Joaquin Hidalgo

CS 2302 MW @1:30pm

Lab 06

**Disjoint forest set(s)**

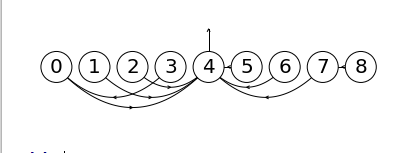
**Introduction:** The problem at hand was to create a disjoint forest set where each cell was its own root thus making (rows\*columns) cells. When determining if each cell is already in its own set, you compare roots. When constructing this disjoint forest set, we either standardly find and union the cells. Or we can find by using compression and union by rank to make more a more effect search.

**Proposed solution design and implementation:** First create a method to count the amount of sets there is in the Disjoint set forest. Here it has a method that runs O(n) time by counting how many -1 there are in the set. Keep removing walls of the maze until there is only one set of numbers. When you pick a random wall, you are given two numbers, one for each cell. Extract both numbers and compare the roots, in standard find, we just look for the root of the cell and find w/ compression is finding the root of the cell with updating each cell to its new root to be compressed. Then after compressing while finding, you determine which of the two cells has a larger set size and add it to the smaller one. Ultimately have the user choose if they want to use standard union and find vs find with compression and union by size.

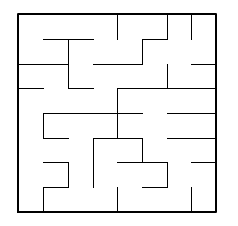
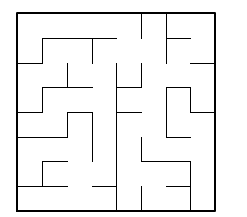
**Experimental results:**

****

Standard find and union with a (4x4) maze

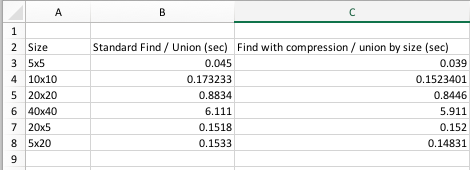
****

Find using compression and union by size with a (3x3) maze

 ****

Maze each cell represent a disjoint forest set and only contains 1 set.

**Running times**



**Conclusion:** I learned how to make a maze where each wall is in a list consisting of 2 numbers (cells) that are in separate cells at first and if so, combine them until there is only 1 set of cells that are connected to each other. By using path compression and union by size, we are able to save time by a few seconds compared to standard union / find.

**Code:**

﻿"""

By: Joaquin Hidalgo

Lab 06

Created on Wed Apr 10 20:26:23 2019

Instructor: Olac Fuentes

TA: Dita and Mali

Last modified Sun 14 Apr 23:48:28

Create a maze where each cell starts off in its own set, randomly

picking walls of the maze to delete, once you delete a wall, you join

the two cells sets and keep on doing until there is only 1 set which is 1 path.

Use standard Union vs Union by size with compression

"""

import matplotlib.pyplot as plt

import numpy as np

import random

from scipy import interpolate

import time

#Create and set disjoint forest by size

def DisjointSetForest(size):

return np.zeros(size,dtype=np.int)-1

#Standard find by root

def FindRoot(S,i):

# Returns root of tree that i belongs to

if S[i]<0:

return i

return FindRoot(S,S[i])

def FindRoot\_c(S,i): #Find with path compression

if S[i]<0:

return i

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

S[i] = r

return r

#Returns true or false if two nums are in set

def CheckSameSet(S,i,j):

# Joins i's tree and j's tree, if they are different

root1 = FindRoot(S,i)

root2 = FindRoot(S,j)

if root1!=root2: # if r1 != r2, then they are different set

return False, root1, root2

return True, root1, root2

def CheckSameSet\_c(S,i,j):

root1 = FindRoot\_c(S,i)

root2 = FindRoot\_c(S,j)

if root1!=root2: # if r1 != r2, then they are different set

return False, root1, root2

return True, root1, root2

#Iterates thru disjoint forest and counts amount of sets

def NumSets(S):

c =0

for i in range(len(S)):

if S[i] <0:

c+=1

return c

# Creates a list with all the walls in the maze

def wall\_list(maze\_rows, maze\_cols):

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 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 draw\_dsf(S):

scale = 30

fig, ax = plt.subplots()

for i in range(len(S)):

if S[i]<0: # i is a root

ax.plot([i\*scale,i\*scale],[0,scale],linewidth=1,color='k')

ax.plot([i\*scale-1,i\*scale,i\*scale+1],[scale-2,scale,scale-2],linewidth=1,color='k')

else:

x = np.linspace(i\*scale,S[i]\*scale)

x0 = np.linspace(i\*scale,S[i]\*scale,num=5)

diff = np.abs(S[i]-i)

if diff == 1: #i and S[i] are neighbors; draw straight line

y0 = [0,0,0,0,0]

else: #i and S[i] are not neighbors; draw arc

y0 = [0,-6\*diff,-8\*diff,-6\*diff,0]

f = interpolate.interp1d(x0, y0, kind='cubic')

y = f(x)

ax.plot(x,y,linewidth=1,color='k')

ax.plot([x0[2]+2\*np.sign(i-S[i]),x0[2],x0[2]+2\*np.sign(i-S[i])],[y0[2]-1,y0[2],y0[2]+1],linewidth=1,color='k')

ax.text(i\*scale,0, str(i), size=20,ha="center", va="center",

bbox=dict(facecolor='w',boxstyle="circle"))

ax.axis('off')

ax.set\_aspect(1.0)

# Main

##################################################################

if \_\_name\_\_ == '\_\_main\_\_':

plt.close("all")

m\_rows= int(input('enter number of rows: '))

m\_cols = int(input('enter number of columns: '))

userChoice = int(input('1. standard union 2. union by size with path compression '))

startTime = time.time()

########################## choice 1

if userChoice == 1:

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

#Creates 'wall\_list' a set of two cells who have a wall between them

wall\_list = wall\_list(m\_rows,m\_cols)

while True:

d = random.randint(0,len(wall\_list)-1) # pick random wall

a,b = wall\_list[d] #obtain the num of the two cells

flag, r1, r2 = CheckSameSet(S,a,b) # checks if same root, standard

if flag == False:

#Union two cell nums

S[r2] = r1

#Pop wall

wall\_list.pop(d)

if (NumSets(S) < 2):

break

draw\_maze(wall\_list,m\_rows,m\_cols) #draw maze

endTime = time.time()

draw\_dsf(S)

print('runtime is: ', endTime-startTime)

########################## choice 2

elif userChoice == 2:

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

#Creates 'wall\_list' a set of two cells who have a wall between them

wall\_list = wall\_list(m\_rows,m\_cols)

while True:

d = random.randint(0,len(wall\_list)-1) # pick random wall

a,b = wall\_list[d] #obtain the num of the two cells

flag, r1, r2 = CheckSameSet\_c(S,a,b) # checks if same root, standard

if flag == False:

#Union two cell nums

if r1!=r2:

if S[r1]>S[r2]: # j's tree is larger

S[r2] += S[r1]

S[r1] = r2

else:

S[r1] += S[r2]

S[r2] = r1

#Pop wall

wall\_list.pop(d)

if (NumSets(S) < 2):

break

draw\_maze(wall\_list,m\_rows,m\_cols) #draw maze

endTime = time.time()

draw\_dsf(S)

print('runtime is: ', endTime-startTime)

else:

print('Please try again...')

**Standards of Conduct and Academic Dishonesty**

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.

* Joaquin Hidalgo