David Amparan

Lab Report 6

April, 14th 2019

Instructor: Dr. Funtes, Olac

T.A: Anindita Nath, Maliheh Zargaran

**Introduction**

A Disjoint Set Forest is an ADT which groups two of the indexes in its array representation are joined onto one set. A simple visual of a DSF is a maze, initially there is no path in the maze and is filled with squares (each index in the DSF). But once we begin joining indexes, the walls disappear and thus show a path from one index to another. With this idea in mind, for this lab that is exaclt what we do. By kindly giving us the beginning of the lab (the function to draw the maze) we draw the maze as long as there is more than 1 set in our DSF.

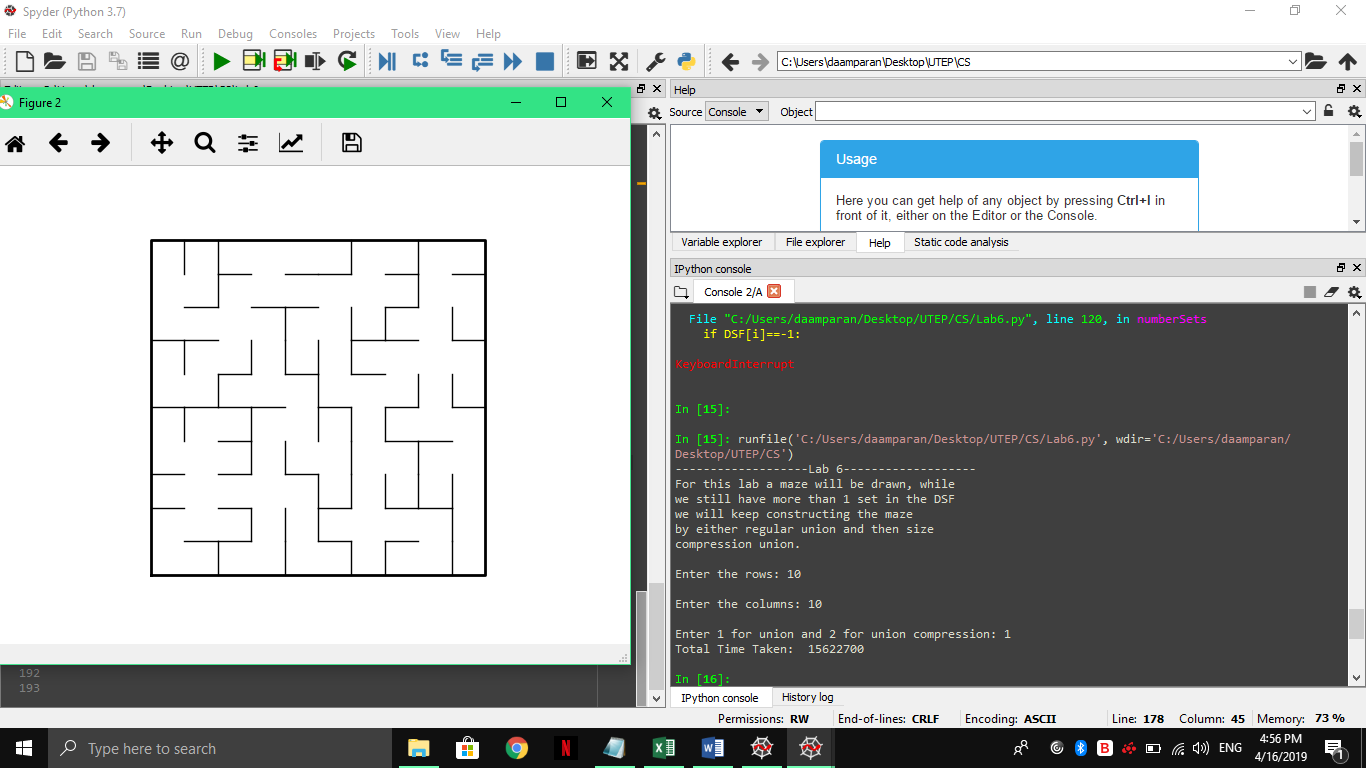
**Proposed Solution**

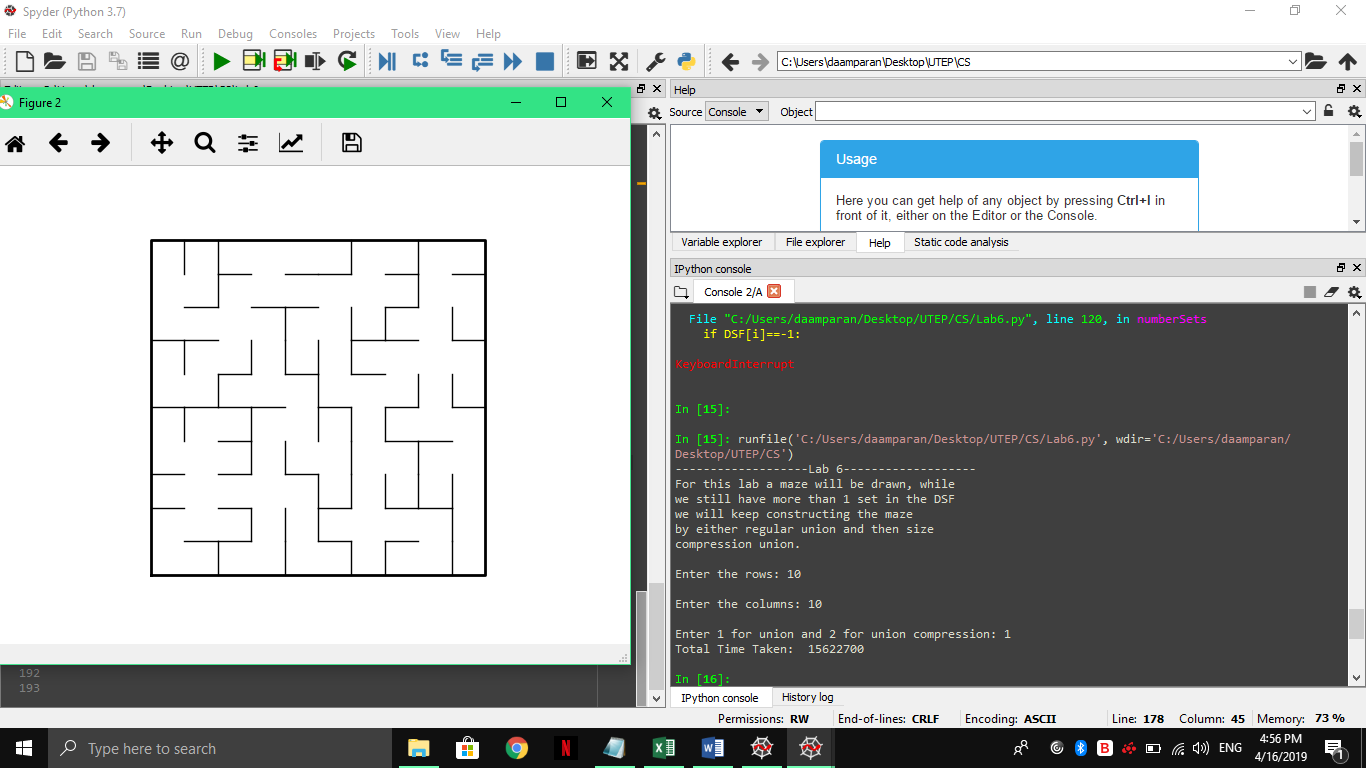
To be able to create a maze as long as we have more than 1 set means we must create a method to count the number of sets. We must also ensure that the picked walls are not part of separate set, therefore we must check this condition and if it is true we will do nothing, if false then we continue and draw the maze.

Count\_set: to count the number of sets within the DSF we simple go through the entire array and find which indexes contain -1, since each set contains a root and the root pointing to -1.

checkSameSet: to check if an item is within the same set we us the handy Find method within the DSF code, if the item returned is the same then they must belong in the same set if not then we return false.

**Experimental Results:**





**Conclusion:**

Lab 6 exemplified the functionality of a DSF data type as well as showing the distinct difference between the regular union as well as union by compression. Not only was the algorithm functionality shown but also the time advantage when done with compression. By building a maze as well we can see the mimicking behavior of how sets eventually can build and entire path or set, which can be useful when deciding what string of tasks to tackle on at once.

**Appendix**

I, David Amparan, certify this assignment was completed by myself with no help nor was copied from another classmate/source. I will take full responsibility if such items are found.

"""

Author: David Amparan

Last Modified on:Apr 13, 2019

Instructor: Fuentes, Olac

T.A: Anindita Nath, Maliheh Zargaran

Purpose: Lab 6 serves the purpose of putting disjoint set forests

