Lab 6

Fernando De Santiago

University of Texas at El Paso

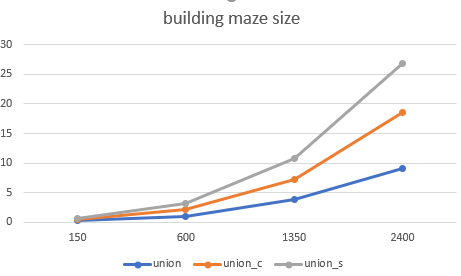
**Introduction**

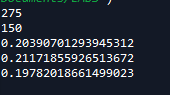
The purpose of this lab was to create a perfect maze. This just required a small change along with adding an extra method that does the same with a union by size using compression, as well as adding union by compression.

# Requirement 1

I started by creating a method called buildmaze1. Buildmaze1 started by having a while loop that called a method called NumSets would loop if the condition of Numsets is greater 1. I then used a value d that was given within the code which would get a random from 0 to the total number of walls. I then created a value that would hold the list value at d. From there I created two new values called C2 and C3 that hold the value of list[0] and list[1]. From there I have an if statement that checks if find(S,C2) is not equal to find(S,C3). If the statement is true then it will do union(S,C2,C3) and it will pop the wall at d till there is only one set. The other two were created the same way only that instead of find it’s find\_c and union is union\_c or union\_by\_size.

**Results**

**sizes are 150, 600, 1350 and 2400**

**275 is number of walls, 150 total cells, then time for union, union with compression, union by size with compression**

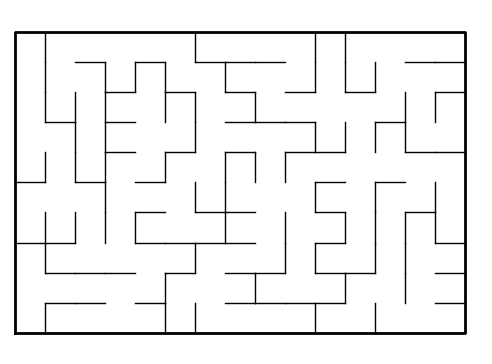


Figure Union

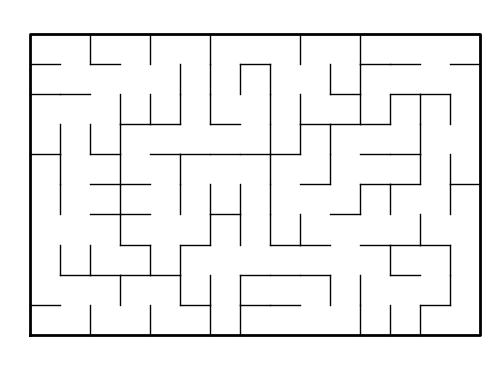


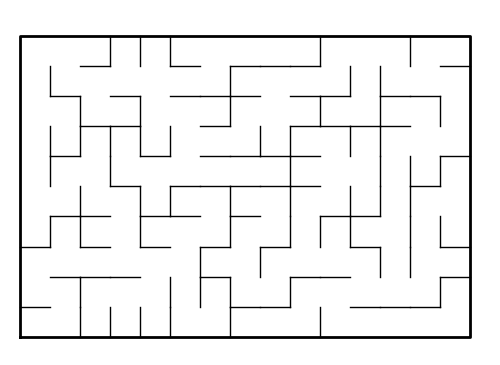
Figure Union with compression

Figure Union by size with compression

**Conclusion**

In conclusion I learned how to read the graph code and implemented the needed changes in the code to reach my goal. There were no problems met with this lab and in conclusion I learned how to create a maze with a given graph code.

**Appendix**

**# -\*- coding: utf-8 -\*-**

**"""**

**Created on Fri Apr 5 10:40:03 2019**

**@author: Fernando**

**"""**

**#CS2302**

**#Fernando De Santiago**

**#LAB5**

**#Olac Fuentes, Anindita Nath and Maliheh Zargaran**

**#last edited 4/12/19 20:56:00 PM**

**#Section M/W 10:30-11:50**

**#purpose: To create a correct maze**

**# Starting point for program to build and draw a maze**

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

**# simple path joiniung any two cells**

**# Programmed by Olac Fuentes**

**# Last modified March 28, 2019**

**import matplotlib.pyplot as plt**

**import numpy as np**

**import random**

**import time**

**def DisjointSetForest(size):#creates sets \* size and fills with -1**

**return np.zeros(size,dtype=np.int,order='C')-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 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**

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

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

**def NumSets(S):#counts all the sets in the set S**

**count=0**

**for i in range(len(S)):**

**if S[i]<0:**

**count+=1**

**return count**

**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# Make j's root point to i's root**

**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**

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

**size=maze\_rows\*maze\_cols**

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

**draw\_maze(walls,maze\_rows,maze\_cols,cell\_nums=True)**

**def buildMaze1():#Creates maze using regular find and union**

**while NumSets(S)>1:**

**d = random.randint(0,len(walls)-1)**

**C1=walls[d]**

**C2=C1[0]**

**C3=C1[1]**

**if find(S,C2)!=find(S,C3):**

**union(S,C2,C3)**

**walls.pop(d)**

**def buildMaze2():#creates maze using path compresssion**

**while NumSets(S)>1:**

**d = random.randint(0,len(walls)-1)**

**C1=walls[d]**

**C2=C1[0]**

**C3=C1[1]**

**if find\_c(S,C2)!=find\_c(S,C3):**

**union\_c(S,C2,C3)**

**walls.pop(d)**

**def buildMaze3():#creates maze using union\_by\_size**

**while NumSets(S)>1:**

**d = random.randint(0,len(walls)-1)**

**C1=walls[d]**

**C2=C1[0]**

**C3=C1[1]**

**if find\_c(S,C2)!=find\_c(S,C3):**

**union\_by\_size(S,C2,C3)**

**walls.pop(d)**

**#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)**

**print(len(walls))**

**print(NumSets(S))**

**start=time.time()**

**buildMaze1()**

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

**end=time.time()**

**print(end-start)**

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

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

**start=time.time()**

**buildMaze2()**

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

**end=time.time()**

**print(end-start)**

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

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

**start=time.time()**

**buildMaze3()**

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

**end=time.time()**

**print(end-start)**

**“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.”**

**Fernando De Santiago**