Jon Munoz

Lab 2

CS2302

Professor: Olac Fuentes

TA: Anindita Nath

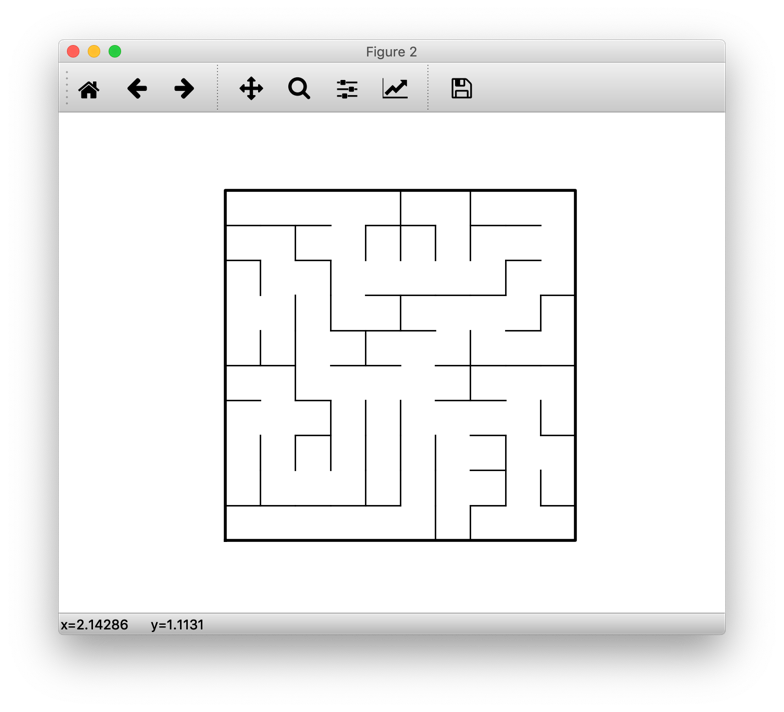
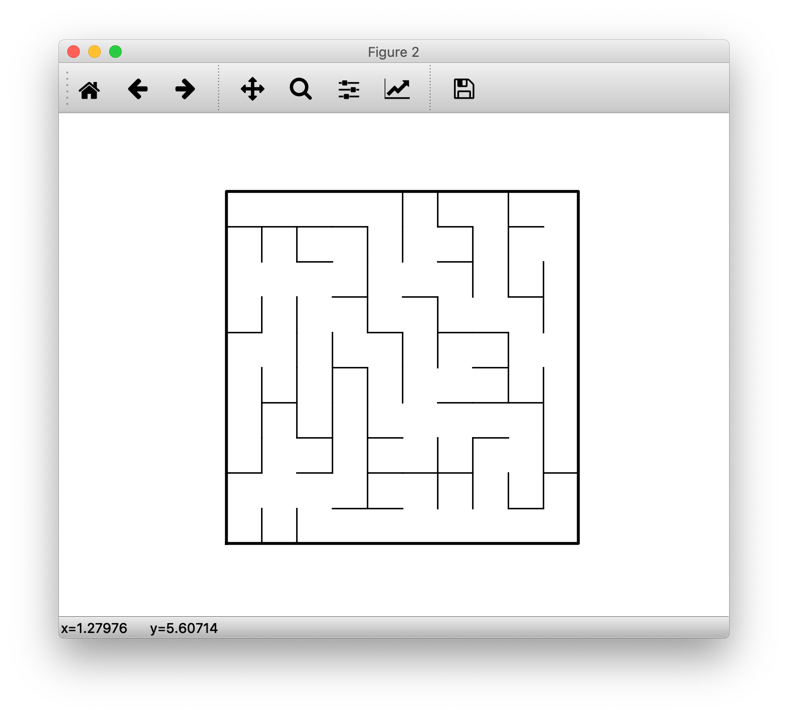
**INTRODUCTION**

For this lab we had to use disjoint set forest to help create a maze of desired sizes. We were given code to randomly chose walls to remove and we had to modify it to only remove walls if the two adjacent cells weren’t already in the same set in the DSF. We had to do these two different ways those being with normal union and find functions and then with union by size and find with path compression. Once we did this we had to then time each method to see which was faster.

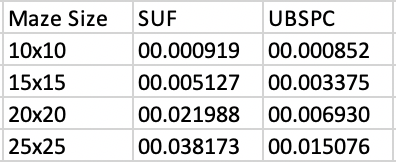
**PROPOSED SOLUTION DESIGN AND IMPLEMENTATION**

When I first started the assignment, I tried to follow the pseudo code that was provided to us. When I translated it to code, I was having problems when it came to the part of removing the wall. I feel like I overcomplicated it, so I ended up trying something else. What I ended up doing was creating a variable that stored the initial number of sets in order to use it as a counter. I then made a while loop that kept executing while that counter variable was greater than one. Within the while loop I set a variable w to a random wall within the list by using random.choice(walls) and once I got that random wall I used another variable and set it to the index of the wall with walls.index(w). Once I had this index I then compared the roots of w[0] and w[1] and if they were different then I popped the wall at the retrieved index and would union the two cells. If a union occurred, then I subtracted one from my counter variable. This process continued until the value of my counter was one and once this happened, I drew the resulting maze. This was my process for both of the different ways that we had to make the maze. I also asked the user for the dimensions that they would like for the maze as to not make it a static sized maze and also to be able to change the size without having to change the code

