Nancy Hernandez

Olac Fuentez

Lab 6 Report

**Introduction**

The purpose of this lab was to use disjoint set forests to build a maze that contains a collection of cells separated by walls in such a way that there is exactly one simple path separating two cells. As well as comparing the times for standard union and union by time.

**Proposed Solution**

In order to create the full maze with all adjacent cells separated by a wall I used the code Professor Fuentes provided. After, I knew I had to create a method that checked if two cells were in the same set. I did this by using the find method, which is in the disjointSetForest file the professor provided, and used it to find (S, a) and (S, b).

I then created the numSet method that would count how many sets there is. This was completed by creating a counter that is made up of a numpy array of zeros with a length of S. Then it goes into a for loop that checks if S[i] is less than zero and if it is then it increases the number of sets. Before I could get started with the rest, I added the methods find\_c and union\_by\_size that were also provided by the professor during lecture.

Finally, the last steps were to assign each cell to a different set in a disjoint set forest S. While the number of sets was more than one and the walls were not none, then I would select a random wall and if the cells were not in the same set then will use the union method to make the cells be in the same set. Then print the maze and record the time it took to complete this task.

These exact same steps were repeated for union by size except for swapping the union method with the union\_by\_size method.

**Setup**

To complete this lab I used an HP Pavilion x360 Convertible with a 2.71 GHz Intel® Core(TM) i5 processor.

**Results**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Time Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| **m** | **n** | **Standard Union Time (sec)** | **Union by Size Time (sec)** |
| 5 | 5 | 0.046862125396728516 | 0.04686427116394043 |
| 10 | 15 | 0.28121447563171387 | 0.17183399200439453 |
| 20 | 25 | 3.5698893070220947 | 0.6248514652252197 |

**Conclusion**

From this lab I learned that union by size is faster than the standard union. It does get significantly faster each time the number of m and n increase. I also learned how using disjoint set forests comes very handy when trying to create a maze of varying sizes. It allows to really have control of the maze and a better understanding of how making the mazes works.

**Appendix**

# Starting point for program to build and draw a maze

# Modify program using disjoint set forest to ensure there is exactly one

# simple path joining any two cells

# Programmed by Olac Fuentes

# Last modified March 28, 2019

import matplotlib.pyplot as plt

import numpy as np

import random

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

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)

print('removing wall ',walls[d])

walls.pop(d)

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

############################

#disjointSetForest file provided by professor

def find(S,i):

# Returns root of tree that i belongs to

if S[i]<0:

return i

return find(S,S[i])

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: # Do nothing if i and j belong to the same set

S[rj] = ri # Make j's root point to i's root

##########################

#code provide by professor in lecture

def find\_c(S, i):

if S[i] < 0:

return i

r = find\_c(S, S[i])

S[i] = r

return r

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

**Academic Dishonesty Statement**

I, Nancy Hernandez, was not involved in any copying from or providing information to another student, possessing unauthorized materials during a test, or falsifying data in laboratory reports. Neither did I participate in any type of collusion involving collaboration with another person to commit an academically dishonest act.