IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 6, June 2016

A Brief Study of Graph Data Structure

Jayesh Kudase[1], Priyanka Bane[2]

B.E. Student, Dept of Computer Engineering, Fr. Conceicao Rodrigues College of Engineering, Maharashtra, India[1, 2]

Abstract: Graphs are a fundamental data structure in the world of programming. Graphs are used as a mean to store and analyse metadata. A graph implementation needs understanding of some of the common basic fundamentals of graph. This paper provides a brief study of graph data structure. Various representations of graph and its traversal techniques are discussed in the paper. An overview to the advantages, disadvantages and applications of the graphs is also provided.

Keywords: Graph, Vertices, Edges, Directed, Undirected, Adjacency, Complexity.

I. INTRODUCTION

Graph is a data structure that consists of finite set of vertices, together with a set of unordered pairs of these vertices for an undirected graph or a set of ordered pairs for a directed graph. These pairs are known as edges/lines/arcs for undirected graphs and directed edge / directed arc / directed line for directed graphs. An edge can be associated with some edge value such as a numeric attribute. These attribute will be based on cost or length or capacity. We can represent the vertices externally also with the help of integer indices or references.

Graphs are very important in the field of computer science. They are used to model real world systems such as Internet where each node represents a router and each edge represents a connection between the routers. The wireframe drawings in computer graphics are another example of graphs.

II. BACKGROUND

A. What is a Graph?

Graph is a pictorial representation of a set of objects, where the object pairs are connected by links. The interconnected objects are called as vertices or nodes of a graph whereas the links connecting these vertices are known as edges.

It is an ordered pair of sets (V, E) where V= set of vertices and E = set of edge joining the vertices pairs. E is a subset of V x V. Simply speaking, each edge connects two vertices, including a case, when a vertex is connected to itself (such an edge is called a loop). Graph structure is shown in Fig. 1.

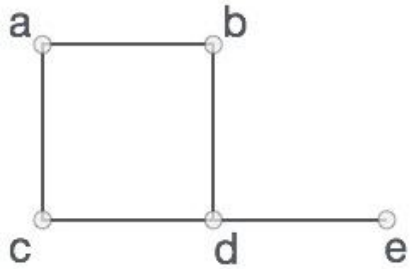


Fig.1 Graph Structure

From the graph shown in Fig. 1, we can write that

V = {a, b, c, d, e}

E = {ab, ac, bd, cd, de}

Adjacency relation: Node B is adjacent to A if there is an edge from A to B.

Paths and reachability: A path from A to B is a sequence of vertices A1… A such that there is an edge from A to A1, from A1 to A2… from An to B. Vertex B is said to be reachable from A if there is a path from A to B.

B. Graph Operations

The basic primary operations provided by a graph data structure are as follows:

- Addition of a vertex: adding a vertex to a graph when needed.
- Removal of a vertex: removing an existing vertex from the graph.
- Get vertex value: returns the value linked with a particular vertex.
- Set vertex value: assigns the value for a particular vertex.
- Addition of an edge: adding an edge between two vertices of a graph.
- Removal of an edge: removing an edge that exists between two vertices as required.
- Get edge value: returns the value linked with the edge joining any two vertices.
- Set edge value: assigns the value to an edge connecting the given vertices.
- Displaying a vertex: display required vertex of a graph.
- Finding all the neighboring vertices Y such that there exists an edge from the vertex X to vertex Y.
- Testing whether vertex X is adjacent to vertex Y which means confirming the existence of an edge from X to Y.
- Counting the number of vertices and number of edges present in the given graph.

C. Types of Graphs

1) Directed Graph: A directed graph is a graph where all the edges are directed from one vertex to another. The order of vertices in the pairs in the edge set matters in this

DOI 10.17148/IJARCCE.2016.5657

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 6, June 2016

type of graph. Thus, a is adjacent to b only if the edge set consists of a pair (a, b). In directed graph edges are drawn as arrows indicating the direction. A directed graph is sometimes called a digraph or a directed network. Directed graph can be cyclic or acyclic. Cycle is a path along the directed edges from a vertex to itself. Example of directed graph is shown in the Fig. 2.

