David Amparan

Instructor: Dr. Fuentes, Olac

TA: Anindita Nath, Maliheh Zargaran

Lab 7 Report

**Introduction**

After computing a maze for Lab 6 using a disjoint set forest and learning about graphs in lecture we combined both concepts and developed the problem for Lab 7. For Lab 7, we reuse the code we computed within Lab 6 but instead of building the maze up until there is only one root; we instead ask the user how many walls they would want removed. Depending on the selection we then display a corresponding message. After displaying this message as well as building the maze, we must apply three separate algorithms to find a solution for the maze. We then must create a method that turns the DSF into a adjacency list. Breath first search, depth first search, and recursive depth first search. After conducting these algorithms we then show the solution for the maze.

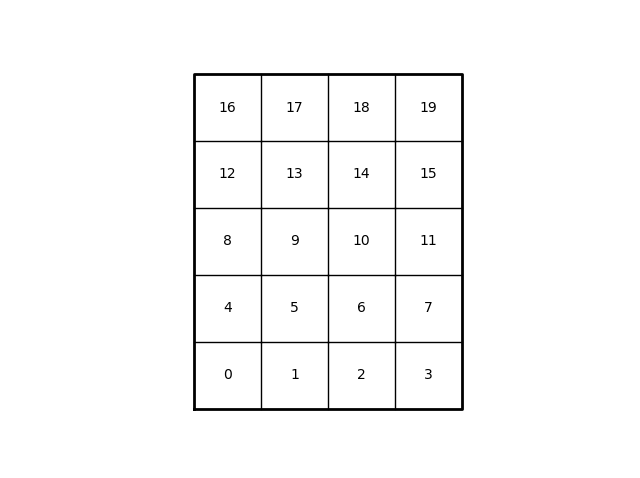
**Propose Solution**

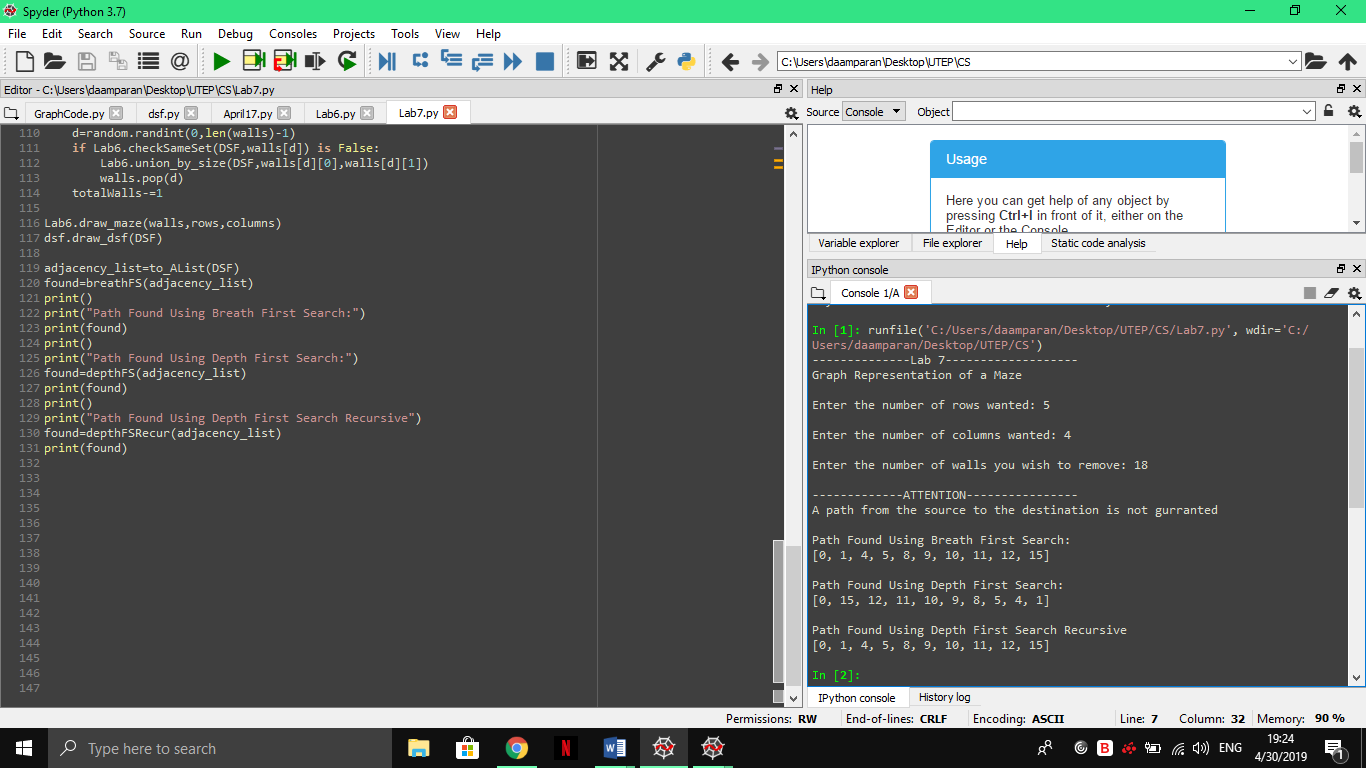
**DSF to Adjacency List**: For this method we must traverse the DSF twice (nested loop); what we must do then is check for same set and if they are then they are connected. If they are then connected we create the list for that vertex

