CS2302 Data Structures

Fall 2019

Lab Report 6

Author: Bryan Ramos

Due: November 15th 2019

Professor: Dr. Olac Fuentes

TA: Anindita Nath

Introduction

The purpose of this lab was to provide deliberate practice with three graph implementations, adjacency lists, adjacency matrices and edge lists, three different ways in which graphs can be represented. Then, by using depth first search and breadth first search, its possible to search within the graphs and determine in what ways both searching methods differ.

Proposed Solution Design and Implementation

The first part of designing the programs for the graph implementations and the main lab file was importing the provided code from the class website. In addition, without knowing it at the time, some of the code needed to solve this problem like inserting edges, deleting edges and implementing the display function were completed in two person groups during the class lectures so that came in handy to finishing the lab successfully. I verified that the three graph implementation classes were working as expected utilizing the test graphs provided code from the class web page.

Next, came writing the conversion functions to go from one type of graph to the other and so on. This was relatively simple as each function followed the same algorithm design. Let's say that for each edge in the graph, use *insert edge* to build a new graph of the different type.

Next, for each graph implementation came writing the search functions for breadth first and depth first. This task was also relatively easy as we had discussed in the class lecture, and followed the pseudocode discussed with Dr. Fuentes on Zybooks. The searching algorithms return a list called path where each index of the list contains its predecessor leading back to the start.

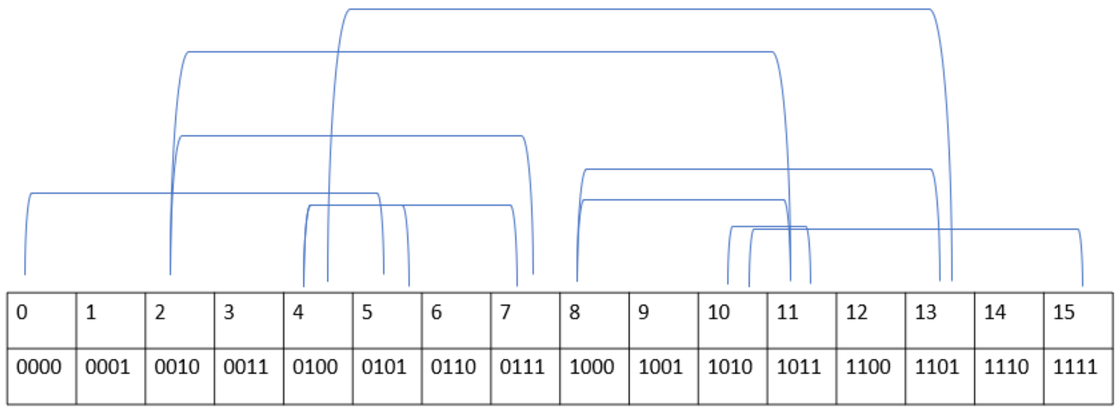
Next, for each graph implementation, I created a function that takes the path list from the previous functions as a parameter as well as the destination vertex and prints the path from the start leading to the destination for both breadth first search and depth first search.

For report purposes and to ensure the graph implementations were written correctly and also the conversion functions work as they should, at the end of my code the console returns the drawn graphs which should all be the same.

Fox, Chicken, Grain, Human Problem, here is how I approached the problem:

You have a fox, a chicken and a sack of grain. You must cross a river with only one of them at a time. If you leave the fox with the chicken he will eat it; if you leave the chicken with the grain he will eat it. How can you get all three across safely? For part 2, you will implement a solution to this problem using a graph search algorithm.

