The purpose of Lab 7 was to utilize what was done in lab 6 to create a maze but this time ask the user how many walls they would like to remove, instead of having the predetermined number of walls to be removed from each cell to create a unique path. That means that depending on the number of walls the user selected to be removed, there was the option that there is one unique path, no path, or even multiple paths. Based on this action, an adjacency list was to be created in representation of the maze. With the adjacency list created, a path from start to finish could be created using one of three options, breadth-first search, depth-first search or depth-first search using recursion. These searches would also be represented in the maze to demonstrate a path if one was available.

Since lab 6 was used to create a maze, the same code was used but was heavily modified. Initially, I decided to add a while true loop where the user is asked how many walls they would like to remove. This is used in case the number of walls to be removed exceed the actual number of walls that exist and if such is the case it will let the user know and ask the same question again. Afterwards the maze is created by using the same code done in lab 6 but modifying it to accomplish the task of removing the number of walls the user wanted. The maze is worked on in the function called Union\_Maze and receives three variables, M, the disjoint set forest, w, the walls in the maze, and m, the number of walls the user wants removed. In this function the adjacency list is created and is done in a different function called, get\_adj\_list, which sends an empty list of lists of the size of the maze rows by the maze columns as well as the list of which walls were popped. This returns the adjacency list which is completed by a for loop in the range of the length of list popped, p. temp0 and temp1 are used to store the individual values in the list of lists in popped and then added to A, which is the empty list of lists, and then returning A to the Union\_Maze function. Afterwards, the adjacency list is returned to what would be G in what is the main in the code. G is then sent to breadth\_first\_search, depth\_first\_search, and depth\_first\_search\_recursion.

The breadth\_first\_seach function receives G and pretty much what it does is that it traverses the maze using the breadth first search method. This is started by creating a list called visited that hold false and is of the same length as the adjacency list. Then, prev is created which is a list that holds -1 and is of the same size as the visited list. The Q variable is then used as a queue which starts empty and is used by importing the queue library. Since the starting point is 0 for this lab, 0 is added to the queue and the first element in visited is changed to true. The rest is done in a while loop that will continue until the queue is empty. Variable v is created to represent the vertex obtained from the queue. Then a for loop is used to traverse all items in the list of lists G and if visited at the specified location is false then it is changed to true, prev at the specified location has the vertex added from the queue and then the item used to specify the location is added to the queue. This is done until all the list is traversed and when the queue is empty then the prev list is returned as the breadth first search sequence. Next is the depth\_first\_search function that works very similar to breadth fist search but like its name, it uses depth first search to traverse the list. The code is the same with the difference that instead of using a queue, a list is used to traverse the adjacency list. Just like the function breadth\_first\_search, depth\_first\_search also returns the list prev. Now with depth\_fist\_search\_recursion, as the name implies, it uses depth first search to traverse the list but calls itself to generate the list prev. Also, it receives integer 0 which is used as the source and this is done because the source will change with every call. Since vertex 0 is the start of the maze, it is why that integer is used when calling this function from the main. The list visited and prev are created in the main since with recursion it would be needed to also use them in each recursion call but python allows for this so that is why this method is used. The similar work is done as in the other searches, visited at the source location is changed to True and then a for loop is used to get the items located in the adjacency list specified by the source and are called t. Afterwards, the if statement is used to make sure that visited at the location specified by t is not true if so, the source, which is the vertex, is added to prev at location t and depth\_first\_search\_recursion is called with G, the adjacency list and t, the item which was at G in the for loop generated previously. When the list is traversed, prev is then returned to main.

Now a path is generated if at least one path exists in the maze. The user is asked to select what method to use from breadth first search, depth first search and depth first search using recursion to generate the path. This is timed with the time library imported to compare the differences between them. What follows are the results of the maze and its path generated by each search.

////////////////////////////////////////////////////////////////////////////////////////////////////

**How many walls do you want to remove: 50**

**A path from source to destination is not guaranteed to exist (m < n-1).**

**Running time for this maze is with maze size: 10 rows, 10 columns, time in milliseconds: 2.0029296875**

**Breadth First Search took: 0.0 milliseconds**

**Breadth First Search: [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]**

**Depth First Search took: 0.0 milliseconds**

**Depth First Search: [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]**

**Depth First Search using recursion took: 0.0 milliseconds**

**Depth First Search using recursion: [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]**

**How do you want to generate a path if at least one is available, select 1 for Breadth First Search, 2 for Depth First Search, or 3 for Depth First Search using recursion: 1**

**It took 0.0 milliseconds to generate a path, (if one exists), from start to goal.**

////////////////////////////////////////////////////////////////////////////////////////////////////

**How many walls do you want to remove: 50**

**A path from source to destination is not guaranteed to exist (m < n-1).**

**Running time for this maze is with maze size: 10 rows, 10 columns, time in milliseconds: 0.996826171875**

**Breadth First Search took: 0.0 milliseconds**

**Breadth First Search: [-1, 0, -1, -1, -1, -1, -1, -1, -1, -1, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]**

**Depth First Search took: 0.0 milliseconds**

**Depth First Search: [-1, 0, -1, -1, -1, -1, -1, -1, -1, -1, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]**

**Depth First Search using recursion took: 0.0 milliseconds**

**Depth First Search using recursion: [-1, 0, -1, -1, -1, -1, -1, -1, -1, -1, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]**

**How do you want to generate a path if at least one is available, select 1 for Breadth First Search, 2 for Depth First Search, or 3 for Depth First Search using recursion: 2**

