Gilbert Velasquez

80574186

CS2302

Lab 6 Report

Instructor: Olac Fuentes

TA’s: Anindita Nath and Maliheh Zargaran

In this Lab we were to design a program that would create a maze with only one path from each cell to another cell. We were to make use of Disjoint Set Forests in order to accomplish this task. This lab was very simple and an Algorithm of how to solve the problem was given to us by Dr. Fuentes. The only other task was to get the time comparisons for running the code with unions by size with compressions and then just regular union.

To Solve this task all we really needed to do was create a loop that’s condition was if we had more than one set in our DSF. Next, we would randomly select a wall in our wall list. If the walls we in the same set then we would do nothing, but if they were in different sets, we would call our union method to make them apart of the same set. To check if they were a part of the same list we just call the find method, which returns the root of an item in the DSF, for each wall. Next, we would just pop that pair of walls from our wall list. The wall list being a list of lists where a pair would be added to the list if they were next to each other.

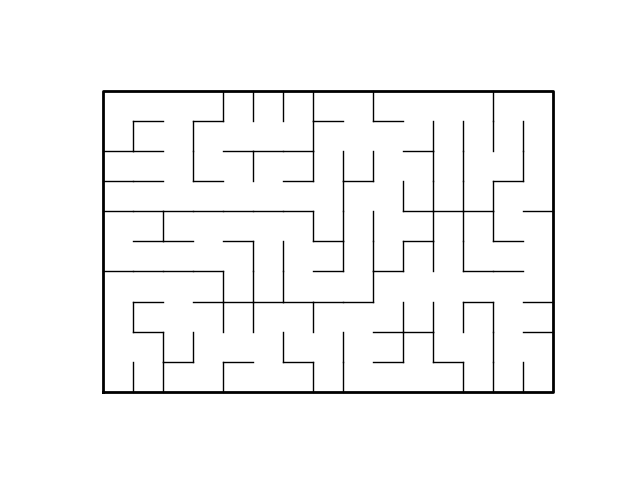
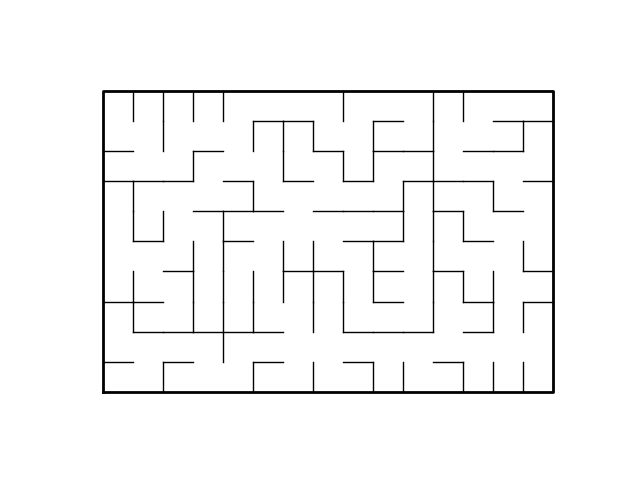
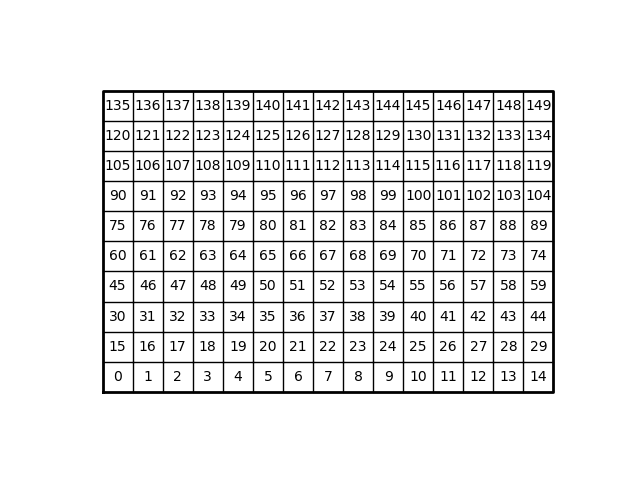
This lab was very straight forward! I enjoyed this lab because it really made me think and utilize all the methods given to us. This Lab taught me that creating algorithms for your code before you start coding is very useful. I was surprised at how simple the problem became when you have an outline for what you plan to do. I will now start my labs by creating an algorithm for them as opposed to just jumping right in to the code.

