Erick Perchez

80582912

CS 2302 Data Structures

4/16/2019

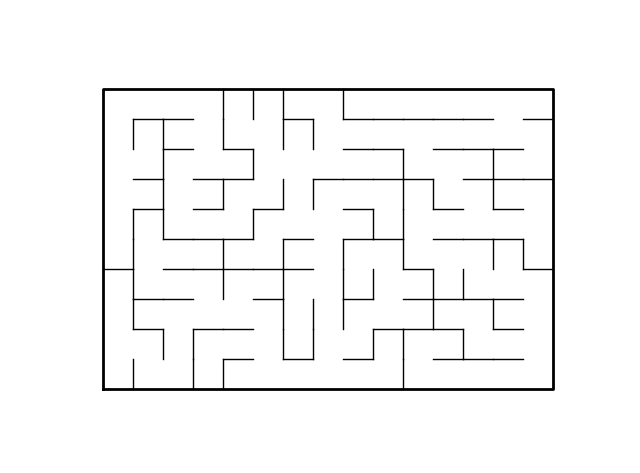
Lab 6

**Introduction:** In this lab we were given one file that defined graphs and we were to make a maze that only had one path from the beginning to end, and having exactly one path to get to any of the cells.

**Proposed solution:**

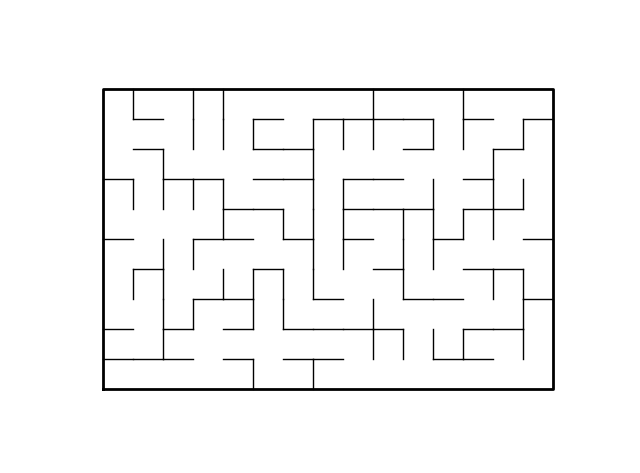
1. Create a maze with Standard Unions- This method takes in a list of walls(explained later) and the disjoint set forest. I use the method given to us called numSets to determine how many sets there are in the disjoint set forest. If there is more than one set, that means that there are still cells that have not been reached. I use a while loop that keeps looping as long as there are more than 1 set in the DSF. I pick a random number between 0 and one less than the length of the list of walls. I also made a method that checks if two numbers are within the same set by using another given method, find, to compare the roots. I use an if conditional to check if in the DSF, W(being the list of walls)[random number][0] and W[random number][1] are in the same set. If they are not in the same set, I use a method given to us, union, to unite the two sets. I then remove (random number) from the list of walls. This repeats until there is only one set in the DSF
2. Create a maze with Compression-This method takes in a list of walls and the disjoint set forest. I use the method given to us called numSets to determine how many sets there are in the disjoint set forest. If there is more than one set, that means that there are still cells that have not been reached. I use a while loop that keeps looping as long as there are more than 1 set in the DSF. I pick a random number between 0 and one less than the length of the list of walls. I also made a method that checks if two numbers are within the same set by using another given method, find, to compare the roots. I use an if conditional to check if in the DSF, W(being the list of walls)[random number][0] and W[random number][1] are in the same set. If they are not in the same set, I use a method that unites two sets depending on their size, called unionBySize, to unite the two sets. I then remove (random number) from the list of walls. This repeats until there is only one set in the DSF
   1. unionBySize- This method receives S (a DSF), i(an integer), and j(an integer). I then initialize two variables, ri, and rj, that use the method find to find the rood of the two integers given respectively. I then check if the roots are the same, if they are not, I check which root is greater. If root of I (ri) is greater, I add S[rj] and S[ri] and set them equal to S[ri]. I then set S[ri] to equal rj (to point to the root). Else, I add S[ri] and S[rj] and set them equal to S[ri] then I make S[rj} equal ri (to point to the root)
   2. sameSet- This method receives S (a DSF), a (an integer), and b (an integer). I then create another two variables that save the root of the given integers respectively. If the roots are the same then it returns True, otherwise, it returns false.
   3. List of Walls- To get the list of walls, I use the method given to us, wall\_list, and input the rows and columns as parameters. This creates a list with all the walls in the maze.
   4. To time- I set the variable start to time.time() right before I begin to build the maze (in both the compression method or the standard method). I then run the maze building method. Soon after it is ran, I set the variable end to time.time() and I subtract start from end and multiply it by 1000, I then round it to 6 overall figures.