**It took 0.0 milliseconds to generate a path, (if one exists), from start to goal.**

////////////////////////////////////////////////////////////////////////////////////////////////////

**How many walls do you want to remove: 50**

**A path from source to destination is not guaranteed to exist (m < n-1).**

**Running time for this maze is with maze size: 10 rows, 10 columns, time in milliseconds: 1.989501953125**

**Breadth First Search took: 0.0 milliseconds**

**Breadth First Search: [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]**

**Depth First Search took: 0.0 milliseconds**

**Depth First Search: [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]**

**Depth First Search using recursion took: 0.0 milliseconds**

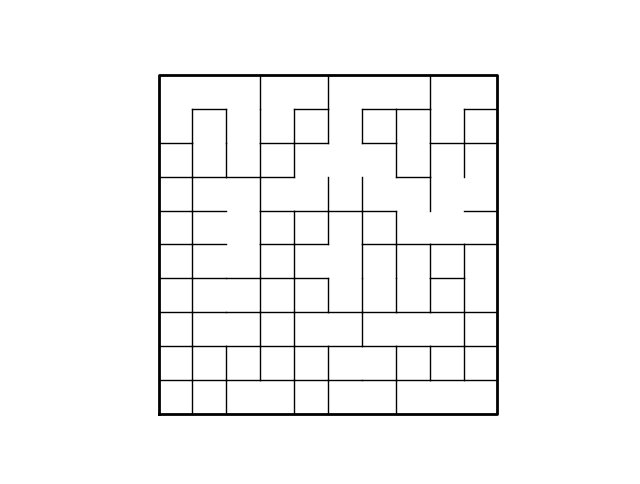
**Depth First Search using recursion: [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]**

**How do you want to generate a path if at least one is available, select 1 for Breadth First Search, 2 for Depth First Search, or 3 for Depth First Search using recursion: 3**

**It took 0.0 milliseconds to generate a path, (if one exists), from start to goal.**

////////////////////////////////////////////////////////////////////////////////////////////////////

Only 50 walls were removed in each iteration and as it can be seen, no path exists for any.



////////////////////////////////////////////////////////////////////////////////////////////////////

**How many walls do you want to remove: 99**

**There is a unique path from source to destination (m = n-1).**

**Running time for this maze is with maze size: 10 rows, 10 columns, time in milliseconds: 4.978515625**

**Breadth First Search took: 0.0 milliseconds**

**Breadth First Search: [-1, 0, 1, 2, 3, 4, 5, 6, 18, 8, 0, 1, 13, 3, 24, 25, 17, 7, 17, 9, 10, 20, 21, 13, 25, 26, 16, 28, 18, 19, 40, 21, 22, 23, 35, 45, 37, 47, 28, 38, 41, 31, 41, 42, 43, 55, 45, 57, 38, 48, 60, 41, 42, 43, 44, 54, 46, 67, 59, 49, 70, 51, 61, 64, 54, 64, 56, 77, 78, 68, 71, 61, 82, 63, 64, 74, 66, 76, 88, 78, 81, 82, 83, 73, 83, 84, 76, 86, 87, 88, 80, 81, 93, 83, 84, 96, 97, 87, 99, 89]**

**Depth First Search took: 0.0 milliseconds**

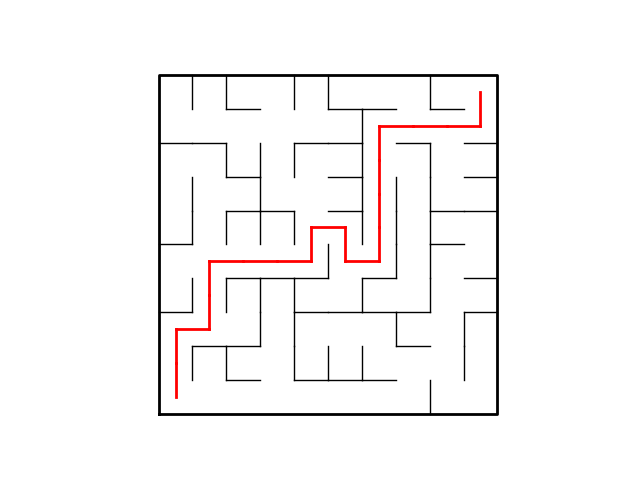
**Depth First Search: [-1, 0, 1, 2, 3, 4, 5, 6, 18, 8, 0, 1, 13, 3, 24, 25, 17, 7, 17, 9, 10, 20, 21, 13, 25, 26, 16, 28, 18, 19, 40, 21, 22, 23, 35, 45, 37, 47, 28, 38, 41, 31, 41, 42, 43, 55, 45, 57, 38, 48, 60, 41, 42, 43, 44, 54, 46, 67, 59, 49, 70, 51, 61, 64, 54, 64, 56, 77, 78, 68, 71, 61, 82, 63, 64, 74, 66, 76, 88, 78, 81, 82, 83, 73, 83, 84, 76, 86, 87, 88, 80, 81, 93, 83, 84, 96, 97, 87, 99, 89]**

