Ivan Perez
80602918
Lab 7 Report

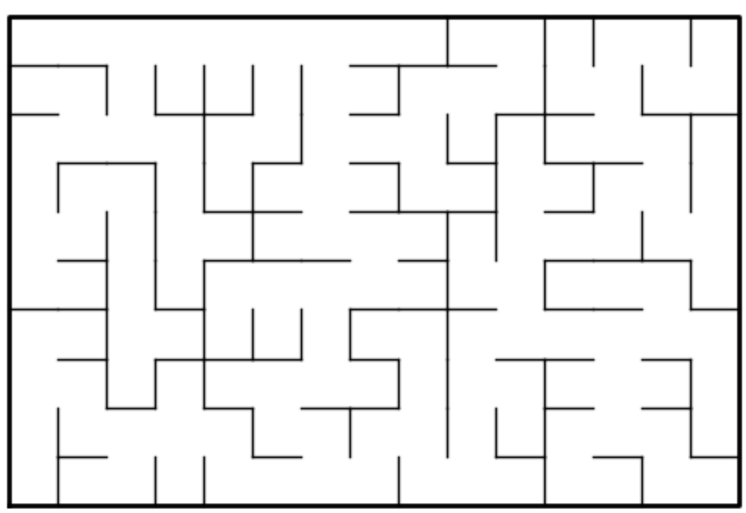
Introduction

The previous lab's maze building algorithm must be revised to allow cases where a maze can be created with any amount of wall removals instead of just removing the amount of cells minus one amount of walls. The code must also ask for user input of how many walls they would like to remove after displaying the amount of cells . Once that's completed, an algorithm must be created to create the adjacency list of the maze. The main part of this lab is to create 3 search algorithms that utilize breadth-first search and depth-first search. One algorithm uses breadth-first search while the other two uses depth-first search but executed differently (One is made to be using stacks while the other uses recursion). The end goal is to display the paths that the algorithm found to the user.

Proposed solution design and implementation

I revised the maze building algorithm to also return the removed walls of the maze. Every wall removal meant that there was a union of vertices so I simply took the removed walls to construct the adjacency list. Implementing the search algorithms is a little more straightforward since the algorithm behind it is already really instructional so shifting it over to actual code seems not too problematic. An issue I can't solve is how to derive the path from start to end with the discovered list. Implementing pyplot is also a bit difficult.

