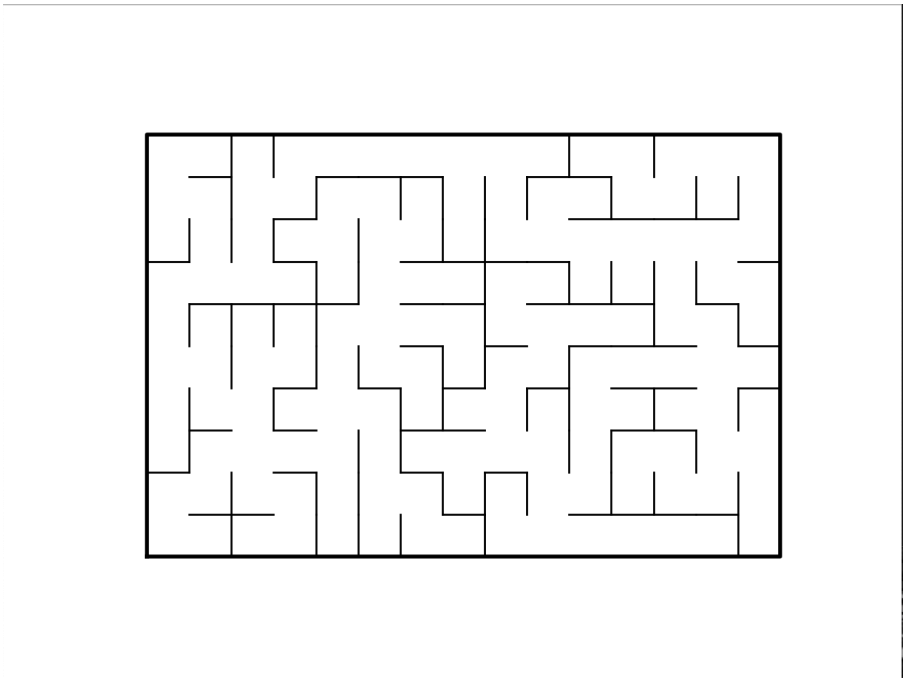
Sergio Gramer – 88521512
Lab #6: Disjoint Set Forest Maze

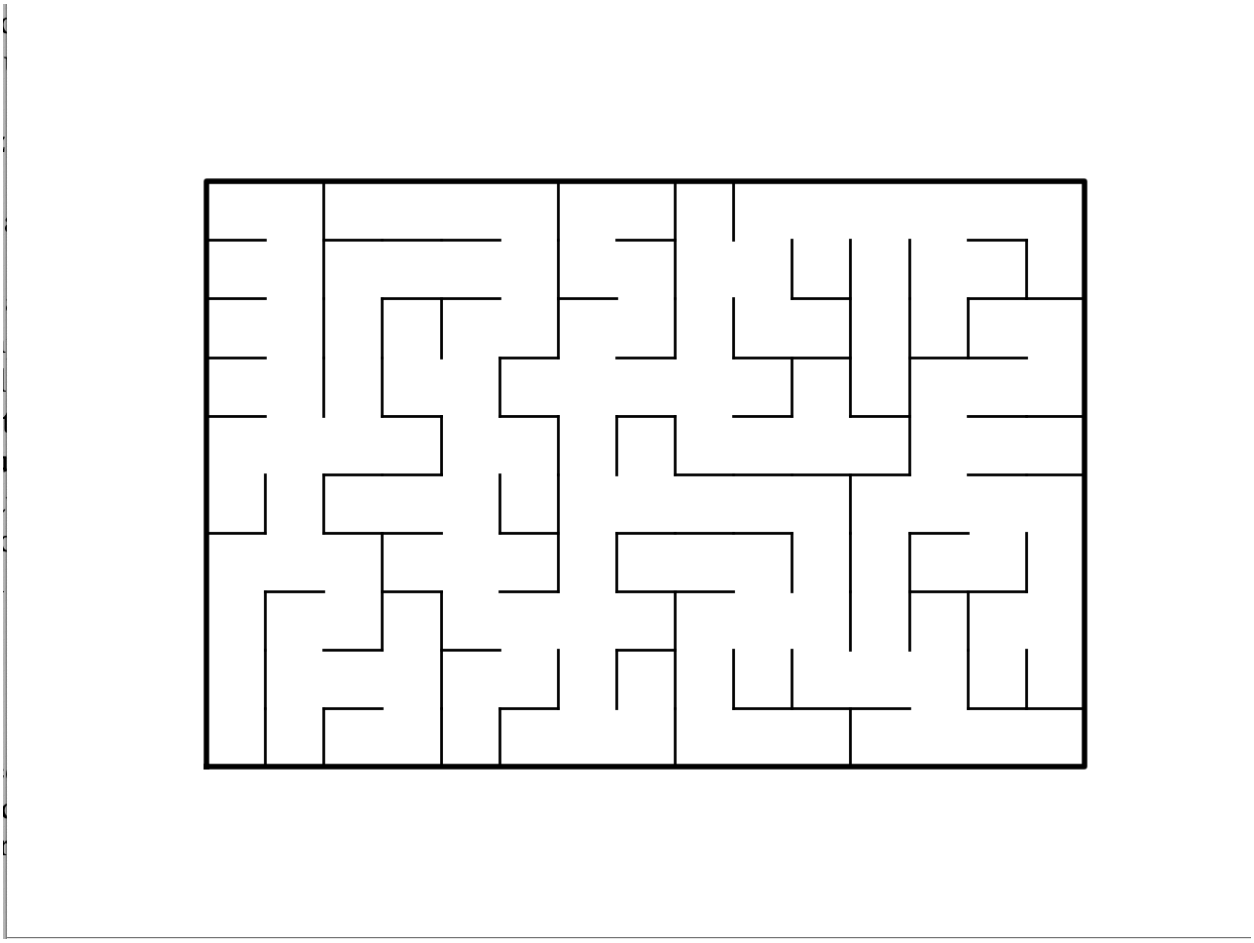
In lab 6 we are tasked with creating a maze out of a disjoin set forest. We are to do this maze either by standard union that is given to us inside of the DSF code or by union by compression which is also given to us within the DSF code. We are to do keep using the union function to create only one set inside of the disjoint set forest. Upon which we would exit the program. We are supposed to be removing walls at random. Using random numbers allows us to see if the program will give us an entirely random maze without repetition or being able to replicate each maze easily.

