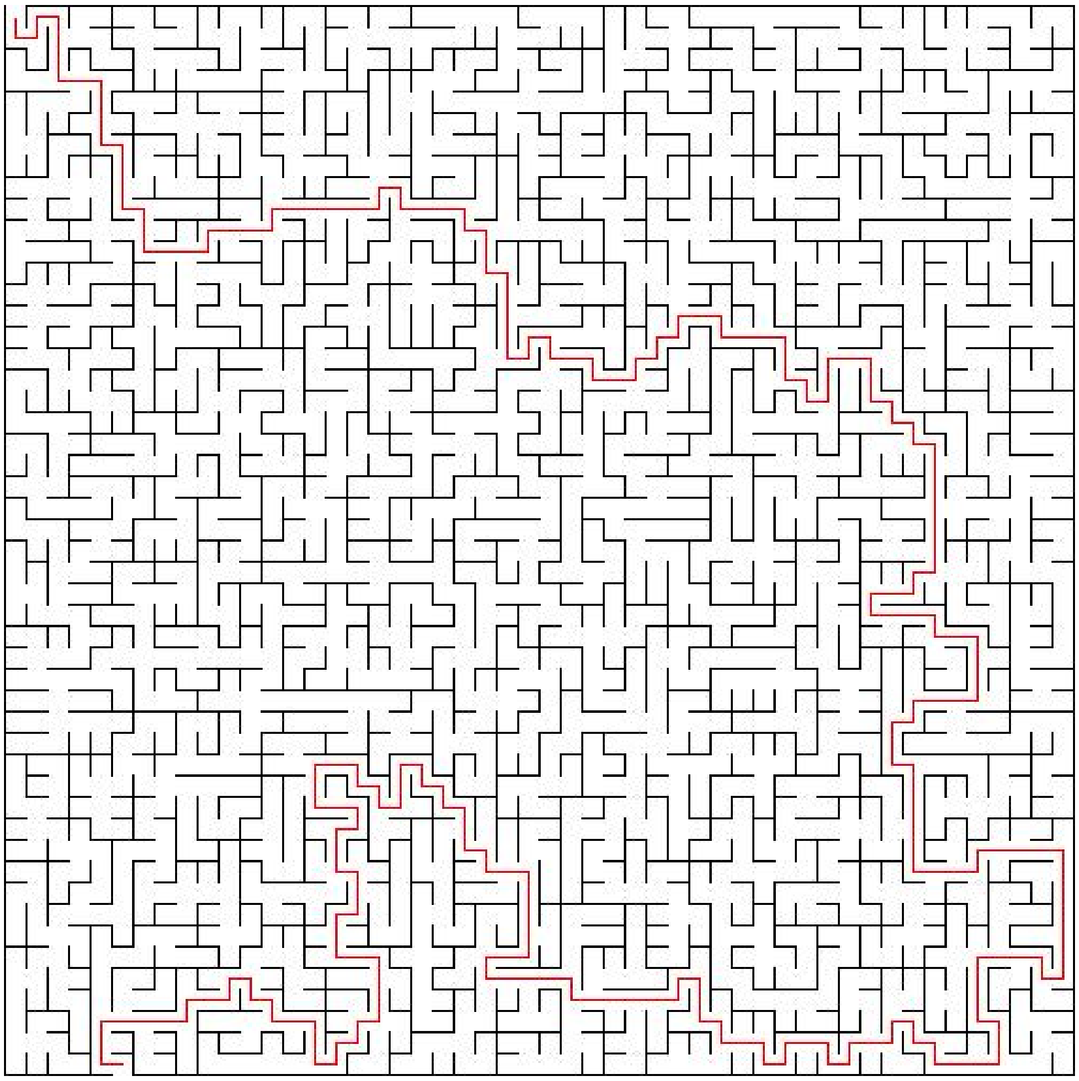
⚠ **Solo MP**

- You are not allowed to work with a partner on this MP. **You must do this MP SOLO (alone).**
- This MP has the most design on your part of any of the MPs.
- We recommend starting **early** so you have time to think about the concepts.
- MP 7.1 is easier than you think! You may want to finish it off early and get the extra credit!

## Goals and Overview

In this MP you will:

- Implement the disjoint sets data structure.
- Create a program to generate random mazes.
- Understand graphs, graph distances, and graph traversal.
- Represent a maze and its solution on a `PNG`.



A solved maze.

## Checking Out the Code

Get the code in the usual way.

The `mp7` directory will contain sample output including all 4 possible 2x2 mazes and one 50x50 maze. We encourage you to test your code by writing your own `main.cpp` that uses your classes in different ways.

## Background: `PNG` Image Handling

Refer to the background information about `EasyPNG` given in previous MPs, such as MP 1 and MP 2.

## Assignment Requirements

These are strict requirements that apply to **both** parts of the MP. Failure to follow these requirements may result in a failing grade on the MP.

- You are required to comment the MP as per the commenting standard described by the Coding Style Policy.