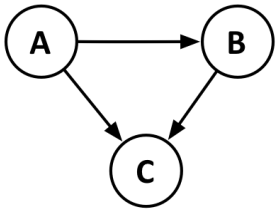


Fig. 2 Directed graph

2) Undirected Graph: A directed graph is a graph where all the edges are bidirectional. The order of vertices in the pairs in the edge set doesn't matter in this type of graph. In undirected graph edges are drawn as straight lines. Example of undirected graph is shown in the Fig. 3.

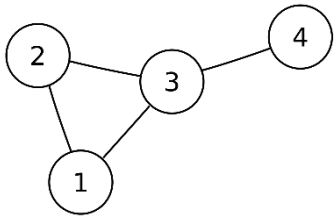


Fig. 3Undirected graph

3) Weighted Graph: A weighted graph is a graph where each edge has an associated numerical value, called weight. Weighted graphs may be either directed or undirected. The weight of the edge is often referred to as the "cost" of the edge. Example of weighted graph is shown in the Fig. 4.

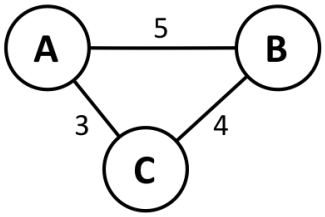


Fig. 4 Weighted graph

4) Space Graphs and Dense Graphs: Consider a graph having n nodes. A graph is said to be a sparse graph if it has less than n^2 edges. For example, a graph with n nodes and n edges or even 2n edges is said to be sparse. Whereas, a graph with close to maximum number of edges is said to be dense.

III.GRAPH REPRESENTATION

Graph is a mathematical structure and finds its applications in various Computer fields. The graph problems should be represented in computer memory using suitable data structures. The choice of graph representation is said to be situation specific. It totally depends on the type of operations to be performed and ease of use. Simple way to represent a graph is using Edge List.

A. Edge List: This structure simply maintains and stores the vertices and the edges into unsorted sequences.
Advantage: Easy to implement and iterate over small edges.

Disadvantage: Finding the edges incident on a given vertex is inefficient since it requires examining the entire edge sequence. That means –

- Difficult to tell how many edges a vertex touches.
- Difficult to tell if an edge exists say from A to B.

Further, Adjacency list and Adjacency matrix are the two standard and widely used ways for the representation of a graph.

B. Adjacency List: This list structure extends the edge list structure by adding incidence containers to each vertex.
Here an array of linked lists is used. Array size will be equal to the number of vertices. Consider an array A[ ]. An entry A[i] represents the linked list of vertices adjacent to the ith vertex. In these, vertices are stored as objects. Each vertex further contains a list of neighboring vertices. This type of representation allows additional data of the vertices to be stored. But these additional data is stored only if edges are stored as objects that mean every vertex store its incident edges and edge stores its incident vertices.
Another representation could be maintaining two lists. First list stores indices corresponding to each vertex in the graph and each of these refer to the second list storing the index of each adjacent vertex to this one. It would be good if we associate weight of each edge with the adjacent vertex in this list. These lists of all the vertices in the graph would be useful if stored in a hash table.
It is also used to represent a weighted graph. The nodes of linked lists will be storing weights of edges. Each node has precisely as many nodes in its adjacency list as it has neighbors. Therefore, an adjacency list is a very space efficient representation of a graph. You would not store more information than actually required.
If a graph has V vertices and E edges then a graph represented using adjacency list will need –

- V+E node instances for a directed graph
- V+2E node instances for an undirected graph

This is generally recommended if it efficiently represent sparse graphs.

Advantages of using adjacency list are as follows:

- Addition of a vertex and connecting new vertices with the existing ones is easier.
- Has a space complexity of O(|V|+|E|).
- It allows us to store graph in more compact form and to get the list of adjacent vertices in O(1) time which is a big benefit for some algorithms.

