# Introduction

The lab we were given today required us to use examples we learned in class. Disjoint set forest allowed us to modify the program given to us and operate on the walls of the maze. The problem we had to solve was modify the wall removing program used so that a disjoin set forest would do it and we could have a solution

# Proposed solution

I first began with creating a DSF and making it the size of rows\*columns. Once that was done, I created a method called one\_set, which took a DSF as the input. In the one set method, I checked if there was more than one set In the DSF by iterating and counting all the -1’s and if the number was greater than 1 it would return false, or true if there was only one. Once that was done, I used a while loop that would iterate while there was more than just one set. Inside the while loop I created a random number the length of walls, and would grab 1 wall from the walls list, and hold it. Next I would check if the root of wall[0] was the same as wall[1] using the find method, if they were not, I would pop the random wall, and have a union with both the cells. Once both cells were full, it would repeat again until there was only 1 set. Print, and complete.

# Experimentation

I messed around with the sizes of the rows and columns and saw how much faster and slower it was when I changed the sizes. I also had an error when submitting the lab because I was checking if wall[0] and wall[1] were the same versus not being the same and was stuck in an infinite loop.

# Conclusion

This lab taught me how to use DSF and see an application for them. I also learned about bugs that could happen when you do not pay attention carefully to what you write. I made a mistake and it cost me 10 extra points because it was late.

# Appendix

1. # Starting point for program to build and draw a maze
2. # Modify program using disjoint set forest to ensure there is exactly one
3. # simple path joiniung any two cells
4. # Programmed by Olac Fuentes
5. # Last modified March 28, 2019
7. **import** matplotlib.pyplot as plt
8. **import** numpy as np
9. **import** random
11. **def** draw\_maze(walls,maze\_rows,maze\_cols,cell\_nums=False):
12. fig, ax = plt.subplots()
13. **for** w **in** walls:
14. **if** w[1]-w[0] ==1: #vertical wall
15. x0 = (w[1]%maze\_cols)
16. x1 = x0
17. y0 = (w[1]//maze\_cols)
18. y1 = y0+1
19. **else**:#horizontal wall
20. x0 = (w[0]%maze\_cols)
21. x1 = x0+1
22. y0 = (w[1]//maze\_cols)
23. y1 = y0
24. ax.plot([x0,x1],[y0,y1],linewidth=1,color='k')
25. sx = maze\_cols
26. sy = maze\_rows
27. ax.plot([0,0,sx,sx,0],[0,sy,sy,0,0],linewidth=2,color='k')
28. **if** cell\_nums:
29. **for** r **in** range(maze\_rows):
30. **for** c **in** range(maze\_cols):
31. cell = c + r\*maze\_cols
32. ax.text((c+.5),(r+.5), str(cell), size=10,
33. ha="center", va="center")
34. ax.axis('off')
35. ax.set\_aspect(1.0)
37. **def** wall\_list(maze\_rows, maze\_cols):
38. # Creates a list with all the walls in the maze
39. w =[]
40. **for** r **in** range(maze\_rows):
41. **for** c **in** range(maze\_cols):
42. cell = c + r\*maze\_cols
43. **if** c!=maze\_cols-1:
44. w.append([cell,cell+1])
45. **if** r!=maze\_rows-1:
46. w.append([cell,cell+maze\_cols])
47. **return** w
49. **from** scipy **import** interpolate
51. **def** DisjointSetForest(size):
52. **return** np.zeros(size,dtype=np.int)-1
54. **def** dsfToSetList(S):
55. #Returns aa list containing the sets encoded in S
56. sets = [ [] **for** i **in** range(len(S)) ]
57. **for** i **in** range(len(S)):
58. sets[find(S,i)].append(i)
59. sets = [x **for** x **in** sets **if** x != []]
60. **return** sets
62. **def** find(S,i):
63. # Returns root of tree that i belongs to
64. **if** S[i]<0:
65. **return** i
66. **return** find(S,S[i])
68. **def** find\_c(S,i): #Find with path compression
69. **if** S[i]<0:
70. **return** i
71. r = find\_c(S,S[i])
72. S[i] = r
73. **return** r
75. **def** union(S,i,j):
76. # Joins i's tree and j's tree, if they are different
77. ri = find(S,i)
78. rj = find(S,j)
79. **if** ri!=rj:
80. S[rj] = ri
82. **def** union\_c(S,i,j):
83. # Joins i's tree and j's tree, if they are different
84. # Uses path compression
85. ri = find\_c(S,i)
86. rj = find\_c(S,j)
87. **if** ri!=rj:
88. S[rj] = ri
90. **def** union\_by\_size(S,i,j):
91. # if i is a root, S[i] = -number of elements in tree (set)
92. # Makes root of smaller tree point to root of larger tree
93. # Uses path compression
94. ri = find\_c(S,i)
95. rj = find\_c(S,j)
96. **if** ri!=rj:
97. **if** S[ri]>S[rj]: # j's tree is larger
98. S[rj] += S[ri]
99. S[ri] = rj
100. **else**:
101. S[ri] += S[rj]
102. S[rj] = ri

105. **def** draw\_dsf(S):
106. scale = 30
107. fig, ax = plt.subplots()
108. **for** i **in** range(len(S)):
109. **if** S[i]<0: # i is a root
110. ax.plot([i\*scale,i\*scale],[0,scale],linewidth=1,color='k')
111. ax.plot([i\*scale-1,i\*scale,i\*scale+1],[scale-2,scale,scale-2],linewidth=1,color='k')
112. **else**:
113. x = np.linspace(i\*scale,S[i]\*scale)
114. x0 = np.linspace(i\*scale,S[i]\*scale,num=5)
115. diff = np.abs(S[i]-i)
116. **if** diff == 1: #i and S[i] are neighbors; draw straight line
117. y0 = [0,0,0,0,0]
118. **else**:      #i and S[i] are not neighbors; draw arc
119. y0 = [0,-6\*diff,-8\*diff,-6\*diff,0]
120. f = interpolate.interp1d(x0, y0, kind='cubic')
121. y = f(x)
122. ax.plot(x,y,linewidth=1,color='k')
123. 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')
124. ax.text(i\*scale,0, str(i), size=20,ha="center", va="center",
125. bbox=dict(facecolor='w',boxstyle="circle"))
126. ax.axis('off')
127. ax.set\_aspect(1.0)
129. #this method checks if s only has one set
130. **def** one\_set(S):
131. count = 0
132. **for** i **in** S:#everytime we find a -1, that means we have a single set, if we have more than one, it is false.
133. **if** i == -1:
134. count += 1
135. **if**(count > 1):
136. **return** False
137. **return** True
139. #######################################
141. plt.close("all")
142. maze\_rows = 10
143. maze\_cols = 15
145. walls = wall\_list(maze\_rows,maze\_cols)
146. S = DisjointSetForest(maze\_rows\*maze\_cols)
147. draw\_maze(walls,maze\_rows,maze\_cols,cell\_nums=True)
148. d = random.randint(0, len(walls)-1)
149. #iterates through the Forest and stops once we have only 1 set in the DSF
150. **while**(one\_set(S) != True):
151. d = random.randint(0,len(walls)-1)
152. hold = walls[d]
153. **if** find(S, hold[0]) != find(S, hold[1]):
154. walls.pop(d)
155. union\_c(S, hold[0], hold[1])
156. draw\_maze(walls,maze\_rows,maze\_cols)

# Running Time

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. C.L