CS 2302 Data Structures

Spring 2019

MW 10:30-11:50 in CCSB 1.0202

LAB # 6

Claudio Angel Garcia

Instructor: Olac Fuentes

TA: Anindita Nath, Maliheh Zargaran

IA: Eduardo Lara

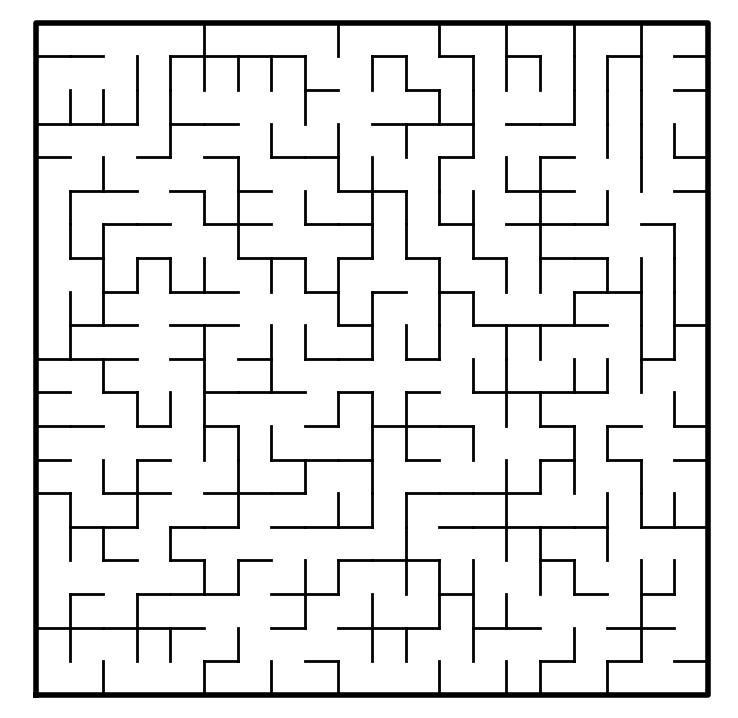
Peer leader: Erick Macik

Introduction

We are tasked to use the disjoint sets forests structure to build a maze I which we will be given a collection of cells that have four walls and the problem will be that we need to remove certain walls to be able to cross between the maze. When need to remove a wall, if the cells that were separated by that wall belonged to different sets, we must unite these sets. We repeated until all cells belong to a single set and then we can display the full maze.