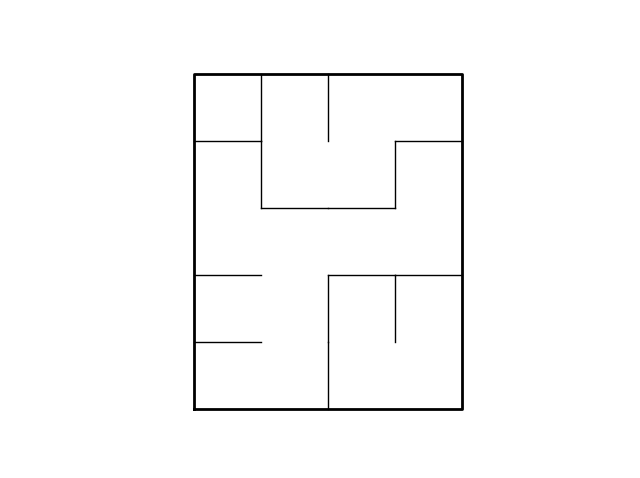
**Breadth First Search**: This type of search involves visiting those cells that are within one as a distance from the starting edge. Then we reach the last edge which is the vertex that has no in-vertexes.

**Depth First Search (Stack)**: By using a stack we visit each cell and its “neighbors” first before the actual cell you begin with. This is then done by for loop then you push into the stack and pop those that aren’t within the visited set.

**Depth First Search (Recursive)**: By then visiting the starting vertex, we then visit these within the starting vertex and make a recursive call in each then return a set if it is in the same set of a DSF

**Drawing Path**: Drawing the path to the goal of the maze would involve using the return value from each of these search methods we draw this line. This is as far as I got and could not complete the goal of the maze.



****

**Experimental Results**

**Conclusion**: For lab 7 what I learned was how to implement the graph search methods as well as how to convert a DSF into a graph with a simple method. Now with this given we could solve a maze since the search algorithms returns a list of the connected cells and thus forming a path from the starting cell to the finishing cell. Yet, like I mentioned earlier I could not solve this part of the lab, once I would try to add the lines they would mess with the original maze lines so on and so forth.

**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.

**Source Code**:

"""

Author: David Amparan

Instructor: Dr. Fuentes, Olac

TA:

Purpose: For lab #7 we will use what we did on lab #6 and transfer that DSF representation

into a graph representation of the maze. When this is done we will find

a Minimum spanning tree with 3 seperate algorithms

"""

import GraphCode

import dsf

import time as tiempo

import Lab6

import random

import matplotlib.pyplot as plt

"""

Method Name: toAL| Parameters: Maze | Return: AL

Functionality: As we take in the maze as a DSF we create a

AL from the

"""

def to\_AList(forest):

toReturn=[]

for i in range(len(forest)):#we check each index with the others to find connections

add=[]

for l in range(len(forest)):#if they belong in the same set then we add them to the list for that vertex

if i == l:

add=[]

elif dsf.find(forest,i) == dsf.find(forest,l):

add.append(l)

toReturn.append(add)

return toReturn

"""

Method Name: breathFS | Parameters: AL |

Functionality: Will determine if there is a path from 0 to walls-1

"Which is the end", and it will return a list that contains the path

"""

def breathFS(AL):

queue=[]

previous=[] #we will implement the queue as our holding then names speak for themselvs

queue.append(0)#we can hardcode zero since this is the beginning

previous.append(0)

#here we begin traversing our graph

while queue != []:

visited=queue.pop(0)

for i in AL[visited]:

if not previous.\_\_contains\_\_(i):

queue.append(i)

previous.append(i)

return previous

"""

Method Name: depthFS | Parameters: AL

Functionality: Depth first search will perform like i said, a depth first search

and return an array with the path found

"""

def depthFS(AL):

stack=[]

previous=[]

#we want to start at 0 so we hard code it

stack.append(0)

while stack != []:#while our stack is not empty

current=stack.pop()

if not previous.\_\_contains\_\_(current):

previous.append(current)

for i in AL[current]:

stack.append(i)

return previous

"""

Method Name: depthFS | Parameters: AL

Functionality: Depth first search will perform like i said, a depth first search

and return an array with the path found, this is a recursive method however

"""

def depthFSRecur(AL,current=0,previous=[]):

if not previous.\_\_contains\_\_(current):

previous.append(current)

for i in AL[current]:

depthFSRecur(AL,i,previous)

return previous

print("--------------Lab 7-------------------")

print("Graph Representation of a Maze")

rows=int(input("Enter the number of rows wanted: "))

columns=int(input("Enter the number of columns wanted: "))

toRemove=int(input("Enter the number of walls you wish to remove: "))

print()

#now we display the message

totalCells=rows\*columns

print("-------------ATTENTION----------------")

if((totalCells-1)>toRemove):

print("A path from the source to the destination is not gurranted")

elif((totalCells-1)<toRemove):

print("There is at least one path from source to destination")

else:

print("There is a unique path from source to destination")

DSF=dsf.DisjointSetForest(rows\*columns)

totalWalls=toRemove

walls=Lab6.wall\_list(rows,columns)

#we begin creating the graph

plt.close("all")

Lab6.draw\_maze(walls,rows,columns,cell\_nums=True)

#part from lab 6

while totalWalls!=0:

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

if Lab6.checkSameSet(DSF,walls[d]) is False:

Lab6.union\_by\_size(DSF,walls[d][0],walls[d][1])

walls.pop(d)

totalWalls-=1

Lab6.draw\_maze(walls,rows,columns)

dsf.draw\_dsf(DSF)

adjacency\_list=to\_AList(DSF)

found=breathFS(adjacency\_list)

print()

print("Path Found Using Breath First Search:")

print(found)

print()

print("Path Found Using Depth First Search:")

found=depthFS(adjacency\_list)

print(found)

print()

print("Path Found Using Depth First Search Recursive")

found=depthFSRecur(adjacency\_list)

print(found)