**Lab #6**

CS 2302

Ana Luisa Mata Sánchez

# Introduction

Use a disjoint set forest to build a maze. The 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.

To build a maze, let M be the number of rows and N be the number of columns of your 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 you remove a wall, if the cells that were separated by that wall belonged to different sets, you must unite these sets. This process is repeated until all cells belong to a single set; at that point you display the maze.

# Proposed solution design and implementation

**Module 1 – create\_standard\_dsf\_maze**

This method creates a disjoint set forest using the standard union and find functions. It checks that the dsf contains only one set, if it does it stops uniting cells and removing walls. If not, it will pick a random wall, see if both numbers that touch the wall are in the same set. If it is not it will make one of them point to the other and remove said wall.

**Module 2 – create\_compressed\_dsf\_maze**

This method creates a disjoint set forest using the compressed union and union by size functions. It checks that the dsf contains only one set, if it does it stops uniting cells and removing walls. If not, it will pick a random wall, see if both cells that touch the wall are in the same set and compress them at the same time. If it is not it will make the smaller set point to the bigger set and remove the wall.

# Experimental results

|  |  |  |
| --- | --- | --- |
| **Maze Size (RowxCol)** | **Non-compressed output** | **Compressed Output** |
| 10x5 | ######## Maze using standard find and union ########  Time it took to create the maze: 0.003989696502685547  Maze row size: 10  Maze column size: 5 | ######## Maze using compressed find and union by size ########  Time it took to create the maze: 0.001993894577026367  Maze row size: 10  Maze column size: 5 |
| 10x10 | ######## Maze using standard find and union ########  Time it took to create the maze: 0.02396869659423828  Maze row size: 10  Maze column size: 10 | ######## Maze using compressed find and union by size ########  Time it took to create the maze: 0.01695418357849121  Maze row size: 10  Maze column size: 10 |
| 10x15 | ######## Maze using standard find and union ########  Time it took to create the maze: 0.13762712478637695  Maze row size: 10  Maze column size: 15 | ######## Maze using compressed find and union by size ########  Time it took to create the maze: 0.04191398620605469  Maze row size: 10  Maze column size: 15 |
| 15x15 | ######## Maze using standard find and union ########  Time it took to create the maze: 0.21445918083190918  Maze row size: 15  Maze column size: 15 | ######## Maze using compressed find and union by size ########  Time it took to create the maze: 0.08380842208862305  Maze row size: 15  Maze column size: 15 |
| 15x20 | ######## Maze using standard find and union ########  Time it took to create the maze: 0.48769330978393555  Maze row size: 15  Maze column size: 20 | ######## Maze using compressed find and union by size ########  Time it took to create the maze: 0.19448113441467285  Maze row size: 15  Maze column size: 20 |
| 20x20 | ######## Maze using standard find and union ########  Time it took to create the maze: 1.4650804996490479  Maze row size: 20  Maze column size: 20 | ######## Maze using compressed find and union by size ########  Time it took to create the maze: 0.30321550369262695  Maze row size: 20  Maze column size: 20 |
| 30x30 | ######## Maze using standard find and union ########  Time it took to create the maze: 15.24419617652893  Maze row size: 30  Maze column size: 30 | ######## Maze using compressed find and union by size ########  Time it took to create the maze: 1.6167094707489014  Maze row size: 30  Maze column size: 30 |

|  |  |  |
| --- | --- | --- |
| Maze Size | Non-compressed Running Time(s) | Compressed Running Time(s) |
| 5x5 | 0.000997782 | 0.001028061 |
| 5x10 | 0.012964964 | 0.002992153 |
| 10x10 | 0.021942139 | 0.015949965 |
| 10x15 | 0.113695383 | 0.029919147 |
| 15x15 | 0.658238411 | 0.092720985 |
| 15x20 | 0.497653961 | 0.146610737 |
| 20x20 | 2.108328104 | 0.286201715 |
| 20x25 | 3.365030289 | 0.330118418 |
| 25x25 | 10.12691498 | 0.715081453 |
| 25x30 | 13.61557746 | 0.994366884 |

# Conclusion

The compressed dsf functions gave quicker times when building the maze, when comparing them to the standard union and find functions. This can clearly be seen in the last graph where the vertical axis is seconds and the horizontal is maze sizes.

