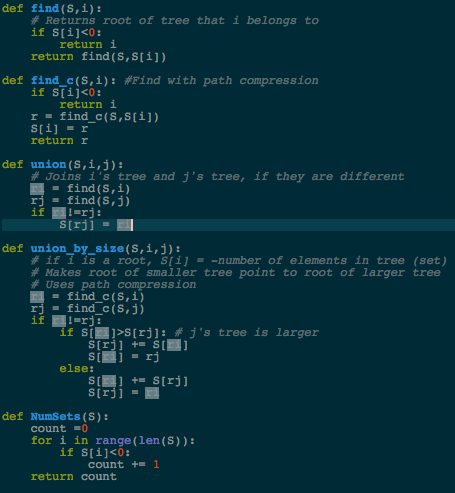
**Lab #6 Report: Disjoint Set Forests.**

**Introduction:**

﻿In this lab assignment we are making a maze with disjoint set forest. A random maze was given to us. when we remove a wall, if the cell that were separated by that wall belonged to different sets, you must unite these sets. The main goal for this lab assignment is to learn to use disjoint set forests as a data structure.

**Pre-methods:**

In the beginning we have the normal method to union two nodes and also another one to union by size whenever we need the set to be compressed. We have also a method *find()* to know the root of a certain node and also *find\_c()* to do the same thing whenever the node compressed. Finally, we have the method *NumSets()* to know the number of set of the forest.

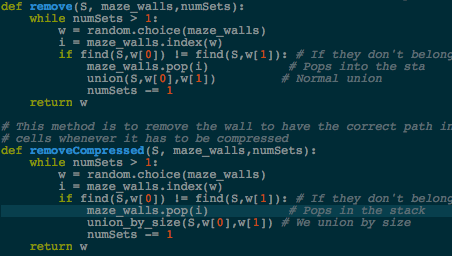
 NumSets() = O(n). Other methods run in constant time.

**Main methods:**

In the first place, we have a method named wall\_list() in charge of creating a list with all the walls in the list. Second we have a method draw\_maze() which is the one that draws the maze by using the matplot library.

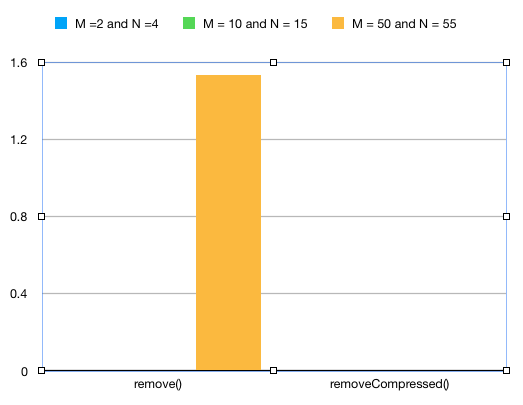
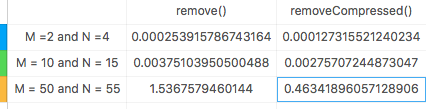
What I needed to do is to create a method to remove walls that way each node could have the right path to have a good maze. I did the method *remove()* which is in charge of removing walls from the maze. It randomly picks walls in the maze and checks if the walls nodes don’t belong to the same root. If this is true then it pops walls into the stack and union the first node to the second one. Finally it just returns the walls.

I did another one named *removeCompressed()* in which it will do the same but it will union by size.

 both run in O(n)

**Conclusion:**

From my point of view, I think this was one of the easiest labs since we had not to do much on it. It was a good assignment because it helped us to learn disjoint set forest as a data structure. Know I feel more comfortable when using disjoint set forests.



﻿

import matplotlib.pyplot as plt

import numpy as np

import random

from scipy import interpolate

import time

# ------------------- PRE-METHODS -------------------------

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:

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):

count =0

for i in range(len(S)):

if S[i]<0:

count += 1

return count

# ------------------- METHODS FOR THE LAB ----------------------

# This method is to remove the wall to have the correct path in the cells

def remove(S, maze\_walls,numSets):

while numSets > 1:

w = random.choice(maze\_walls)

i = maze\_walls.index(w)

if find(S,w[0]) != find(S,w[1]): # If they don't belong to the same root

maze\_walls.pop(i) # Pops into the sta

union(S,w[0],w[1]) # Normal union

numSets -= 1

return w

# This method is to remove the wall to have the correct path in the

# cells whenever it has to be compressed

def removeCompressed(S, maze\_walls,numSets):

while numSets > 1:

w = random.choice(maze\_walls)

i = maze\_walls.index(w)

if find(S,w[0]) != find(S,w[1]): # If they don't belong to the same root

maze\_walls.pop(i) # Pops in the stack

union\_by\_size(S,w[0],w[1]) # We union by size

numSets -= 1

return w

# This method the maze

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)

# This method creates a list with all the walls in the maze

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 = 55

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

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

# runs the remove() method

#time1 = time.time()

#remove(S,walls,NumSets(S))

#print("--- %s seconds ---" % (time.time() - time1))

# Runs the removeCompressed method

time1 = time.time()

removeCompressed(S,walls,NumSets(S))

print("--- %s seconds ---" % (time.time() - time1))

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

#for i in range(len(walls)//2): #Remove 1/2 of the walls

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

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

# walls.pop(d)

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.

* David A. Davis