CSCI 2270: Data Structures Final Project WriteUp

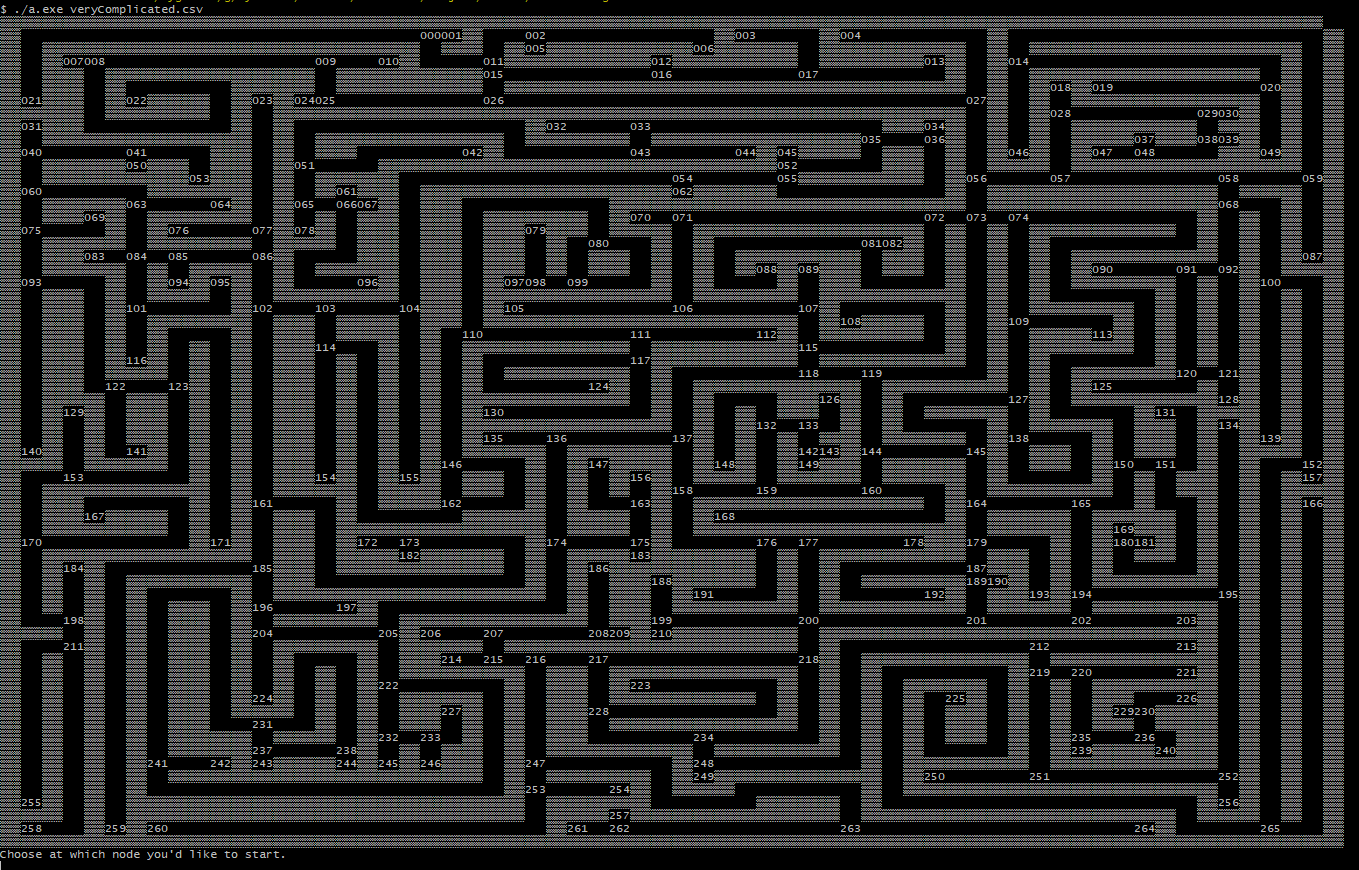
Adam Buencamino, Benjamin Page, Luke Roberson

Abstract:

The main idea of our project is to solve mazes and find the fastest solution with relatively small time-complexity. We’ve completed a realistic goal to solve two dimensional mazes with O(n) complexity where n is the number of intersection points in the maze. The goals we set out for ourselves included: finding the shortest path between two user specified nodes, being able to process a maze of any dimension as long as the ‘paths’ and ‘walls’ can be indicated by a single pixel, and printing out a cohesive representation of the graph for easy user interaction.