**Experimental Analysis:**

**1.**

Size of Maze: 150

Time for Regular Union: 0.024005413055419922 Seconds

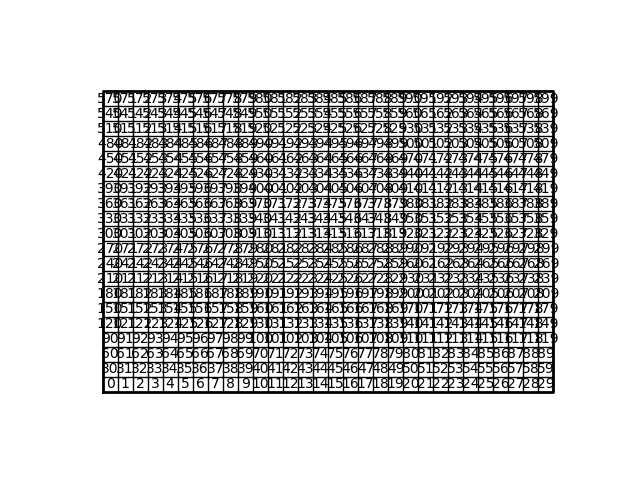
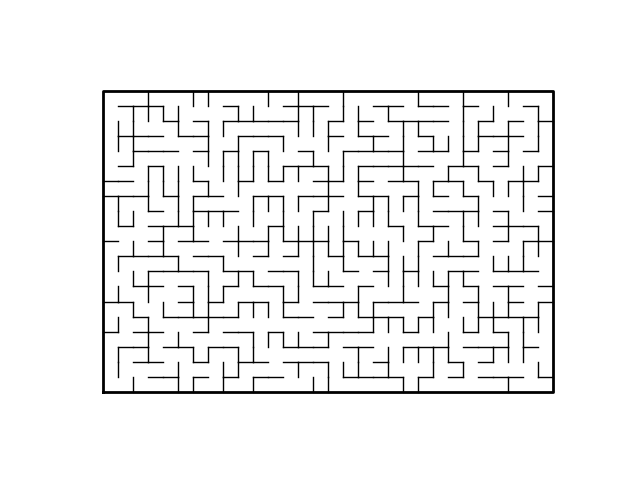
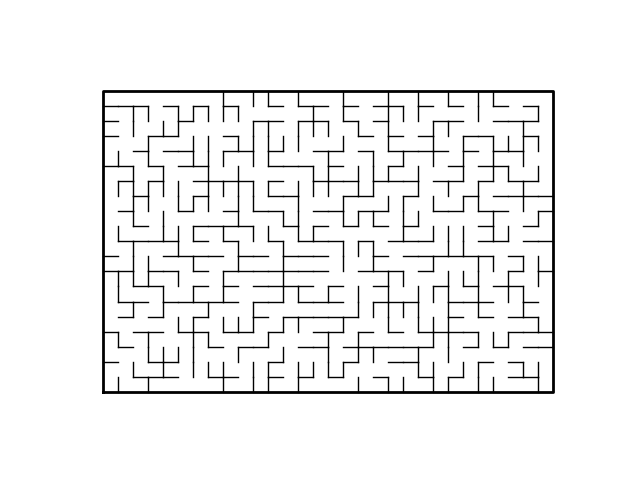
Time for Union by Size with Compression: 0.015003204345703125 Seconds 

**2.**

Size of Maze: 600

Time for Regular Union: 0.48010945320129395 Seconds

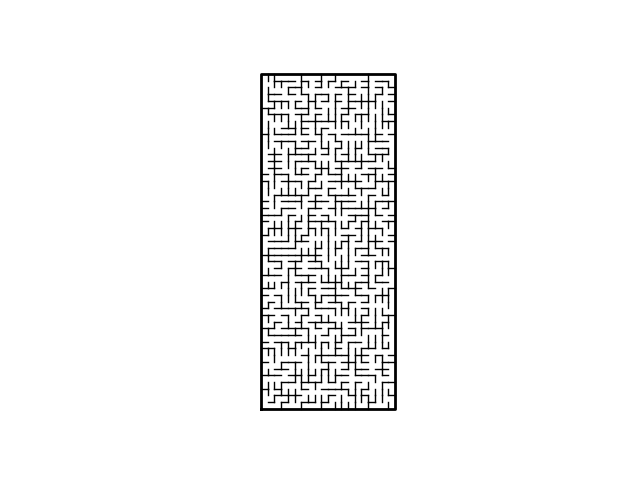
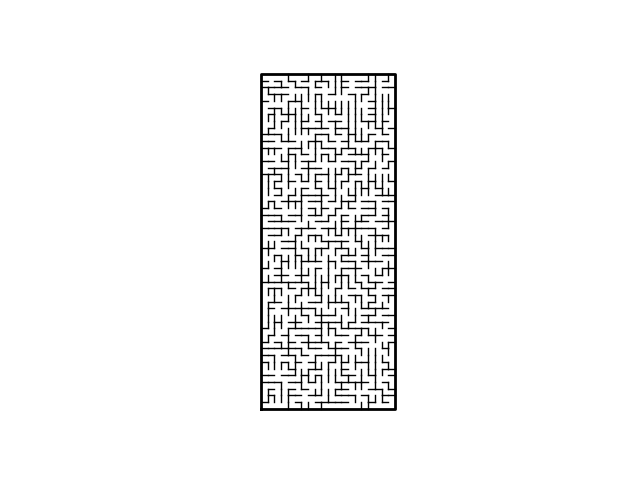
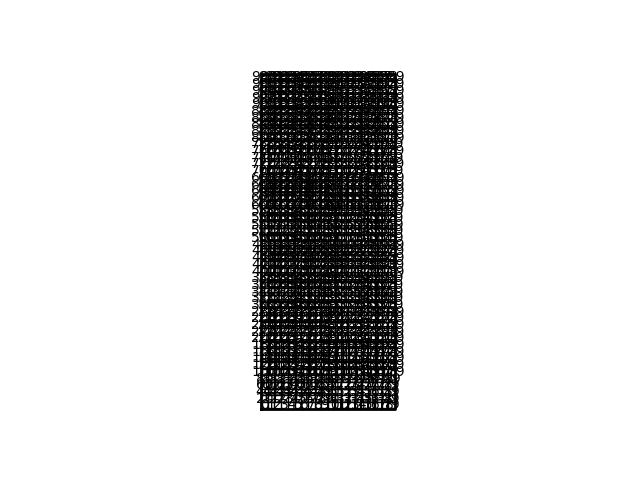
Time for Union by Size with Compression: 0.39708876609802246 Seconds