**Depth First Search using recursion took: 0.000997304916381836 milliseconds**

**Depth First Search using recursion: [-1, 0, 1, 2, 3, 4, 5, 6, 18, 8, 0, 1, 13, 3, 24, 25, 17, 7, 17, 9, 10, 20, 21, 13, 25, 26, 16, 28, 18, 19, 40, 21, 22, 23, 35, 45, 37, 47, 28, 38, 41, 31, 41, 42, 43, 55, 45, 57, 38, 48, 60, 41, 42, 43, 44, 54, 46, 67, 59, 49, 70, 51, 61, 64, 54, 64, 56, 77, 78, 68, 71, 61, 82, 63, 64, 74, 66, 76, 88, 78, 81, 82, 83, 73, 83, 84, 76, 86, 87, 88, 80, 81, 93, 83, 84, 96, 97, 87, 99, 89]**

**How do you want to generate a path if at least one is available, select 1 for Breadth First Search, 2 for Depth First Search, or 3 for Depth First Search using recursion: 1**

**It took 17.93505859375 milliseconds to generate a path, (if one exists), from start to goal.**



//////////////////////////////////////////////////////////////////////////////////////////////////

**How many walls do you want to remove: 99**

**There is a unique path from source to destination (m = n-1).**

**Running time for this maze is with maze size: 10 rows, 10 columns, time in milliseconds: 5.9853515625**

**Breadth First Search took: 0.0 milliseconds**

**Breadth First Search: [-1, 11, 3, 13, 3, 15, 5, 6, 7, 8, 0, 21, 22, 12, 15, 16, 26, 27, 28, 9, 10, 20, 21, 22, 23, 26, 36, 26, 27, 28, 31, 21, 42, 23, 33, 25, 46, 27, 28, 29, 30, 31, 43, 33, 43, 44, 56, 57, 47, 48, 60, 50, 51, 43, 53, 45, 55, 56, 48, 69, 70, 51, 52, 64, 65, 66, 56, 66, 67, 68, 71, 72, 82, 72, 75, 76, 86, 78, 79, 69, 90, 82, 92, 84, 74, 75, 96, 77, 87, 79, 91, 81, 93, 94, 84, 96, 97, 87, 88, 89]**

**Depth First Search took: 0.0 milliseconds**

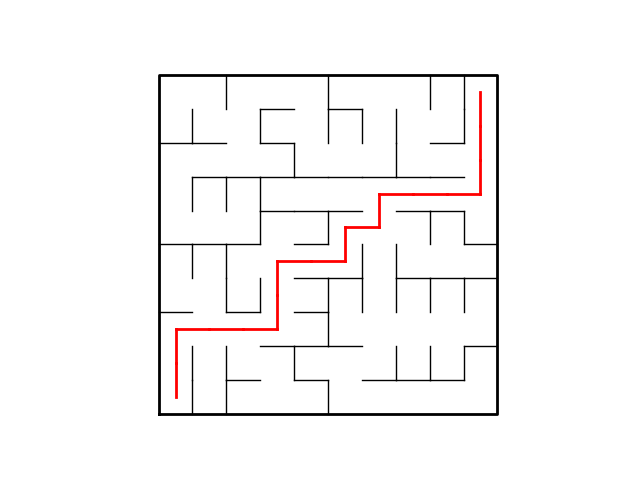
**Depth First Search: [-1, 11, 3, 13, 3, 15, 5, 6, 7, 8, 0, 21, 22, 12, 15, 16, 26, 27, 28, 9, 10, 20, 21, 22, 23, 26, 36, 26, 27, 28, 31, 21, 42, 23, 33, 25, 46, 27, 28, 29, 30, 31, 43, 33, 43, 44, 56, 57, 47, 48, 60, 50, 51, 43, 53, 45, 55, 56, 48, 69, 70, 51, 52, 64, 65, 66, 56, 66, 67, 68, 71, 72, 82, 72, 75, 76, 86, 78, 79, 69, 90, 82, 92, 84, 74, 75, 96, 77, 87, 79, 91, 81, 93, 94, 84, 96, 97, 87, 88, 89]**

**Depth First Search using recursion took: 0.0 milliseconds**

**Depth First Search using recursion: [-1, 11, 3, 13, 3, 15, 5, 6, 7, 8, 0, 21, 22, 12, 15, 16, 26, 27, 28, 9, 10, 20, 21, 22, 23, 26, 36, 26, 27, 28, 31, 21, 42, 23, 33, 25, 46, 27, 28, 29, 30, 31, 43, 33, 43, 44, 56, 57, 47, 48, 60, 50, 51, 43, 53, 45, 55, 56, 48, 69, 70, 51, 52, 64, 65, 66, 56, 66, 67, 68, 71, 72, 82, 72, 75, 76, 86, 78, 79, 69, 90, 82, 92, 84, 74, 75, 96, 77, 87, 79, 91, 81, 93, 94, 84, 96, 97, 87, 88, 89]**

**How do you want to generate a path if at least one is available, select 1 for Breadth First Search, 2 for Depth First Search, or 3 for Depth First Search using recursion: 2**

**It took 15.959228515625 milliseconds to generate a path, (if one exists), from start to goal**

.

