David Rozmajzl

FILE MANIFEST
1. Project4.pdf
2. buildMaze.cpp
3. maze.cpp
4. maze.h
5. DisjSets.cpp
6. DisjSets.h
7. mazeCell.h

Project 4

The purpose of this project was to create a maze using disjoint sets. The user can enter any two integers greater than one to be the dimensions of the maze. For example, if the user were to enter 3 rows and 5 columns, the maze would have 15 cells each surrounded by four walls. The first and last cells are opened. The program then randomly generates two numbers that will be the index of the cells. If the two cells are neighbors, the *smashWall()* function breaks down the wall between the cells if one exists, and the two cells are run through the *unionSets()* function which adds them to the same set. When two cells are connected, the *find()* method indicates that they are in the same set.

Because the program can only break down one wall at a time, the user has the option to show the maze after each wall breaks down or just show the start and end maze. By entering ‘y’, the program displays the maze after each wall break. Entering ’n’ shows only the start and end maze. A good reason to pick ’n’ is when the maze is very large because the larger it gets, the longer it takes to print the maze. And it also requires more wall breaks.

a. If two cells are connected in the maze, they are in the same set within the DisjSets class. However, the indices are not organized in actual sets. Instead, we use the vector *s* in the DisjSets class. The *unionSet()* method organizes the indices in such a way that if an index is in the same “set” as another index, they both point back to the same root that is indicated by a negative number. The *find()* method is what finds out which root node a cell index points to. If *find(*first index*)* returns the same number as *find(*second index*)*, then the two indices are in the same “set” and, therefore, connected in the maze.