Michael Gonzalez

CS 2302 Data Structures

Olac Fuentes

04/16/2019

Lab 6 - Maze

For this lab, we will use a disjoint set forest to build a maze. The maze should contain a collection of cells separated by walls in such a way that there is exactly one simple path (that is, a path that does not visit any cell more than once) separating any two cells.

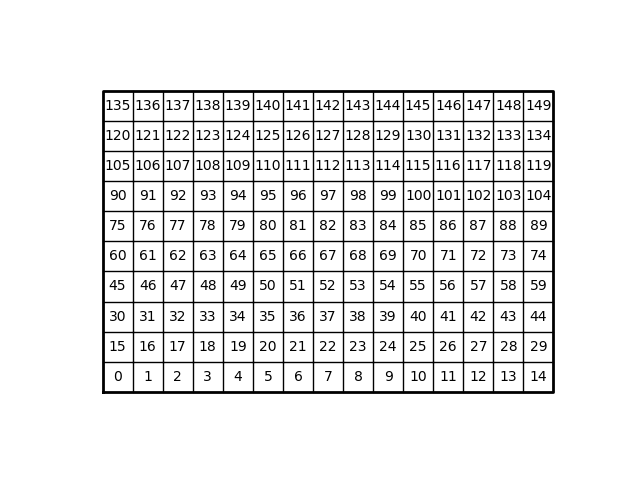
My solution for This lab is fairly simple, first I create a function called CountSets. This method takes a disjoint set forest S. First thing it does is to make a counter called num\_sets and sets it to zero. Next, it goes into a for loop, from zero to the range of the length of S,( for i in range(len(S)) ). Then it goes into an if statement and makes a comparison if the current S[i] is less than zero making it the end of the set, if true it adds 1 to the counter. Out of the if whether the condition was true or false it calls the disjoint set forest function find\_c(S,i) to compressed the path of S[i] as specified in the lab assignment. Once it is out of the for loop it simply returns the counter which is the number of sets.

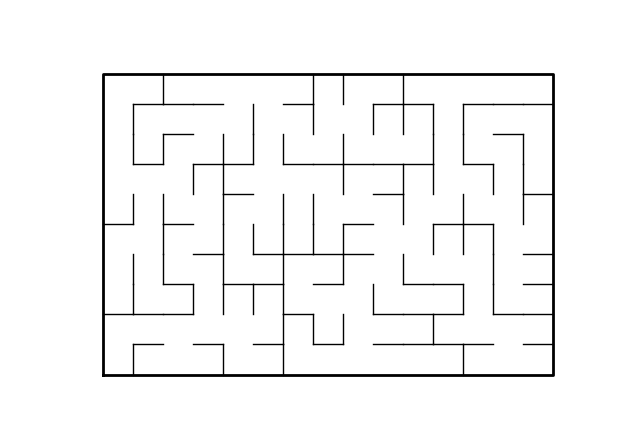
Next, for the main part of the code, first use the plot command plt.close(“all”) then set maze rows and maze cols to whatever number you want them to be.next I created a variable called num\_cells which stores the multiplication between rows and columns. Using the wall\_list method provided in class I created the list of walls and also using the given function draw\_maze it drew a maze with all the numbers in their respective slot to verify that its creation was a success. Here I made a variable to store the current time to test the running time for my code. I then created the disjoint set forest using the function DisjointSetForest with the number of cells in my maze, and I decided to draw it to check that the forest was created successfully. Then the code goes into a while loop with the condition, while CountSets(S)>1. Meaning that the next part will run until there is only 1 set which is what we want. Inside the while loop I assign d random integer between zero and the number of walls -1, next it prints that it will be removing the d wall (wall[d]), then it goes into an if statement where if the two cells involved in the wall removal are part of different sets then it unites them by the union function or the union by size function and pops the d from the wall list walls. Then, I draw the set forest and the maze to check if they have been made correctly. Lastly it records the elapsed time of the entire process which is the current time minus the start time from the beginning and prints out the running time for the maze with size M\*N. averaged out to a time of 2 seconds for the 10\*15 maze