\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

**How many walls do you want to remove: 99**

**There is a unique path from source to destination (m = n-1).**

**Running time for this maze is with maze size: 10 rows, 10 columns, time in milliseconds: 3.615234375**

**Breadth First Search took: 0.993896484375 milliseconds**

**Breadth First Search: [-1, 0, 1, 2, 14, 15, 7, 17, 7, 19, 0, 12, 22, 12, 24, 14, 15, 27, 28, 29, 10, 20, 21, 13, 23, 35, 25, 26, 27, 28, 40, 32, 33, 23, 24, 34, 46, 27, 37, 29, 41, 31, 32, 44, 54, 35, 45, 46, 38, 48, 40, 41, 42, 43, 55, 45, 57, 58, 48, 49, 61, 62, 63, 53, 65, 55, 76, 57, 58, 59, 80, 70, 71, 72, 75, 65, 75, 76, 77, 69, 81, 91, 81, 93, 83, 95, 87, 77, 78, 99, 91, 92, 93, 94, 95, 96, 86, 87, 88, 98]**

**Depth First Search took: 0.0 milliseconds**

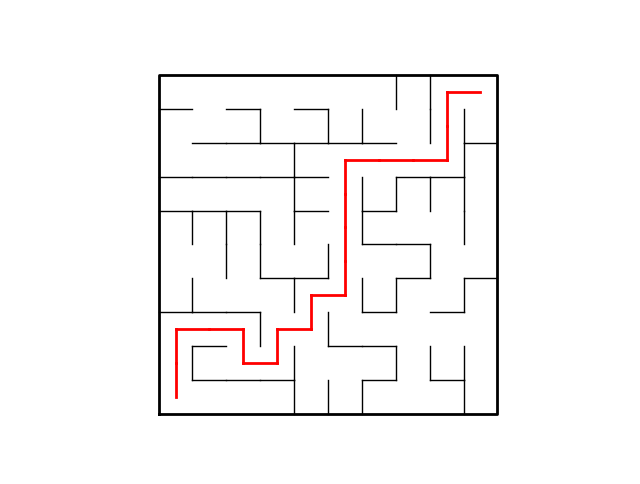
**Depth First Search: [-1, 0, 1, 2, 14, 15, 7, 17, 7, 19, 0, 12, 22, 12, 24, 14, 15, 27, 28, 29, 10, 20, 21, 13, 23, 35, 25, 26, 27, 28, 40, 32, 33, 23, 24, 34, 46, 27, 37, 29, 41, 31, 32, 44, 54, 35, 45, 46, 38, 48, 40, 41, 42, 43, 55, 45, 57, 58, 48, 49, 61, 62, 63, 53, 65, 55, 76, 57, 58, 59, 80, 70, 71, 72, 75, 65, 75, 76, 77, 69, 81, 91, 81, 93, 83, 95, 87, 77, 78, 99, 91, 92, 93, 94, 95, 96, 86, 87, 88, 98]**

**Depth First Search using recursion took: 0.0 milliseconds**

**Depth First Search using recursion: [-1, 0, 1, 2, 14, 15, 7, 17, 7, 19, 0, 12, 22, 12, 24, 14, 15, 27, 28, 29, 10, 20, 21, 13, 23, 35, 25, 26, 27, 28, 40, 32, 33, 23, 24, 34, 46, 27, 37, 29, 41, 31, 32, 44, 54, 35, 45, 46, 38, 48, 40, 41, 42, 43, 55, 45, 57, 58, 48, 49, 61, 62, 63, 53, 65, 55, 76, 57, 58, 59, 80, 70, 71, 72, 75, 65, 75, 76, 77, 69, 81, 91, 81, 93, 83, 95, 87, 77, 78, 99, 91, 92, 93, 94, 95, 96, 86, 87, 88, 98]**

**How do you want to generate a path if at least one is available, select 1 for Breadth First Search, 2 for Depth First Search, or 3 for Depth First Search using recursion: 3**

**It took 17.954345703125 milliseconds to generate a path, (if one exists), from start to goal.**



//////////////////////////////////////////////////////////////////////////////////////////////

As one can see, 99 walls were removed from this maze which generates only one unique path. In this case a path is drawn, and the comparisons show that depth fist search is the fastest from the other two. Next the number of walls removed will be greater generating a path and while another path may also exist.

**How many walls do you want to remove: 120**

**There is at least one path from source to destination (m > n-1).**

**Running time for this maze is with maze size: 10 rows, 10 columns, time in milliseconds: 6.035888671875**

**Breadth First Search took: 0.0 milliseconds**

**Breadth First Search: [-1, 11, 1, 2, 3, 4, 5, 6, 7, 8, 0, 10, 11, 12, 13, 16, 17, 7, 8, 18, 10, 20, 12, 13, 25, 35, 25, 17, 18, 28, 20, 41, 22, 32, 33, 34, 46, 27, 28, 49, 30, 51, 32, 33, 43, 44, 45, 46, 47, 59, 40, 50, 42, 63, 53, 54, 55, 47, 48, 58, 61, 62, 52, 62, 63, 64, 56, 57, 58, 68, 60, 72, 62, 63, 73, 74, 66, 67, 68, 78, 81, 82, 72, 84, 74, 84, 85, 88, 89, 79, 91, 92, 82, 83, 84, 94, 86, 96, 97, 98]**

