**Course Name:** 2302 **Author:** Olugbenga Iyiola **ID:** 80638542 **Instructor:** Olac Fuentes **TA:** Nath Anindita/ Malileh Zargaran **LAB #6 Report**

**Introduction**

The purpose of this lab is to use a disjoint set forest to build a maze which contains a collection of cells separated by walls in such a way that there is exactly one simple path (that is, a path that does not visit any cell more than once) separating any two cells.

A disjoint-set data structure is a [data structure](https://en.wikipedia.org/wiki/Data_structure) that tracks a [set](https://en.wikipedia.org/wiki/Set_(mathematics)) of elements [partitioned](https://en.wikipedia.org/wiki/Partition_of_a_set) into a number of [disjoint](https://en.wikipedia.org/wiki/Disjoint_sets) or non-overlapping subsets. It provides near-constant-time operations to add new sets, to merge existing sets, and to determine whether elements are in the same set.(Wikipedia 2019)

**Proposed Solution Design and Implementation**

To build the maze, the following procedures are followed;

* Let M be the number of rows and and N be the number of columns of the square maze.
* When all walls are present, each of the M ∗ N cells in the maze belongs to a different set.
* Thus you have M ∗ N sets in your disjoint set forest.
* When a wall is removed, if the cells that were separated by that wall belong to different sets, these sets are united.
* This process is repeated until all cells belong to a single set.
* At that point the maze is displayed.

The following pseudocode illustrates the process to build the maze:

*Create full maze with all adjacent cells are separated by a wall*

*Assign each cell to a different set in a disjoint set forest S*

*While S has more than one set*

*Select a random wall w =[c1,c2]*

*If cells c1 and c2 belong to different sets, remove w and join c1’s set and c2’s set*

*otherwise do nothing*

*Display maze*

**Experimental Result**

System Specification: HP Windows 10, 1.60GHZ Intel® Celeron® , 4.GB RAM, 64-bit operating system

The results of the various test cases using different sizes from the file for each of the algorithms are shown below:

**Creating Maze Using Standard Union**

|  |  |
| --- | --- |
| **Number of Cells(Rows \* Cols)** | **Runtime in nanoseconds** |
| **100** | **22171618** |
| **225** | **103527285** |
| **400** | **226172058** |
| **600** | **2076615030** |
| **1000** | **2701149734** |
|  |  |

O(n)

**Creating Maze Using Union-by-Size**

|  |  |
| --- | --- |
| **Number of Cells(Input)** | **Runtime in nanoseconds** |
| **100** | **55878980** |
| **225** | **219802515** |
| **400** | **766078616** |
| **600** | **2521313989** |
| **1000** | **3122238948** |
|  |  |

O(n)

**CONCLUSION**

In summary Standard Union uses Find () to determine the roots the cells belong to. If the roots are distinct, the cells are combined by attaching the root of one to the root of the other. If this is done naively, the height of the trees can grow O(n).

{\displaystyle O(n)}

Union by size always attaches the shorter tree to the root of the taller tree. Thus, the resulting tree is no taller than the originals unless they were of equal height, in which case the resulting tree is taller by one node. Without path compression, union by size, the height of trees can grow unchecked as O(n).(Wikipedia 2019)

**Appendix**

***Programmed by Olac Fuentes***

# Implementation of disjoint set forest

# Programmed by Olac Fuentes

# Last modified March 28, 2019

import matplotlib.pyplot as plt

import numpy as np

from scipy import interpolate

def DisjointSetForest(size):

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

def dsfToSetList(S):

#Returns aa list containing the sets encoded in 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 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

def draw\_dsf(S):

scale = 30

fig, ax = plt.subplots()

for i in range(len(S)):

if S[i]<0: # i is a root

ax.plot([i\*scale,i\*scale],[0,scale],linewidth=1,color='k')

ax.plot([i\*scale-1,i\*scale,i\*scale+1],[scale-2,scale,scale-2],linewidth=1,color='k')

else:

x = np.linspace(i\*scale,S[i]\*scale)

x0 = np.linspace(i\*scale,S[i]\*scale,num=5)

diff = np.abs(S[i]-i)

if diff == 1: #i and S[i] are neighbors; draw straight line

y0 = [0,0,0,0,0]

else: #i and S[i] are not neighbors; draw arc

y0 = [0,-6\*diff,-8\*diff,-6\*diff,0]

f = interpolate.interp1d(x0, y0, kind='cubic')

y = f(x)

ax.plot(x,y,linewidth=1,color='k')

ax.plot([x0[2]+2\*np.sign(i-S[i]),x0[2],x0[2]+2\*np.sign(i-S[i])],[y0[2]-1,y0[2],y0[2]+1],linewidth=1,color='k')

ax.text(i\*scale,0, str(i), size=20,ha="center", va="center",

bbox=dict(facecolor='w',boxstyle="circle"))

ax.axis('off')

ax.set\_aspect(1.0)

if \_\_name\_\_ == "\_\_main\_\_":

plt.close("all")

S = DisjointSetForest(8)

print(S)

draw\_dsf(S)

union(S,7,6)

print(S)

draw\_dsf(S)

union(S,0,2)

print(S)

draw\_dsf(S)

union(S,6,3)

print(S)

draw\_dsf(S)

union(S,5,2)

print(S)

draw\_dsf(S)

union(S,4,6)

print(S)

draw\_dsf(S)

print('Sets encoded by DSF:',dsfToSetList(S))

T = DisjointSetForest(8)

union(T, 7 , 0 )

union(T, 1 , 6 )

union(T, 3 , 0 )

union(T, 0 , 6 )

union(T, 3 , 4 )

union(T, 2 , 5 )

union(T, 6 , 0 )

union(T, 0 , 3 )

union(T, 4 , 2 )

union(T, 1 , 7 )

print(T)

draw\_dsf(T)

print('Sets encoded by DSF:',dsfToSetList(T))

U = DisjointSetForest(8)

for i in range(len(U)):

union(U, i , 0 )

print(U)

draw\_dsf(U)

Uc = DisjointSetForest(8)

for i in range(len(Uc)):

union\_c(Uc, i , 0 )

print(Uc)

draw\_dsf(Uc)

Us = DisjointSetForest(8)

for i in range(len(Us)):

union\_by\_size(Us, i , 0 )

print(Us)

draw\_dsf(Us)

**Wikipedia**

[**https://en.wikipedia.org/wiki/Sorting\_algorithm#Comparison\_of\_algorithms**](https://en.wikipedia.org/wiki/Sorting_algorithm#Comparison_of_algorithms)

**Academic Dishonesty**

This work was done by me without any act or practice of academic dishonesty

**SIGNATURE**

**OLUGBENGA IYIOLA(OT)**

**……………………………………………….**