**CS2302 - Data Structures**

**Spring 2019**

**Lab Report 6**

April 16, 2019

Sebastian Gomez

**Introduction:**

Correct the program given and modify it to work as a disjoint set forest and display the correct maze. Use standard union and union by size to join the sets, and to show the difference, use different sizes to see the difference between them in matters of running time.

**Implementation:**

First, I passed the code from the disjoint set forest to the code given to display the maze. Then I saw how the maze done the given code looked like so I could compare with the final or correct one. I removed the for loop and followed the pseudocode provided in the Lab 6 file. I created a disjoint set forest of size M\*N because we M\*N cells, where M represents the number of rows and N the number of columns. After creating the disjoint set forest, I displayed it to see the sets. After that I created two methods which removed the walls, but the difference between them was how the sets joined. The first method was a maze by standard union and the second one union by size, and the difference between them is that union by size uses path compression and standard union does not. For the methods I used a while loop that kept removing walls until there was only one ser. In order to do so, I assigned a random value to a variable called d that varied from zero to the total number of walls minus 1 for it to stay in the boundary of number of walls. Then I created a variable c1 which I assigned the first the element of the list of wall d and in order to find its parent or to which root it pointed, I called the function find. I did the same for variable c2 but instead of assigning the first value, I assigned the second value of the list of wall d. By now having these two values, I did an if statement to compare them, that if they belonged to different sets, it would remove the wall between them and then it would join them to belong to the same set. This action repeated until all of the sets belonged to one set. The method union by size did the same process but the only difference was that I used find\_c to assign the values to c1 and c2 which find\_c used path compression, the to join the sets, it used union by size to join them. Also, I displayed the disjoint set forest after the unions were finalized to show the difference between the two methods and how it was only one set. After completing the methods, I printed out the running time and did the following table:

|  |  |  |
| --- | --- | --- |
| Lengths | Running Time of Standard Union | Running Time of Union by Size |
| Rows = 11  Columns = 15 | Time = 0.03332185745239258 nanoseconds | Time = 0.011464595794677734  nanoseconds |
| Rows = 20  Columns = 30 | Time =  0. 266345739364624 nanoseconds | Time =  0. 38820648193359375 nanoseconds |
| Rows = 50  Columns = 55 | Time = 8.079598903656006 nanoseconds | Time = 10.693266153335571 nanoseconds |
| Rows = 70  Columns = 80 | Time = 33.087238073349 nanoseconds | Time = 33.13569211959839 nanoseconds |

As lengths increase, we can see how standard union removes the walls and joins the sets faster than union by size.

**Conclusion:**

This lab showed how useful can be disjoint set forests and it different ways to perform them. I learned how to modify a code to do the same thing but in a better and correct way. I expect to keep learning more about ways of making things simpler that are applicable in real life problems.

**I, Sebastian Gomez, certify that this project is entirely my own work, I wrote, debugged, and tested the code being presented, performed 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.**

**Appendix:**

# -\*- coding: utf-8 -\*-

"""

@author: Sebastian Gomez

Course: Data Structure 2302

Assignment: Lab 6

Instructor: Olac Fuentes

T.A: Anindita Nath and Maliheh Zargaran

Purpose: Modify the given code to function with disjoint set forest and display the correct maze.

"""

import matplotlib.pyplot as plt

import numpy as np

import random

from scipy import interpolate

import time

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 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\_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

def NumSets(S):

count =0

for i in range(len(S)):

if S[i]<0:

count += 1

return count

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 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)

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)#displaying before creating the maze

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

draw\_dsf(S)#displaying before union

def mazeByUnion(S,walls):

while NumSets(S)>1:

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

c1 = find(S,walls[d][0])

c2 = find(S,walls[d][1])

if c1 != c2:

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

walls.pop(d)

def mazeByUnionBySize(S,walls):

while NumSets(S)>1:

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

c1 = find\_c(S,walls[d][0])

c2 = find\_c(S,walls[d][1])

if c1 != c2:

union\_by\_size(S,walls[d][0],walls[d][1])

walls.pop(d)

start = time.time()

mazeByUnion(S,walls)

end = time.time()

print(end-start)

draw\_dsf(S)#to check if there is only one set

draw\_maze(walls,maze\_rows,maze\_cols)#displaying the maze