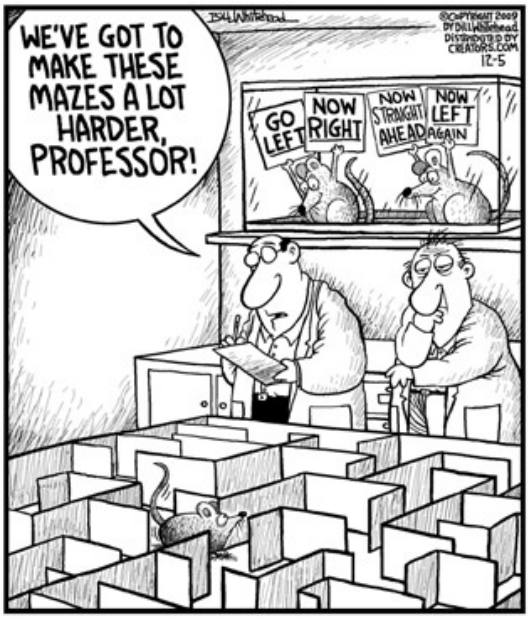
CS 302 – Assignment #9

Purpose: Learn concepts regarding disjoint sets.
Due: Monday (11/05) → Must be submitted on-line before class.
Points: Part A → 100 pts, Part B → 50 pts

Assignment:

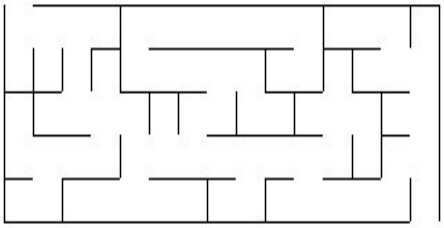
Part A:

Design and implement a C++ class, ***disjointSets***, to implement a disjoint sets[1] data structure. The disjoint sets class will use arrays for the parent links and the ranks (height estimates). The disjoint sets class will perform path compression (for the find operation). A test program is provided to allow testing the disjoint set class.



Once the disjoint set class is working, create a new class ***mazeGen*** to generate a maze using the disjoint sets method. The class will provide some simple functions to generate a maze and create a maze output file and a text version of the maze. The maze output file can be used to create a graphic image of the file (as shown).

A maze generation main will be provided that performs a series of tests. Refer to the UML descriptions for implementation details.



Part B:

Create and submit a brief write-up including the following:

- Name, Assignment, Section.
- Summary of the ***disjoint sets*** data structure.
- Note some applications for the disjoint sets data structure.
- Big-O for the various operations (find and union).

Submission:

- Submit a compressed zip file of the program source files, header files, and makefile via the on-line submission by 4:00 PM.
- Submit a copy of the write-up (open document, word, or PDF format).

All necessary files must be included in the ZIP file. The grader will download, uncompress, and type **make**. You must have a valid, working *makefile*.

1 For more information, refer to: http://en.wikipedia.org/wiki/Disjoint-set_data_structure

Maze Generation

In general, a good maze will have a choice of multiple paths, where some paths lead to the exit and some lead to a dead-end. There should be no unreachable areas in the maze as that can make the maze to easy.

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

As shown above, a maze can be logically represented as a basic grid, with each cell numbered. A maze ***wall*** can be represented as the boundary between two cells in the grid. For example, **`WALL 5 6`** is highlighted in the grid. It should be obvious that a wall can only exist between adjacent cells. As such, **`WALL 11 17`** would not be valid. Additionally, we assume all external walls are solid, excluding the start and exit walls.

Since each cell can be viewed as a set, for this example we have 25 disjoint sets, {0}, {1}, {2}, {3}, …, {24}. If we perform a *find(5)* and a *find(6)*, we will determine if they are currently connected or not. Specifically, if *find(5)* ≠ *find(6)* they are not in the same set and thus not connected. If they are not already in the same set, we can combine them and logically remove the wall by performing a *union(5, 6)*. However, if they are already in the same set (through other walls already being removed), then the wall should stay.

To generate mazes, start with a logical maze grid with all the walls in place. Generate and store all the interior walls for that maze.

- Randomly choose a wall.
- If that wall connects two disjoint sets of cells, then remove the wall and union the two sets. *Note*, can just mark the wall as being removed.
- Continue until all cells are in one set.

All walls not removed are part of the final maze.