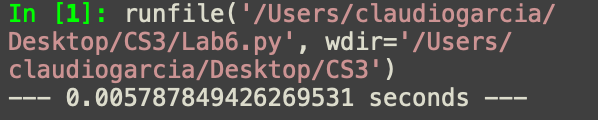
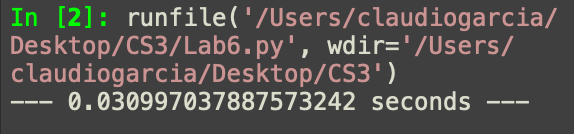
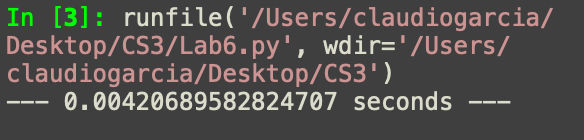
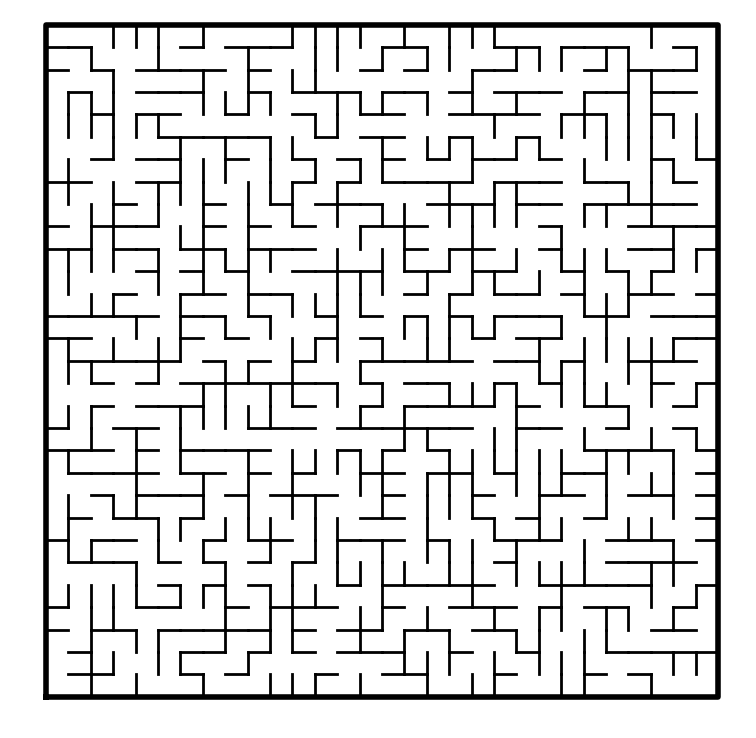
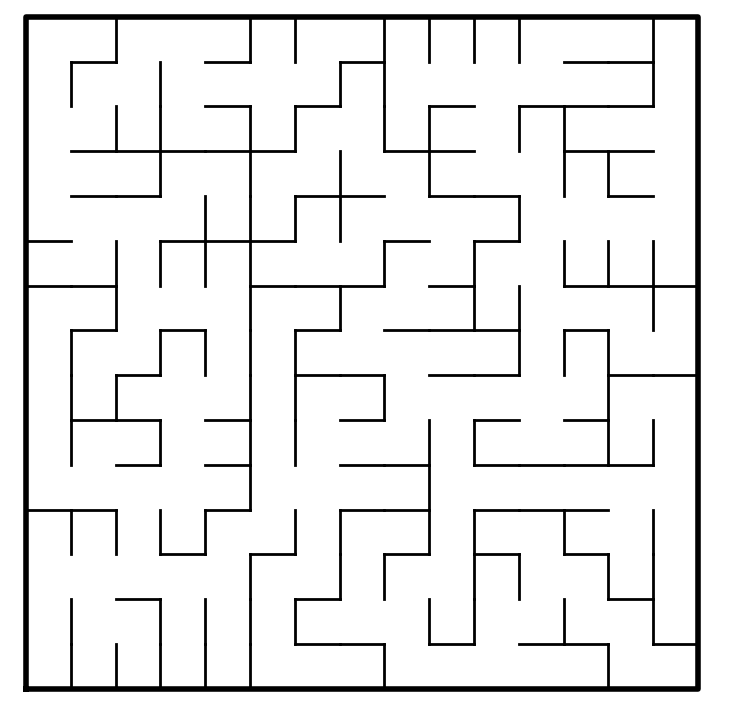
Solution

Given the maze and the display and code of a wrong maze, the only thing to do is to create a remove method that can pop (remove) a wall if root of w[0] is not the same as w[1] and then combine the wall after the deletion. For this we just create a method remove in which we have a while loop that has as base case while numSets is greater than one, numSets being the number of sets in the def, then we choose random wall to remove and store it on w, and then we get the position of this wall to be later used in the removal if w at index and w at index 1 don’t have the same root, if true then pop at index [i] and create a union between those two, we decrement numSets by 1 and return w. For the removal with path compression is the same but with union by size instead of using standard union.



