CS 2302

Lab 6 Report

Author: David Ayala

ID: 80617838

Instructor: Olac Fuentes

TA: Anindita Nath

**Introduction**

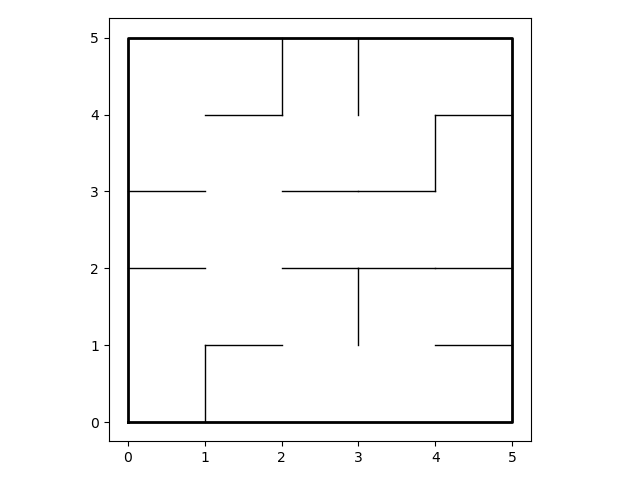
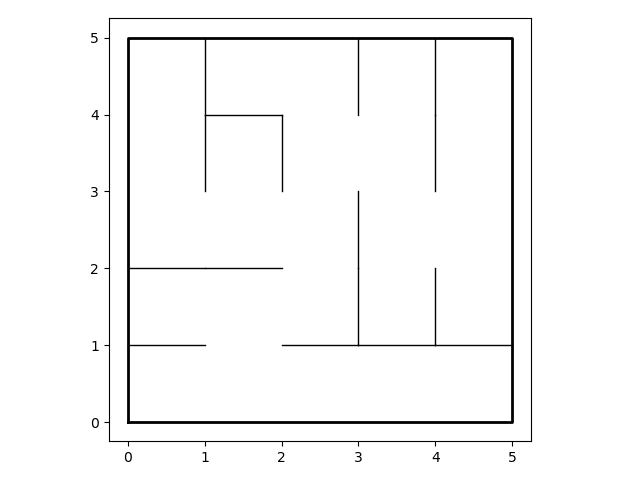
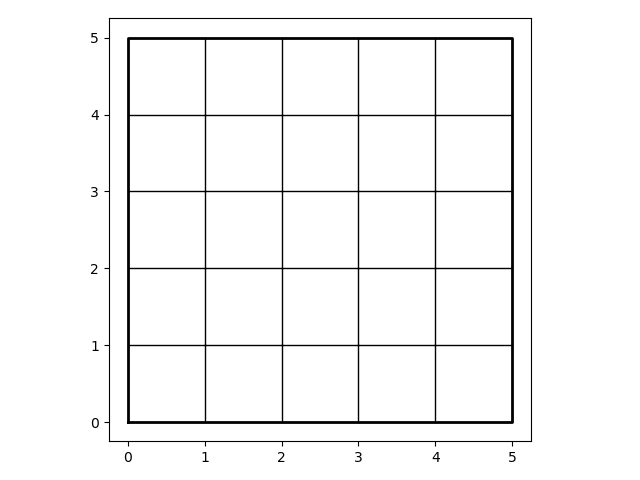
The purpose of this lab was to use a disjoint set forest to build a maze. Your maze should contain a collection of cells separated by walls in such a way that there is exactly one simple path (that is, a path that does not visit any cell more than once) separating any two cells. Perform a comparison of running times of the program using standard union and union by size with path compression for various maze sizes.

