JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue

JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)

An International Scholarly Open Access, Peer-reviewed, Refereed Journal

Study on Traversal of Graphs Based on Complexity.

Chintan Rana, Dr viral Parekh, Dr Harsha Padheriya

Chintanrana606@gmail.com,viralparekh@sal.edu.in,hpadheriya@gmail.com

SALITER, Assistant Professor, SAL College of Engineering

ABSTRACT

This research focuses on the depth-first search (DFS) and breadth-first search (BFS) techniques used in data structures and gives an idea of how complex they are. In addition to the O-notation, the essay also looks at spatial and temporal complexity. This research study initially examines BFS and DFS-based graph and tree traversal in order to reach our result.

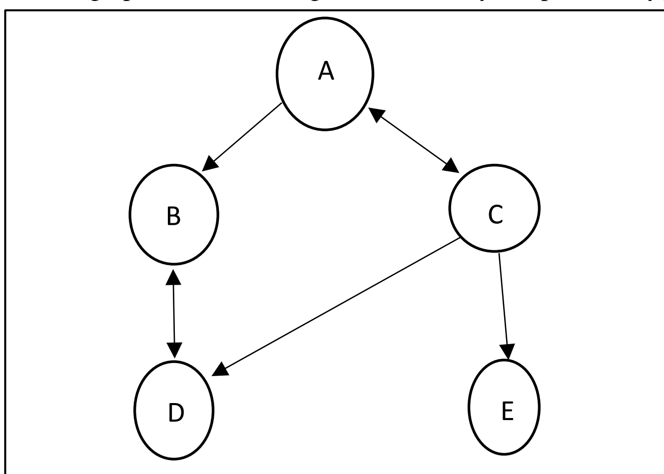
Keywords—BFS, DFS, time complexity, space complexity, O-notation

I. Introduction

Graphs are among the most intriguing data structures in computer science, and they are crucial to computing. In terms of traversing a graph, BFS and DFS are the two most often used techniques. There are several structural similarities between graphs and trees. The truth is that the tree originated from the graph data structure. To do tasks like searching for a certain node, one can traverse the graph. It may also be extensively changed to find a path between two nodes, check for links in the graph, seek for loops, and so forth. Using graphs, one may model real-world problems such as imitating the actions of air traffic controllers, simulating the movement of cities connected by highways, and more.

II. Graph

A triple called a graph (G) is made up of a vertex set (V), an edge set (E), and a relation connecting two vertices known as the endpoints of each edge (not necessarily distinct). G is formally stated as G = (V, E), where V is a set .If all of the variables v and w are equivalent to V: (v, w) E (w, v) E, then G = (V, E) undirected. The message is directed if not. There are no loops or many edges in a basic graph. Here, each edge e in E(G) may be specified by one of the endpoints u ,v in V (G).



[Figure 1: Diagram of graph]

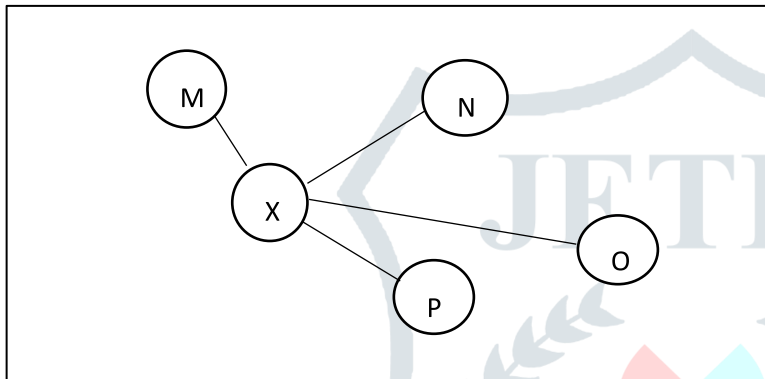
The nodes A, B, C, D, and E make up the graph's set of nodes, while its set of arcs is made up of { (A, B),(A, C),(B, D),(C, E), and (C, D). The graph's directed and ordered arcs are made up of a pair of nodes.

Graph theory is mostly used in computer applications to create graph algorithms. There are several methods for solving problems that may be represented as graphs. These techniques must first be used to resolve the theoretical problems connected with graphs before tackling the related computer science practical problems may be addressed. Algorithms include the following:

- Algorithms for finding a data structure's element (DFS, BFS).
- In a network, the shortest route algorithm.
- A minimal spanning tree's discovery.
- Identifying graph planarity.
- Algorithms to find adjacency matrices.
- Algorithms to find the cycles in a graph and so on.

III. TREES

The tree concept is especially significant when used in graph applications. The graph of a tree is not closed. A tree cannot have parallel or self-looping edges, as is made plain by the definition, and it must instead be a simple graph.



[**Figure 2 : Diagram of A Tree**] A Tree has the

following properties:

- In a tree, T, there is only one path that connects every pair of vertices.
- An n-vertex tree has n-1 edges.
- A tree is any linked graph with n vertices and n-1 edges.
- If and only if a graph is minimally linked, it qualifies as a tree.