Experiments/Results

1a. Maze size 10\*15:

Union: Union

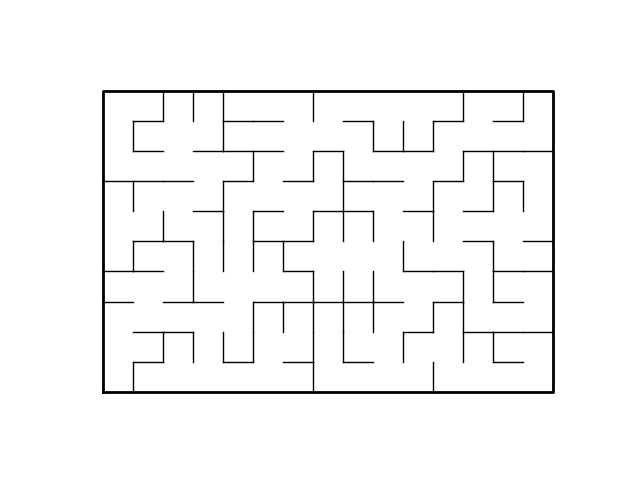
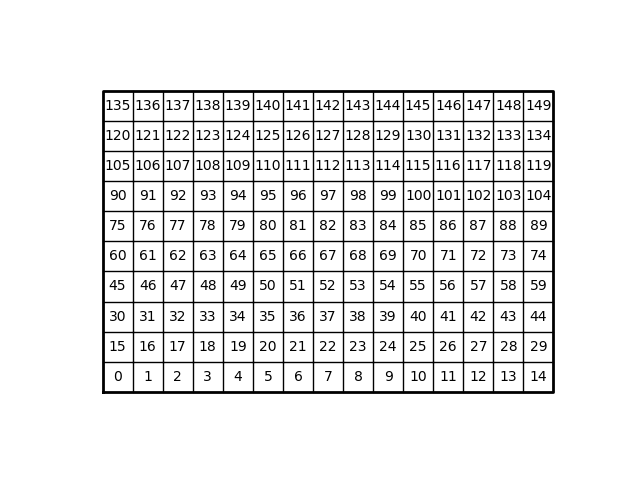




Running time is: 1 seconds

1b. Maze size 10\*15:

