LAB 6

CS2302 – Data Structures

Quinones Rodriguez, Diego A

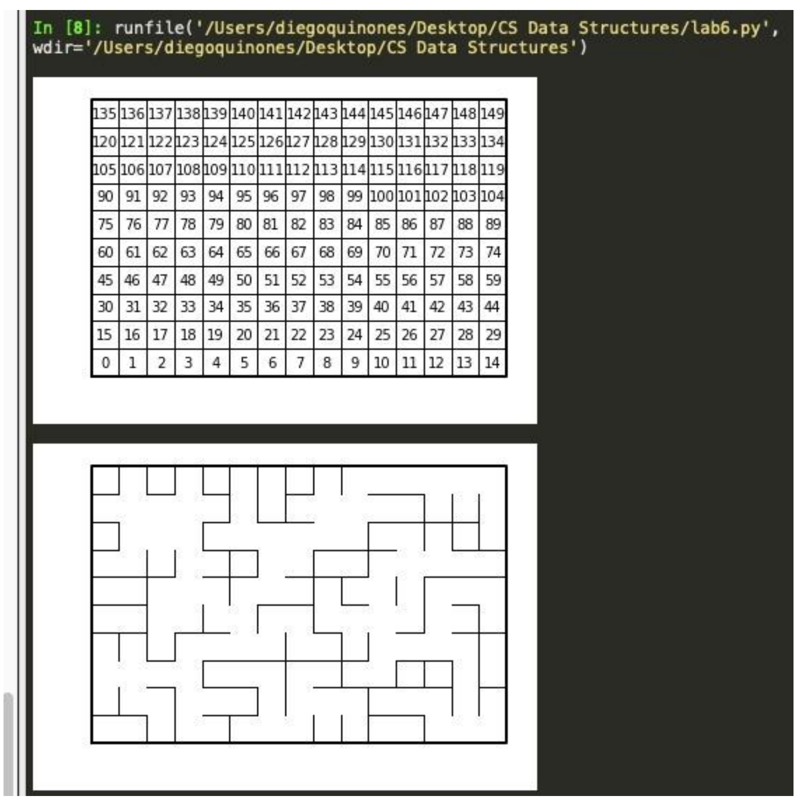
Report

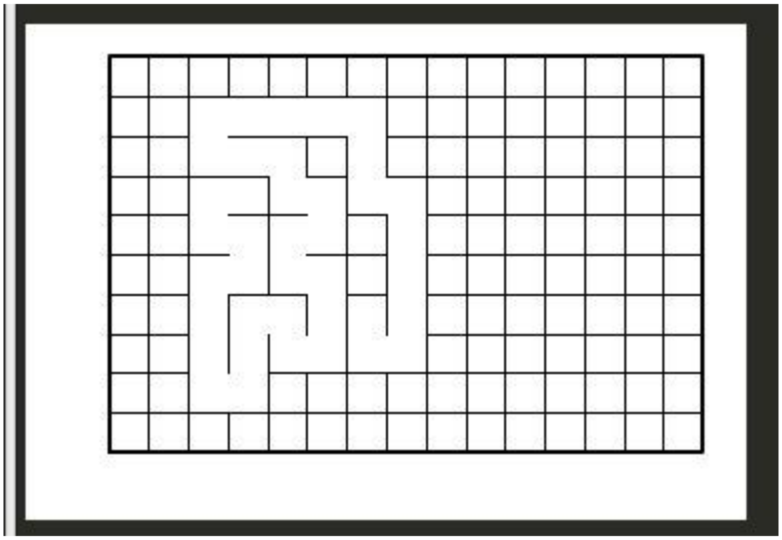
On lab 6 we had the assignment of creating a maze with the use of a DSF (disjoint set forest). We had to create a code that would delete walls in the table to create a path for the maze.

My final code ended having 17 different methods that will work in conjunction to create the maze table that had a total 150 blocks being 15 blocks wide and 10 blocks tall.

I personally was not able to make it so it creates the full maze, mine only deletes a couple of random walls.

