CS3 Lab 6 Report

Daniela Flores

Intro

Lab 6 consisted of drawing a maze with the help of a disjoint set forest. We were given the code that drew an incomplete maze by randomly removing half of the walls. We had to draw a maze that one would be able to find one path from once cell to another. We had to implement two types of union; standard union and union by compression to compare their running times.

Proposed Solution and Implementation:

I first started by analyzing the code Dr.Fuentes provided us to draw the maze by commenting out certain lines of code to see what role they played to better understand what I was working with. After that I started to implement the method that'll create the maze. I first started this method by setting its parameters to be the list of walls, and the number of rows and columns of the maze. I then initialized a disjoint set forest with its size being the number of rows times the number of columns. A while loop then is used to keep looping till there's no more than 1 set in the forest. Inside this while loop I got a variable to store a random number between 0 and the length of the list of walls minus one. Another variable I then initialized which stored the wall which index would be the variable containing the random number. I then split the walls set of coordinates to get the adjacent cells and stored those in variables. an if statement then checks if the roots of both cells aren't the same, if they are not the same, the union function then is called with the parameters being the disjoint set forest and both cells to get their roots to be the same, then the wall is popped. After the while loop is done iterating, the finished maze is drawn by calling the method Dr.Fuentes provided us with.

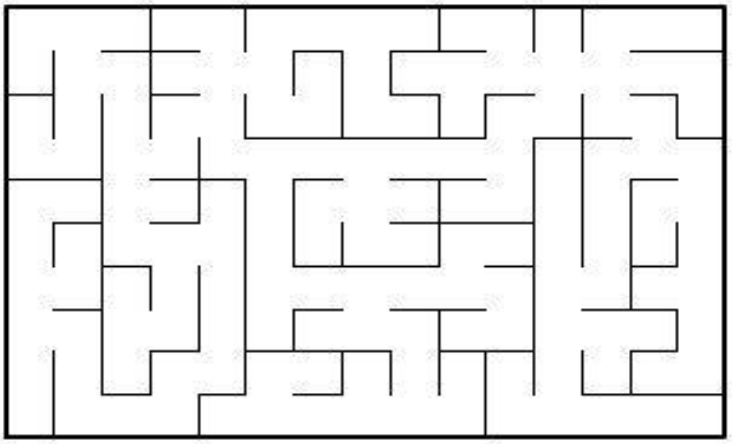
The other method which I wrote was one that kept track of the number of sets in the disjoint set forest. The parameters for this method was the disjoint set forest only. I implemented in this method a for loop which iterated through the array of the disjoint set forest. Inside of this for loop an if statement checked if the root of the given index was –1, and if it was the counter would be incremented by one. After the for loop is done iterating an if statement checks if the counter is bigger than one, a True is returned, otherwise a False is returned. This method is used in the while loop in the method that created the maze, as long as this method kept returning true, that while loop would keep iterating.

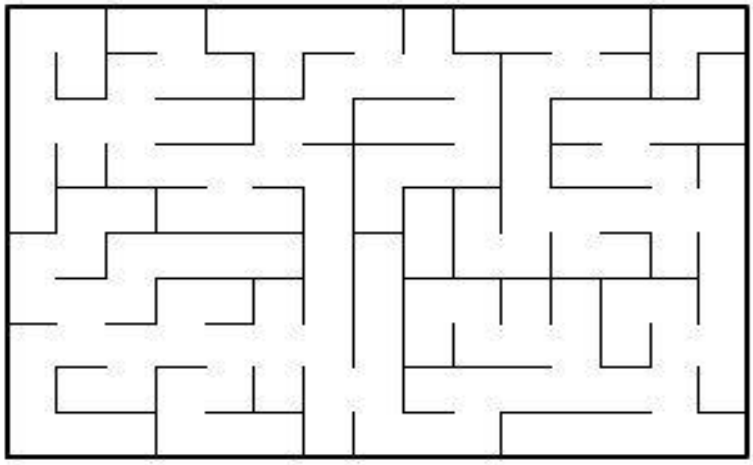
Another method that I had to implement into my program were the methods to union by compression. These methods were also provided by Dr.Fuentes, I just wrote them into my program and called them in the method which created the maze but this time using using union by size with compression.

Experimental Results:

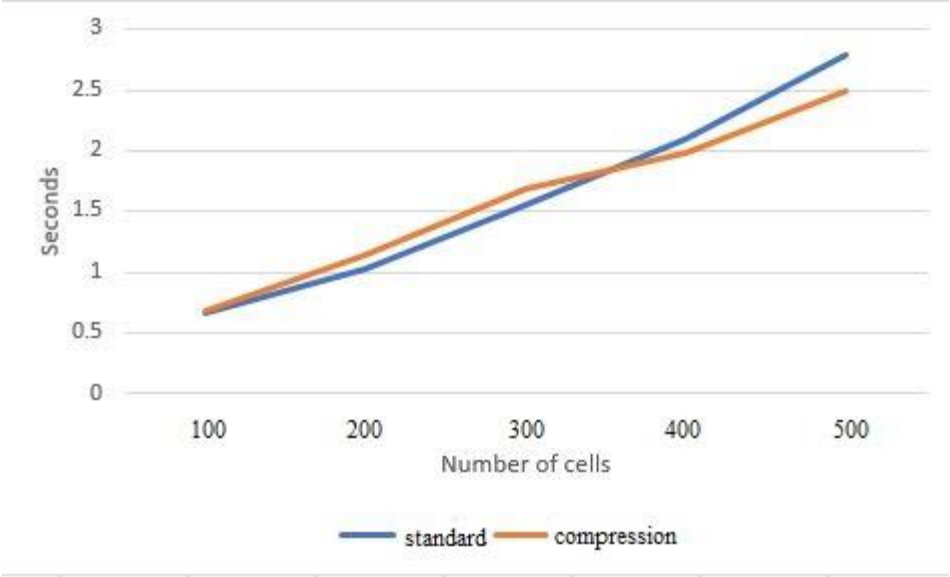
I experimented with the program first provided by Dr.Fuentes as I needed to know what every of his methods did and what role they played. I also experimented with the number of rows and columns of the maze to see is my program didn’t have any bugs. I also had to experiment a lot with the walls list as at first, I didn’t really know how to get the adjacent cells that the wall was in between. The following image is what my program’s output is when run:

```
In [3]: runfile('C:/Users/flore/.spyder-py3/cs3Lab6.py',
wdir='C:/Users/flore/.spyder-py3')
*Maze with standard union
*Followed by maze with union by size with path compression
```





Running Times:



Standard	Compression
0.352	0.332
0.664	0.678
1.036	1.14
1.557	1.68
2.096	1.978
2.781	2.5

Conclusion:

What I leaned from this lab is how to work with disjoint set forests. I also learned on how there's two types of unions which were standard union and union by size with path compression. What I also learned from this lab is how one is able to draw a maze that's contains x cells in python.

Appendix:

#Course:CS2302

#aAuthor:Daniela Flores

#Lab6

#Instructor:Olac Fuentes

#T.A: Dita Nath

#date of last Mod: 4/13/19

#purpose of program: draw a maze with the help of a disjoint set forest to better

#understand this data structure

```
import matplotlib.pyplot as plt
import numpy as np
import random
import time

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


#########################################################
#union by compression
def union_by_size(S,i,j):
    ri = findC(S,i)
    rj = findC(S,j)
    if ri!=rj:
        if S[ri] > S[rj]:
            S[rj] += S[ri]
            S[ri] = rj
        else:
            S[ri]+=S[rj]
            S[rj] = ri



#finds the root using path compression
def findC(S,i):
    if S[i] < 0:
        return i
    r = findC(S,S[i])
    S[i] = r
    return r



#########################################################
#method that returns True if theres more than one set in the disjoint set forest
# and false otherwise
def numOfSets(forest):
```

```
    count = 0
    #counts number of sets
    for i in range(len(forest)):
        if forest[i] == -1:
            count = count +1
    if count>1:
        return True
    else:
        return False


#method that will keep popping walls if the adjacent cells are not in the same set
#draws maze with standard union
def wallsAsforests(walls,row,cols):


    forest = DisjointSetForest(row*cols)
    #while the disjoint set forest has more than 1 set this loop will keep iterating
    while numOfSets(forest) == True:
        #gets random integer
        d = random.randint(0,len(walls)-1)
        #stores random wall
        randomWall = walls[d]
        #splits the wall's cordinate to get adjacent cells
        cellX = randomWall[0]
        cellY = randomWall[1]
        #checks if cells are not in same set
        if findC(forest,cellY) != findC(forest,cellX):
            #gets both cells to same set
            union(forest,cellX,cellY)
            #removes wall
```

```
            walls.pop(d)
    #draws finished maze
    draw_maze(walls,row,cols)

#method that draws maze with path compression
def wallsAsforestsC(walls,row,cols):

    forestC = DisjointSetForest(row*cols)
    #while the disjoint set forest has more than 1 set this loop will keep iterating
    while numOfSets(forestC) == True:
        #gets random integer
        d = random.randint(0,len(walls)-1)
        #stores random wall
        randomWall = walls[d]
        #splits the wall's cordinate to get adjacent cells
        cellX = randomWall[0]
        cellY = randomWall[1]
        #checks if cells are not in same set
        if findC(forestC,cellY) != findC(forestC,cellX):
            #gets both cells to same set
            union_by_size(forestC,cellX,cellY)
            #removes wall
            walls.pop(d)
    #draws finished maze
    draw_maze(walls,row,cols)



############################################################
maze_rows = 10
```

```
maze_cols = 15
walls = wall_list(maze_rows,maze_cols)
walls2 =  wall_list(maze_rows,maze_cols)
print('*Maze with standard union\n*Followed by maze with union by size with path compression')

#start = time.time()
wallsAsforests(walls,maze_rows,maze_cols)
#elapsedTime = time.time() - start
#print('standard:',elapsedTime)
#
#start = time.time()
wallsAsforestsC(walls2,maze_rows,maze_cols)
#elapsedTime = time.time() - start
#print('compression:',elapsedTime)
```

Contract:

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 inappropiate assistance to any student in the class.

X