**Depth First Search took: 0.0 milliseconds**

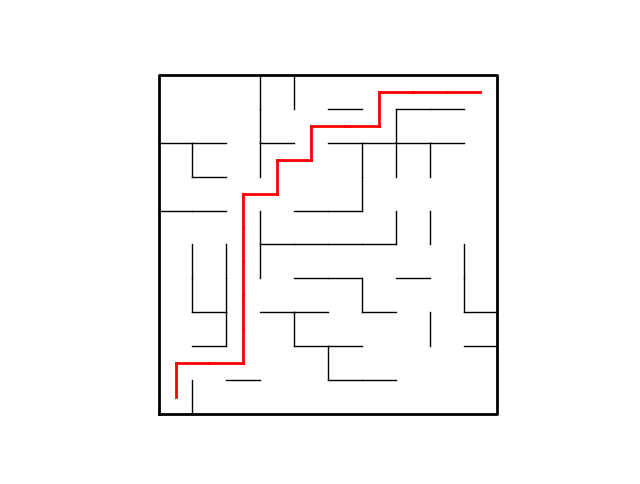
**Depth First Search: [-1, 2, 3, 13, 14, 4, 5, 6, 7, 8, 0, 10, 22, 23, 13, 16, 26, 27, 8, 9, 10, 20, 32, 22, 25, 26, 27, 37, 38, 28, 20, 41, 33, 34, 35, 25, 37, 38, 48, 49, 30, 51, 52, 33, 43, 44, 47, 57, 47, 59, 40, 50, 51, 63, 55, 56, 66, 67, 68, 58, 61, 62, 52, 62, 63, 64, 67, 68, 78, 79, 60, 72, 62, 63, 64, 65, 66, 67, 79, 89, 81, 82, 72, 84, 74, 84, 96, 88, 89, 99, 91, 81, 82, 83, 84, 94, 95, 96, 97, 98]**

**Depth First Search using recursion took: 0.0 milliseconds**

**Depth First Search using recursion: [-1, 11, 1, 2, 5, 6, 7, 8, 9, 19, 0, 10, 22, 3, 4, 16, 26, 7, 28, 18, 30, 20, 23, 13, 25, 26, 27, 17, 38, 28, 40, 41, 22, 34, 35, 25, 46, 36, 48, 49, 50, 51, 32, 33, 43, 44, 45, 46, 58, 59, 51, 52, 42, 54, 55, 56, 66, 47, 68, 58, 61, 62, 52, 62, 74, 64, 67, 57, 69, 79, 60, 72, 62, 63, 73, 65, 66, 67, 68, 89, 90, 80, 72, 84, 74, 84, 85, 88, 89, 99, 91, 92, 82, 83, 95, 96, 86, 96, 97, 98]**

**How do you want to generate a path if at least one is available, select 1 for Breadth First Search, 2 for Depth First Search, or 3 for Depth First Search using recursion: 1**

**It took 15.955810546875 milliseconds to generate a path, (if one exists), from start to goal.**



\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

**How many walls do you want to remove: 120**

**There is at least one path from source to destination (m > n-1).**

**Running time for this maze is with maze size: 10 rows, 10 columns, time in milliseconds: 4.987060546875**

**Breadth First Search took: 0.9970703125 milliseconds**

**Breadth First Search: [-1, 0, 1, 13, 3, 15, 5, 17, 18, 8, 0, 12, 2, 12, 13, 14, 6, 18, 28, 18, 10, 20, 12, 22, 14, 15, 25, 37, 38, 28, 20, 21, 31, 32, 24, 25, 46, 47, 37, 49, 30, 31, 52, 33, 34, 44, 45, 46, 47, 48, 40, 41, 62, 43, 64, 56, 57, 47, 48, 58, 50, 51, 61, 53, 74, 64, 76, 66, 58, 68, 60, 70, 71, 63, 73, 74, 75, 67, 68, 78, 70, 71, 83, 73, 83, 84, 85, 88, 78, 88, 80, 90, 91, 83, 84, 96, 86, 87, 88, 98]**

**Depth First Search took: 0.0 milliseconds**

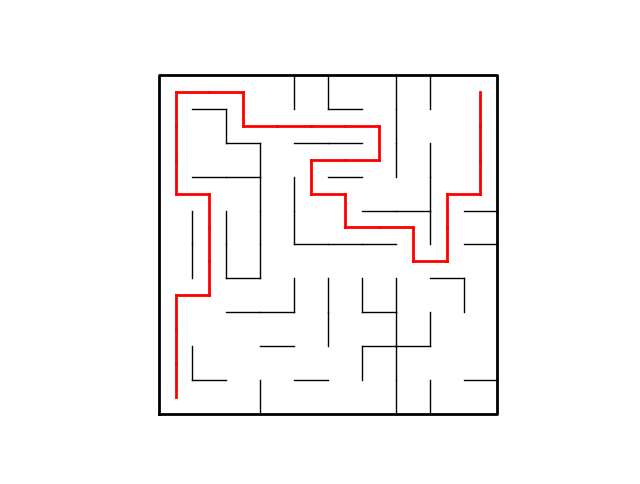
**Depth First Search: [-1, 0, 12, 4, 5, 15, 5, 17, 18, 8, 0, 12, 13, 14, 24, 14, 6, 18, 19, 29, 10, 20, 12, 24, 34, 15, 25, 37, 29, 39, 20, 30, 31, 43, 44, 45, 46, 47, 28, 49, 30, 31, 52, 44, 45, 46, 47, 57, 47, 48, 60, 41, 62, 43, 64, 65, 55, 56, 48, 58, 61, 51, 61, 53, 74, 64, 76, 77, 58, 68, 60, 70, 71, 83, 75, 76, 86, 87, 68, 69, 70, 80, 92, 82, 83, 84, 85, 88, 89, 79, 80, 90, 91, 92, 84, 96, 86, 87, 88, 89]**