Screenshots





Running Times

.777

.69

.808

.84

Code

```
#
Starting
point
for
program
to build
and draw
a maze
            # Modify program using disjoint set forest to ensure there is exactly one
            # simple path joiniung any two cells
            # Programmed by Diego Quinones
            # Last modified March 28, 2019


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

```python
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):
    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 path(S):
    lengths =0
```

```python
    index = 0
    for s in range(len( S)):
        if len(S[s])>lengths:
            lengths = len(S[s])
            index= s
    p = []
    p.append(S[index][0])
    p+=S[index]
    return p


def findAdjacent(walls,n):
    adjacent = []
    for i in walls:
        if i[0]==n:
            adjacent.append(i[1])
        if i[1]==n:
            adjacent.append(i[0])
    return adjacent
def adjacents(walls,maze_rows,maze_cols ):
    adj = [ ]
    for i in range(maze_rows*maze_cols):
        adj.append(findAdjacent(walls,i))
    return adj
def wall_finder(walls,a,b):
    for i in range(len(walls)):
        if walls[i]==[a,b] or  walls[i] == [b,a]:
                return i
    return None




def roots(S):
    root = []
    for s in range(len(S)):
        if len(S[s])==1:
            r = S[s][0]
            root.append(r)
    return root


def maze(S,walls,adjacents):
```

```
    d = random.randint(0,len(S)-1)
    v=[]
    for i in range(len(S)*2):
        if adjacents[d]!=[]:
            r = random.randint(0,len(adjacents[d])-1)
            if find(S,d)!=find(S,adjacents[d][r]):
                union(S,d,adjacents[d][r])
                walls.pop(wall_finder(walls,d,adjacents[d][r]))
                v.append(d)
                r = adjacents[d][r]
                adj = adjacents[d]
                adj2 = adjacents[r]
                cell = r
                adj.remove(r)
                adj2.remove(d)
                if len(adj)==1:
                    other = adj[0]
                    adj3=adjacents[other]
                    adj3.remove(d)
                    adjacents[other]=adj3
                    adj=[]
                adjacents[d]=adj
                adjacents[r]=adj2
                d = cell
        else:
             d = v.pop()
    return S


def maze_compression(S,walls,adjacents):
    b = random.randint(0,len(S)-1)
    v =[]
    while len(adjacents)>0:
        if adjacents[d]!=[]:
            k = random.randint(0,len(adjacents[d])-1)
            if find_c(S,b)!=find_c(S,adjacents[b][k]):
                union_c(S,b,adjacents[b][k])
                walls.pop(wall_finder(walls,d,adjacents[b][k]))
                v.append(b)
                k = adjacents[b][k]
                adj = adjacents[b]
                adj1 = adjacents[k]
                cell = k
```

```
                adj.remove(k)
                adj1.remove(b)
                if len(adj)==1:
                    other = adj[0]
                    adj2=adjacents[other]
                    adj2.remove(b)
                    adj=[]
                    adjacents[other]=adj2
                adjacents[b]=adj
                adjacents[k]=adj1
                b = cell
        else:
             b = v.pop()
    return S


def maze_height(S,walls,adjacents):
    v =[]
    b = random.randint(0,len(S)-1)
    while len(adjacents)>0:
        if adjacents[d]!=[]:
            k = random.randint(0,len(adjacents[b])-1)
            if find(S,b)!=find(S,adjacents[b][k]):
                union_by_size(S,b,adjacents[b][k])
                walls.pop(wall_finder(walls,b,adjacents[b][k]))
                v.append(b)
                k = adjacents[b][k]
                adj1 = adjacents[k]
                adj = adjacents[b]
                cell = k
                adj1.remove(d)
                adj.remove(k)

                if len(adj)==1:
                    other = adj[0]
                    adj2=adjacents[other]
                    adj2.remove(b)
                    adjacents[other]=adj2
                    adj=[]
                adjacents[b]=adj
                adjacents[k]=adj1
                b = cell
        else:
```

```python
            b = v.pop()
    return S




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


def find(S,i):
    if S[i]<0:
        return i
    return find(S,S[i])


def find_c(S,i):
    if S[i]<0:
        return i
    r = find_c(S,S[i])
    S[i] = r
    return r

def dsfToSetList(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 union(S,i,j):
    ri = find(S,i)
    rj = find(S,j)
    if ri!=rj:
        S[rj] = ri
def union_c(S,i,j):
    ri = find_c(S,i)
    rj = find_c(S,j)
    if ri!=rj:
        S[rj] = ri
```

```python
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
```

```python
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)
for i in range(len(walls)//2): #Remove 1/2 of the walls
    d = random.randint(0,len(walls)-1)
    walls.pop(d)
```

```python
draw_maze(walls,maze_rows,maze_cols)
walls = wall_list(maze_rows,maze_cols)
disjoint = DisjointSetForest(maze_rows*maze_cols)
adj = adjacents(walls,maze_rows,maze_cols)
disjoint= maze(disjoint,walls,adj)
disjoint= dsfToSetList(disjoint)
pa=path(disjoint)
walls.append(pa)
draw_maze(walls,maze_rows,maze_cols)
```

Academic Honesty Certification

“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.” -Diego Quinones