**EXPERIMENTAL RESULTS**

Below I have the outputs of my mazes with the first being the standard union and find maze and the second being the union by size and find with path compression maze, both are also 10 by 10 sized mazes:

Below I have the running times of the maze construction with various sizes with each time being in seconds/milliseconds. SUF is the standard union and find while UBSPC is union by size with path compression:



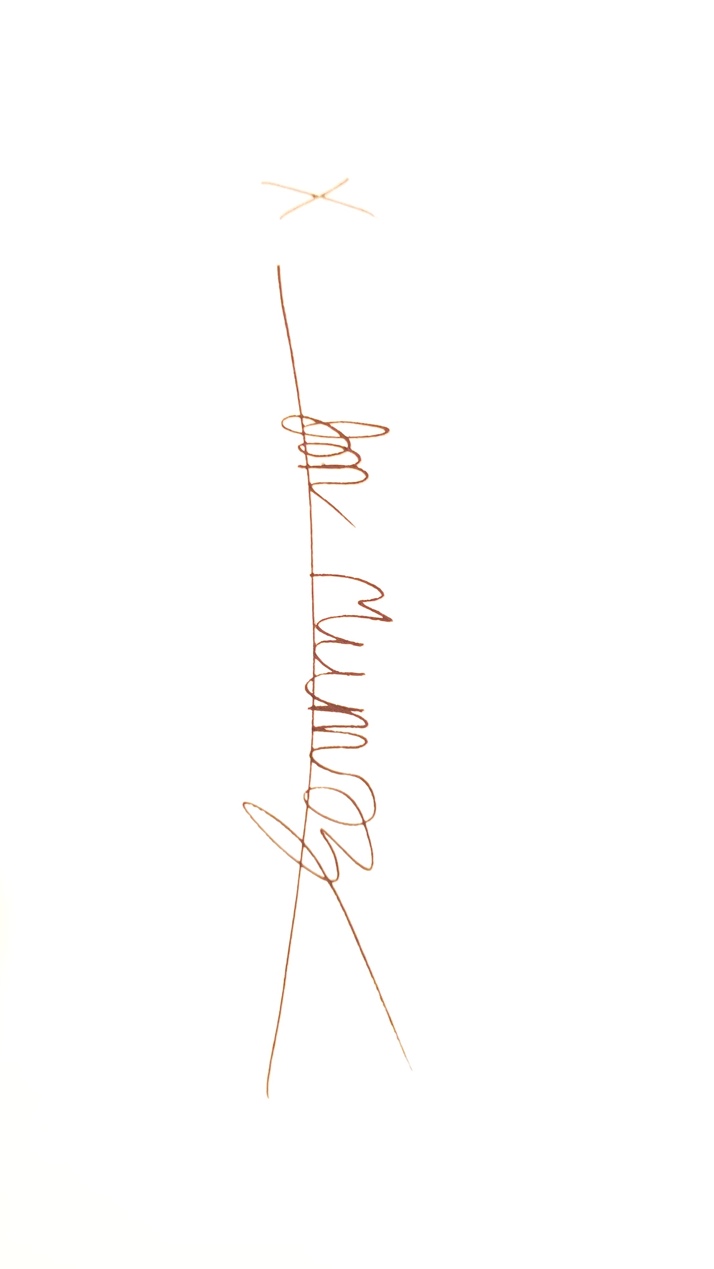
**CONCLUSION**

At the end of this lab I got more acquainted with disjoint set forest and I learned how to make a maze. One other thing that is important to me that I learned is that you can chose a random index within a list and get the index of that position with the two functions random.choice() and .index()