Experimental results

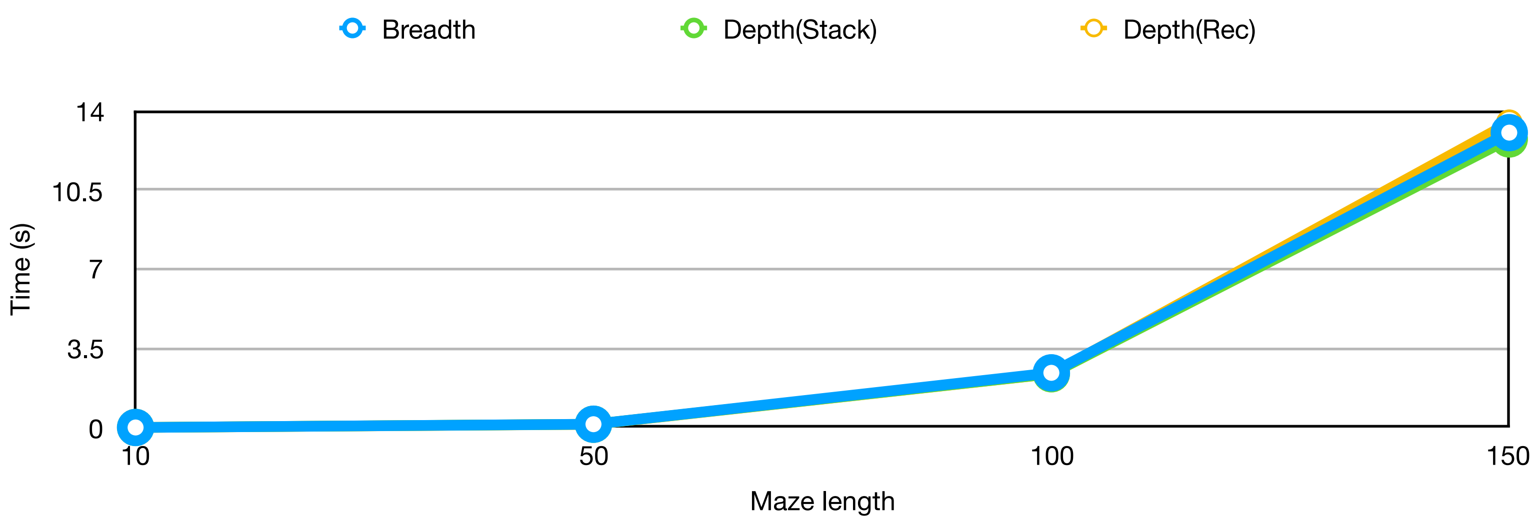


```
Number of cells:  150

How many walls would you like to remove? 149
There is a unique path from source to destination
Breadth First Search
[0, 15, 30, 31, 45, 16, 46, 17, 2, 18, 1, 3, 19, 33, 4, 5, 6, 7, 21,
22, 20, 23, 35, 8, 38, 34, 36, 9, 53, 37, 51, 10, 24, 52, 66, 39, 65,
67, 40, 54, 50, 64, 68, 82, 25, 55, 49, 81, 83, 56, 70, 80, 96, 57,
69, 85, 95, 97, 111, 42, 58, 84, 86, 100, 112, 126, 27, 41, 43, 59,
73, 87, 101, 115, 113, 127, 141, 26, 28, 44, 72, 102, 98, 128, 140,
142, 11, 13, 29, 71, 103, 99, 129, 125, 139, 143, 12, 14, 88, 118,
114, 130, 110, 124, 138, 89, 117, 145, 109, 123, 137, 74, 104, 116,
132, 144, 94, 122, 136, 119, 131, 147, 107, 135, 146, 148, 106, 108,
133, 105, 121, 93, 134, 90, 120, 78, 149, 75, 63, 79, 60, 76, 61, 91,
92, 77, 62, 47, 32, 48]
Depth First using stack
[0, 15, 30, 45, 46, 31, 16, 17, 18, 33, 19, 4, 5, 6, 21, 20, 35, 36,
51, 66, 67, 82, 83, 81, 96, 111, 126, 141, 142, 143, 140, 139, 138,
137, 136, 135, 122, 107, 108, 93, 78, 79, 63, 106, 121, 120, 105, 90,
75, 76, 91, 92, 77, 62, 47, 48, 32, 60, 61, 123, 124, 125, 110, 109,
94, 127, 112, 113, 128, 129, 130, 145, 144, 114, 98, 99, 97, 95, 80,
68, 65, 64, 49, 50, 37, 34, 7, 22, 23, 38, 53, 52, 8, 9, 24, 39, 54,
55, 70, 85, 100, 115, 101, 86, 87, 102, 103, 118, 117, 132, 147, 148,
133, 134, 149, 131, 146, 116, 88, 89, 104, 119, 74, 69, 84, 56, 57,
58, 73, 72, 71, 59, 44, 29, 42, 43, 41, 27, 28, 13, 14, 26, 11, 12,
40, 25, 10, 3, 2, 1]
Depth First using recursion
[0, 15, 30, 31, 16, 17, 2, 1, 18, 3, 19, 4, 5, 6, 7, 22, 23, 8, 9, 10,
24, 39, 40, 25, 54, 55, 56, 57, 42, 27, 26, 11, 12, 28, 13, 14, 41,
43, 58, 59, 44, 29, 73, 72, 71, 70, 69, 84, 85, 86, 87, 102, 103, 88,
89, 74, 104, 119, 118, 117, 116, 132, 131, 146, 147, 148, 133, 134,
149, 100, 101, 115, 38, 53, 52, 21, 20, 35, 34, 36, 37, 51, 66, 65,
50, 64, 49, 67, 68, 82, 81, 80, 96, 95, 97, 111, 112, 113, 98, 99,
128, 129, 114, 130, 145, 144, 126, 127, 141, 140, 125, 110, 109, 94,
139, 124, 138, 123, 137, 122, 107, 106, 105, 90, 75, 60, 61, 76, 91,
92, 77, 62, 47, 32, 48, 121, 120, 108, 93, 78, 63, 79, 136, 135, 142,
143, 83, 33, 45, 46]
```

I was not able to display the found path but I was able to print the discovered list. I also computed the running times of the algorithms shown below

```
[10, 10]
0.0003781318664550781
0.00045800209045410156
0.0003712177276611328
[50, 50]
0.14500999450683594
0.14391493797302246
0.1398477554321289
[100, 100]
2.417449951171875
2.374760150909424
2.3724472522735596
[150, 150]
13.036717891693115
12.756168127059937
13.471085786819458
```



The results and graph shows that the depth first search using recursion took the least amount of time more often than the other search methods.