Experimentation

Wall removal with path compression



Rows: 15

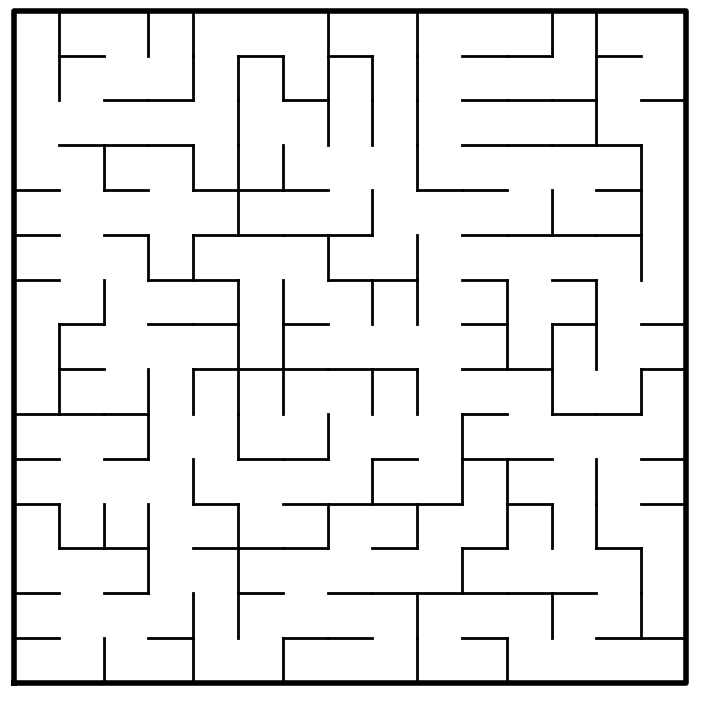
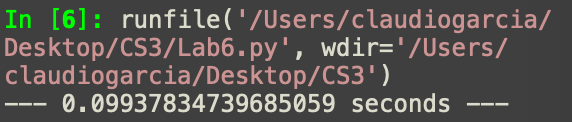
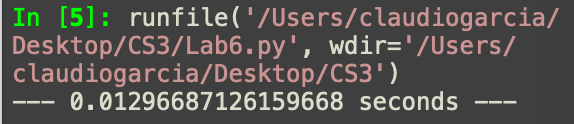
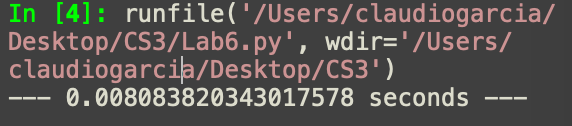
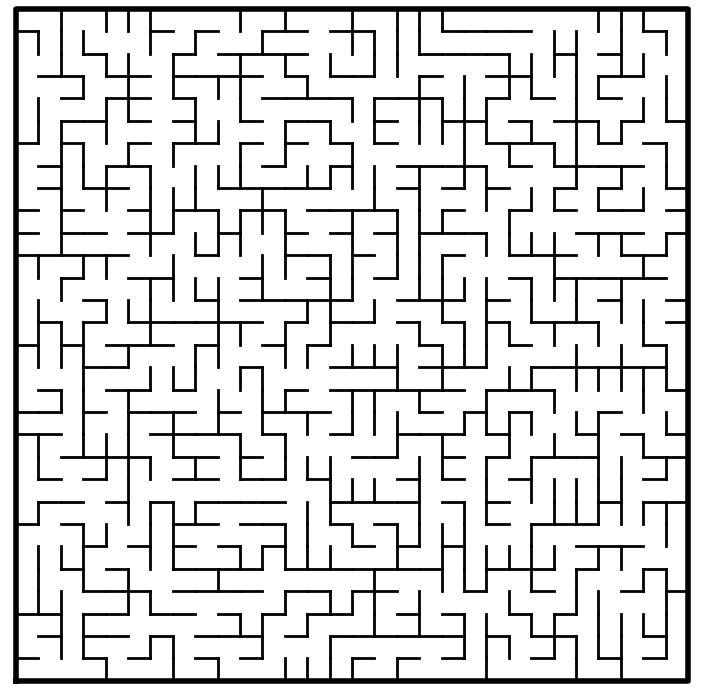
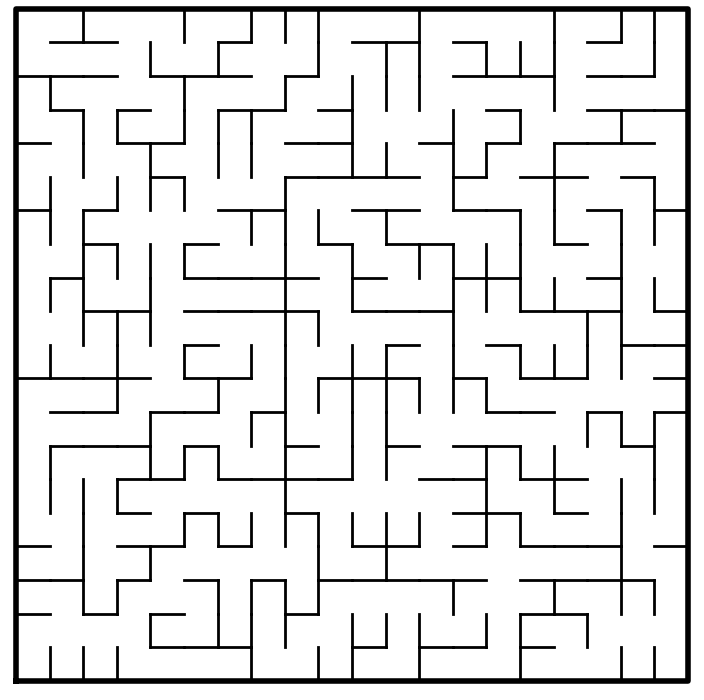
Columns: 15