**Depth First Search using recursion took: 0.0 milliseconds**

**Depth First Search using recursion: [-1, 0, 1, 4, 5, 15, 5, 17, 18, 8, 20, 21, 2, 12, 13, 25, 6, 18, 28, 18, 21, 22, 23, 24, 14, 35, 25, 37, 29, 39, 20, 32, 33, 43, 44, 45, 46, 38, 28, 49, 30, 31, 52, 53, 43, 44, 45, 46, 58, 48, 40, 41, 62, 63, 55, 56, 57, 47, 68, 58, 50, 51, 61, 73, 54, 66, 76, 66, 69, 79, 60, 70, 71, 83, 64, 74, 75, 67, 68, 89, 81, 71, 83, 93, 85, 86, 76, 77, 87, 99, 80, 90, 91, 92, 84, 96, 86, 87, 88, 98]**

**How do you want to generate a path if at least one is available, select 1 for Breadth First Search, 2 for Depth First Search, or 3 for Depth First Search using recursion: 2**

**It took 28.923828125 milliseconds to generate a path, (if one exists), from start to goal.**



//////////////////////////////////////////////////////////////////////////////////////////////////

**How many walls do you want to remove: 120**

**There is at least one path from source to destination (m > n-1).**

**Running time for this maze is with maze size: 10 rows, 10 columns, time in milliseconds: 7.017822265625**

**Breadth First Search took: 0.0 milliseconds**

**Breadth First Search: [-1, 0, 12, 2, 14, 6, 16, 6, 7, 8, 0, 21, 22, 23, 24, 5, 26, 16, 17, 9, 10, 20, 21, 22, 23, 24, 25, 26, 27, 28, 31, 21, 31, 23, 44, 25, 37, 27, 39, 29, 30, 31, 52, 42, 45, 35, 47, 37, 38, 48, 40, 41, 51, 43, 53, 45, 46, 56, 68, 58, 50, 51, 61, 62, 63, 64, 65, 66, 67, 59, 60, 61, 71, 72, 73, 74, 75, 76, 68, 69, 70, 80, 72, 82, 83, 84, 96, 97, 89, 99, 80, 81, 82, 83, 93, 85, 95, 96, 97, 98]**

**Depth First Search took: 0.0 milliseconds**

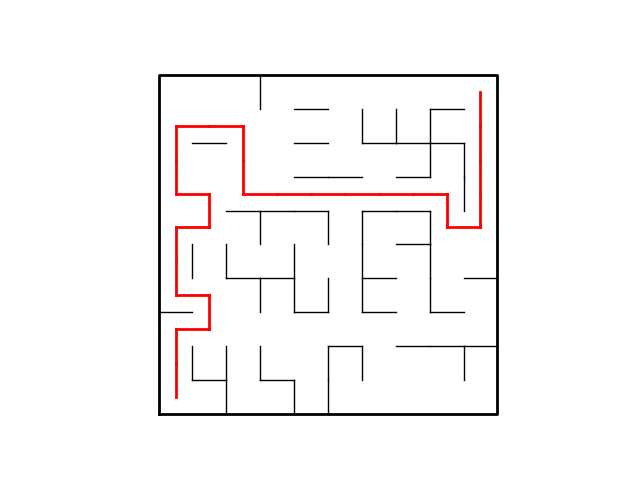
**Depth First Search: [-1, 0, 12, 2, 14, 6, 16, 17, 18, 8, 0, 21, 22, 23, 13, 5, 26, 16, 17, 9, 10, 20, 21, 22, 23, 24, 25, 26, 27, 28, 40, 21, 22, 23, 44, 25, 37, 27, 39, 29, 50, 51, 52, 42, 54, 44, 47, 37, 38, 48, 51, 61, 51, 43, 53, 45, 46, 56, 48, 58, 70, 71, 61, 73, 63, 66, 67, 68, 58, 79, 80, 70, 73, 74, 75, 76, 66, 76, 68, 89, 90, 82, 83, 73, 83, 75, 96, 97, 89, 99, 91, 92, 82, 83, 93, 94, 95, 96, 97, 98]**

**Depth First Search using recursion took: 0.0 milliseconds**

**Depth First Search using recursion: [-1, 0, 12, 2, 14, 6, 16, 6, 18, 8, 0, 21, 22, 14, 24, 5, 26, 7, 17, 9, 10, 20, 23, 13, 25, 26, 27, 28, 29, 39, 31, 21, 22, 23, 44, 25, 37, 27, 48, 38, 30, 51, 43, 53, 45, 35, 47, 37, 49, 59, 40, 50, 42, 54, 44, 45, 46, 56, 68, 58, 61, 51, 72, 62, 63, 64, 65, 66, 67, 59, 60, 72, 82, 74, 75, 85, 75, 76, 68, 69, 70, 80, 81, 93, 83, 84, 96, 97, 89, 79, 91, 92, 82, 94, 95, 96, 97, 98, 99, 89]**

**How do you want to generate a path if at least one is available, select 1 for Breadth First Search, 2 for Depth First Search, or 3 for Depth First Search using recursion: 3**

**It took 24.935302734375 milliseconds to generate a path, (if one exists), from start to goal.**



//////////////////////////////////////////////////////////////////////////////////////////

