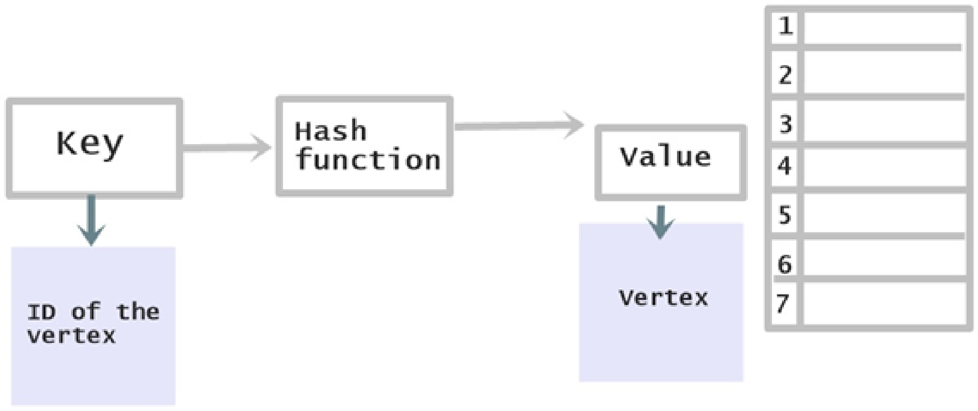
Laboratory practice No. 1: Implementation of graphs

María Sofía Uribe
Universidad Eafit
Medellín, Colombia
msuribec@eafit.edu.co

Isabel Cristina Graciano
Universidad Eafit
Medellín, Colombia
icgracianv@eafit.edu.co

3) Practice for final project defense presentation

3.1 The first exercise was developed by using hash tables which allowed us to reduce the time required for search and insertion. Each vertex has a LinkedList that holds Its neighbors. These lists hold pairs made of an integer (for the neighbor's ID) and a reference to the edge that connects the current vertex to its neighbor.



3.2 An implementation of a graph through an adjacency matrix requires at least n^2 bits of memory (because each Boolean takes 1 bit) , in a graph with 300,000 vertices that would mean

$$300{,}000 \ * 300{,}000 \text{ bits} = \ 90000000000 \text{ bits} = \ 11.25 \text{ Gigabytes}$$

3.3 To solve this problem we used Hash tables, Hash Map in Java, given that the keys (IDS) are unique, having non-sequential keys is not a problem.

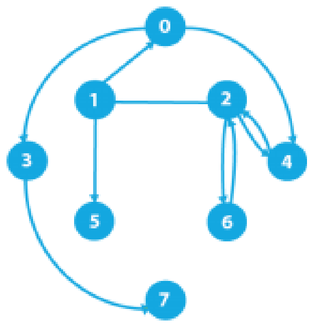
3.4 The structure used was an adjacency list (collection of lists) where each node's neighbors are stored. A DFS is performed to determine whether the graph is bipartite, using two arrays one to keep track of the vertices that have already been visited and another one to keep track of the color (given that there's only a two-color palette, using true and false is adequate). As we traverse the graph, we assign a color to each vertex that is different from its parent's color, if there is an edge connecting two vertices of equal color the graph is not bipartite

3.5 The time complexity of the algorithm provided is $O(V+E)$

3.6 V: number of vertices of the graph, E: number of edges of the graph

4) Practice for midterms

4.1



	0	1	2	3	4	5	6	7
0				1	1			
1	1		1			1		
2		1			1		1	
3								1
4			1					
5								
6			1					
7								

The edge that connects vertices 1 and 2 does not have an arrow pointing at either so we have taken it as a bidirectional arrow.

4.2 Solution

0 -> [3,4]
1 -> [0,2,5]
2 -> [1,4,6]
3 -> [7]
4 -> [2]
5 ->
6 -> [2]
7 ->

The edge that connects vertices 1 and 2 does not have an arrow pointing at either so we have taken it as a bidirectional arrow.

4.3 B. is $O(n^2)$
The space complexity of an adjacency list (worst case) is $O(n^2)$ where n is the number of vertices, an example of the worst case is a completely connected graph.

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

5) *Recommended reading (optional)*

Summary:

Graphs represent connections or relations (edges) between real world objects (called nodes or vertices). The vertices adjacent (connected by an edge) to a given vertex are its neighbors. A graph is connected if there's at least one path (A sequence of edges) from every vertex to every other vertex. A graph is not to be confused with a binary tree, given that the former one allows for a vertex to have more than two children.