**“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.”**

-Ana Luisa Mata Sánchez

# Appendix

|  |
| --- |
| # Author: Ana Luisa Mata Sanchez |
|  | # Course: CS2302 |
|  | # Assignment: Lab #6 |
|  | # Instructor: Olac Fuentes |
|  | # Description: Program to draw mazes using dsf |
|  | # T.A.: Anindita Nath, Maliheh Zargaran |
|  | # Last modified: 04/11/2019 |
|  | # Purpose: To compare time differences between compressed and non-compressed disjoint set forest |
|  |  |
|  | import matplotlib.pyplot as plt |
|  | import numpy as np |
|  | import random |
|  | import time |
|  |  |
|  | ###################################### Code provided and written by Dr. Fuentes ###################################### |
|  | 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\_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 |
|  |  |
|  | ###################################### MY CODE ###################################### |
|  |  |
|  | #Method that removes walls and creates the dsf using the standard union and find methods |
|  | def create\_standard\_dsf\_maze(S,walls): |
|  | #If there is only one set it means that all cells are reacheable from any cell |
|  | while len(dsfToSetList(S))>1: |
|  | #Finds a wall to remove |
|  | d = random.randint(0,len(walls)-1) |
|  | #If the elements that share a wall are not in the same set, remove it |
|  | if find(S,walls[d][0]) != find(S,walls[d][1]): |
|  | #make the elements belong to the same set |
|  | union(S,walls[d][0],walls[d][1]) |
|  | #remove the wall |
|  | walls.pop(d) |
|  |  |
|  | #Method that removes walls and creates the dsf using the union by size and compressed find methods |
|  | def create\_compressed\_dsf\_maze(SC,wallsC): |
|  | #If there is only one set it means that all cells are reacheable from any cell |
|  | while len(dsfToSetList(SC))>1: |
|  | #Finds a wall to remove |
|  | dC = random.randint(0,len(wallsC)-1) |
|  | #If the elements that share a wall are not in the same set, remove it |
|  | if find\_c(SC,wallsC[dC][0]) != find\_c(SC,wallsC[dC][1]): |
|  | #make the elements belong to the same set |
|  | union\_by\_size(SC,wallsC[dC][0],wallsC[dC][1]) |
|  | #remove the wall |
|  | wallsC.pop(dC) |
|  |  |
|  | plt.close("all") |
|  | maze\_rows = 10 |
|  | maze\_cols = 15 |
|  |  |
|  | #wall list & dsf for standard method |
|  | walls = wall\_list(maze\_rows,maze\_cols) |
|  | S = DisjointSetForest(maze\_rows\*maze\_cols) |
|  |  |
|  | #wall list & dsf for compressed method |
|  | wallsC = wall\_list(maze\_rows,maze\_cols) |
|  | SC = DisjointSetForest(maze\_rows\*maze\_cols) |
|  |  |
|  | #draw initial maze |
|  | draw\_maze(walls,maze\_rows,maze\_cols,cell\_nums=True) |
|  |  |
|  | print("######## Maze using standard find and union ########\n") |
|  |  |
|  | iStandardMazeT = time.time() |
|  | create\_standard\_dsf\_maze(S,walls) |
|  | fStandardMazeT = time.time() |
|  | draw\_maze(walls,maze\_rows,maze\_cols) |
|  |  |
|  | print("Time it took to create the maze:", fStandardMazeT-iStandardMazeT) |
|  | print("Maze row size:", maze\_rows) |
|  | print("Maze column size:", maze\_cols) |
|  |  |
|  | print("\n######## Maze using compressed find and union by size ########\n") |
|  |  |
|  | iCompressedMazeT = time.time() |
|  | create\_compressed\_dsf\_maze(SC,wallsC) |
|  | fCompressedMazeT = time.time() |
|  | draw\_maze(wallsC,maze\_rows,maze\_cols) |
|  |  |
|  | print("Time it took to create the maze:", fCompressedMazeT-iCompressedMazeT) |
|  | print("Maze row size:", maze\_rows) |
|  | print("Maze column size:", maze\_cols) |