Introduction

The object of this assignment is to create a maze using disjoint set forest. When two cells don’t belong to the same set, it means there is no path from those two cells. Thus the implementation is to unionize when two cells are not in the same set and remove the wall. Walls are defined in a array and by removing an appropriate wall, the maze will be drawn with single a path. In addition, we are comparing standard union and union with compressed. At the beginning, the program asks whether the user wants to use standard union or union with compression.

**Part-1:** Create maze using standard union

First, walls (array) is create with given rows and columns size. If the maze is drawn now, it will show a table separate with wall(frame) inn each cell. Since we are using disjoint set forest in order to create a maze, I defined “S” which is a DSF variable which contains the whole cells pointed to the root. The maze is complete when there is only one set which means there is only one set in disjoint set forest. Thus, while number of sets is greater than 1, it will remove or unionize the sets. The wall to test is selected randomly between 0 to the length of walls to create different maze every time. Random number is sent to union method, but it returns true if two numbers (cells) were in different sets and false otherwise. If true, pop the array at the random number that was selected.

**Part-2:** Create maze using union with path compression

In this part, the idea of implementation is exactly the same as part 1. The only difference is that instead of standard union, we are using union\_c (union with compressed path) to avoid having a long path in the sets.

Output: Standard union

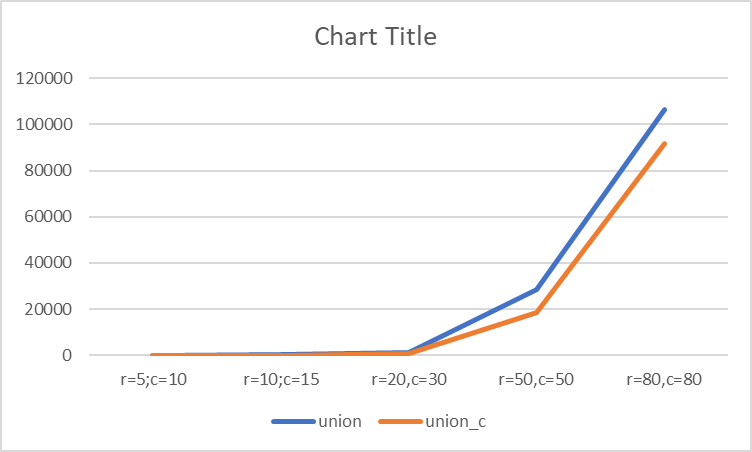
A close up of a piece of paper

Description automatically generatedA close up of a logo

Description automatically generated

Type 1 to use standard union, type 2 to use union with path compressions. 1

Time: 63.9836 milisecons.



Output: union\_c

A close up of a piece of paper

Description automatically generatedA close up of a logo

Description automatically generated

Type 1 to use standard union, type 2 to use union with path compressions. 2

Time: 39.9897 milisecons.

"""""""""""""""""""""""""""""""""""""""

Lab 6 - Disjoint Set Forest - Create a maze

04/13/2019 - Ken M. Amamori

CS2302 MW 10:30 - 11:50

Professor: Olac Fuentes

TA: Anindita Nath, Maliheh Zargaran

"""""""""""""""""""""""""""""""""""""""

import matplotlib.pyplot as plt

import numpy as np

import random

from scipy import interpolate

import time

import sys

#Find with path compression

def find\_c(S,i):

if S[i]<0:

return i

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

S[i] = r

return r

# Returns root of tree that i belongs to

def find(S,i):

if S[i]<0:

return i

return find(S,S[i])

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

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

ri = find\_c(S,i)

rj = find\_c(S,j)

if ri!=rj: # Do nothing if i and j belong to the same set

S[rj] = ri

return True #return true since i and j joined

return False

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

def union(S,i,j):

ri = find(S,i)

rj = find(S,j)

if ri!=rj: # Do nothing if i and j belong to the same set

S[rj] = ri # Make j's root point to i's root

return True #return true since i and j joined

return False

#create a array(dsf) with give size

def DisjointSetForest(size):

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

#return the number of sets in S

def count\_sets(S):

c = 0

for i in S:

if i==-1:

c+=1

return c

#draw maze

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)

#create a list with

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")

maze\_rows = 10

maze\_cols = 15

c = input("Type 1 to use standard union, type 2 to use union with path compressions.\t")

walls = wall\_list(maze\_rows,maze\_cols) #create a maze separeated by walls

draw\_maze(walls,maze\_rows,maze\_cols,cell\_nums=True) #draw maze with number in each cell

start = time.time()\*1000 #get the starting time

S = DisjointSetForest(maze\_rows\*maze\_cols) #create dsf to create maze

while count\_sets(S)>1: #while there are more than 1 set

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

if c=='1': #standard union

if union(S, walls[d][0], walls[d][1]): #if two numbers were in different sets

walls.pop(d) #remove

elif c=='2': #union with path compression

if union\_c(S, walls[d][0], walls[d][1]): #if two numbers were in different sets

walls.pop(d) #remove

else: #exit program

sys.exit("You chose the wrong input number!")

end = time.time()\*1000 #get the ending time

print("Time: ", round(end-start, 4), "milisecons.") #

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

plt.show()

**Conclusion:**

Throughout this lab, I learned how to manage well a disjoint set forest and its implementation in other project. I found that disjoint set forest is nice to use when there is a path or a flow that leads to one point to another. Once I realized how the maze will be built based on disjoint set forest, the design and the implementation was not hard as I expected. This lab was a very good practice to compare between standard union and union\_c. The result was obvious when I compare the time result. Therefore, in this lab, I have learned and experienced new things what I can utilize in the future.

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.