**Introduction**

This assignment was given to help understand how sets can be applied to computer science. The main objective to complete consists of producing a correctly-drawn maze using find, union, and union by size with path compression. The goal of the assignment was to unite two sets until there is one set left. The way to accomplish this is by selecting two random numbers in the sets and removing their corresponding walls in the maze, if the first number is not in the same set as the second one, and if the second number is not in the same set as the first one.

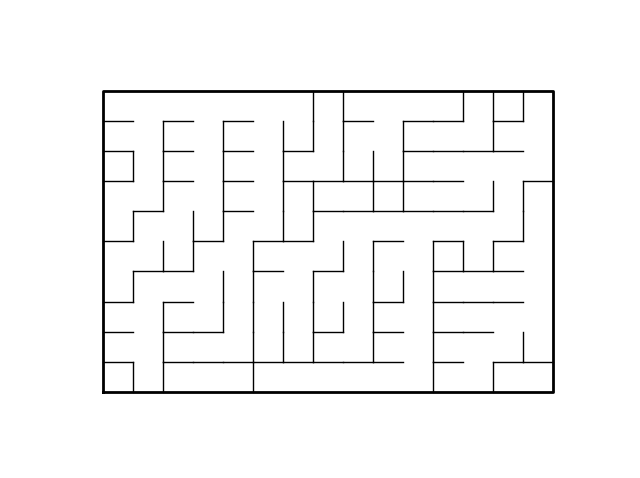
**Proposed Solution Design and Implementation**

Using standard union, the main objective should be to use a while loop that repeats until there is exactly one set left in the disjoint set forest.Each iteration of the loop should unite two items in the set if the items are not within the same set.

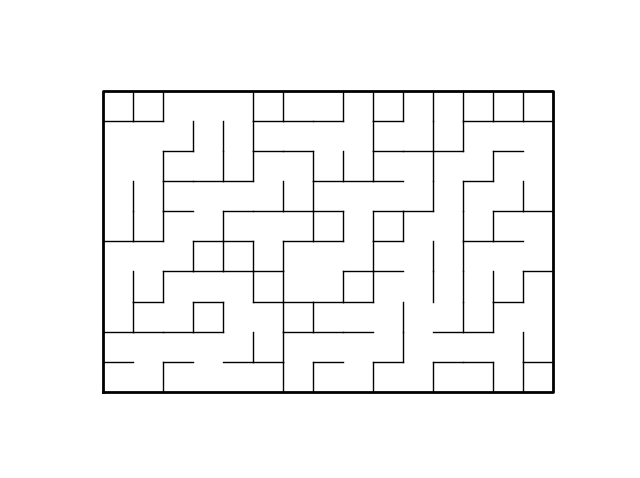
Using union by size with compression, the main objective should be to use a while loop that repeats until there is exactly one set left in the disjoint set forest. Each iteration of the loop should unite two items in the set if the items are not within the same set. This method proves to be the fastest as the union by compression ensures that each node in the set points directly to the root.

**Experimental Results**

Maze with standard find and union functions:



Maze with find with compression and union by size with compression functions:



**Conclusions**

Having completed this assignment somewhat poorly, many things can be learned from this lesson. Firstly, there should be more studying done with disjoint set forests. Secondly, the use of Python’s list manipulation still needs to be perfected. In conclusion, the use of Python’s lists and disjoint set forests can be improved indefinitely.

**Appendix**

"""

Course: CS 2302

Author: Jacob Montenegro

Lab: 6

Instructor: Dr. Olac Fuentes

T.A.: Anindita Nath, Maliheh Zargaran

Date of Last Modification: 4/12/2019

Program's Purpose: Use disjoint set forests to produce optimized maze of one set

"""

# Implementation of disjoint set forest

# Programmed by Olac Fuentes

# Last modified March 28, 2019

import matplotlib.pyplot as plt

import numpy as np

from scipy import interpolate

import random

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 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)

# 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

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

#checks if items in same set

def SameSet(S,i,j):

return find(S,i) == find(S,j)

#sameSet with path compression

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

return find\_c(S,i) == find\_c(S,j)

def numSets(S):

c = 0

for s in S:

if s < 0:

c += 1

return c

#returns the indices of roots

def getIndex(S):

index = 0

L = []

for s in S:

index += 1

if s < 0:

L.append(index)

return L

#Standard union and find

def Standard():

plt.close("all")

maze\_rows = 10

maze\_cols = 15

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

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

i = 0

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

# this is set to 25 because the program never reaches just 1 set, it stops around 20 and 30 total sets

# this ensures that the program does not enter an infinite loop

while numSets(S) > 25:

#random number chosen from the list

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

#ensures that the index is within that of the walls list

if d < len(walls):

#checks that the numbers do not belong to the same set

if SameSet(S,d,walls[d][0]) is False:

#unites them if they're not in the same set

union(S,d,walls[d][0])

#removes the wall

walls.pop(d)

#returns the indices of all the negative numbers

L = getIndex(S)

i = 0

while numSets(S) > 10 and len(L) > 15:

#ensures that the loop never enters an infinite loop

if i == 50000:

break

i += 1

#selects a random number

d = random.randint(0,max(L))

if d < len(walls):

#loops through the indexes that have not been united yet

for l in L:

if l < len(walls) and SameSet(S,walls[l][0],walls[d][0]) is False:

union(S,walls[l][0],walls[d][0])

walls.pop(d)

L.remove(l)

#shuffles the item

random.shuffle(L)

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

#Method for union by size and compression

def BySizeComp():

plt.close("all")

maze\_rows = 10

maze\_cols = 15

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

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

i = 0

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

#set to 30 to ensure that the loop doesn't enter an infinite loop

while numSets(S) > 30:

i += 1

#chooses random number

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

if d < len(walls):

if SameSet\_c(S,d,walls[d][0]) is False:

union\_by\_size(S,d,walls[d][0])

walls.pop(d)

#list of all the indices that may or may not have been combined in union with other sets

L = getIndex(S)

i = 0

while numSets(S) > 25 and len(L) > 10:

#selects a random number

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

if d < len(walls):

#checks that the items are not in the same set

if L[0] < len(walls) and SameSet\_c(S,L[0],walls[d][0]) is False:

union\_by\_size(S,L[0],walls[d][0])

walls.pop(d)

#ensures that the removed item is not repeated

L.remove(L[0])

#randomizes the list to ensure that a new item is chosen each iteration

random.shuffle(L)

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

Standard()

BySizeComp()

**I certify that this project is entirely my own work. I wrote, debugged, and tested the code 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. Signed, Jacob M. Montenegro.**