Robert Marc

80487972

CS 2302

MW 1:30-3:00

Lab 5 Report

Introduction

The purpose of this lab was to take existing code and use disjoint set forests to create mazes.

Proposed Solutions

My solution to this lab was to create a disjoint set forest alongside the array that holds the wall pairs. As walls are removed, union is called on the cells with the wall between them removed. Once the entire forest is merged into a single set, the method ends and the maze is printed.

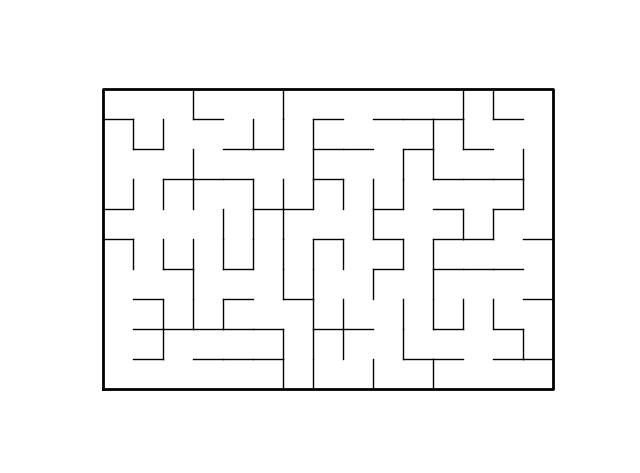
Experimental Results

For N rows and columns:

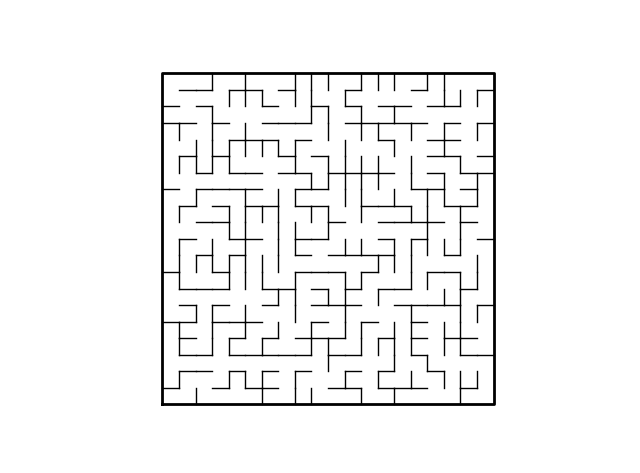
Average Number of Seconds for a given N across 10 tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N = | 10 | 20 | 30 | 40 |
| regMaze | 0.08 | 2.46 | 27.4 | 188.47 |
| compMaze | 0.05 | 0.87 | 3.47 | 22.27 |
| sizeMaze | 0.07 | 0.81 | 2.73 | 18.03 |

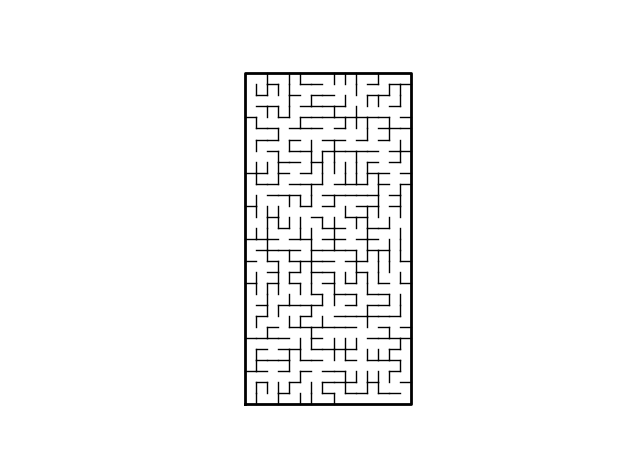
Outputs



10x15 maze built with regMaze (uses standard union, no path compression)



20x20 built with compMaze (uses union with path compression)



30x15 built with sizeMaze (uses union by size)

Conclusions

This lab was easy to figure out what needed to be done to produce correct mazes. At first I thought I did something wrong because it took me very little time to produce a solution. But when each method produced a correct maze for varying sizes, I realized that it was probably meant to be a simple solution.

Academic Honesty Statement

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.”