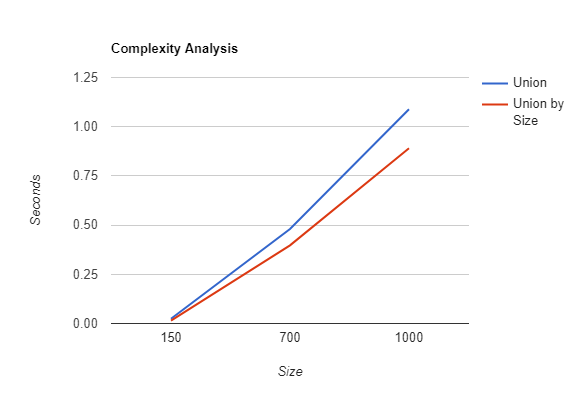
**3.**

Size of Maze: 1000

Time for Regular Union: 1.0892448425292969 Seconds

Time for Union by Size with Compression: 0.8912169933319092 Seconds

**Time Complexity:**



From the given Graph we can conclude that the Big O of these two is Linear therefore it is O(n). Unions by size does however do it faster. We can conclude that the randomness does play a factor in how fast the program runs seeing as it can be very lucky or unlucky with the walls it picks.

**Academic Honesty:**

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

Gilbert Velasquez

Gilbert Velasquez

**Source Code:**

#Gilbert Velasquez

# CS2302 MW 1:30-2:50

# lab 6

# Instructor Olac Fuentes

# TA Anindita Nath and Maliheh Zargaran

# Date of Last Modification 4/14/2019

#The purpose of this lab was to get a better understanding of Disjoint Set Forests. In this lab we were to use the DSF to create a maze that had exactly

#one path. This was achieved by using DSF and the idea that we can get a single path if we have a DSF with only one root. This lab taught me a lot! Espically not to overthink things!

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 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

ri = find(S,i)

rj = find(S,j)

if ri!=rj:

S[rj] = ri

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

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

# Uses path compression

ri = find\_c(S,i)

rj = find\_c(S,j)

if ri!=rj:

S[rj] = ri

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 NumSets(S): # Returns the number of sets in a DSF

count = 0

for i in range(len(S)):

if S[i]< 0:

count += 1

return count

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 = 50

maze\_cols = 20

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

DSF = DisjointSetForest(maze\_rows\*maze\_cols) # Creates DSF

walls2 = wall\_list(maze\_rows,maze\_cols) # second wall list for use with compressions

DSF2 = DisjointSetForest(maze\_rows\*maze\_cols)# Second DSF for compressions

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

print("Size of Maze: " , maze\_rows \* maze\_cols)

start = time.time() #Time for uninons without compressions

while NumSets(DSF)>1: #Check to see if we have more than one set

d = random.randint(0,len(walls)-1) #select a random wall

if find(DSF,walls[d][0]) != find(DSF,walls[d][1]): # if they belong to different sets

#print(find(DSF,walls[d][0]))

#print(find(DSF,walls[d][1]))

union(DSF,walls[d][0],walls[d][1]) #here we combine them without compressions

#print(S)

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

walls.pop(d) # get rid of that wall from our list

end = time.time()

print("Time for Regular Union:" , end - start, " Seconds")

start2 = time.time() # Time for size and compressions union

while NumSets(DSF2)>1: #Check to see if we have more than one set

d = random.randint(0,len(walls2)-1)

if find(DSF2,walls2[d][0]) != find(DSF2,walls2[d][1]): # if they belong to different sets

union\_by\_size(DSF2,walls2[d][0],walls2[d][1]) #here we combine them here with size and compression union

walls2.pop(d) # get rid of that wall from our list

end2 = time.time()

print("Time for Union by Size with Compression:", end2 - start2 , " Seconds")

draw\_maze(walls,maze\_rows,maze\_cols) #Draws maze without compressions

draw\_maze(walls2,maze\_rows,maze\_cols) #Draws maze with size and compression uninon