To represent the walls, a multi-dimension array will be used. The array will be dynamically allocated to the exact number of possible interior walls for the given give size. The number of interior walls and resulting wall indexes will be based on the maze size. For the 5x5 maze, there are exactly 40 interior walls. The walls for the 5x5 maze are listed before for reference.

first wall	second wall
0	1
1	2
2	3
3	4
5	6
6	7
7	8
8	9
10	11
11	12

first wall	second wall
12	13
13	14
15	16
16	17
17	18
18	19
20	21
21	22
22	23
23	24

first wall	second wall
0	5
1	6
2	7
3	8
4	9
5	10
6	11
7	12
8	13
9	14

first wall	second wall
10	15
11	16
12	17
13	18
14	19
15	20
16	21
17	22
18	23
19	24

Walls that are removed, can be marked with invalid values (such as -1, -1).

Class Descriptions

- Disjoint Sets Class
  The disjoint sets class will implement functions specified below.

disjointSets
-setSize: int
-links: *int
-ranks: *int
-MIN_SIZE = 10: static constexpr int
+disjointSets(int=10)
+~disjointSets()
+entries() const: int
+printSets() const: void
+setUnion(int, int): int
+setFind(int): int

Function Descriptions

- The *disjointSets()* constructor should initialize the links and ranks arrays to an empty state. The parameter must be checked to ensure it is ≥ MIN_SIZE. If invalid, the default value should be used.
- The *~disjointSets()* destructor should delete the dynamically allocated memory.
- The *entries()* function should return the current set size.
- The *printSets()* function should print the current disjoint set status, links and ranks, with the index. Refer to the example output for formatting.
- The *setUnion()* function should perform a *union-by-rank* operation between the two passed sets and return the parent. The passed set numbers must be range checked.
- The *setFind()* functions should search the parent of the passed set. *Note,* this may be the set itself. The passed set number must be range checked. Additionally, the function should perform path compression.

- Maze Generation Class
  The maze generation class will implement functions specified below.

mazeGenerator
-rows: int
-cols: int
-walls: **int
-MIN_ROWS = 10: static constexpr int
-MIN_COLS = 10: static constexpr int
-MAX_ROWS = 100000: static constexpr int
-MAX_COLS = 100000: static constexpr int
+mazeGenerator()
+~mazeGenerator()
+getSize(int &, int &) const: void
+setSize(int, int): bool
+generate(): void
+printMazeData(const string) const: bool
+printMazeText() const: void
+getArguments(int, char *[], int &, int &, string &, bool &): bool
-randomize(): void

Function Descriptions

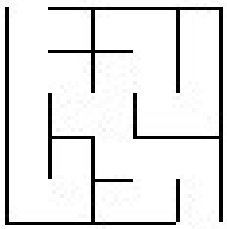
- The *mazeGenerator()* constructor should initialize the class variables as appropriate.
- The *~mazeGenerator()* destructor should delete the dynamically allocated memory.
- The *getSize()* function should return the current maze size, rows and columns in the order, of the current maze.
- The *setSize()* function should set the current maze size, rows and columns in the order, of the current maze, ensuring they are both ≥ MIN_ROWS and MIN_COLS and ≤ MAX_ROWS and MAX_COLS. If not, the function should return false. If the arguments are valid, the multi-dimension walls array, of the appropriate size, should be created and initialized.
- The *generate()* function should generate a new maze based on the disjoint sets method (see description).
- The *randomize()* function will re-order the alls array in a random order using the permutations method (as discussed in class). Select a random index, swap with the last and repeat shrinking the size.
- The *getArguments()* function accept and validate the command line arguments. The arguments include:

```
./maze -r <rows> -c <columns> -o <fileName> [-p]
./maze -o <fileName> [-p] -c <rows> -r <columns>
```

  Which can be entered in any order (two possible ways shown). The **-r** sets/returns the number of rows (integer), the **-c** sets/returns the number of columns (integer), the **-o** sets/returns the output file name (string), and the optional **-p** sets/returns if the maze should be displayed to the console (boolean). The validity of the numeric values is checked, not the range (as that is done in the main).