Conclusions
The difficult part of this lab was trying to implement the peplos. Understanding the search algorithms was pretty straight forward but displaying those results took a lot more work than I was expecting. This lab was very useful in seeing and understanding search algorithms and was a great lab to do.

Appendix

```
######################################################################################
#
#
# Course: CS2302 (MW - 1:30pm)
# Author: Ivan Perez
# Assignment: Lab 7 Graphs
# Instructor: Olac Fuentes
# T.A.: Anindita Nath
# Last Modified: May 5, 2019
# Purpose: Using methods of the previous lab that makes mazes, This code's purpose
# is to revise the maze creation by allowing the user to determine how many
# walls will be removed. This code also uses breadth first search and depth first
# search to find the start and end path.
#
######################################################################################
#
import matplotlib.pyplot as plt
import numpy as np
```

```
import random

# Disjoit SetForest object and methods
def DisjointSetForest(size):
    return np.zeros(size,dtype=np.int)-1


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_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



# Maze creation methods #
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 createMaze(M,N,u):
    #Creates Maze with Compression
    dsf = DisjointSetForest(M*N)
    walls = wall_list(M, N)
    extra = 0
    removed = []
    if u >= M*N:
        extra = u - M*N
        u = M*N -1
    while u > 0:
        d = random.randint(0,len(walls)-1)
        i = walls[d][0]
        j = walls[d][1]
        if find_c(dsf, i) != find_c(dsf, j):
            # if values are not part of the same set
            removed.append(walls.pop(d)) # removes wall
```

```
            union_by_size(dsf, i , j ) #unites the two sets
            u -= 1

    while extra > 0 and len(walls)>0:
        d = random.randint(0,len(walls)-1)
        i = walls[d][0]
        j = walls[d][1]

        removed.append(walls.pop(d)) # removes wall
        union_by_size(dsf, i , j ) #unites the two sets
        extra -= 1

    return removed,walls


############################### Assignment Code #######################3
def adjacencyList(removed, s):
    al = np.empty(s, dtype=object)
    for i in range(s):
        al[i] = []
    for i in range(len(removed)):
        al[removed[i][0]].append(removed[i][1])
        al[removed[i][1]].append(removed[i][0])
    for i in range(len(al)):
        al[i].sort()
    return al

# Search Methods #
def breadth_first(al):
    # breadth first search using a Queue
    Q = [0] # Queue
    discovered = [0]
    while len(Q)>0:
        value = Q[0]
        del Q[0]
        location = al[value] #checks adjacency of vertex
        for i in range(len(location)):
            #adds to Queue if it hasnt already been visited
            if location[i] not in discovered:
                Q.append(location[i])
                discovered.append(location[i])
    return discovered
```

```
def depth_firstStack(al):
    # depth first search using a stack
    S = [0]
    visited = []
    while len(S)>0:
        value = S[len(S)-1]
        del S[len(S)-1]
        if value not in visited:
            place = al[value]
            visited.append(value)
            for i in range(len(place)):
                S.append(place[i])
    return visited

def depth_firstRec(al, cur, visited):
    # depth first search using recursion
    if cur not in visited:
        visited.append(cur)
        place = al[cur]
        for i in range(len(place)):
            visited = depth_firstRec(al, place[i], visited)
    return visited

 ###################### Main Method ##############################
wall = wall_list(10,15)

x = 10
y = 15
n = x*y

print('Number of cells: ',n)
i = int(input('How many walls would you like to remove? '))

if i < n-1:
    print('A path from source to destination is not guaranteed to exist')
elif i == n-1:
    print('There is a unique path from source to destination')
else:
    print('There is at least one path from source to destination')


plt.close("all")
removed, walls = createMaze(10,15, i)
```

```
al = adjacencyList(removed, 150)
bfs = breadth_first(al)
dfs = depth_firstStack(al)
dfr = depth_firstRec(al, 0, [])

print('Breadth First Search')
print(bfs)

print('Depth First using stack')
print(dfs)

print('Depth First using recursion')
print(dfr)

draw_maze(walls,10,15)
```

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.

Ivan Perez