Michael Goheen
CS 3114
4/12/12

Project 3 Report

Graph Data Structure

The graph data structure uses a 2-dimensional array to hold rooms which point north, east, south, and west to walls. These values are Booleans that tell whether the adjacent walls currently exist. They are false for a missing wall. The edges are modeled with an x, y, and boolean flag to tell whether the edge is horizontal or vertical. The graph also holds information about the number of rooms, columns, and rows. Each vertex and edge can be accessed in constant time.

Random Wall List

The random wall list is modeled in an array of 1-dimensional internal walls. For each internal wall, a random number between 0 and remaining internal walls determines the index for which wall to pick out of the array. Once the wall is picked, it is swapped with the last element and the total number of remaining internal walls is decremented. This ensures that the next random number will not pick the same wall over and over again. Each wall can be picked and removed from this list in constant time.

Union/Find Data Structure

After a random wall is picked, the rooms adjacent to that wall are retrieved. Then, a method is called to determine if these two rooms are in the same set. If they are in the same set, the wall is not removed. If they are not in the same set, the wall is removed and the two rooms are unioned into the same set. The union is done by height and the find (or getLeader method as I called it) does a path compression. Together with union by height and path compression, the operations should be near constant time.

BFS

The BFS uses a queue in order to find the shortest path. It starts with an initial room and begins adding adjacent rooms to the queue. As each adjacent room is dequeued, it looks at that room until the destination room is found. At this point, it prints the path from stored pointers to the parent room. My implementation appears to be a little buggy some of the time on larger mazes. Not quite sure why this is. However, the algorithm is done in O (V+E) time as every operation to get data in the maze is in constant time. It looks at all rooms (V = C*R) and all edges (E).

DFS

The DFS uses recursion to get each of the paths in the maze. It looks through all the adjacent rooms to the initial room and recurses on that room until it gets to the destination room. It keeps track of the path of rooms and edges along the way by marking and unmarking them appropriately. Once it gets to the destination room in the base case, it prints the maze with the path found and updates the number of paths by 1. It also keeps track of the number of rooms in the path by incrementing and decrementing it appropriately. It does this until every path is found. I’m not entirely sure, but I think the DFS is done in around O ((V+E) * (some constant) W). Since W, or the number of walls removed after initial wall removal, creates some constant * W cycles in the maze and DFS has to look through all possible paths.