Appendix

# 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

"""

@Course: CS2302 MW 1:30-2:50 pm

@Author: Robert Marc, 80487972

@Assignment: Lab 6

@Instructor: Dr. Olac Fuentes

@TAs: Anindita Nath and Maliheh Zargaran

@Date of Last Modification: 4/14/19 @6:08PM

@Purpose: Use Disjoint set forests to create a maze

"""

import matplotlib.pyplot as plt

import numpy as np

import random

from scipy import interpolate

import time

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

def DisjointSetForest(size):

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

def dsfToSetList(S):

#Returns aa list containing the sets encoded in S

sets = [ [] for i in range(len(S)) ]

for i in range(len(S)):

sets[find(S,i)].append(i)

sets = [x for x in sets if x != []]

return sets

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

"""

Modified to return True if a union is made, or False if nothing is changed

"""

ri = find(S,i)

rj = find(S,j)

if ri!=rj:

S[rj] = ri

return True

return False

def union\_c(S,i,j):

# Joins i's tree and j's tree, if they are different

# Uses path compression

"""

Modified to return True if a union is made, or False if nothing is changed

"""

ri = find\_c(S,i)

rj = find\_c(S,j)

if ri!=rj:

S[rj] = ri

return True

return False

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

"""

Modified to return True if a union is made, or False if nothing is changed

"""

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

return True

else:

S[ri] += S[rj]

S[rj] = ri

return True

return False

def draw\_dsf(S):

scale = 30

fig, ax = plt.subplots()

for i in range(len(S)):

if S[i]<0: # i is a root

ax.plot([i\*scale,i\*scale],[0,scale],linewidth=1,color='k')

ax.plot([i\*scale-1,i\*scale,i\*scale+1],[scale-2,scale,scale-2],linewidth=1,color='k')

else:

x = np.linspace(i\*scale,S[i]\*scale)

x0 = np.linspace(i\*scale,S[i]\*scale,num=5)

diff = np.abs(S[i]-i)

if diff == 1: #i and S[i] are neighbors; draw straight line

y0 = [0,0,0,0,0]

else: #i and S[i] are not neighbors; draw arc

y0 = [0,-6\*diff,-8\*diff,-6\*diff,0]

f = interpolate.interp1d(x0, y0, kind='cubic')

y = f(x)

ax.plot(x,y,linewidth=1,color='k')

ax.plot([x0[2]+2\*np.sign(i-S[i]),x0[2],x0[2]+2\*np.sign(i-S[i])],[y0[2]-1,y0[2],y0[2]+1],linewidth=1,color='k')

ax.text(i\*scale,0, str(i), size=20,ha="center", va="center",

bbox=dict(facecolor='w',boxstyle="circle"))

ax.axis('off')

ax.set\_aspect(1.0)

def regMaze():

"""

Uses standard union algorithm to construct maze

Uses provided algorithm bod modified into a while loop that continues until

everything has been put into a single set.

If a union is made, pop the current wall pair.

"""

while len(dsfToSetList(maze)) > 1:

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

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

if union(maze,walls[d][0],walls[d][1]):

walls.pop(d)

def compMaze():

"""

Uses union algorithm with path compression to construct maze

Uses provided algorithm bod modified into a while loop that continues until

everything has been put into a single set.

If a union is made, pop the current wall pair.

"""

while len(dsfToSetList(maze)) > 1:

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

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

if union\_c(maze,walls[d][0],walls[d][1]):

walls.pop(d)

def sizeMaze():

"""

Uses union by size algorithm with path compression to construct maze

Uses provided algorithm bod modified into a while loop that continues until

everything has been put into a single set.

If a union is made, pop the current wall pair.

"""

while len(dsfToSetList(maze)) > 1:

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

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

if union\_by\_size(maze,walls[d][0],walls[d][1]):

walls.pop(d)

plt.close("all")

print("Enter number of rows in maze: ")

maze\_rows = int(input())

print("Enter number of columns in maze: ")

maze\_cols = int(input())

print("Choose DSF Implementation: ")

print("Type 1 for regular DSF, 2 for path compression, and 3 for Union by Size")

choice = input()

elapsed = 0

if choice== "1":

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

maze = DisjointSetForest(maze\_rows\*maze\_cols)

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

regMaze()

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

elif choice == "2":

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

maze = DisjointSetForest(maze\_rows\*maze\_cols)

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

compMaze()

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

elif choice == "3":

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

maze = DisjointSetForest(maze\_rows\*maze\_cols)

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

sizeMaze()

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

else:

print("Input not recognized, please try again.")