DOI 10.17148/IJARCCE.2016.5657

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 6, June 2016

Disadvantages of using adjacency list are as follows:

- Queries like whether there is an edge from vertex u to vertex v are not efficient and require time complexity, O(V).
- It does not allow us to make an efficient implementation, if dynamic change of vertices number is required.

Example of adjacency list representation is as shown in Fig. 5.

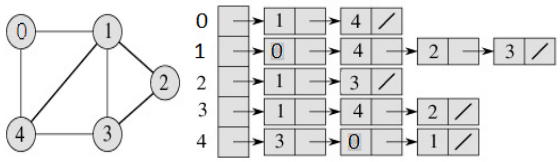


Fig5 Adjacency List Representation of a given graph

C. Adjacency Matrix: The adjacency matrix structure augments the edge list structure with a matrix where each row and column corresponds to a vertex. It is a two dimensional matrix form where the rows represent source vertices and columns represent destination vertices. Data on edges and vertices is stored externally. Between each pair of vertices, cost of one edge is to be stored. This shows which vertices are adjacent to one another. We know that two vertices are said to be adjacent if there is an edge connecting them. For a graph of n vertices, the dimensions of adjacency matrix will be n*n.

In case of directed graph, suppose if vertex j is adjacent to vertex i then there will be an edge from i to j and vice-versa. For a given vertex i, its adjacent vertices will be determined by looking at the row entry (i, [1…n]) of the matrix. If the value is true then it indicates that there exists an edge from i to j and false indicates no edge exists.

In case of undirected graph, the matrix values will be populated with Boolean values. The values of (i, j) and (j, i) are equal which means adjacency matrix for undirected graph is always symmetric along the diagonal.

In weighted graph, the Boolean values will be the costs of the edges connecting two vertices of a graph. Generally adjacency matrix is used to represent weighted graphs. If adj[i][j] = w , then we will say that there is an edge from vertex i to vertex j with weight w (can be any positive number). There are some cases where zero can also be the possible weight of the edge, then we have to store some sentinel value for non-existent edge, which can be a negative value; since the weight of the edge is always a positive number.

An adjacency matrix requires an n^2 element array so for parse graphs much of the matrix will be empty. Also, for undirected graphs half of the graph is repeated information. Hence these matrices are said to be space inefficient.

The memory use of an adjacency matrix is $O(n^2)$ where n = number of vertices.

Advantages of using adjacency matrix are as follows:

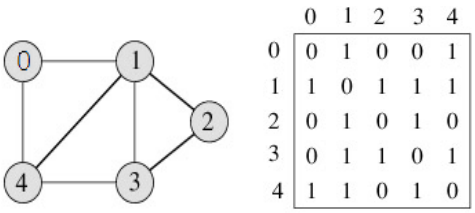
- Easy to understand, implement and convenient to work with.
- Removing an edge involves time complexity of O(1).
- Queries like whether there is an edge from vertex ‘u’ to vertex ‘v’ are efficient and require time complexity, O(1).

Disadvantages of using adjacency matrix are as follows:

- Space complexity is of the $O(V^2)$ where V = number of vertices.
- Sparse matrix has less number of edges but the space complexity is still the same.
- Adding an edge involves time complexity of $O(V^2)$.
- If the number of nodes in the graph may change, matrix representation is too inflexible (especially if we don’t know the maximal size of the graph).

This is preferred if the graph is of dense type where $|E| \sim V^2$.

Example of adjacency matrix representation is as shown in Fig. 6.



	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0

Fig. 6. Adjacency Matrix Representation of a given Graph

We can say that Although the linked list representation requires very less memory as compared to the adjacency matrix, the simplicity of adjacency matrix makes it preferable when graph are reasonably small.

Adjacency information in an array can be viewed as a function.

1) Merits of representing graphs as functions are as follows:

- Simple and easy to understand.
- Easily adaptable to different types of graphs.