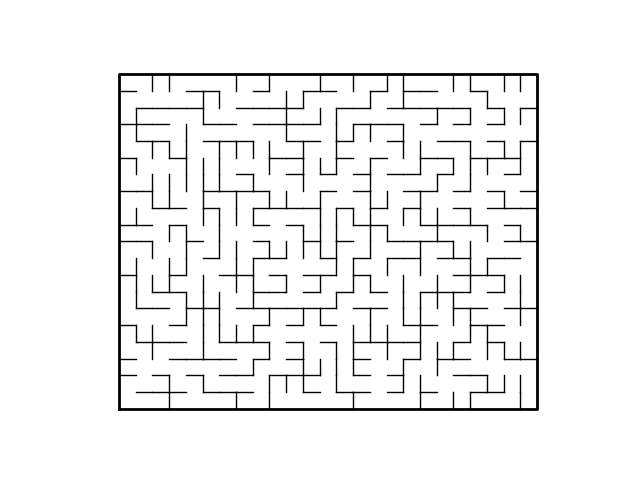
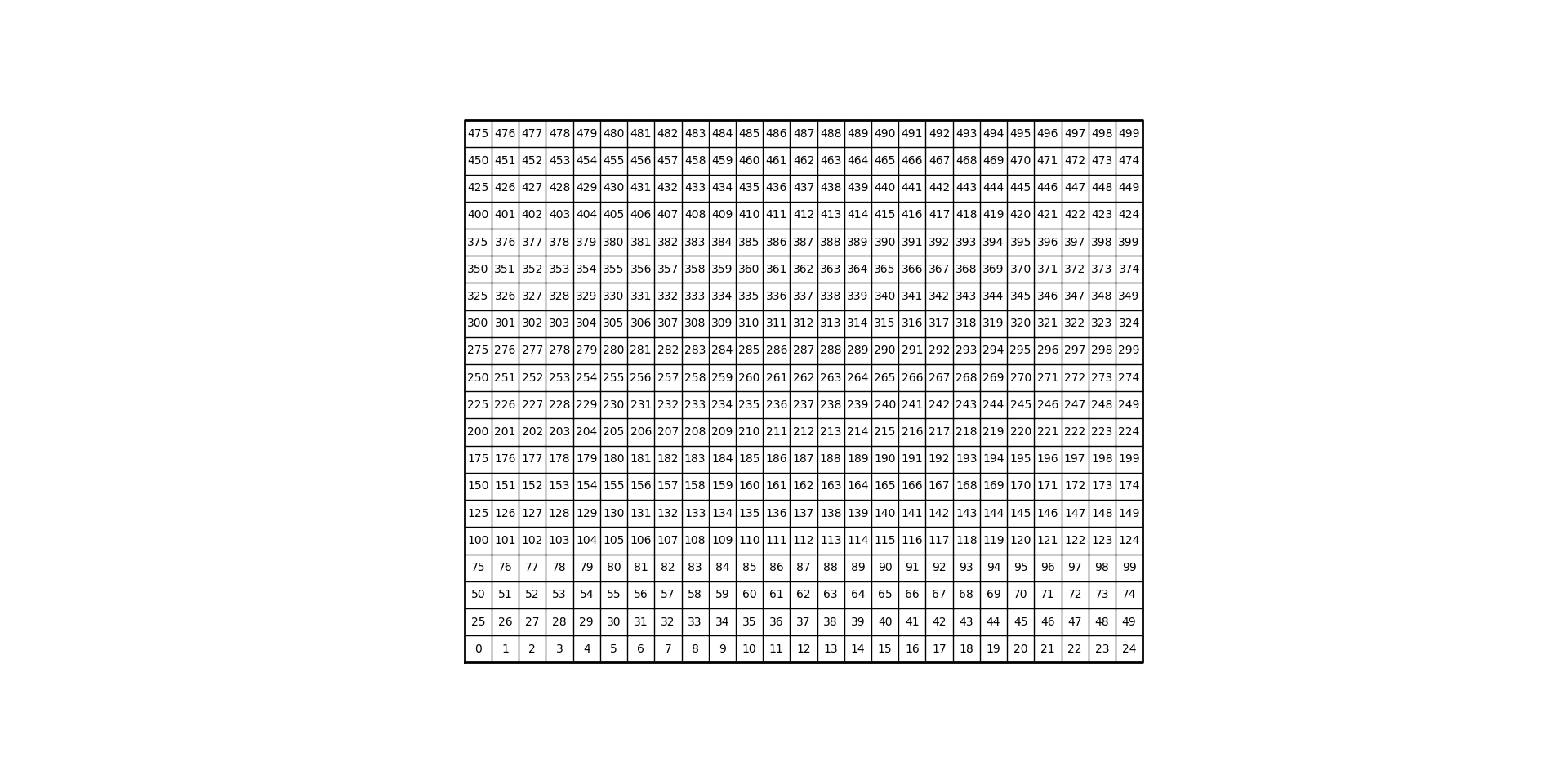
Union: Union\_by\_size



Running time is: 1 seconds

1b. Maze size 20\*25:

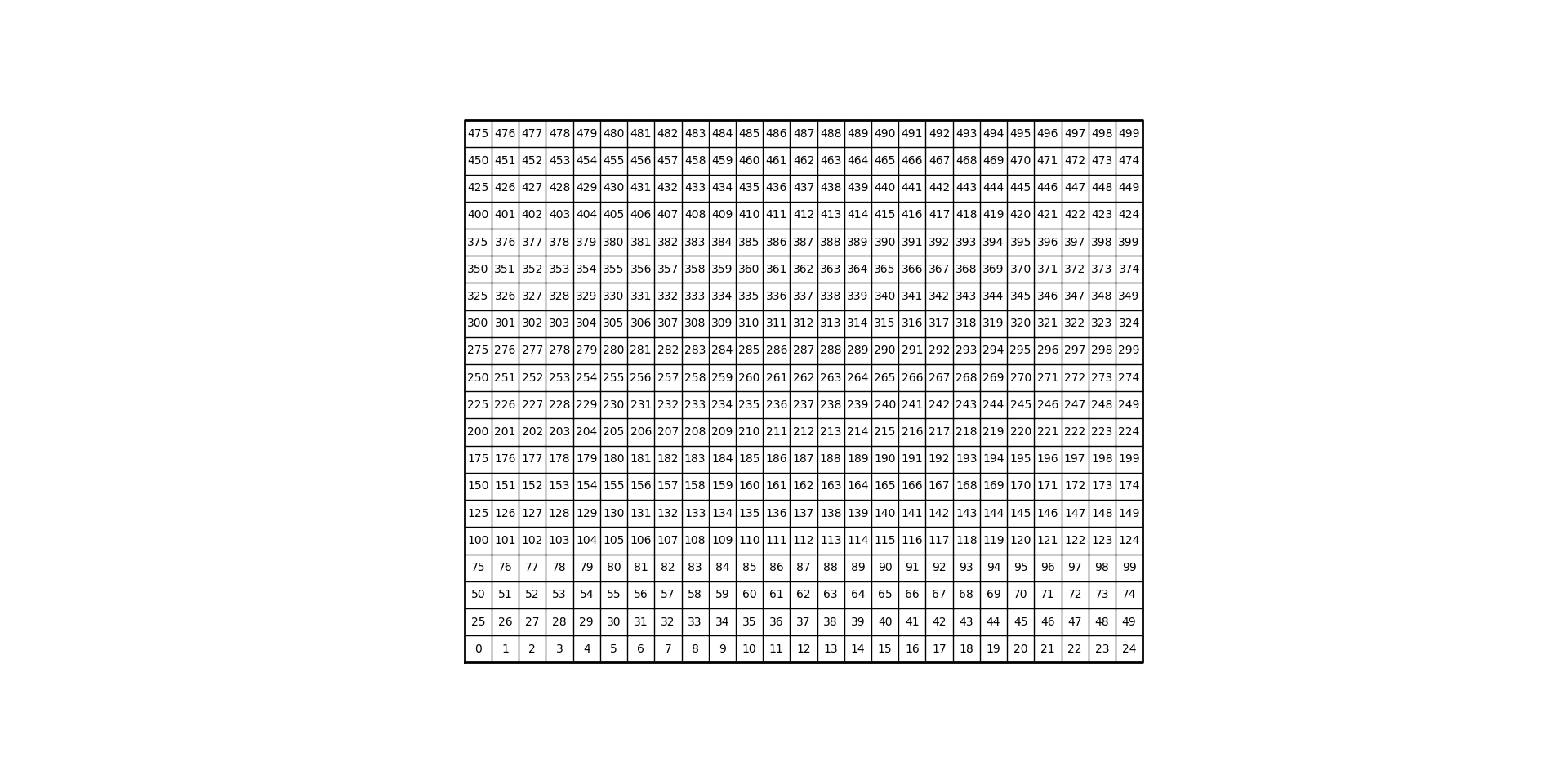
Union: Union:

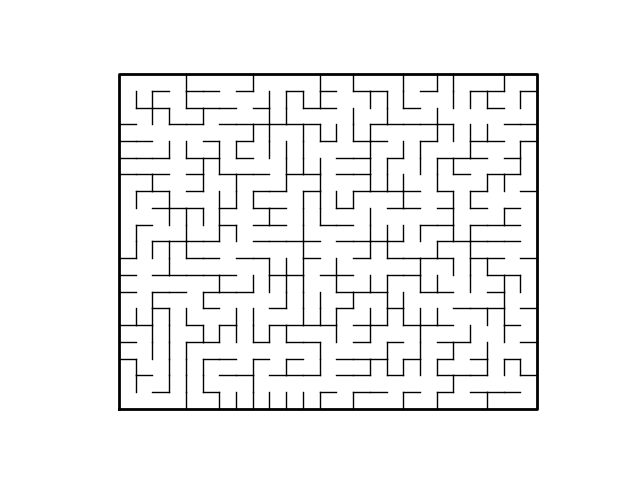


Running time 5 seconds

1b. Maze size 20\*25:

Union: Union\_by\_size

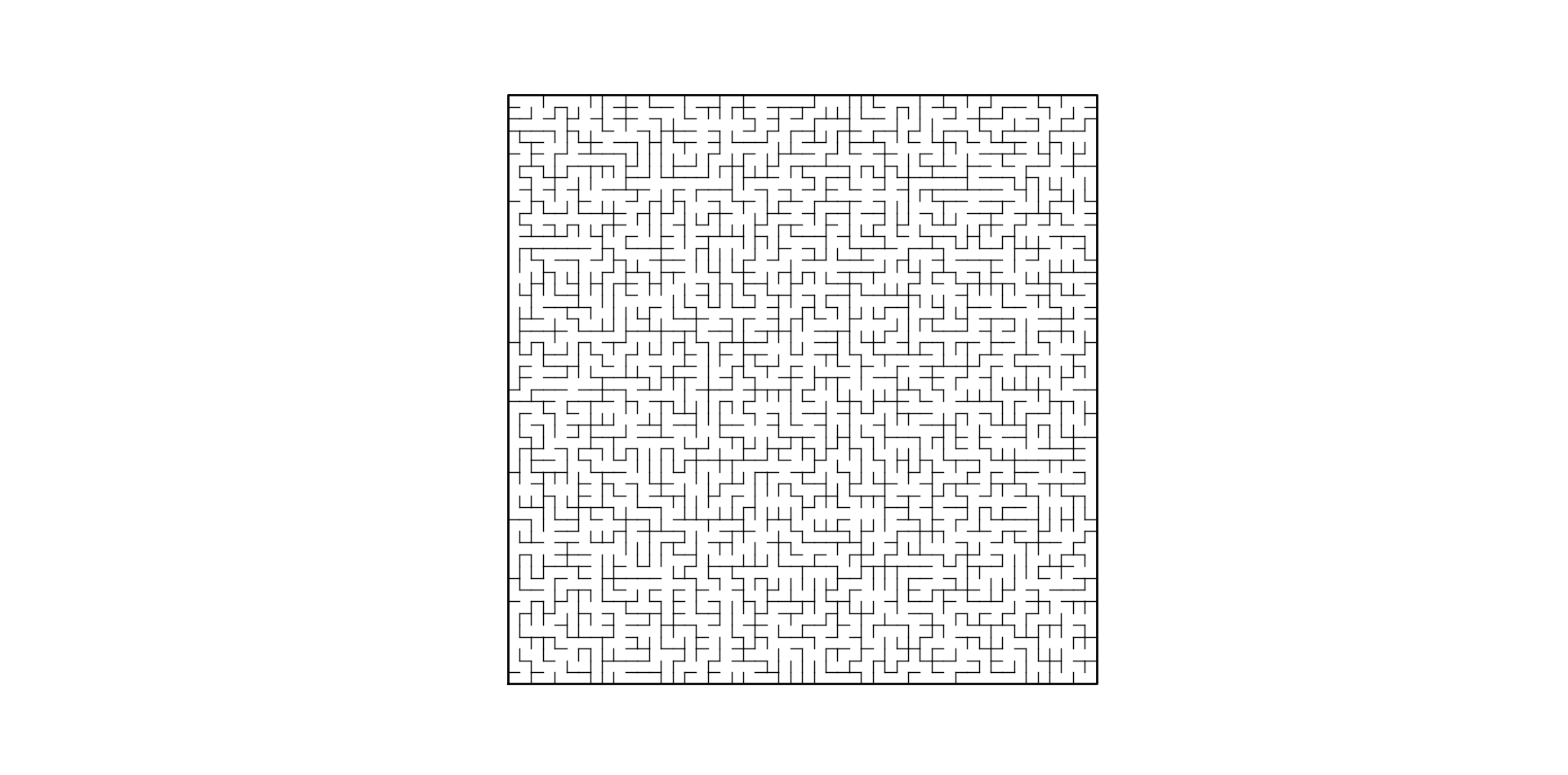




Running time is: 5 Seconds

1c. Maze size 50\*50:

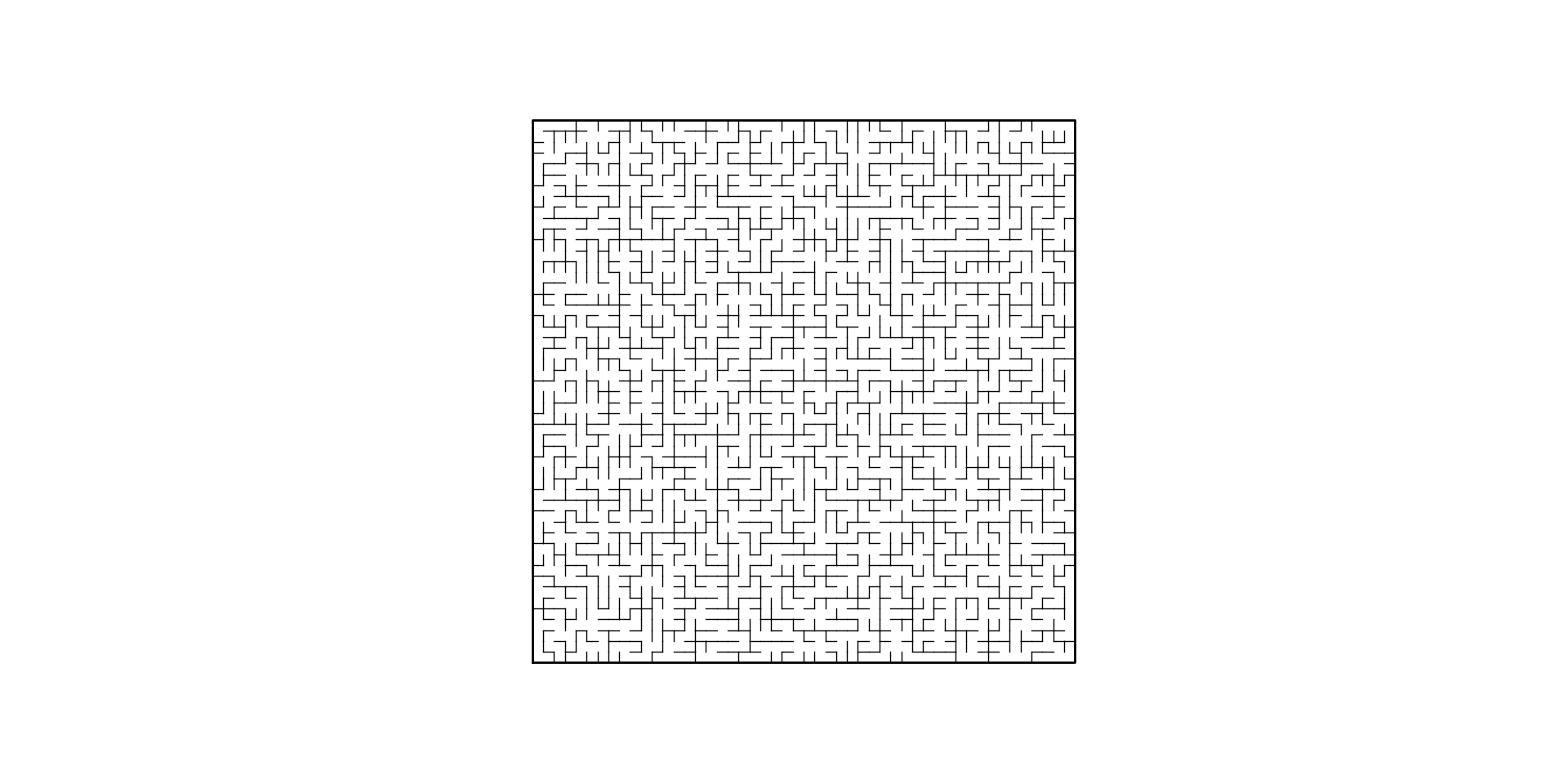
Union: Union



Running time is: 76 Seconds

1c. Maze size 50\*50:

Union: Union\_by\_size:



Running time is: 71 Seconds

Conclusion

In conclusion, I learned how to design a maze using the data structure if disjoint set forests, and how to calculate its running time. I have now a better appreciation Disjoint set forests and its real-life application. I have become more comfortable with coding in python than in lab 5 and I believe that I will be able to learn more from future labs to come and hopefully make some of my friends to solve my mazes.

I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class.

– Michael Gonzalez

Appendix – code

#Author: Michael Gonzalez

#Course: CS 2302 Data Structures

#Lab 6

#TA: Anindita Nath & Eduuardo Lara

#Purpose:the purpose of this lab is to modify the code given to create a maze.

#Your maze should contain a collection of cells separated by walls in such a way

#that there is exactly one simple path (that is, a path that does not

#visit any cell more than once) separating any two cells.

# Last modified April,12 2019

import matplotlib.pyplot as plt

import numpy as np

import random

import time

import dp

def draw\_maze(walls,maze\_rows,maze\_cols,cell\_nums=False):

fig, ax = plt.subplots()

for w in walls:

if w[1]-w[0] ==1: #vertical wall

x0 = (w[1]%maze\_cols)

x1 = x0

y0 = (w[1]//maze\_cols)

y1 = y0+1

else:#horizontal wall

x0 = (w[0]%maze\_cols)

x1 = x0+1

y0 = (w[1]//maze\_cols)

y1 = y0

ax.plot([x0,x1],[y0,y1],linewidth=1,color='k')

sx = maze\_cols

sy = maze\_rows

ax.plot([0,0,sx,sx,0],[0,sy,sy,0,0],linewidth=2,color='k')

if cell\_nums:

for r in range(maze\_rows):

for c in range(maze\_cols):

cell = c + r\*maze\_cols

ax.text((c+.5),(r+.5), str(cell), size=10,

ha="center", va="center")

ax.axis('off')

ax.set\_aspect(1.0)

def wall\_list(maze\_rows, maze\_cols):

# Creates a list with all the walls in the maze

w =[]

for r in range(maze\_rows):

for c in range(maze\_cols):

cell = c + r\*maze\_cols

if c!=maze\_cols-1:

w.append([cell,cell+1])

if r!=maze\_rows-1:

w.append([cell,cell+maze\_cols])

return w

###################################################

# LAB START

def CountSets(S):

num\_sets = 0

for i in range(len(S)):

if S[i]<0:

num\_sets += 1

dp.find\_c(S,i)

return num\_sets

plt.close("all")

maze\_rows = 10

maze\_cols = 15

num\_cells = maze\_rows\*maze\_cols

walls = wall\_list(maze\_rows,maze\_cols)

draw\_maze(walls,maze\_rows,maze\_cols,cell\_nums=True)

start = time.time()

S = dp.DisjointSetForest(num\_cells)

dp.draw\_dsf(S)

while CountSets(S) > 1:

d = random.randint(0,len(walls)-1)

print('removing wall ',walls[d])

if dp.find(S,walls[d][0]) != dp.find(S,walls[d][1]):

dp.union(S,walls[d][0],walls[d][1])

# dp.union\_by\_size(S,walls[d][0],walls[d][1])

walls.pop(d)

dp.draw\_dsf(S)

draw\_maze(walls,maze\_rows,maze\_cols)

elapsed\_time = time.time()-start

print("Running time for Maze construction:", round(elapsed\_time),"seconds")