Here we can see that the breadth first search was the fastest in comparison with the other searches. The path function is used to generate the red line formed to show the path in the figure. The function draw\_maze from lab 6 is used as a parameter with the user chosen search, the size of the maze n, (maze columns times maze rows), maze columns - .5 and maze rows - .5. Creating the path was the most difficult part of this lab as it was very difficult to determine how the line from the start to the goal would be done. The decided way to do this was by using recursion and as a matter of fact there are up to four ways the recursion call can be made. Everything is handled by an if statement that goes into other if statements that create the red line. This initial if statement determines if the current item in the list generated by the selected search of position determined by the size of the maze, (columns times rows – 1, (since we start at 0)) is not -1. This is because only the starting point, cell 0, or the bottom left corner, should be the only one with a -1. If there is another -1 anywhere on the search list, that means that there isn’t any path available in the maze. This also means that the actual path is drawn from the goal, (top right corner) to the start of the maze. This is also the reason that the search list is called previous since it will start at the end of the list and go back. The variables x, x1, y and y1 are used to represent the walls for the rows and columns and are subtracted by .5 when path is first called to plot the red line at the center of the cells. Each if statement inside the first if statement uses the conditions the variable vertex being equal to previous(vertex) and some combination of addition or subtraction of maze columns or 1. What this means is that if the condition is met the wall from that cell was removed and as such a path may be possible. Then the recursion call is made which checks the next wall to find out if a wall was removed and in case there isn’t any removal other than the one where the path was being created, will mean that there is no path and will go back to each of the calls until it finds a different wall that was removed and find its path to cell 0.

The path as stated has four recursion calls giving it a recursive equation of T(n) = 4T(n-1) and means that the runtime has an exponential growth of O(4^n). Based on the data, the path, if generated, is what takes the longest to create in every run. For the Union\_Maze the big o notation would be O(n) since it contains a while loop that depends on m, the number of walls to be removed and for the creation of the temp list which is used to create the adjacency list that is not inside the while loop. Even though the get\_adj\_list funcition is inside this function, the big o of it is O(n) and if added it would be O(n + n) which is O(2n) which will end up being O(n). For breadth\_fist\_search and depth\_fist\_search, will have the same big o notation since they are similar. They run at O(n^2) due the for loop inside the while loop that depend on variables. The depth\_first\_search\_recursion is different since it has a recurrence equation of T(n) = T(n-1) + n. This means that the big o notation will be O(n^2) making all the searches similar in runtime with no clear advantage. One can notice with the data though that maybe breadth first search might be the slowest just slightly since in some runs it took about .9 milliseconds to generate the list while the other took 0.

What I learned for this project was how breadth fist search and both depth first search worked. While their code is different, they will generate the same list and have the same runtime. I have a slight preference for the recursive one just because it is less code. Also, I further learned how to read a list of lists as this helped with providing more experience on how to manipulate them. I still feel that my weak point are the images since I had a difficult time created the line for the path. After much investigation and thought I was able to pull it off and have complete understanding.

Appendix:

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

"""

Created on Wed Apr 17 14:06:40 2019

@author: Gerardo Armenta

Instructor: Dr. Olac Fuentes

TA: Anindita Nath, Maliheh Zargaran, Erik Macik, Eduardo Lara

Purpose: To use the previous lab 6 and create a maze that asks the user to select the number of walls they want removed and then create an adjacency list of each path created from each wall removal. Then implement breadth first search, depth first search and depth first search using recursion. Finally create a path from start, cell 0, to goal, top right corner.

"""

import matplotlib.pyplot as plt

import numpy as np

import time

import random

import queue

def DisjointSetForest(size):

# Creates disjoint set forest all set to -1 (the root)

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

return True

return False

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

return True

return False

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]:

S[rj] += S[ri]

S[ri] = rj

else:

S[ri] += S[rj]

S[rj] = ri

def NumSets(S):

# Counts the number of sets of a disjoint set forest

count =0

for i in S:

if i < 0:

count += 1

return count

def draw\_maze(walls,maze\_rows,maze\_cols,cell\_nums=False):

# Draws a maze

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)

return ax

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

# Creates the adjacency list for the graph

def get\_adj\_list(p, A):

for i in range(len(p)):

temp0 = p[i][0]

temp1 = p[i][1]

A[temp0].append(temp1)

A[temp1].append(temp0)

return A

# Creates the path from start to goal

# Will check for a path recursively from goal to start by checking if path exists

def path(plot, previous, vertex, x, y) :

# Since only element at previous[0] should equal -1 if a path exists

# if a -1 is found before hand, it means no path exists.

if previous[vertex] != -1:

# path is ploted in red from removed wall

if vertex == (previous[vertex] + maze\_cols) :

x1 = x

y1 = y - 1

path(plot, previous, previous[vertex], x1, y1)

plot.plot([x1, x], [y1, y], linewidth = 2, color = 'r')

if vertex == (previous[vertex] - maze\_cols) :

x1 = x

y1 = y + 1

path(plot, previous, previous[vertex], x1, y1)

plot.plot([x1, x], [y1, y], linewidth = 2, color = 'r')

if vertex == (previous[vertex] + 1) :

x1 = x - 1

y1 = y

path(plot, previous, previous[vertex], x1, y1)

plot.plot([x1, x], [y1, y], linewidth = 2, color = 'r')

if vertex == (previous[vertex] - 1) :

x1 = x + 1

y1 = y

path(plot, previous, previous[vertex], x1, y1)

plot.plot([x1, x],[y1, y], linewidth = 2, color = 'r')