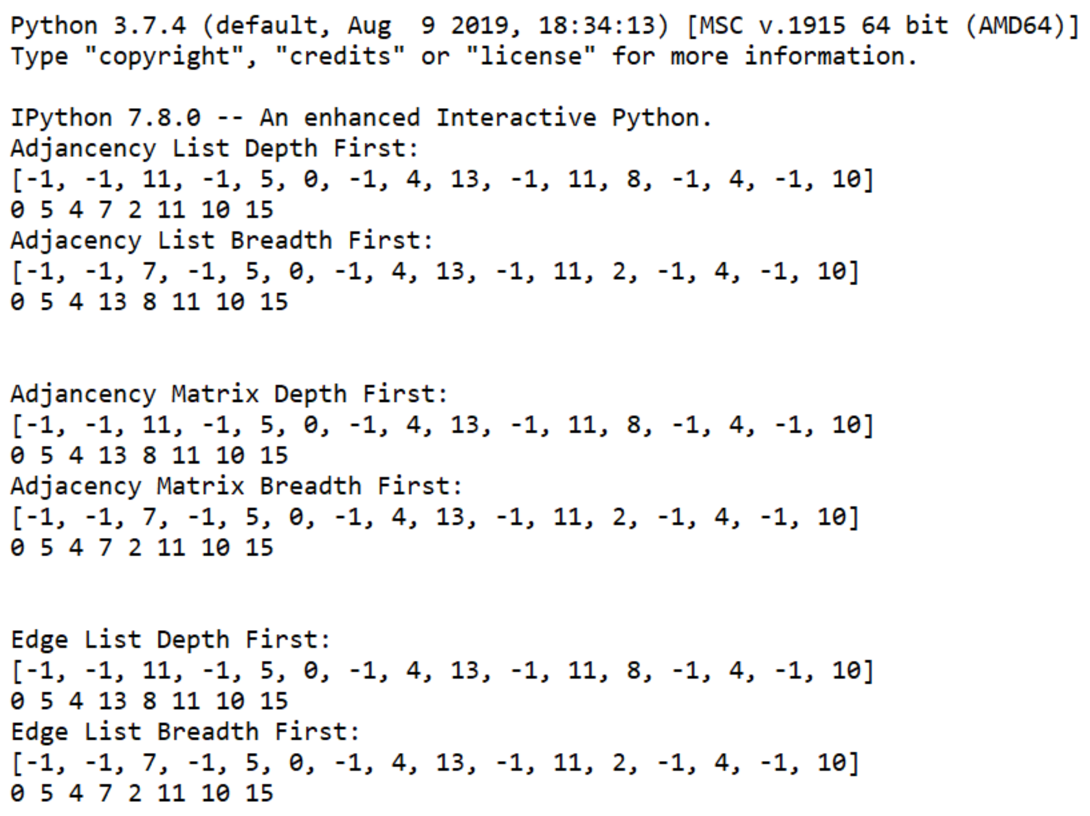
Following the lab 6 instructions, each scenario that could occur is encoded as bits. For the scenarios that follow the rules, they are valid, and the scenarios should be connected. The valid edges were then encoded in graphs.



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

Experimental Results

Depth First and Breadth First Search algorithms output: Below is the output for depth first and breadth first search for all three implementations. The result the code should print is the quickest path from vertex 0 to vertex 15.



```
Python 3.7.4 (default, Aug  9 2019, 18:34:13) [MSC v.1915 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.8.0 -- An enhanced Interactive Python.
Adjancency List Depth First:
[-1, -1, 11, -1, 5, 0, -1, 4, 13, -1, 11, 8, -1, 4, -1, 10]
0 5 4 7 2 11 10 15
Adjancency List Breadth First:
[-1, -1, 7, -1, 5, 0, -1, 4, 13, -1, 11, 2, -1, 4, -1, 10]
0 5 4 13 8 11 10 15


Adjancency Matrix Depth First:
[-1, -1, 11, -1, 5, 0, -1, 4, 13, -1, 11, 8, -1, 4, -1, 10]
0 5 4 13 8 11 10 15
Adjancency Matrix Breadth First:
[-1, -1, 7, -1, 5, 0, -1, 4, 13, -1, 11, 2, -1, 4, -1, 10]
0 5 4 7 2 11 10 15


Edge List Depth First:
[-1, -1, 11, -1, 5, 0, -1, 4, 13, -1, 11, 8, -1, 4, -1, 10]
0 5 4 13 8 11 10 15
Edge List Breadth First:
[-1, -1, 7, -1, 5, 0, -1, 4, 13, -1, 11, 2, -1, 4, -1, 10]
0 5 4 7 2 11 10 15
```

