Armando Ortiz

April 16, 2019

CS 2302 Data Structures

Lab Report 6

UTEP

Instructor: Olac Fuentes

**Introduction**

The purpose of the program is to create a maze using union and union by size which performs path compression. The functions to draw the maze, create the walls, and basic disjoint set forest functions were provided from the classroom website. The code had to be modified using the union functions to create a single set, and remove walls to create paths that lead to each space in the maze.

**Proposed Solution Design and Implementation**

The main problem of this program was figuring out how to remove walls in order to create a correct maze. The original program included code that would remove random walls. The random integer was used to pick the location of a wall within the set of walls. Then the find function was used to find the roots of those walls, to make sure that they were in separate sets. If they were in separate sets, the two numbers were then united using the union function. With the wall being popped/removed. This makes it so every space is reachable, but still creating a path to the end of the maze. The sets were subtracted by one, and this was done in the loop until there was more than one set left. The maze created with the union by size function was done the same way, but just switching out the union function with the union by size function.

**Experimental Results**

**Output**

10 x 15 Maze by union

A screenshot of a social media post

Description automatically generated

10 x 15 Maze by union by size

A screenshot of a cell phone

Description automatically generated

**Runtimes**

|  |  |  |
| --- | --- | --- |
| **Maze Size** | **Maze by union** | **Maze by union by size** |
| **10 x 15** | 0.0011 | 0.0018 |
| **20 x 30** | 0.0222 | 0.0056 |
| **30 x 45** | 0.0945 | 0.0178 |
| **40 x 60** | 0.3167 | 0.0273 |
| **50 x 75** | 0.822 | 0.0565 |

**Conclusion**

Using disjoint set forests to create paths can be quite simple and quick. Using union by size is faster than simply using union, and can be figured out similarly to using a union function. It was surprising how counting the number of sets worked the same for each kind of union function. The most exciting portion was being able to see a drawing of the path, and how it creates a correct, solvable maze.

﻿**Appendix – Source Code**

"""

Author: Armando Ortiz

Course: CS 2302 Data Structures

Assignment: Lab 6

Instructor: Olac Fuentes

T.A.: Anindita Nath, Maliheh Zargaran

Last Modification: April 16, 2019

Purpose: The purpose of this program is to create two mazes

- one using union

- one using union by size which uses path compression

"""

import matplotlib.pyplot as plt

import numpy as np

import random

import time

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): #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

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

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

# Draws the maze

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

# Create Maze using Union

def mazeU(walls,D):

sets = len(D)

while sets>1:# While there is more than one set

w = random.randint(0,len(walls)-1)# picks a random wall in the set

c1 = find(D,walls[w][0])# Finds root of set

c2 = find(D,walls[w][1])# Finds root of Set

if c1 != c2:# If sets are different

union(D,c1,c2)# uses Union

sets-=1

walls.pop(w)# removes wall

# Create Maze using Union By Size

def mazeUBS(walls,D):

sets = len(D)

while sets >1: #While there is more than one set

w = random.randint(0,len(walls)-1)#npicks a random wall in the set

c1 = find(D,walls[w][0])#nFinds root of set

c2 = find(D,walls[w][1])#nFinds root of set

if c1!= c2:#nIf sets are different

union\_by\_size(D,c1,c2)# Uses union by compression

sets -=1

walls.pop(w)# removes wall

plt.close("all")

maze\_rows = 50

maze\_cols = 75

# Maze using regular union

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

D1 = DisjointSetForest(maze\_rows\*maze\_cols)

start = time.time()

mazeU(walls,D1)

end = time.time()

mazeURuntime = end - start

print("Maze by union runtime",round(mazeURuntime,4))

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

# Maze using union by size

walls2 = wall\_list(maze\_rows,maze\_cols)

D2 = DisjointSetForest(maze\_rows\*maze\_cols)

start = time.time()

mazeUBS(walls2,D2)

end = time.time()

mazeUBSRuntime = end - start

print("Maze by union by size runtime",round(mazeUBSRuntime,4))

draw\_maze(walls2,maze\_rows,maze\_cols)

A screenshot of a cell phone

Description automatically generated