# Traverses adjacency list using depth-first search

def depth\_first\_search(G):

start = time.time()\*1000

visited = [False] \* len(G) # creates a list of boolean False of length G

prev = [-1] \* len(G) # creates a list of -1 of length G

S = [] # A list used as a stack that starts empty

S.append(0) # Starting point is vertex 0 so it is appended fist

visited[0] = True # Visisted list is changed to True for 0, the starting point

# Traversal starts from 0

while S: # While the stack is not empty

v = S.pop() # v is for the vertext which is popped from S

for t in G[v]:

if not visited[t]: # if visited[t] is False

visited[t] = True

prev[t] = v

S.append(t)

stop = time.time()\*1000

print('Depth First Search took: ', stop-start, ' milliseconds')

return prev

# Traverses adjacency list using depth-first search with recursion

def depth\_first\_search\_recursion(G, source):

visited[source] = True # Source starts at 0 and moves on recursively

for t in G[source]:

if not visited[t]: # if visited[t] is False

prev[t] = source # the source is appended to prev list

depth\_first\_search\_recursion(G, t)

return prev

# Travereses adjacency list using breadth-first search

def breadth\_first\_search(G):

start = time.time()\*1000

visited = [False] \* len(G) # creates a list of boolean False of length G

prev = [-1] \* len(G) # creates a list of -1 of length G

Q = queue.Queue() # a queue that starts empty

Q.put(0) # Starting point is vertex 0 so it is placed fist

visited[0] = True # Visisted list is changed to True for 0, the starting point

# Traversal starts from 0

while Q.empty() is False: # While the queue is not empty

v = Q.get() # v is for the vertext which is popped from Q

for t in G[v]:

if not visited[t]: # if visited[t] is False

visited[t] = True

prev[t] = v # the vertex is assigned to prev list

Q.put(t)

stop = time.time()\*1000

print('Breadth First Search took: ', stop-start, ' milliseconds')

return prev

def Union\_Maze(M, w, m): # creates a maze and asks the user how many walls to remove

popped = []

start\_union = time.time() \* 1000

while m > 0:

d = random.randint(0, len(walls)-1) # d equals the wall to be removed at random

if NumSets(M) == 1:

popped.append(walls.pop(d))

m -= 1

elif union\_c(M, walls[d][0], walls[d][1]) is True:

popped.append(walls.pop(d))

m -=1

stop\_union = time.time() \* 1000

print('Running time for this maze is with maze size: ',

maze\_rows, ' rows, ', maze\_cols, ' columns, time in milliseconds: ', (stop\_union-start\_union))

temp\_list = []

for i in range(maze\_rows\*maze\_cols):

temp\_list.append([])

adj\_list = get\_adj\_list(popped, temp\_list)

return adj\_list

plt.close("all")

maze\_rows = 10

maze\_cols = 10

n = maze\_rows\*maze\_cols # Total number of cells in the maze

walls = wall\_list(maze\_rows,maze\_cols) # Creates the walls for the maze

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

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

# This loop created in case user wants to remove more walls than the ones in the maze

while True:

m = int(input('How many walls do you want to remove: ')) # Asks user for number of walls to be removed

# This checks if the user wants to remove more walls than there are in the maze.

if m > len(walls)-1:

print('The number of walls you want to remove exeeds the number of walls that are present.')

continue # This will cause the loop to continue and ask the user to select less walls to remove

if m > n-1:

print('There is at least one path from source to destination (m > n-1).')

break # Will exit loop

if m < n-1:

print('A path from source to destination is not guaranteed to exist (m < n-1).')

break # Will exit loop

if m == n-1:

print('There is a unique path from source to destination (m = n-1).')

break # will exit loop

G = Union\_Maze(M, walls, m) # Creates maze

visited = [False] \* len(G) # variable created to be used by depth\_first\_search\_recursion

prev = [-1] \* len(G) # variable created to be used by depth\_first\_search\_recursion

bfs = breadth\_first\_search(G)

print('Breadth First Search: ', bfs)

dfs = depth\_first\_search(G)

print('Depth First Search: ', dfs)

start\_dfsr = time.time()

dfsr = depth\_first\_search\_recursion(G, 0)

stop\_dfsr = time.time()

print('Depth First Search using recursion took: ', stop\_dfsr-start\_dfsr, ' milliseconds')

print('Depth First Search using recursion: ', dfsr)

line = draw\_maze(walls,maze\_rows,maze\_cols) # used to send to path to generate path from start to goal

# Path will generate a red line from start to goal if path is available

ans = int(input('How do you want to generate a path if at least one is available, select 1 for Breadth First Search, 2 for Depth First Search, or 3 for Depth First Search using recursion: '))

start\_path = time.time()\*1000

if ans == 1: # Will generate a path using breadth first search

path(line,bfs,(n)-1,maze\_cols-.5,maze\_rows-.5)

elif ans== 2: # Will generate a path using depth first search

path(line,dfs,(n)-1,maze\_cols-.5,maze\_rows-.5)

else: # Will generate a path using depth first search using recursion

path(line,dfsr,(n)-1,maze\_cols-.5,maze\_rows-.5)

stop\_path = time.time()\*1000

print('It took ', stop\_path-start\_path, ' milliseconds to generate a path, (if one exists), from start to goal.')

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

plt.show()

Academic dishonesty includes but is not limited to cheating, plagiarism and collusion. Cheating may involve copying from or providing information to another student, possessing unauthorized materials during a test, or falsifying data (for example program outputs) in laboratory reports. Plagiarism occurs when someone represents the work or ideas of another person as his/her own. Collusion involves collaborating with another person to commit an academically dishonest act. Professors are required to - and will - report academic dishonesty and any other violation of the Standards of Conduct to the Dean of Students.