Yahir Rivas

Lab 6 Report

Instructor: Dr. Olac Fuentes

Teaching Assistant: Nath, Anindita

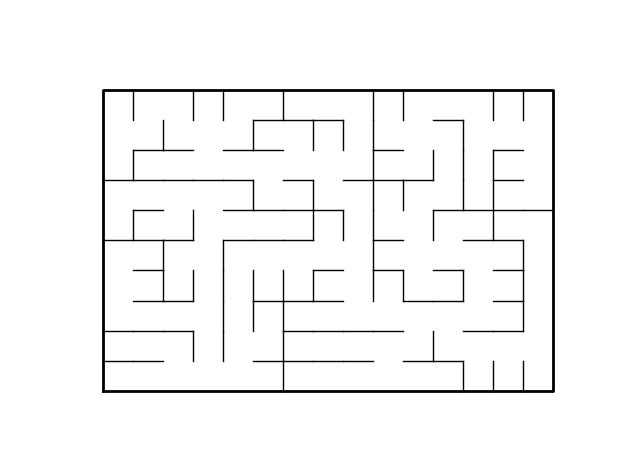
CS 2302

MW 1:30 - 2:50

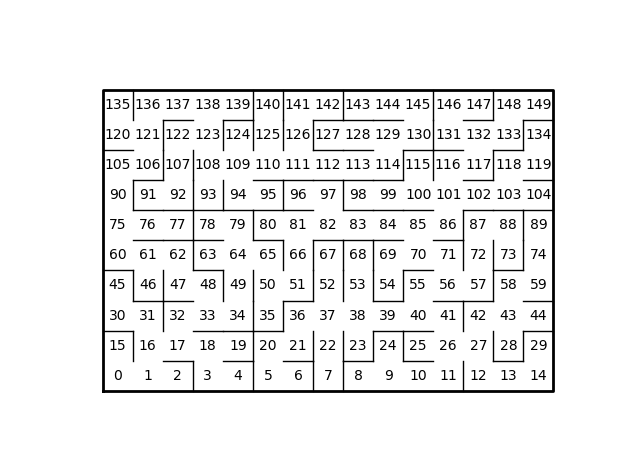
**Lab 6 Report**

The objective of this lab was to use a disjoint set forest to build a maze. The way this worked was by checking if two cells belong in the same set, if they don’t then I had to remove a wall to create a set between those two sets and repeat that process for every single cell until all cells belong in the same set, meaning that every cell can be accessed from any other cell in the maze. We implemented this with standard union and with compression, then comparing both methods.

The wat I implemented standard union was by creating a disjoint set forest, in that way we can stop uniting cells whenever there is only one root, meaning all cells are in the same set. We use random number cells to check if they are already in the same set, if they aren’t the wall must be removed so that they both are in the same set, therefore creating a path between them.



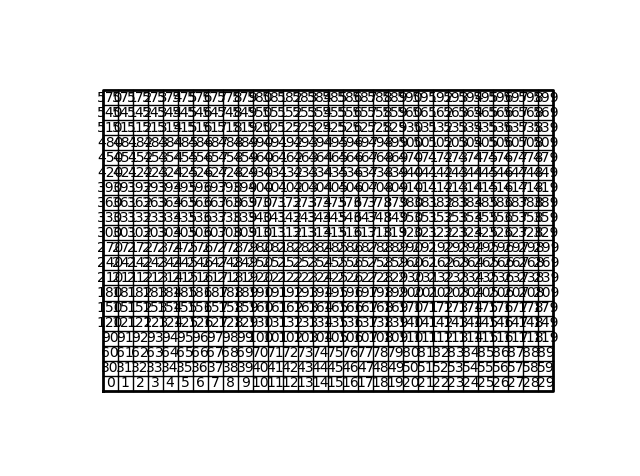
Maze after union has been made.

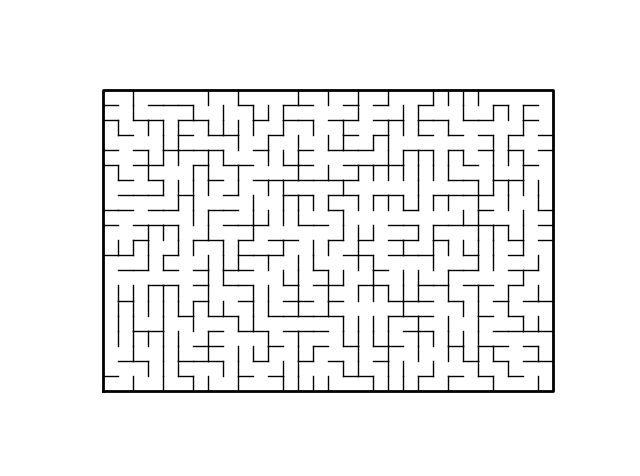


Maze with cell numbers and removed walls.

Running time for standard union with 10 rows and 15 columns: 0.01700139045715332

Running time for standard union with 10 rows and 15 columns: 0.013833761215209961



Running time for standard union with 20 rows and 30 columns: 0.25911974906921387

Running time for compression with 20 rows and 30 columns: 0.38153672218322754

import matplotlib.pyplot as plt

import numpy as np

import random

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 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

plt.close("all")

maze\_rows = 10

maze\_cols = 15

def DisjointSetForest(size):

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

def numSets(S): #return the number of sets

count = 0

for i in S:

if i < 0: #if it is -1 then it is a root so add 1

count += 1

return count

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

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

ri = find\_c(S,i)

rj = find\_c(S,j)

if ri!=rj:

S[rj] = ri

return True

return False

plt.close("all")

def countSets(S):

c = 0

for i in S:

if i==-1:

c+=1

return c

def unionSize(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: #if different root

if S[ri] > S[rj]: #if ri is bigger than rj then rj goes to ri

S[rj] += S[ri]

S[ri] = rj

return True

else:

S[ri] += S[rj] #if rj is bigger than ri then ri goes to rj

S[rj] = ri

return True

return False

plt.close("all")

maze\_rows = 20

maze\_cols = 30

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

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

S = DisjointSetForest(maze\_rows\*maze\_cols)#use a dsf to create maze

start = time.time()

while countSets(S) > 1:

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

if union(S,walls[d][0],walls[d][1]): #if they are in different sets

walls.pop(d) #remove wall

end = time.time()

rt = end - start

print("The running time for standard union is: ", rt )

'''

start = time.time()

while countSets(S) > 1:

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

if union\_c(S,walls[d][0],walls[d][1]):#if they are in different sets

walls.pop(d)#remove wall

end = time.time()

rt = end - start

print("The running time for compression is: ", rt )

'''

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

**Conclusion**

Using disjoint set forests is very useful when trying to unite sets. In this case it was used to create a path from any cell by only having one root in the disjoint set forest, meaning all cells were in one set.

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.