Rows: 30

Columns: 30

Rows: 20

Columns: 20



Rows: 15

Columns: 15

Rows: 20

Columns: 20

Experimentation

Wall removal with standard union

Rows: 30

Columns: 30

Conclusion

With this lab I learned more about disjoint sets forests and how to implement them in a construction of a maze.

Appendix

import matplotlib.pyplot as plt

import numpy as np

import random

import time

def DisjointSetForest(size):

return np.zeros(size,dtype=np.int)-1

def find(S,i):

# Returns root of tree that i belongs to

if S[i]<0:

return i

return find(S,S[i])

def find\_c(S,i): #Find with path compression

if S[i]<0:

return i

r = find\_c(S,S[i])

S[i] = r

return r

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 # Make j's root point to i's root

def union\_by\_size(S,i,j):

# if i is a root, S[i] = -number of elements in tree (set)

# Makes root of smaller tree point to root of larger tree

# Uses path compression

ri = find\_c(S,i)

rj = find\_c(S,j)

if ri!=rj:

if S[ri]>S[rj]: # j's tree is larger

S[rj] += S[ri]

S[ri] = rj

else:

S[ri] += S[rj]

S[rj] = ri

def setAmount(s): #sets in teh disjoint set forests

c=0

for i in range(len(s)):

if s[i]<0:

c+=1

return c

def remove(s, walls, numSets): #remove random parts of the wall

while numSets>1:

w = random.choice(walls) #random wall selection

i = walls.index(w) #position of wall

if find(s, w[0]) != find(s, w[1]): #if root of w[0] is not the same as w[1]

walls.pop(i) #wall removal

union(s, w[0], w[1]) #wall union after the removal

numSets -=1

return w

def removeComp(s, walls, numSets): #remove random parts of the wall

while numSets>1:

w = random.choice(walls) #random wall selection

i = walls.index(w) #position of wall

if find(s, w[0]) != find(s, w[1]): ##if root of w[0] is not the same as w[1]

walls.pop(i) #wall removal

union\_by\_size(s, w[0], w[1]) #wall union after the removal

numSets -=1

return w

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

plt.close("all")

maze\_rows = 30

maze\_cols = 30

walls = wall\_list(maze\_rows,maze\_cols) #list of walls in the maze

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

S = DisjointSetForest(maze\_rows\*maze\_cols) #new disjoint sets forests

#combining rows and columns

numSets = setAmount(S) #sets in the dsf

start\_time = time.time()

remove(S, walls, numSets)

print("--- %s seconds ---" % (time.time() - start\_time))

#start\_time = time.time()

#removeComp(S, walls, numSets)

#print("--- %s seconds ---" % (time.time() - start\_time))

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

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.

* Claudio Garcia