IV. BFS

The term BFS refers to breadth-first search (BFS). Both the graph traversal method and the BFS traversal approach visit all descendants of a visited node before visiting any descendants of those descendants. BFS typically results in very tall, slender trees. BFS is implemented using a queue in order to express the fact that the first node visited is also the first node whose successors are visited. The BFS method uses a queue, which is initially empty, assum-ed to be the case. The array mark's entries are assumed to be all unique and have never been visited. If the graph is disconnected, then BFS must be called on each node of each linked component. It is necessary to label a node before adding it to the BFS queue. thereby preventing it from being added to the queue more than once. When the queue is empty, the algorithm ends.

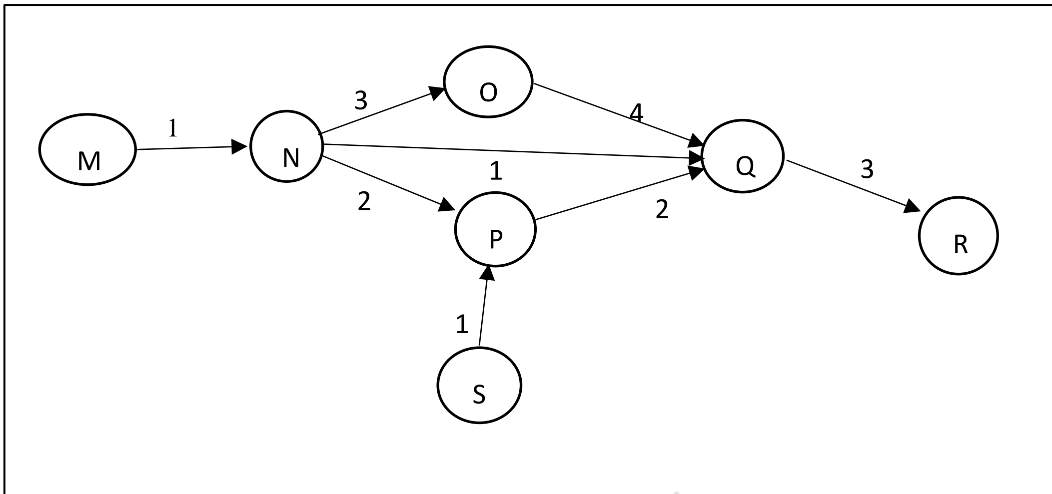
A BFS spanning tree doesn't have any forward edges since every node near a visited node has been visited before or is a son of a visited node in a spanning tree. Every cross edge within a single tree in a directed graph connects to a node either higher up the tree or at the same level as the tree. Given that every back edge is also a forward edge on an undirected graph, a BFS spanning has no back edges. The preceding paragraph of BFS creates a tree.

The algorithm for BFS is as below:

i. Print v, mark v as visited, and add v to the queue.
ii. Repeat steps 3 to 5, while queue is not empty
iii. Remove anything from the queue, then give it to v.
iv. Assign the address of adjacent list of v to adj.
v. Repeat steps a to c, while adj ≠NULL.
vi. A. Assign node of adj to v.

B. If node v is unvisited, then mark v visited, print v and insert v into the queue.
C. Assign the next node pointer of adjacent list to adj. [adj=adj→next].

vii. Exit



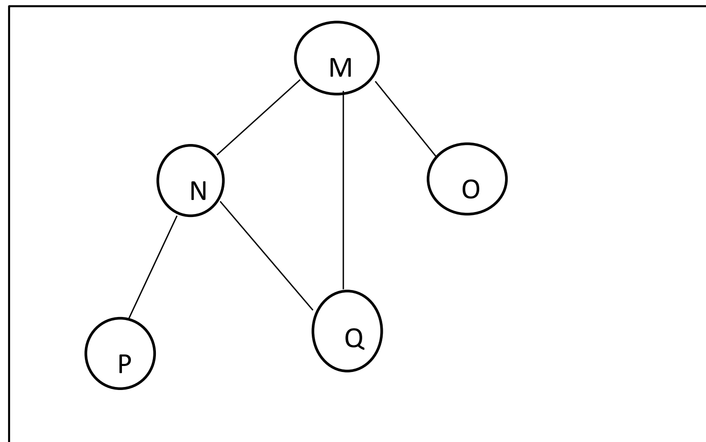
[**Figure 3 : BFS traversal**]

The top node, M, should be visited first. The following step involves going to the neighbouring, unvisited node of M and N. Then look for O, P, and Q, N's nearby unexplored nodes. A search for nearby nodes finds no additional unexplored nodes in the area of O. R was found to be Q's unvisited neighbouring node when Q was verified after that. There is a visitation record for . Upon arriving at P, you noticed that it shared a border with the previously unvisited node S, making S's status change to visited. We now know that all nodes have been traversed. The output of the above-described tree traversal is M,N,O,P,Q,R and S.