A graph can be directed (edges have a direction indicated by an arrow) or undirected.

The most common data structures to represent graphs are adjacency matrices, two-dimensional arrays in which the elements indicate whether an edge is present between two vertices), and adjacency lists, which can be an array of lists or a list of lists.

If there's a need to traverse through every vertex of the graph, a search of two kinds could be performed: DFS or BFS.

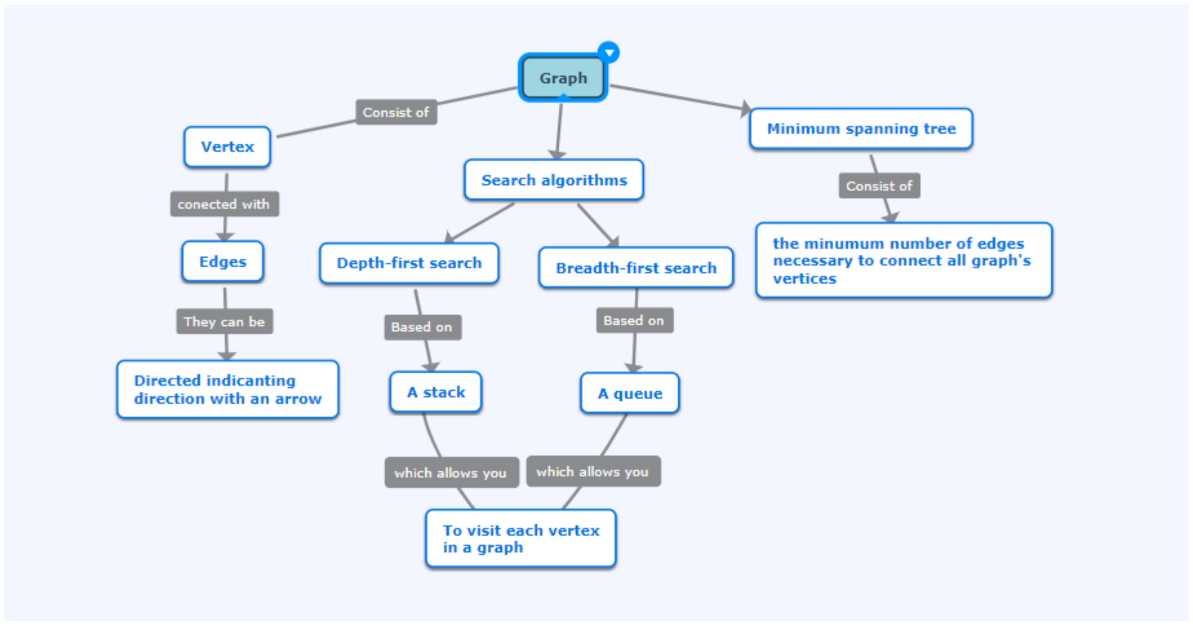
- DFS uses a stack to keep track of where it should go in case of reaching a dead end.

- DFS visits the adjacent vertices of a certain vertex first when possible, if not, it recovers the last vertex of the stack.
- BFS uses a queue.
- BFS visits all the vertices adjacent to one vertex before moving on (like traversing by levels) if there are no more unvisited vertices it removes one from the queue.

A minimum spanning tree (MST) consists of the minimum number of edges necessary to connect all a graph's vertices.

Concept diagram based on the key theoretical elements of "Robert Lafore, Data Structures and Algorithms in Java (2nd edition), Chapter 13: Graphs. 2002"

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473



6) Team work and gradual progress (optional) Discussing

6.1 Meeting minutes

TEAM MEMBER	DATE	DONE	DOING	TO DO
ISABEL	31/01/2019	Discussing task distribution		Plan data structure for exercise 2.1
SOFIA	31/01/2019	Discussing task distribution		Plan data structure for exercise 1
ISABEL	1/02/2019	Data structure works	Solve online exercise	Optional Reading (concept diagram)
SOFIA	1/02/2019	Data structure works	Solve online exercise	Optional Reading(concept map)
ISABEL	2/02/29	Concept map	Program exercise 1	Answer exercise 3
SOFIA	2/02/29	Summary of reading	Program exercise 1	Answer exercise 3

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

ISABEL	3/02/29	Answer exercise 3.1-3.3	Code comments
SOFIA	3/02/29	Answer exercise 3 .4-3.6	Upload code
ISABEL	3/02/29	Code comments	
SOFIA	3/02/29	Upload code and report	

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473