2) Demerits of representing graphs as functions are as follows:

- Graph must be known statistically at compile time.
- Cannot be extended to accommodate queries about the set of Vertices or the set of Edges.

One way to overcome the cons of using functions to represent graph is to use arrays instead.

1) Merits of representing graphs as arrays are as follows:

- Simple and easy to understand and easily adaptable to different types of graphs.
- Can be accessed efficiently and constructed at run-time.

2) Demerits of representing graphs as arrays are as follows:

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 6, June 2016

- The domain of Vertices must be a type that can be used as an index into an array.
- Requires that graph access be a Command rather than a computation

IV. GRAPH TRAVERSAL TECHNIQUES

Graph traversal means visiting all the nodes of the graph. A structured system is required by many application of graph to examine the vertices and edges of a graph. There are two graph traversal methods as follows:

1. Breadth First Search (BFS)
2. Depth First Search (DFS)

Breadth First Search:

Given an input graph G = (V, E) and a source vertex S, from where the searching starts. We mark the vertex S as visited and then visit all of its adjacent nodes. Now one of the adjacent node is selected for exploration. The procedure is repeated until all the nodes are visited. Thus, BFS systematically traverse the edges of G to explore every vertex that is reachable from S. A queue is used to keep a track of the progress of traversing the neighbour nodes. Implementation of BFS is as shown in the Fig. 7.

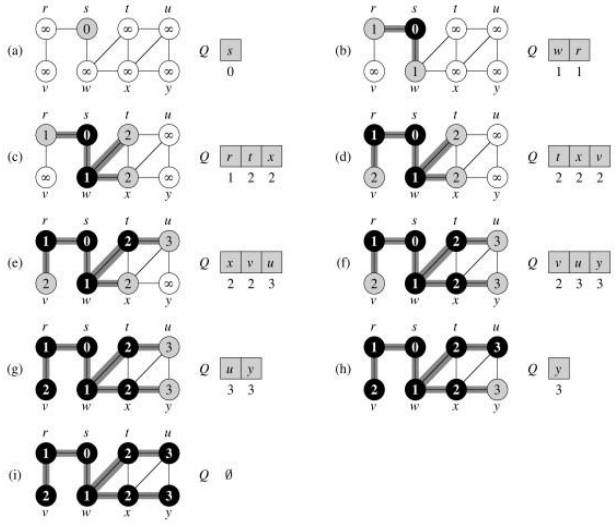


Fig.7 Implementation of BFS

Algorithm:

1. Input the vertices of the graph and its edges G = (V, E)
2. Input the source vertex and assign it to the variable S.
3. Add or push the source vertex to the queue.
4. Repeat the steps 5 and 6 until the queue is empty (i.e., front > rear)
5. Pop the front element of the queue and mark it as visited.
6. Push the vertices, which is neighbor to just popped element, if it is not in the queue and is not visited.

Depth First Search:

Given an input graph G = (V, E) and a source vertex S, from where the searching starts. We mark the vertex S as visited and then visit one of its adjacent nodes. We will mark this adjacent node as visited. The procedure is repeated until all the vertices of the graph are visited recursively. A stack is used in the implementation of DFS. Implementation of DFS is as shown in the Fig. 8.

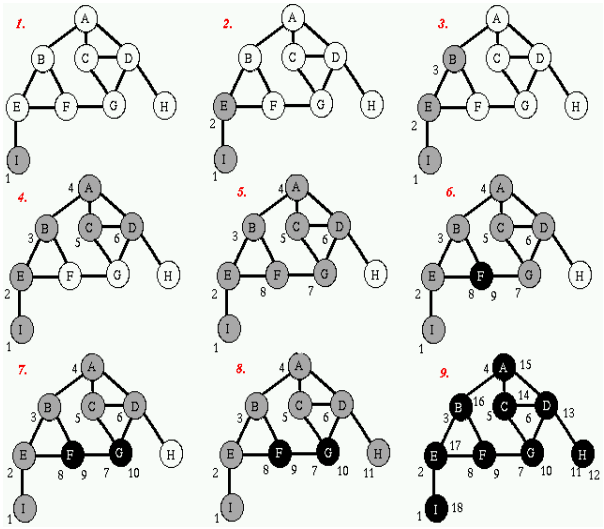


Fig. 8 Implementation of DFS

Algorithm:

1. Input the vertices and edges of the graph G = (V, E).
2. Input the source vertex and assign it to the variable S.
3. Push the source vertex to the stack.
4. Repeat the steps 5 and 6 until the stack is empty.
5. Pop the top element of the stack and mark it is visited.
6. Push the vertices which is neighbour to just popped element, if it is not in the stack and is not visited.
7. Exit.

V. APPLICATIONS OF GRAPHS

- In road networks, we can consider the intersections as vertices and the road segments between them as the edges. Many map programs such as Google maps, Bing maps and Apple IOS 6 maps makes use of such networks to find the best routes between locations. They are used for studying traffic patterns, traffic light timings and many aspects of transportation.
- Directed graph can be used to map the links between pages within a website. In this case each web page is a vertex and each hyperlink is a directed edge. These graphs are also used to analyse ease of navigation between different parts of the site.
- In case of power grid and water network, vertices represent connection points and edges represent the wires or pipes between them. Graphs can be used to minimize the cost to build this infrastructure that matches the required demands.
- Scene graphs represent the logical or spatial relationships between objects in a scene. Scene graphs are widely used in graphics and computer games industry.
- Graph theory is also widely used in sociology as a way, for example, to measure actors' prestige or to explore rumour spreading, notably through the use of social network analysis software.

DOI 10.17148/IJARCCE.2016.5657

- A common problem in AI is to find some goal that satisfies a list of constraints. For example, for a University to set its course schedules, it needs to make sure that certain courses don't conflict, that a professor isn't teaching two courses at the same time, that the lectures occur during certain timeslots, and so on. Constraint satisfaction problems like this are often modelled and solved using graphs.

VI. CONCLUSION

Graphs are a commonly used data structure because they can be used to model many real-world problems. The graph makes large data quite simpler to work with. Graphs are a very effective visual tool because they have the capacity to present the information quickly as well as easily. Graphs have the ability to reveal a trend or comparison. That is the main reason why the graphs are commonly used by different media and also in business. Thus various representations of a graph, advantages and disadvantages and their applications have been studied.

REFERENCES

[1] Danny Sleator, "Parallel and Sequential Data Structures and Algorithms,15-210 (fall 2013) ", Sept. 24 , 2013.
[2] Nykamp DQ, "Undirected graph definition" on Math Insight. Available: http://mathinsight.org/definition/undirected_graph
[3] MSDN contributors, Available: https://msdn.microsoft.com/en-us/library/aa289152(v=vs.71).aspx
[4] Wikipedia contributors, Available: https://en.m.wikipedia.org/wiki/Graph_(abstract_data_type)
[5] Wiki books contributors, Available: https://en.m.wikibooks.org/wiki/Data_Structures/Graphs
[6] Tutorials point contributors, Available: http://www.tutorialspoint.com/data_structures_algorithms/graph_data_structure.htm
[7] Graphs in computer science, Available: http://web.cecs.pdx.edu/~sheard/course/Cs163/Doc/Graphs.html
[8] Slideshare contributors, Available: http://www.slideshare.net/AbhishekPachisia/matrix-representation-ofgraph?next_slideshow=1
[9] Stackoverflow contributors, Available: http://stackoverflow.com/questions/703999/what-are-good-examples-of-problems-that graphs-can-solve-better-than-the-alterna
[10] Ds lecture notes on Graph, Available: http://www.ggu.ac.in/download/Class-Note13/ds%20lecture%20notes%20graph12.11.13.pdf.

DOI 10.17148/IJARCCE.2016.5657