International Journal of Advanced Technology in Engineering and Science www.ijates.com
Volume No.02, Issue No. 09, September 2014 ISSN (online): 2348 – 7550

A REVIEW ON GRAPHS IN DATA STRUCTURES

Ayush Bansal[1], Abhishek Nehra[2], Anurag Vats[3]

[1, 2, 3] Student (B.tech 3rd sem) Department of Computer Science Engineering Dronacharya College of Engineering Gurgaon-123506, (India)

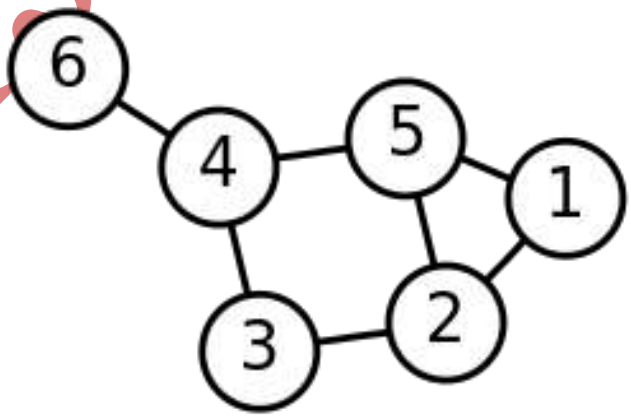
ABSTRACT

*In computer science, a **data structure** is a particular way of organizing data in a computer so that it can be used efficiently. One of the most important structures used in this field is a graph which is formed by nodes and edges. Graphs prove to be beneficial in problems concerned with objects which are related to each other in some manner. Frequent use of graphs in practice has led to extensive research in "**graph theory**", in which there is a large number of known problems for graphs and for most of them there are well-known solutions. This paper gives a review on the graphs in data structures including the terminology used in graph theory. The paper also discusses the different types of graphs available in data structures.*

Keywords: Graph, Algorithms, Edge, Endpoint, Data Structures.

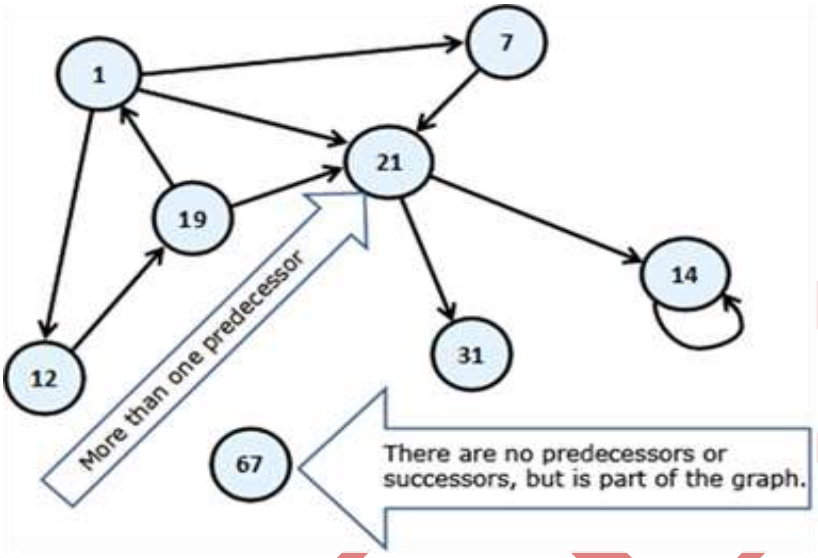
I. INTRODUCTION

A **graph** is a mathematical structure consisting of a set of vertices (also called nodes) $\{v_1, v_2, \ldots, v_n\}$ and a set of edges $\{e_1, e_2, \ldots, e_m\}$. An edge is a pair of vertices $\{v_i, v_j\}$ $i, j \in \{1..n\}$. The two vertices are called the edge *endpoints*. Graphs are ubiquitous in computer **science**. They are used to model real-world systems such as the Internet (each node represents a router and each edge represents a connection between routers); airline connections (each node is an airport and each edge is a flight); or a city road network (each node represents an intersection and each edge represents a block). The wireframe drawings in computer graphics are another example of graphs.



A Labeled Graph for 6 Vertices and 7 Edges.

The **graphs** are very useful and fairly common data structures. They are used to describe a wide variety of **relationships between objects** and in practice can be related to almost everything. As we will see later, trees are a subset of the graphs and also lists are special cases of trees and thus of graphs, i.e. the graphs represent a generalized structure that allows modeling of very large set of real-world situations.



The circles in Fig. 1 are known as **vertices (nodes)** and the arrows connecting them are **directed edges**. The vertex of which the arrow comes out is called **predecessor** of that the arrow points. For example "19" is a predecessor of "1". In this case, "1" is a **successor** of "19". Unlike the structure tree, here each vertex can have more than one predecessor. Like "21", it has three – "19", "1" and "7". If two of the vertices are connected with edge, then we say these two vertices are **adjacent** through this edge.