- You must name all files, public functions, public member variables (if any exist), and executables **exactly** as we specify in this document.
- Your code must produce the **exact** output that we specify: nothing more, nothing less. Output includes standard and error output and files such as Images.
- Your code must compile on the EWS machines using `clang++`. Being able to compile on a different machine is **not** sufficient.
- Your code must be submitted correctly by the **due date and time**. Late work is not accepted.
- Your code must not have any memory errors or leaks for full credit. ASAN tests will be performed separately from the functionality tests.
- Your public function signatures must match ours **exactly** for full credit. If using different signatures prevents compilation, you will receive a zero. Tests for `const`-correctness may be performed separately from the other tests (if applicable).

## Assignment Description

You will be implementing a `Disjoint` set data structure and then implementing a random maze generator and solver. The assignment is broken up into the two following parts:

- MP 7.1 — The `DisjointSets` data structure.
- MP 7.2 — The `SquareMaze` random maze generator and solver.

As usual, we recommend implementing, compiling, and testing the functions in MP 7.1 before starting MP 7.2. Submission information is provided for each part in the respective sections below.

## MP 7.1: The `DisjointSets` data structure

The `DisjointSets` class should be declared and defined in `dsets.h` and `dsets.cpp`, respectively. Each `DisjointSets` object will represent a family of disjoint sets, where each element has an integer index. It should be implemented with the optimizations discussed in lecture, as up-trees stored in a single `vector` of `int`s. Specifically, use path compression and union-by-size. Each element of the vector represents a node. (Note that this means that the elements in our universe are indexed starting at 0.) A nonnegative number is the index of the parent of the current node; a negative number in a root node is the negative of the set size.

Note that the default compiler-supplied Big Three will work flawlessly because the only member data is a `vector<int>` and this vector should initially be empty.

### The `addelements` function

See the Doxygen for this function.

### The `find` function

See the Doxygen for this function.

### The `setunion` function

See the Doxygen for this function.

### Testing MP 7.1

The following command can be used to compile the `DisjointSets` test executable:

```
make mp7.1
```

The following command can be used to run the test executable:

```
./testdsets
```

As usual, an ASAN version is also compiled:

```
./testdsets-asan
```

Provided monad test cases are available as well.

### Grading Information — MP 7.1

The following files are used to grade MP7:

- `dsets.cpp`
- `dsets.h`
- `partners.txt`

All other files including your testing files will not be used for grading.

## MP 7.2: The `SquareMaze` random maze generator and solver

The `SquareMaze` class should be declared and defined in `maze.h` and `maze.cpp`, respectively. Each `SquareMaze` object will represent a randomly-generated square maze and its solution. **Note that by "square maze" we mean a maze in which each cell is a square; the maze itself need not be a square.** As always, we recommend reading the whole specification before starting.

> ⚠ **`setWall` and `canTravel`**
>
> You should triple check that `setWall` and `canTravel` function exactly according to spec, as an error in these functions will not be caught by making your own mazes but can cost you a majority of the points during grading.

### Videos

- Cycle Prevention / Detection

### The `makeMaze` function

See the Doxygen.

### The `canTravel` function

See the Doxygen.

### The `setWall` function

See the Doxygen.

### The `solveMaze` function

See the Doxygen.

### The `drawMaze` function

See the Doxygen.

### The `drawMazeWithSolution` function

See the Doxygen.

## Testing MP 7.2

Since your mazes will be randomly generated, we cannot provide you with any sample images to diff against. However, we have provided you with all four possible 2x2 mazes. If you have your program create and solve a 2x2 maze, the resulting image (with solution) should match one (and only one) of the provided images `m0.png`, `m1.png`, `m2.png`, and `m3.png`. We strongly suggest that you diff against these to make sure that you have formatted the output image correctly.

We provide some basic code to test the functionality of `SquareMaze`.

The following command can be used to compile the `SquareMaze` test executable:

```
make mp7.2
```

The following command can be used to run the test executable:

```
./testsquaremaze
```

As usual, an ASAN version is also compiled:

```
./testsquaremaze-asan
```

You can compare the console output of your program with the expected by comparing it with the file `soln_testsquaremaze.out`.

Provided monad test cases are available as well.

## Runtime Concerns

You should strive for the best possible implementation. This MP can be implemented so that the given `testsquaremaze.cpp` runs in less than a quarter of a second on the EWS linux machines. To have a high probability of finishing within the time constraints of the grading script, make sure you can run the given `testsquaremaze.cpp` in under 3 seconds on an unencumbered machine. You can time MP7 by running the command `time ./testsquaremaze`.

## Grading Information — MP 7.2

- We will use `canTravel` to reconstruct your randomly generated maze in our own maze class, to check that it's a tree and to compare your implementation to a correct implementation of this MP. This will require `height*width*2` calls to `canTravel`. Therefore, it is very important that `canTravel` works, and works quickly (constant time). If it doesn't work, you will lose a lot of points.
- We will use `setWall` to replace your maze with our own, for the purpose of testing all of your other functions independently of your `createMaze`.

**⚠ `setWall` and `canTravel`**

You should triple check that `setWall` and `canTravel` function exactly according to spec, as an error in these functions will not be caught by making your own mazes but can cost you a majority of the points during grading.

The following files are used to grade MP7:

- `dsets.cpp`
- `dsets.h`
- `maze.cpp`
- `maze.h`
- `partners.txt`

All other files (including your `main.cpp`) will not be used for grading.