Initially I saw the code that the professor had provided. I attempted to create them both inside of the same method. After carefully re-reading the instructions I decided that I would do what I did last lab and give the user a choice of whether doing Standard Union by typing “stand” or using Path Compression by typing “comp”. By splitting the problem I was able to take the professors original code and make it randomly remove walls from the maze. Since the maze was already being created, I just modified the random numbers to fit my needs. I created an if statement that would check if the random numbers walls would be able to be removed. I kept getting errors because I didn’t have the union returning anything. I didn’t realize this until a few attempts. I then modified both of the methods to return true if they were able to be combined or false if they weren’t. This allowed both of my methods to be successful.

```
In [100]: runfile('/Users/SergioGramer/Desktop/Spring2019/CS2302/Lab 6 Maze/lab6.py', wdir='/Users/SergioGramer/Desktop/Spring2019/CS2302/Lab 6 Maze')
Type "standard" to use Standard Union
Type "comp" for Union by Compression

comp
Chose size
It took  0:00:00.180947  seconds to create the maze using Path Compression.
```



```
In [99]: runfile('/Users/SergioGramer/Desktop/Spring2019/CS2302/Lab 6 Maze/lab6.py', wdir='/Users/SergioGramer/Desktop/Spring2019/CS2302/Lab 6 Maze')
Type "standard" to use Standard Union
Type "comp" for Union by Compression

standard
Chose standard
It took  0:00:00.180863  seconds to create the maze using Standard Union.
```



In conclusion, I learned that DSF can be used to create random mazes. This is especially useful when you are using methods for union and compression union. The running times are very similar but do in fact change when you select with compression or standard union. Both of them are doing essentially the same thing except one sends back item being searched with the root being the root of the whole set. Because of the recursive call inside of the find method we see that the running times are a bit high. When calculated I calculated O(n) = 3^n.

Appendix: Source Code:

```
import matplotlib.pyplot as plt
import numpy as np
import random
from datetime import datetime



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
# Implementation of disjoint set forest
# Programmed by Olac Fuentes
# Last modified March 28, 2019

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


def dsfToSetList(S):
    #Returns aa list containing the sets encoded in S
    sets = [ [] for i in range(len(S)) ]
    for i in range(len(S)):
        sets[find(S,i)].append(i)
    sets = [x for x in sets if x != []]
```

```
    return sets

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
        return True #uses true or false to return for whether the method
    return False   #should execute or skip

def union_c(S,i,j):
    # Joins i's tree and j's tree, if they are different
    # Uses path compression
    ri = find_c(S,i)
    rj = find_c(S,j)
    if ri!=rj:
        S[rj] = ri #uses true or false to return for whether the method
        return True #should execute or skip
    return False

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[ri] = rj
        else:
            S[ri] += S[rj]
            S[rj] = ri

def NumSets(S):
    count =0
    for i in S:
        if i < 0:
            count += 1
    return count

def MazeStandardUnion(S) : #will use the standard union method to create the maze
    while NumSets(S) > 1 : #will execute so long as the number of sets is greater than 1
        d = random.randint(0, len(walls)-1) # d is the random integer we create to remove walls based
on that integer
        if union(S, walls[d][0], walls[d][1]) is True : #uses the true or false statement inside of the
union
            walls.pop(d)                          #to decide if we execute

def MazeCompression(S) : #will use the Union_c with compression to create the maze
    while NumSets(S) > 1 :
```

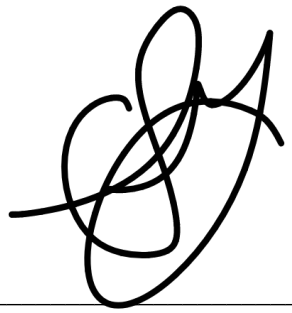
```
        d = random.randint(0, len(walls)-1)
        if union_c(S, walls[d][0], walls[d][1]) is True:
            walls.pop(d)

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)
M = DisjointSetForest(maze_rows * maze_cols)
print('Type "standard" to use Standard Union \nType "comp" for Union by Compression')
choice = input()
if choice == "standard" :
    TimeStart = datetime.now()
    print('Chose standard')
    MazeStandardUnion(M)
    draw_maze(walls,maze_rows,maze_cols)
    plt.show()
    TimeEnd = datetime.now()
    print('It took ', TimeEnd - TimeStart, ' seconds to create the maze using Standard Union.')

elif choice == "comp" :
    TimeStart = datetime.now()
    print('Chose size')
    MazeCompression(M)
    draw_maze(walls,maze_rows,maze_cols)
    plt.show()
    TimeEnd = datetime.now()
    print('It took ', TimeEnd - TimeStart, ' seconds to create the maze using Path Compression.')
else :
    print("Please type the provided choices")
```

ACADEMIC CERTIFICATION

I “Sergio Gramer” 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.

Signed: 04/15/2019

Sergio Gramer



_____