II. GRAPHS IN DATA STRUCTURES

A graph may be either *undirected* or *directed*. Intuitively, an undirected edge models a "two-way" or "duplex" connection between its endpoints, while a directed edge is a one-way connection, and is typically drawn as an arrow. A directed edge is often called an *arc*. Mathematically, an undirected edge is an unordered pair of vertices, and an arc is an ordered pair. For example, a **road network** might be modelled as a directed graph, with one-way streets indicated by an arrow between endpoints in the appropriate direction, and two-way streets shown by a pair of parallel directed edges going both directions between the endpoints.

III. REPRESENTATION OF GRAPHS

Different data structures for the representation of graphs are used in practice:

- Adjacency list

  Vertices are stored as records or objects, and every vertex stores a list of adjacent vertices. This data structure allows the storage of additional data on the vertices. Additional data can be stored if edges are also stored as objects, in which case each vertex stores its incident edges and each edge stores its incident vertices.

- Adjacency matrix

  Adjacency matrix is a two-dimensional matrix, in which the rows represent source vertices and columns represent destination vertices. Data on edges and vertices must be stored externally. Only the cost for one edge can be stored between each pair of vertices.

- Incidence matrix

  Incidence matrix is a two-dimensional Boolean matrix, in which the rows represent the vertices and columns represent the edges. The entries indicate whether the vertex at a row is incident to the edge at a column.

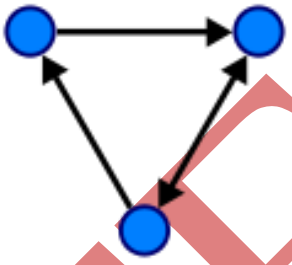
The following table gives the time complexity cost of performing various operations on graphs, for each of these representations. In the matrix representations, the entries encode the cost of following an edge. The cost of edges that are not present are assumed to be ∞.

	Adjacency list	**Adjacency matrix**	**Incidence matrix**
Storage	$O(\|V\| + \|E\|)$	$O(\|V\|^2)$	$O(\|V\| \cdot \|E\|)$
Add vertex	$O(1)$	$O(\|V\|^2)$	$O(\|V\| \cdot \|E\|)$
Add edge	$O(1)$	$O(1)$	$O(\|V\| \cdot \|E\|)$
Remove vertex	$O(\|E\|)$	$O(\|V\|^2)$	$O(\|V\| \cdot \|E\|)$
Remove edge	$O(\|E\|)$	$O(1)$	$O(\|V\| \cdot \|E\|)$
Query: are vertices u, v adjacent? (Assuming that the storage positions for u, v are known)	$O(\|V\|)$	$O(1)$	$O(\|E\|)$
Remarks	When removing edges or vertices, need to find all vertices or edges	Slow tor remove vertices, because matrix must be resized/copied	Slow to add or remove vertices and edges, because matrix must be resized/copied

Adjacency lists are generally preferred because they efficiently represent sparse graphs. An adjacency matrix is preferred if the graph is dense, that is the number of edges $|E|$ is close to the number of vertices squared, $|V|^2$, or if one must be able to quickly look up if there is an edge connecting two vertices.

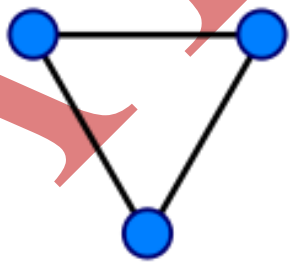
A) Directed Graph

The number of edges with one endpoint on a given vertex is called that vertex's **degree**. In a directed graph, the number of edges that point *to* a given vertex is called its **in-degree**, and the number that point *from* it is called its **out-degree**. Often, we may want to be able to distinguish between different nodes and edges. We can associate labels with either. Such a graph is called **labeled**.



B) Undirected Graphs

In a directed graph, the edges point from one vertex to another, while in an **undirected graph**, they merely connect two vertices.



C) Weighted Graphs

We may also want to associate some cost or weight to the traversal of an edge. When we add this information, the graph is called **weighted**. An example of a weighted graph would be the distance between the capitals of a set of countries. Directed and undirected graphs may both be weighted. The operations on a weighted graph are the same with addition of a weight parameter during edge creation.

VI. CONCLUSION

Graphs are very important in data structures. Graph theory is an integral part of data structures that offers solutions to a wide range of problems and is thus extensively used. In this paper we have reviewed the basics of graphs and the types of graphs in data structures.

REFERENCES

[1] http://en.wikipedia.org/wiki/Graph_(abstract_data_type)
[2] http://stackoverflow.com/questions/14296415/whats-the-complexity-of-going-from-one-graph-representation-to-another
[3] http://www.introprogramming.info/english-intro-csharp-book/read-online/chapter-17-trees-and-graphs/
[4] http://en.wikipedia.org/wiki/Data_structure
[5] http://gr12computers.wikispaces.com/Graph+Theory
[6] http://howtoprogramwithjava.com/the-5-basic-concepts-of-any-programming-language-concept-3/
[7] http://technicalsanju.blogspot.com/
[8] http://www.exforsys.com/tutorials/c-algorithms/graph-theory.html