Lab 6 Adrian Monreal

This lab was given to us by Dr.Fuentes with the intent to teach us and test our knowledge of disjoint set forest by creating a maze. We were required to create a maze given spreadsheet of cells we were to create one path so that all the cells are connected someway and contain only one root, how were we going to create the maze was by removing walls that connected each cell. Dr. Fuentes gave us the code to display the maze and create the maze list so all we had to create was the disjoint set forest and link all the nodes. My brainstorming for this function was as presented below, While S has more than one set choose a random wall from the list of walls that the wall list function gives you. Check to see if they belong to the same set, if they do not union the 2 sets, then pop that wall from the wall list, and if they do belong to the same set do not do anything.

*"""*

*Adrian Monreal*

*Lab 6*

*CS2302*

*"""*

import dsf

import matplotlib.pyplot as plt

import numpy as np

import random

"""

# Starting point for program to build and draw a maze

# Modify program using disjoint set forest to ensure there is exactly one

# simple path joining any two cells

# Programmed by Olac Fuentes

"""

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)

#M #n

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

"""

While S has more than one set

choose a random wall from the list of walls that

the wall list function gives you

check to see if they belong to the same set

if they do not union the 2 sets

if they do belong to the same set dont do anything

then pop that wall from the wall list

"""

def dsfMaze(rows,columns,wallList):

cells = rows \* columns

s = dsf.DisjointSetForest(cells)

while dsf.NumSets(s)> 1:

curr = random.randint(0,len(wallList)-1)

wall = wallList[curr]

if dsf.find(s,wall[0]) != dsf.find(s,wall[1]):

dsf.union(s, wall[0], wall[1])

wallList.pop(curr)

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)

dsfMaze(maze\_rows,maze\_cols,walls)

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

“I Adrian Monreal 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.”