Introduction:

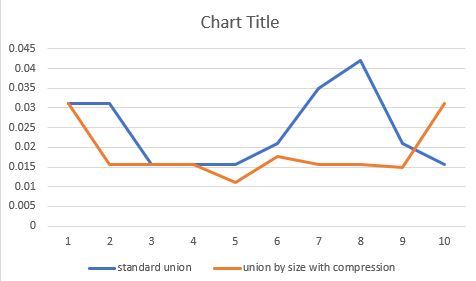
The problem that was to be solved was to make a maze out of disjoint set forests.

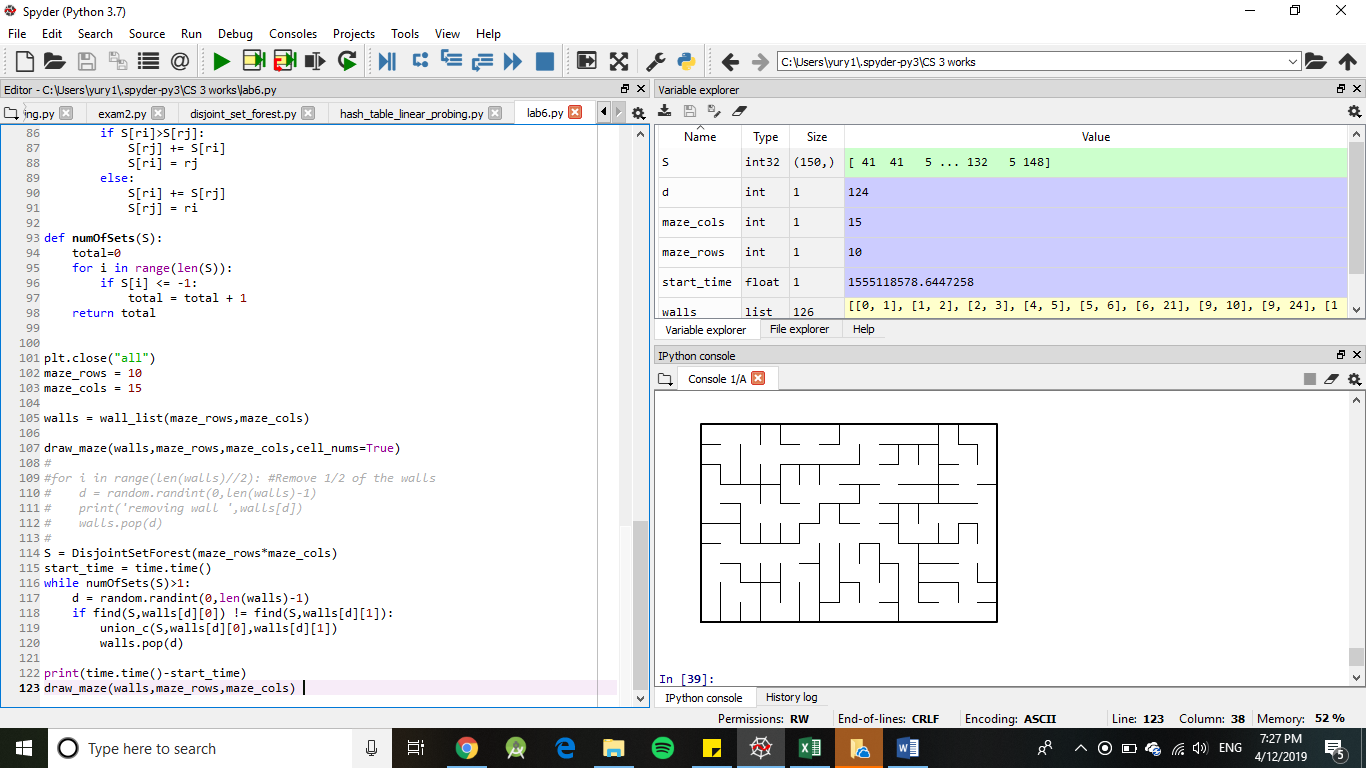
Solution:

I first attempted at the problem by writing pseudocode and trying to make sense of it. I then implemented it into code and ran several times. Each time I ran it I got close to solving the problem. I finally followed it up with actually coming up with the solution.

Results:

The experiments included running the program several times, and finding out the outputs. The running times.





Conclusions:

I learned that disjoint set forests are very useful.

I Yury Ionov 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.



Source code:

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 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: # Do nothing if i and j belong to the same set

S[rj] = ri # Make j's root point to i's root

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):

ri = find\_c(S,i)

rj = find\_c(S,j)

if ri!=rj:

if S[ri]>S[rj]:

S[rj] += S[ri]

S[ri] = rj

else:

S[ri] += S[rj]

S[rj] = ri

def numOfSets(S):

total=0

for i in range(len(S)):

if S[i] <= -1:

total = total + 1

return total

plt.close("all")

maze\_rows = 10

maze\_cols = 15

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

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)

#

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

start\_time = time.time()

while numOfSets(S)>1:

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

if find(S,walls[d][0]) != find(S,walls[d][1]):

union\_c(S,walls[d][0],walls[d][1])

walls.pop(d)

print(time.time()-start\_time)

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