Graphs Constructed: It is necessary to check that all of the graph implementations (adjacency list, adjacency matrix and edge list) build the same graph with the same edges. To confirm this, we can see if the graphs drawn as output are the same:

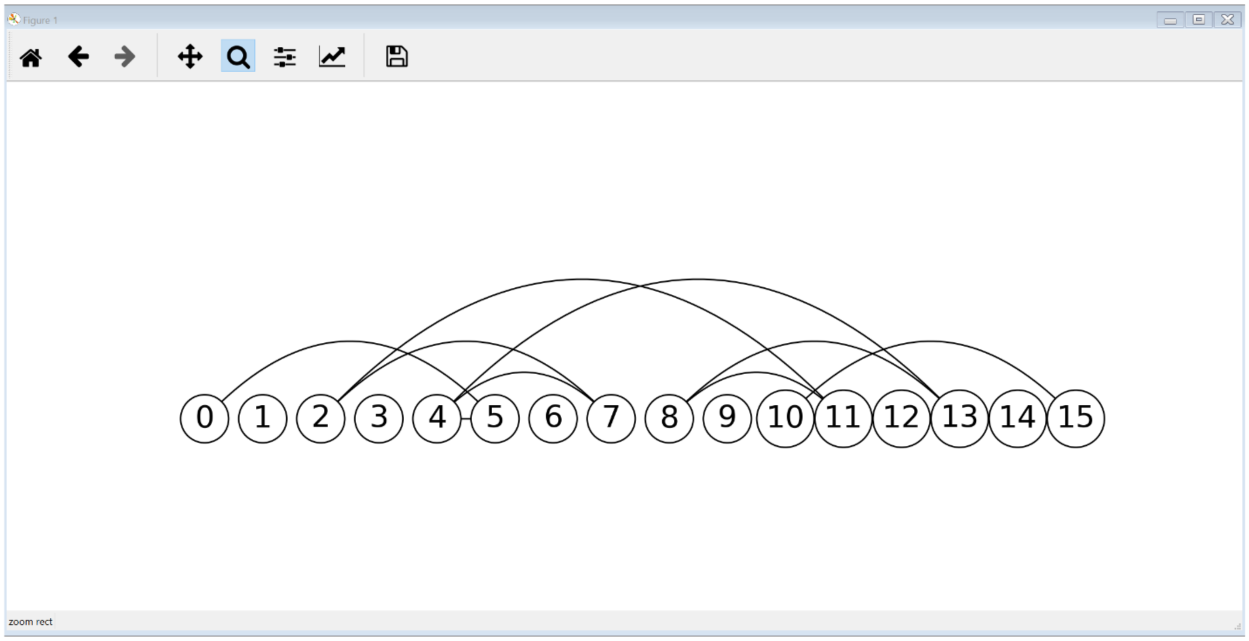


Figure 1
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
zoom rect

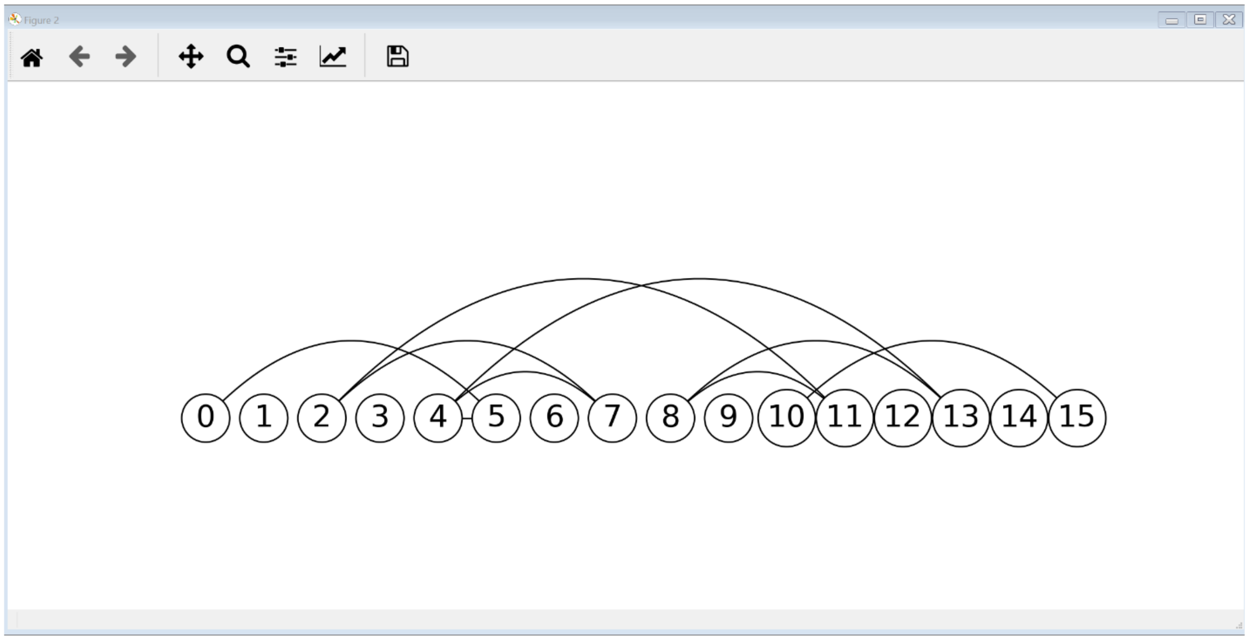
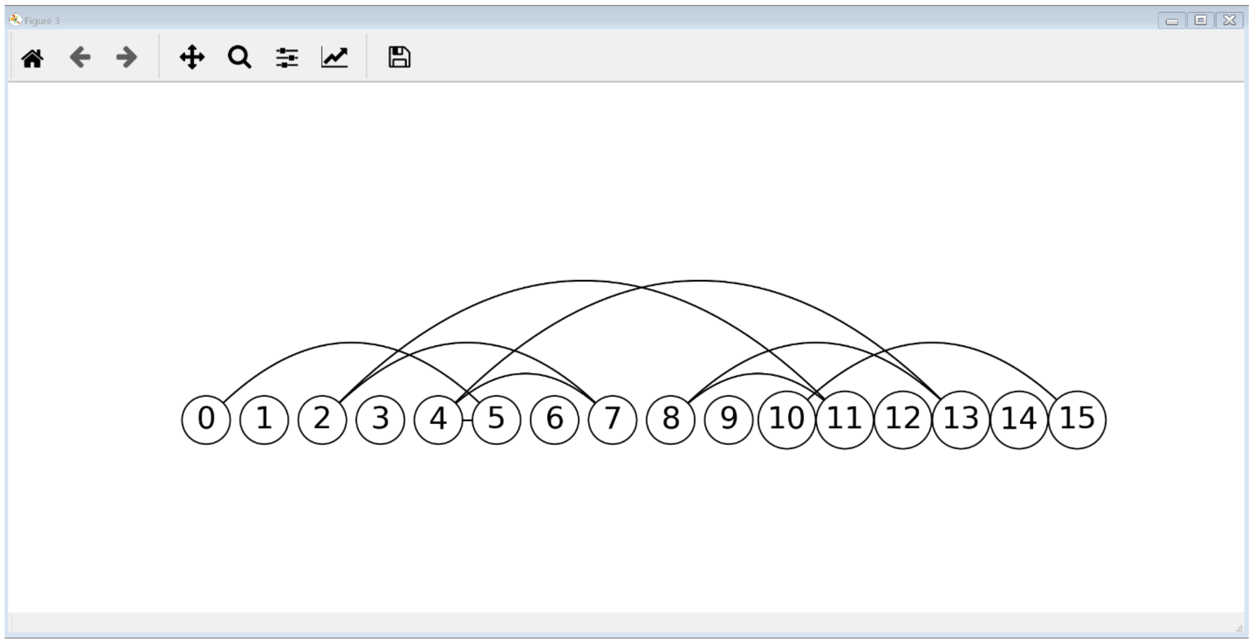