- The *printMazeData()* function should print the maze data to a file starting with the size (ROWS ***r*** COLS ***c***) and then the data about where the walls are located (e.g., WALL 1 2, etc.). If the file can be successfully created, the function should return true, and false otherwise. For example, for the 5 by 5 test maze, the file would contain the following:

```
ROWS 5 COLS 5
WALL 1 6
WALL 12 13
WALL 13 18
WALL 17 22
WALL 23 24
WALL 11 16
WALL 1 2
WALL 21 22
WALL 15 16
WALL 2 7
WALL 3 4
WALL 10 11
WALL 8 9
WALL 6 7
WALL 14 19
WALL 16 17
```



  The maze image file generated from this data is shown on the right for reference.
- The *printMazeText()* function should print maze in textual format. For example, for the 5 by 15 maze, the output would look like the following:

```
+  +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|  |                 |  |        |  |  |  |  |  |
+  +--+  +  +--+--+--+  +  +--+--+  +  +  +  +
|        |  |        |              |        |
+--+  +  +  +  +  +  +--+  +  +  +--+  +--+  +
|     |  |     |  |     |  |  |     |  |     |
+  +--+  +  +--+--+--+  +--+--+  +--+  +  +--+
|  |  |  |     |        |     |        |     |
+  +  +--+--+  +--+--+  +  +--+  +  +--+--+  +
|        |     |                 |  |        |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+  +
```

  The calling routine will display appropriate titles.

Refer to the example executions for output formatting. Make sure your program includes the appropriate documentation. See Program Evaluation Criteria for CS 302 for additional information. ***Note, points will be deducted for especially poor style or inefficient coding.***

ImageMagick[2]

An open source utility for image conversions, imageMagick, will be used. ImageMagick is a software suite to create, edit, compose, or convert bitmap images. It can read and write images in a variety of formats like GIF, JPEG, PNG, BMP, Postscript, and TIFF. It must be installed prior to use. Installation is as follows:

```
sudo apt update
sudo apt upgarde
sudo apt install imagemagick
```

You must be connected to the Internet for the installation.

2 For more information, refer to: https://www.imagemagick.org/script/index.php

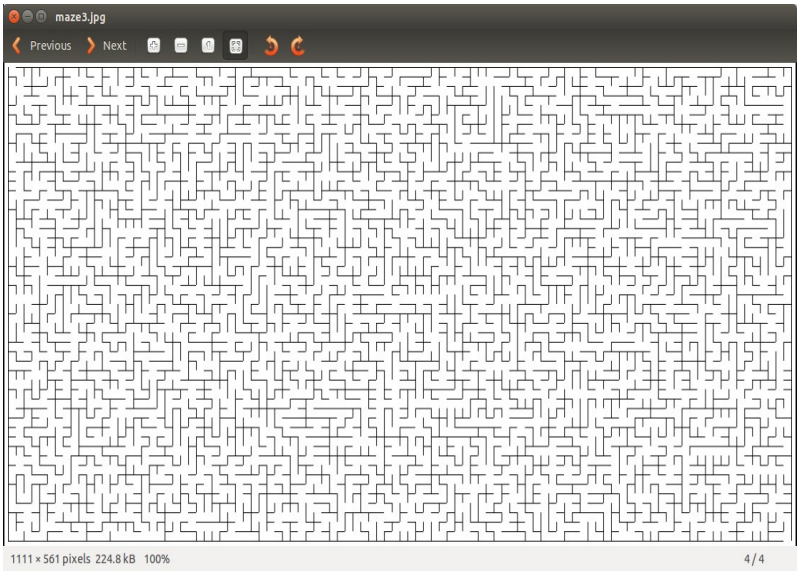
Maze Data File Conversion

A provided program will read the maze data file and convert to a image (PPM) format. The utility *ImageMagick* can convert the PPM file into a JPG or GIF image. First, install *imageMagick* via the software center.

The provided program and ImageMagick can be used as follows:

```
cat mazeFile3.txt | ./maze_ppm 5 | convert - maze3.jpg
```

Which will read the **mazeFile3.txt** file, pipe it to the maze conversion program (which output PPM data, and then pipe that to the *imageMagick* conversion utility which outputs in JPG format.