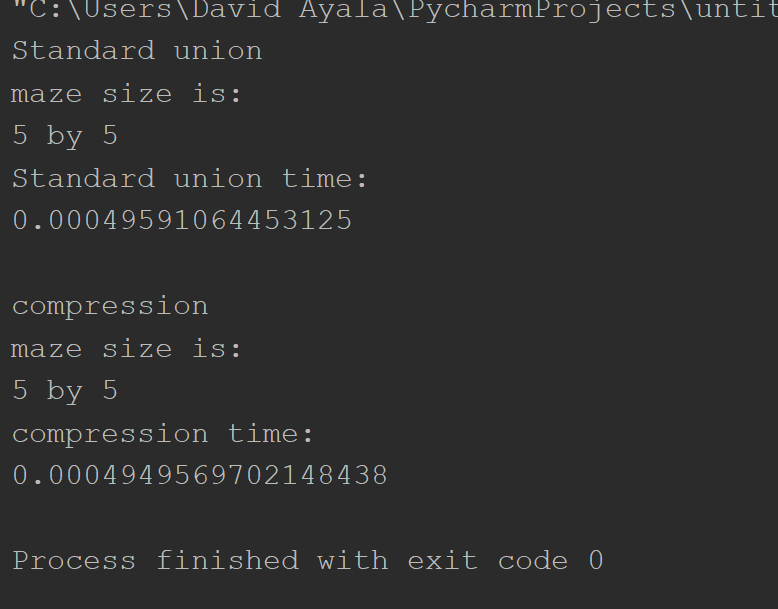
**Proposed Solution & Design Implementation**

Since Professor Fuentes had given us the code to make the maze and to able to draw it all I had to do was remove the walls by using standard union and union by size with path compression for various maze sizes. So first I need the number of sets once I had that, then I need an index to start from. I got that by getting the length of the of the walls, once I had that I then had to check if the current set was the same as the next set, if they weren’t then I would combine the two using standard union and I would delete the wall after that. I did the same thing for compression but instead of using standard union I would use compression.

**Experimental Results**

|  |  |  |
| --- | --- | --- |
| **Size** | **Standard Union** | **Compression** |
| **5 by 5 (25)** | 0.0004959 | 0.0004949 |
| **10 by 10 (100)** | 0.0039696 | 0.0059523 |
| **15 by 15 (225)** | 0.0258016 | 0.0287673 |
| **20 by 20 (400)** | 0.0838232 | 0.1155726 |

