Lab 6 Report

Introduction:

In this lab we are tasked to make a maze with only one path using a DSF and an edge list.

Proposed Solution:

You have a maze with walls you create a dsf with the number of walls and the number of sets. Then you remove a random wall only if it is not part of the same set and if the number of sets is more than 1. This takes about O(|E|) because you need to traverse the edge-list. The dsf would be O(n). We also need to use normal union and union by size.

Conclusion:

The Disjoint set forest makes it easier to identify what walls to delete.

Appendix:

import matplotlib.pyplot as plt

import numpy as np

import random

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 union(S,i,j):

ri = find(S,i)

rj = find(S,j)

if ri!=rj:

S[rj] = ri

def find\_c(S,i):

if S[i]<0:

return i

else:

r = find\_c(S,S[i])

S[i] = r

return r

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[rj] = rj

else:

S[ri] += S[rj]

S[rj] = ri

def standard\_dsf(M,N):

S = DisjointSetForest(M\*N)

for i in range(len(S)-1):

union(S,i,i+1)

return S

def compressed\_dsf(M,N):

S = DisjointSetForest(M\*N)

for i in range(len(S)-1):

union\_by\_size(S,i,i+1)

return S

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

plt.close("all")

graph = input("Do you want to remove walls randomly, sequentially or by compression?")

maze\_rows = 10

maze\_cols = 15

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

print(walls)

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

if graph == 'random':

for i in range(len(walls)//2): #Remove random walls

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

print('removing wall ',walls[d])

walls.pop(d)

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

if graph == 'sequentially':

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

for i in range(len(walls)-1):

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

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

Academic Honesty:

Academic Agreement:

I 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.

Andres Arellanes