Laboratory 6 – Graphs

CS 2302 – Data structures Fall 2019

Carlos cardenas

INSTRUCTOR: OLAC FUENTES

Contents

[Introduction 2](#_Toc15996822)

[Implementation 2](#_Toc15996823)

[Experimental results 3](#_Toc15996824)

[Conclusion 9](#_Toc15996825)

[Honesty Certification 14](#_Toc15996827)

# Introduction

From: <https://www.geeksforgeeks.org/graph-and-its-representations/>

Graph is a data structure that consists of following two components:  
**1.** A finite set of vertices also called as nodes.  
**2.** A finite set of ordered pair of the form (u, v) called as edge. The pair is ordered because (u, v) is not same as (v, u) in case of a directed graph(di-graph). The pair of the form (u, v) indicates that there is an edge from vertex u to vertex v. The edges may contain weight/value/cost.

Graphs are used to represent many real-life applications: Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network. Graphs are also used in social networks like linkedIn, Facebook. For example, in Facebook, each person is represented with a vertex(or node). Each node is a structure and contains information like person id, name, gender and locale.

Following is an example of an undirected graph with 5 vertices.  
A picture containing object

Description automatically generated

Following two are the most commonly used representations of a graph.  
**1.** Adjacency Matrix  
**2.** Adjacency List  
There are other representations also like, Incidence Matrix and Incidence List. The choice of the graph representation is situation specific. It totally depends on the type of operations to be performed and ease of use.

**Adjacency Matrix:**  
Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.

The adjacency matrix for the above example graph is:  
Adjacency Matrix Representation

Pros: Representation is easier to implement and follow. Removing an edge takes O(1) time. Queries like whether there is an edge from vertex ‘u’ to vertex ‘v’ are efficient and can be done O(1).

Cons: Consumes more space O(V^2). Even if the graph is sparse(contains less number of edges), it consumes the same space. Adding a vertex is O(V^2) time.  
  
**Adjacency List:**  
An array of lists is used. Size of the array is equal to the number of vertices. Let the array be array[]. An entry array[i] represents the list of vertices adjacent to the**i**th vertex. This representation can also be used to represent a weighted graph. The weights of edges can be represented as lists of pairs. Following is adjacency list representation of the above graph.

Adjacency List Representation of Graph

**Edge List:**

Finally, and edge list representation, is an array that only includes the edges in a graph. Each edge is represented as: (source, destination, weight).

With our implementation of edge list, a reference to the number of vertices is included in the constructor.

# Implementation

We were given an incomplete file to fill in the missing functions to implement the graphs as AL, AM, and EL. Modifying the parameters and functions in each file was done to implement the desired type of graph. Then, to create another type of graph from our given graph, we simply call the constructor for the desired implementation, and construct a list with the values of our current graph.

The tricky part of this lab was the puzzle of the river crossing. To implement this, I represented the states as 0 to 15. With the desired path being ﻿[0, 5, 4, 7, 11, 10, 15]. To do this, search algorithms were implemented in the graph: Depth first search and breadth first search.

We insert the desired edges and the we use one of these two algorithms to traverse the graph and give us the desired output using the representation of its binary number.

.

# Experimental Results

As an adjacency list:

A close up of text on a black background

Description automatically generated

As an adjacency matrix:

A close up of a keyboard

Description automatically generatedA close up of a keyboard

Description automatically generatedA screenshot of a cell phone

Description automatically generated

As an edge list:

A close up of text on a black background

Description automatically generated

Puzzle:

As an edge list:

A screenshot of a cell phone

Description automatically generated

As an adjacency list:

A close up of a logo

Description automatically generated

As an adjacency matrix:

A close up of a logo

Description automatically generated

Graph output:

A screenshot of a cell phone

Description automatically generated

# 

# Conclusion

Graphs are very useful in determine the best path, which can be really useful in the real world (airplanes, buses, Facebook,etc.). This lab definitely prepared me for the use of graphs in the future. I personally like adjacency lists because they seem to have the better runtimes overall. Edge lists may cause problems if done incorrectly, and adjacency matrices will take a lot of space.

# 

# Honesty Certification

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 provide inappropriate assistance to any student in the class.

11/ 01/ 2019

Carlos Cardenas Date