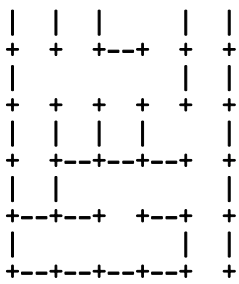
Example Execution:

Below is an example output for the test script and a program execution for the main.

```
ed-vm% ./testDJclass
*****************************************************************
CS 302 - Assignment #10
Disjoint Sets

Initial State:
  index:  0  1  2  3  4  5  6  7  8  9
  links: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
  ranks:  1  1  1  1  1  1  1  1  1  1

---------------------------------------
  union(0,1) -> 1
  union(2,3) -> 3
  union(4,5) -> 5

New State 1:
  index:  0  1  2  3  4  5  6  7  8  9
  links:  1 -1  3 -1  5 -1 -1 -1 -1 -1
  ranks:  1  2  1  2  1  2  1  1  1  1

---------------------------------------
  union(1,3) -> 3
  union(5,6) -> 5
  union(5,7) -> 5
  union(5,8) -> 5
```

```
New State 2:
  index:  0  1  2  3  4  5  6  7  8  9
  links:  1  3  3 -1  5 -1  5  5  5 -1
  ranks:  1  2  1  3  1  2  1  1  1  1

---------------------------------------
  setFind(1): 3
  setFind(2): 3
  setFind(4): 5
  setFind(7): 5

New State 3:
  index:  0  1  2  3  4  5  6  7  8  9
  links:  1  3  3 -1  5 -1  5  5  5 -1
  ranks:  1  2  1  3  1  2  1  1  1  1

---------------------------------------
  union(0123,45678) -> 3

New State 4:
  index:  0  1  2  3  4  5  6  7  8  9
  links:  1  3  3 -1  5  3  5  5  5 -1
  ranks:  1  2  1  3  1  2  1  1  1  1

---------------------------------------
  setFind(3): 3
  setFind(5): 3
  setFind(7): 3

New State 5:
  index:  0  1  2  3  4  5  6  7  8  9
  links:  1  3  3 -1  5  3  5  3  5 -1
  ranks:  1  2  1  3  1  2  1  1  1  1

---------------------------------------
  setFind(0): 3

New State 6:
  index:  0  1  2  3  4  5  6  7  8  9
  links:  3  3  3 -1  5  3  5  3  5 -1
  ranks:  1  2  1  3  1  2  1  1  1  1

---------------------------------------
  setFind(4): 3
  setFind(6): 3
  setFind(8): 3

Final State:
  index:  0  1  2  3  4  5  6  7  8  9
  links:  3  3  3 -1  3  3  3  3  3 -1
  ranks:  1  2  1  3  1  2  1  1  1  1


*****************************************************************
Game Over, thank you for playing.
ed-vm%
ed-vm%
ed-vm%
```

```
ed-vm%
ed-vm% ./main -p
****************************************************************
CS 302 - Assignment #10
Maze Generator


Maze file mazeFile0.txt created.
----------------------------------------
Small Maze:

+  +--+--+--+--+
|     |     |  |
+  +--+--+  +  +
|     |     |  |
+  +  +  +  +  +
|  |     |     |
+  +--+  +--+--+
|  |  |        |
+  +  +--+  +  +
|     |     |  |
+--+--+--+--+  +

----------------------------------------

Maze file mazeFile1.txt created.
----------------------------------------
Small Maze:

+  +--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|  |                 |  |        |  |  |  |  |
+  +--+  +  +--+--+--+  +  +--+--+  +  +  +  +
|        |  |        |              |        |
+--+  +  +  +  +  +  +--+  +  +  +--+  +--+  +
|     |  |     |  |     |  |  |     |  |     |
+  +--+  +  +--+--+--+  +--+--+  +--+  +  +--+
|  |  |  |     |        |     |        |     |
+  +  +--+--+  +--+--+  +  +--+  +  +--+--+  +
|        |     |                 |  |        |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+  +

----------------------------------------

Maze file mazeFile2.txt created.
----------------------------------------
Small Maze:

+  +--+--+--+--+
|  |     |     |
+  +  +--+--+  +
|  |     |     |
+  +  +  +  +  +
|     |     |  |
+--+--+--+  +  +
|           |  |
+--+--+  +  +--+
|        |     |
+--+  +  +--+--+
|     |        |
+--+  +--+  +  +
|     |     |  |
+  +--+  +--+--+
|     |        |
+--+  +--+--+  +
|     |  |     |
+  +  +  +--+--+
|  |           |
+--+--+--+  +--+
```



```
---------------------------------------

Maze file mazeFile3.txt created.

Maze file mazeFile4.txt created.

Maze file mazeFile5.txt created.

**********************************************************************
Game Over, thank you for playing.
ed-vm%
```