**APPENDIX**

**SOURCE CODE**

|  |
| --- |
| #Jon Munoz |
|  | #CS2302 Data Structures |
|  | #Lab 6 |
|  | #Instructor:Olac Fuentes |
|  | #TA:Anindita Nath |
|  | #Last Modified 4/12/19 |
|  |  |
|  | ########################################################################### |
|  | # Starting point for program to build and draw a maze |
|  | # Modify program using disjoint set forest to ensure there is exactly one |
|  | # simple path joiniung any two cells |
|  | # Programmed by Olac Fuentes |
|  | # Last modified March 28, 2019 |
|  |  |
|  | import matplotlib.pyplot as plt |
|  | import numpy as np |
|  | import random |
|  | import datetime |
|  |  |
|  | 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 |
|  |  |
|  | #the below code randomly selected walls to remove (this was given to us) |
|  | #this is the method that removes walls |
|  | #for i in range(len(walls)//2): #Remove 1/2 of the walls |
|  | # print(len(walls)) |
|  | # d = random.randint(0,len(walls)-1) |
|  | # print('removing wall ',walls[d]) |
|  | # walls.pop(d) |
|  | #draw\_maze(walls,maze\_rows,maze\_cols) |
|  | ########################################################################### |
|  |  |
|  | def DisjointSetForest(size): |
|  | return np.zeros(size,dtype=np.int)-1 |
|  |  |
|  | #finds the root of a given number i |
|  | def find(S,i): |
|  | if S[i]<0:#if S[i] < zero then youre at the root return i |
|  | return i |
|  | return find(S,S[i])#recursive call if S[i] is not less that 0 |
|  |  |
|  | #finds the root of a given number i using path compression |
|  | def findC(S,i): |
|  | if S[i] < 0:#if S[i] < zero then youre at the root return i |
|  | return i |
|  | root = findC(S,S[i])#recursively get to the root |
|  | S[i] = root#directly set the root of i to the root value |
|  | return root |
|  |  |
|  | #combines the two given values using their size to choose who get combined |
|  | def unionbySize(S,i,j): |
|  | ri = findC(S,i)#find the root of the first value |
|  | rj = findC(S,j)#find the root of the second value |
|  | if ri!=rj:#if the roots are not the same then you complete the union |
|  | if S[ri] > S[rj]:#if S[ri] is greater than S[rj] then you combine S[rj] to S[ri] |
|  | S[rj] += S[ri] |
|  | S[ri] = rj |
|  | else:#rif S[rj] is greater than S[ri] then you combine S[ri] to S[rj] |
|  | S[ri]+=S[rj] |
|  | S[rj] = ri |
|  |  |
|  | #counts the number of sets in the given disjoint set forest |
|  | def NumSets(S): |
|  | count = 0#counter |
|  | for i in range(len(S)):#traverse the entire DSF |
|  | if S[i] < 0:#if an value is less than 0 then add one to the counter |
|  | count += 1 |
|  | return count |
|  |  |
|  | #union the two given values |
|  | def union(S,i,j): |
|  | # Joins i's tree and j's tree, if they are different |
|  | ri = find(S,i) |
|  | rj = find(S,j) |
|  | if ri!=rj: # Do nothing if i and j belong to the same set |
|  | S[rj] = ri |
|  |  |
|  | plt.close("all") |
|  | ans1 = input("How many rows would you like? ")#this line gets the user input for the desired number of rows |
|  | print("Number of rows:", ans1) |
|  | rows = int(ans1)#parse the input to an int |
|  | ans2 = input("How many columns would you like? ")#this line gets the user input for the desired number of columns |
|  | print("Number of columns:", ans2) |
|  | columns = int(ans2)#parse the input to an int |
|  |  |
|  | maze\_rows = rows#set maze\_rows to the value of rows |
|  | maze\_cols = columns#set maze\_cols to the value of columns |
|  |  |
|  | walls = wall\_list(maze\_rows,maze\_cols) |
|  |  |
|  | draw\_maze(walls,maze\_rows,maze\_cols,cell\_nums=True) |
|  |  |
|  | S = DisjointSetForest(maze\_rows \* maze\_cols)#create a DSF with the size of the value of rows times the value of columns |
|  |  |
|  | numberOfSets = NumSets(S)#numberOfSets is the original number of sets to begin with |
|  |  |
|  | #Make maze using normal union and find |
|  | start = datetime.datetime.now()#start time of maze construction |
|  | while numberOfSets > 1:#keep doing the below code until there is only one set |
|  | w = random.choice(walls)#set w to a random list within the bigger list |
|  | ind = walls.index(w)#get the index of the chosen list w |
|  | if find(S,w[0]) != find(S,w[1]):#find the parents of the number in index 0 and 1 of the list and if they are different union them |
|  | walls.pop(ind)#pop the wall connecting the two numbers since we want to "union" them |
|  | union(S,w[0],w[1])#use union method to connect the two values within the DSF |
|  | numberOfSets -= 1#one set is removed so subtract 1 from the numberOfSets value |
|  | end = datetime.datetime.now()#end time of creating maze |
|  | elapsed = end - start#total time |
|  | print(elapsed) |
|  |  |
|  | draw\_maze(walls,maze\_rows,maze\_cols)#draw the new maze |
|  |  |
|  | ##Make maze using union by size and path comnpresion |
|  | #start = datetime.datetime.now()#start time of maze construction |
|  | #while numberOfSets > 1:#keep doing the below code until there is only one set |
|  | # w = random.choice(walls)#set w to a random list within the bigger list |
|  | # ind = walls.index(w)#get the index of the chosen list w |
|  | # if findC(S,w[0]) != findC(S,w[1]):#find the parents of the number in index 0 and 1 of the list using path compression and if they are different union them |
|  | # walls.pop(ind)#pop the wall connecting the two numbers since we want to "union" them |
|  | # unionbySize(S,w[0],w[1])#use unionbySize method to connect the two values within the DSF |
|  | # numberOfSets -= 1#one set is removed so subtract 1 from the numberOfSets value |
|  | #end = datetime.datetime.now()#end time of creating the maze |
|  | #elapsed = end - start#total time |
|  | #print(elapsed) |
|  | #draw\_maze(walls,maze\_rows,maze\_cols)#draw the new maze |



“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 provide inappropriate assistance to any student in the class.”