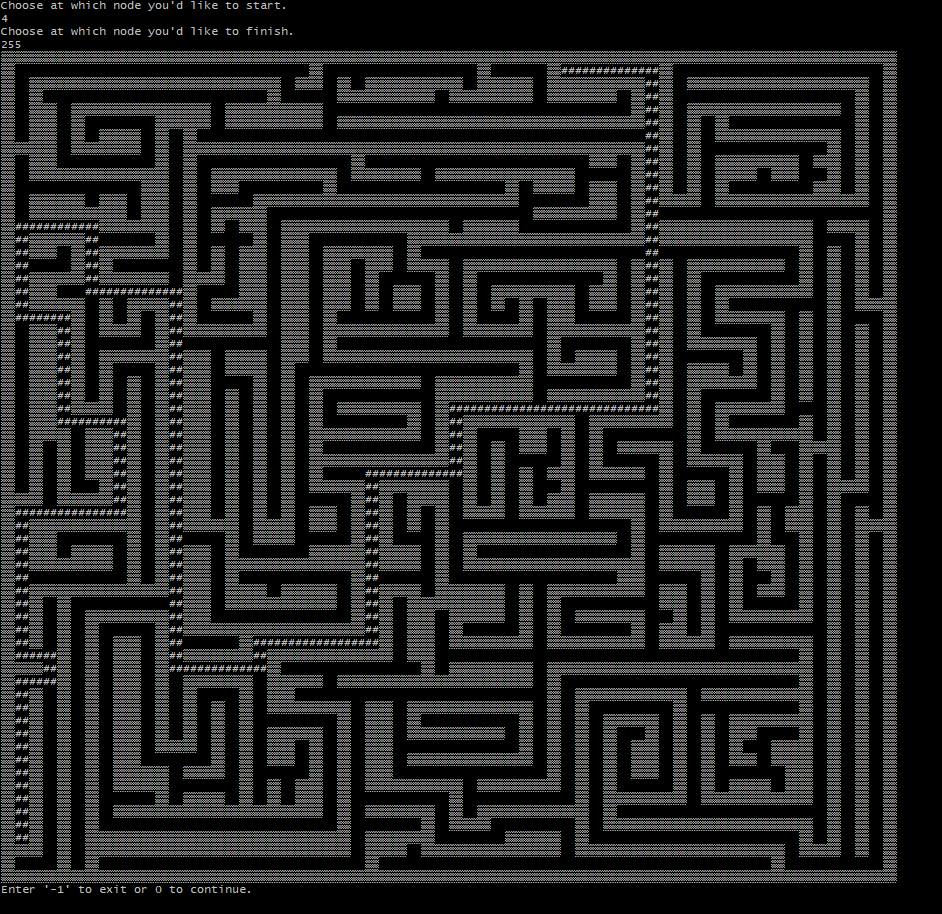
Data Structure:

The data structures we’ll be using for this algorithm are graphs, stacks, and arrays. Using a custom implemented graph for our data structure, we first read in an image file to MatLab and output a csv of 1s and 0s (paths and walls), next our c++ code reads in this file and checks for any 1 surrounded by other 1s to decide where intersections and dead ends exist in the maze. The are both inserted into our weighted graph where the number of pixels between vertices are the weight of the edges. We’ve created a mapCrawler function that can traverse between vertices whether or not there are corners in the connecting path. The stack will hold the path of traversal where a node will be added every time the algorithm traverses, and popped every time it back tracks. The array will hold the directions as ints (1-4) and upon finding the best solution will output the path as directions (N,E,S,W) respectively.



Solution:

We’ve implemented a dijkstra type algorithm for traversing the graph. The custom aspect coming in when we save the path travelled. To get a nice visual output we save the vertices visited in a specific path in a ‘parent’ list. A modified version of our map crawler function to recursively traverse backwards from the end node, through its consecutive parents, eventually reaching the start node. As the function ‘crawls’ it saves each coordinate of the map that it touches to 2d bool array which we then use to superimpose the path travelled onto the original maze representation.



Conclusion:

While there were definitely a lot of unexpected hiccups in trying to implement all of our planned function, we were able to produce a final product that we’re very happy with. Our code is a two-step process, first going through MatLab to convert the image file to csv and then going into the bulk of our work which processes the csv to a graph. We had to make a couple original functions that over the course of our experimentation became more and more efficient. We are most proud of mapCrawler pathFinder, and printMaze which became the basis for other helper functions later in the process. Many hours were spent perfecting these functions so that we could produce a product capable of the O(n), n being number of vertices, time complexity we set out to achieve. We are proud to present our maze solving algorithm.