into practice by helping us create a maze. This maze will be formed when each

item within itis part of a set

"""

# 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

import matplotlib.pyplot as plt

import numpy as np

import random

from scipy import interpolate

import time as tiempo

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 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

"""

Method Name: numberSets | Parameters: DSF

Functionality: Will return the number of sets within the DSF

"""

def numberSets(DSF):

#we will traverse entire list to find the -1

numSets=0

for i in range(len(DSF)):

if DSF[i]==-1:

numSets+=1

return numSets

"""

Method Name:checkSameSet | Parameters:DSF, walls

Functionality: Checking the given walls within the DSF to

determine if they are within the same set; if they are not

return true

"""

def checkSameSet(DSF,walls):

same=False

if find(DSF,walls[1]) == find(DSF,walls[0]):

same=True

return same

print("-------------------Lab 6-------------------")

print("For this lab a maze will be drawn, while")

print("we still have more than 1 set in the DSF")

print("we will keep constructing the maze")

print("by either regular union and then size")

print("compression union.")

m = int(input("Enter the rows: "))

n = int(input("Enter the columns: "))

userChoice=int(input("Enter 1 for union and 2 for union compression: "))

#create the dsf

DSF = DisjointSetForest(m\*n)

plt.close("all")

walls = wall\_list(m,n)

"""

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,m,n)

"""

if userChoice==1:

#application for regular union

draw\_maze(walls,m,n,cell\_nums=True)

start=tiempo.time\_ns()

while numberSets(DSF)>1:

#here we attain the random walls

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

if checkSameSet(DSF,walls[d])==False:

union(DSF, walls[d][0], walls[d][1])

walls.pop(d)

end=tiempo.time\_ns()

draw\_maze(walls,m,n)

print("Total Time Taken: ",end-start)

elif userChoice==2:

#application for compression size

draw\_maze(walls,m,n,cell\_nums=True)

start=tiempo.time\_ns()

while numberSets(DSF)>1:

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

if checkSameSet(DSF,walls[d])==False:

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

walls.pop(d)

end=tiempo.time\_ns()

draw\_maze(walls,m,n)

print("Total Time Taken:", end-start)