****

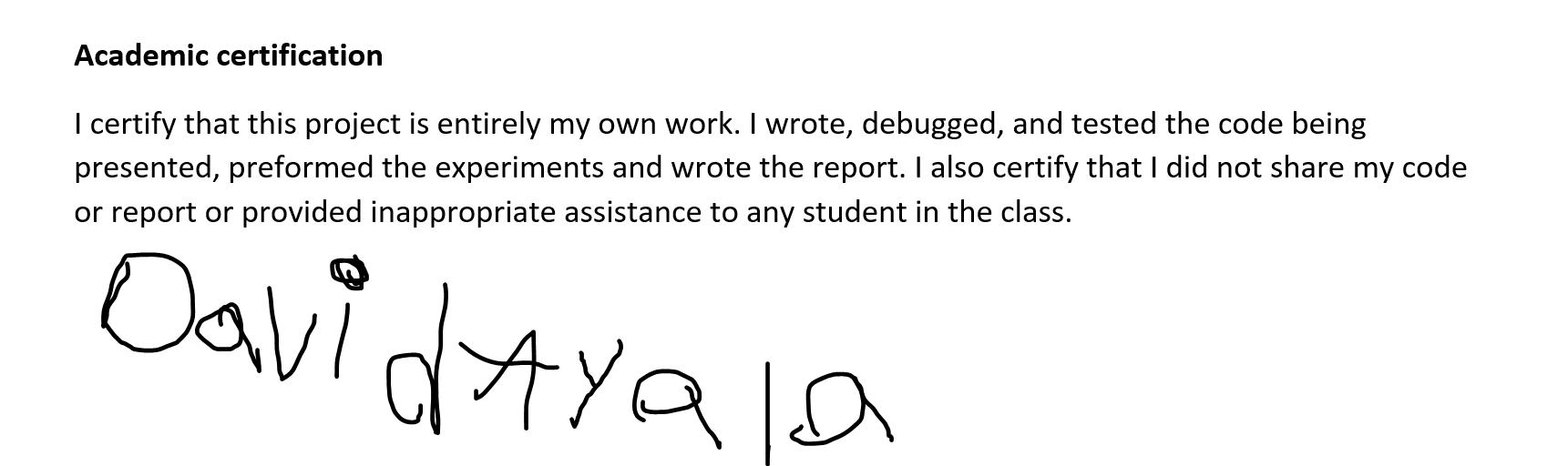


**Conclusion**

This lab allowed us to learn how to transverse a binary Search tree(BST) and how to obtain certain values in the tree. Basically, this whole lab was about BST and the different things we could do with them particularly the questions that he asked us to do(Display the binary search tree as a figure, make an Iterative version of the search operation, build a balanced binary search tree given a sorted list as input, Extracting the elements in a binary search tree into a sorted list and Printing the elements in a binary tree ordered by depth.). This lab also will give us some ideas on how to do the next lab.

**Appendix**

# Course:CS 2302 MW 1:30-2:50, Author:David Ayala  
# Assignment:Lab #6, Instructor: Olac Fuentes  
# Teaching Assistant: Maliheh Zargaran, Date of last Modification: 4/16/2019  
# Purpose of program: build and draw a maze  
# using disjoint set forest to ensure there is exactly one  
# simple path joining any two cells by using standard union and union by size with path  
# compression for various maze sizes  
  
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('on')  
 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 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 numberOfSets(S):  
 # Returns the number of sets(roots)  
 sets = 0  
 for i in range(len(S)):  
 if S[i] < 0:  
 sets += 1  
 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 MazeStanard(Sets,Walls):  
 while numberOfSets(Sets) > 1:  
 temp = random.randint(0, len(Walls)-1)  
 if find(Sets, Walls[temp][0]) != find(Sets, Walls[temp][1]):  
 union(Sets, Walls[temp][0], Walls[temp][1])  
 Walls.pop(temp)  
  
def MazeCompression(Sets,Walls):  
 while numberOfSets(Sets) > 1:  
 temp = random.randint(0, len(Walls)-1)  
 if find\_c(Sets, Walls[temp][0]) != find\_c(Sets, Walls[temp][1]):  
 union\_by\_size(Sets, Walls[temp][0], Walls[temp][1])  
 Walls.pop(temp)  
  
plt.close("all")  
  
notNegative = True  
  
Columns = 20  
Rows = 20  
  
if Columns < 1 or Rows < 1:  
 notNegative = False  
else:  
 Walls = wall\_list(Rows, Columns)  
 Sets = DisjointSetForest(Rows \* Columns)  
  
notNegative2 = True  
Columns2 = 20  
Rows2 = 20  
  
if Columns2 < 1 or Rows2 < 1:  
 notNegative2 = False  
else:  
 Walls2 = wall\_list(Rows2, Columns2)  
 Sets2 = DisjointSetForest(Rows2 \* Columns2)  
  
if notNegative is True:  
 print("Standard union")  
 print('maze size is:')  
 print(Rows, 'by', Columns)  
 draw\_maze(Walls, Rows, Columns)  
 start = time.time()  
 MazeStanard(Sets, Walls)  
 end = time.time()  
 draw\_maze(Walls, Rows, Columns)  
 print("Standard union time:")  
 print(end - start)  
 plt.show()  
else:  
 print("Standard union")  
 print('no Negative numbers or zeros please')  
  
print()  
  
if notNegative2 is True:  
 print("compression")  
 print('maze size is:')  
 print(Rows2, 'by', Columns2)  
 draw\_maze(Walls2, Rows2, Columns2)  
 start2 = time.time()  
 MazeCompression(Sets2, Walls2)  
 end2 = time.time()  
 draw\_maze(Walls2, Rows2, Columns2)  
 print("compression time:")  
 print(end2 - start2)  
 plt.show()  
else:  
 print("compression")  
 print('no Negative numbers or zeros please')



David Ayala

4/16/19