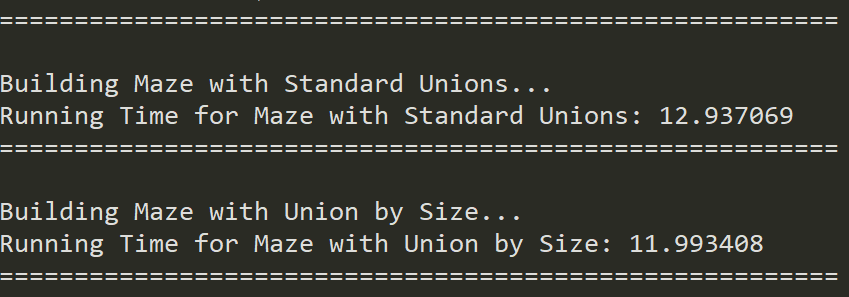
**Experimental Results:**



Maze with Standard Union

Maze with Union By Size

\



**Conclusion:** I learned a lot on this lab. While learning to navigate and manipulate a DSF, I bettered my skills in iterative methods and as well to traverse lists.

**Appendix:**

'''

Course: CS2302

Author: Erick Perchez

Assignment: Lab 6

Instructor: Dr. Fuentes

TA: Andita Nath

Date: 04/15/2019

Purpose: To use DSF to create a maze where each block is only accessible

through one path instead of multiple paths

'''

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 find\_c(S, i):

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

def unionBySize(S, i, j):

ri = find\_c(S, i)

rj = find\_c(S, j)

if ri != rj:

if S[ri] > S[rj]: #Checks if root i is greater than root of j

S[rj] += S[ri]

S[ri] = rj #points i to j

else:

S[ri] += S[rj]

S[rj] = ri #points j to i

def numSets(S):

sets = 0

for i in range(len(S)):

if S[i] < 0:

sets += 1

return sets

def sameSet(S,a,b):

rA = find(S,a) #Checks for the roots of both indices

rB = find(S,b)

if rA == rB:

return True

return False

def buildMaze\_Standard(W, S):

while numSets(S) > 1: #if theres more than 1 set, we arent done

rand = random.randint(0, len(W)-1)

if sameSet(S, W[rand][0], W[rand][1]) is False: #checks if they dont belong in the same set

union(S, W[rand][0], W[rand][1]) #creates union between the two sets

W.pop(rand) #removes rand from the maze

return W

def buildMaze\_Compression(W, S):

while numSets(S) > 1: #if theres more than 1 set, we arent done

rand = random.randint(0, len(W)-1)

if sameSet(S, W[rand][0], W[rand][1]) is False: #checks if they dont belong in the same set

unionBySize(S, W[rand][0], W[rand][1]) #creates union between the two sets

W.pop(rand) #removes rand from the maze

return W

'''

===============================================================================

Main

'''

plt.close("all")

maze\_rows = 10

maze\_cols = 15

Walls = wall\_list(maze\_rows, maze\_cols)

S = np.zeros(maze\_rows \* maze\_cols, dtype = np.int)-1 #DSF

print('========================================================')

print()

print('Building Maze with Standard Unions...')

start = time.time()

Maze = buildMaze\_Standard(Walls, S)

end = time.time()

print('Running Time for Maze with Standard Unions:', round((end - start) \* 1000, 6))

draw\_maze(Maze, maze\_rows, maze\_cols)

print('========================================================')

print()

SN = np.zeros(maze\_rows \* maze\_cols, dtype = np.int)-1 #DSF

WallsN = wall\_list(maze\_rows, maze\_cols)

print('Building Maze with Union by Size...')

startN = time.time()

MazeN = buildMaze\_Compression(WallsN, SN)

endN = time.time()

print('Running Time for Maze with Union by Size:', round((endN - startN) \* 1000, 6))

draw\_maze(MazeN, maze\_rows, maze\_cols)

print('========================================================')

“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”