Figure 2
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15



For the searching algorithms, breadth first and depth first, the same results were found. This is correct as the goal was to construct the same graph using different graph implementations. The algorithm also provided another solution to the problem once it was calculated in the console, showing this problem has more than one solution not just one solution.

Conclusion

Graph implementations are diverse, graphs can be implemented using adjacency lists, adjacency matrices and edge lists. I was unable to calculate the running times for searching as the console would always print 0.0 for the running time. Nevertheless, through this lab, the lectures and with deliberate practice, I learned that graphs have important properties:

- Graphs can be weighted
- Graphs can be directed
- Depth first algorithm utilizes a stack
- Breadth first algorithm utilizes a queue

Furthermore, breadth first and depth first search can be extremely useful in real world applications.

Appendix

```
'''
Course: CS2302 Data Structures Fall 2019
Author: Bryan Ramos [88760110]
Assignment: Lab 6
Instructor: Dr. Olac Fuentes
TA: Anindita Nath
Last Modified: November 18th 2019
Purpose: Implement Adjacency List, Adjacency Matrix and Edge List
graph representations.
'''

import graph_AL as graphAL
import graph_AM as graphAM
import graph_EL as graphEL

# build three graph representation:
# - adjacency list
# - adjacency matrix
# - edge list
def buildGraphRepresentations():
    # vars
    v = 16
    weighted = False
    directed = False

    # build adjacency list
    AL = graphAL.Graph(v, weighted, directed)
    AL.insert_edge(0, 5)
    AL.insert_edge(2, 11)
    AL.insert_edge(2, 7)
    AL.insert_edge(4, 5)
    AL.insert_edge(4, 7)
    AL.insert_edge(4, 13)
    AL.insert_edge(8, 11)
    AL.insert_edge(8, 13)
    AL.insert_edge(10, 11)
    AL.insert_edge(10, 15)

    # build adjacency matrix, build edge list
    AM = AL.as_AM()
    EL = AL.as_EL()

    # return all three representations
    return AL, AM, EL

# for lab report
def graphs(AL, AM, EL):
    AL.draw()
    AM.draw()
    EL.draw()

if __name__ == "__main__":

    AL, AM, EL = buildGraphRepresentations()

    # adjacency list
    # depth first
    print("Adjancency List Depth First:")
    print(AL.depthFirst(0, 15))
    AL.printDepthFirst(0, 15)
    # breadth first
    print("Adjancency List Breadth First:")
    print(AL.breadthFirst(0, 15))
    AL.printBreadthFirst(0, 15)

    print()
    print()
    # adjancency matrix
    # depth first
    print("Adjancency Matrix Depth First:")
    print(AM.depthFirst(0, 15))
    AM.printDepthFirst(0, 15)
    # breadth first
    print("Adjacency Matrix Breadth First:")
    print(AM.breadthFirst(0, 15))
    AM.printBreadthFirst(0, 15)

    print()
    print()

    # edge list
    # depth first
    print("Edge List Depth First:")
    print(EL.depthFirst(0, 15))
    EL.printDepthFirst(0, 15)
    # breadth first
    print("Edge List Breadth First:")
    print(EL.breadthFirst(0, 15))
    AM.printBreadthFirst(0, 15)

    print()
    print()

    # for lab report
    graphs(AL, AM, EL)
```

I, Bryan Ramos, 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.