V. DFS

The term "DFS," or "depth-first search" (DFS). In contrast to a breadth first search, a depth first search of a graph stops a vertex v's investigation as soon as a new vertex is discovered. Investigating the new vertex u at the moment. Once this new vertex is studied, the study of u is continued. The search is complete once every accessible vertex has been carefully examined. Recursive definition of this search method is the simplest. Every time it is practical, DFS uses a technique that looks "deeper" in the graph. The predecessor subgraph that DFS generates could include many trees since it's possible for the search to be repeated from a variety of sources.

```
Algortihm of DFS(V)
{
        visited [v]=1
        For each vertex w adjacent from v do
        {
                If (visited[w]=0) then
                DFS(w);
        }
}
```



[figure 4 : DFS traversal]

Go to node A in the top row and mark it as visited first. Then look for A, B, and C's surrounding undiscovered nodes. B is visited and it is documented as such. Search for nodes that are near to node B next, locate and visit D, then go on to E. Adjacent, the unvisited node C, which is next to A, is visited. The outcomes of the tree traversal mentioned above are A, B, D, E, and C. To find acyclic graphs, we use DFS. In both directed and undirected graphs, a cycle exists if a back edge is present in a DF forest. Using a DFS traversal, the nodes may be sorted in reverse topological order.

VI. ALGORITHM EFFICIENCY AND COMPLEXITY

1. The O Notation

Sequential processes known as algorithms may make use of a variety of resources, depending on the particular instance of the issue they are used to solve (time, space, etc.). As a result, O-notation, a method for analysing an algorithm's resource use, is referred to as a worst-case analysis since it consistently offers bounds that hold true for even the worst events of a given magnitude. A polynomial time bound algorithm is considered time efficient if and only if its runtime is O(np) for some p N. Only when an algorithm has a logarithmic space constraint, or when its runtime is O(log p(n)), can it be said to be space-efficient.

2. Complexity (Space, Time)

Whether an algorithm works quickly or slowly affects how difficult it is to compute. Complexity is defined as the relationship between the time variable T(n) and the input size n in mathematics. We want to quantify the time required by an algorithm without relying on implementation details. The implementation affects T(n). Depending on variables like processor speed, instruction set, disc speed, compiler brand, and others, a similar approach may need varying amounts of time for the same inputs. A workaround involves asymptotically calculating the performance of each method. We'll define time T(n) as the amount of elementary "steps," assuming that the amount of time needed for each basic "step" is constant (specified in any way).

Let's examine two examples from the past when two numbers were added. By digit-by-digit adding two integers, we will generate a "step" in our computational model (or bit by bit). Therefore, we say that there are n stages involved in adding two n-bit integers. Thus, T(n) = c * n, where c is the amount of time needed to add two bits, gives the overall computation time. Because adding two bits, let's say c1 and c2, might take different amounts of time on different machines, the time needed to add two n-bit integers is T(n) = c1 * n and T(n) = c2 * n, respectively. This shows that different machines produce different slopes but that time T(n) increases linearly with input size.

One of the core concepts in computer science is the method of abstracting away specifics and calculating the rate of resource utilisation in terms of input size.

VI.2.1 Space Complexity: Space needed by algorithms is the sum of the following components :-

- a fixed component that is unaffected by factors like the quantity and size of inputs and outputs. This section often contains the instruction space (code space), simple variable space, fixed size component variable space, constant space, and so on.
- A variable route contains the space needed for the recursion stack, the space for referenced variables, and component variables whose sizes are determined by the particular problem instance being handled.

The space requirement S(P) of any algorithm P may therefore be written as S(P) c + Sp (instance characteristics) where c is a constant.

VI.2.2 Time Complexity: The time T(p) taken by a program P is the sum of compile time and the run time (execution time). The compile time does not depend on the instance characteristics. Run time is denoted by tp. Tp(n)=ca ADD(n) + cs SUB(n) + cm MUL(n) + cd DIV(n)+….

ADD, SUB, MUL, DIV, and so forth are functions whose values are the number of additions, subtractions, multiplications, divisions, and so forth that are carried out when codes for P are used on an instance containing the letter n. The time required for addition, subtraction, multiplication, division, and other operations is indicated by the letter n, which also stands for instance properties such as ca, cs, cm, cd, and so on.

Only by experimentation can the value of tp(n) for every given n be determined. On a specific computer, the programme is typed, compiled, and executed. Tp(n) is calculated after physically clocking the execution time. The length of time it takes for a programme to execute on a multi-user system varies on variables like system load and the number of concurrently executing processes.

3. Complexity of BFS and DFS:-

If we measure the efficiency of DFS algorithm, by traversing the successors of all the nodes, it is O(n^2). Therefore DFS search using adjacency matrix representation efficiency is O(n+n^2).

If the adjacency list representation is used, traversing each successor of every node is O(e), where e is the number of edges in the graph. Assuming the graph nodes are organised in an array or a link list, the efficiency of DFS traversal using adjacency lists is O(n+e), as traversing all n nodes is O. (n).

As long as each node is only visited once, BFS is as successful as DFS at including all arcs originating from it.
It is therefore O(n2) efficient for the adjacency matrix form and O(n+e) efficient for the adjacency list format.

VII. Conclusion

The challenges that traversal techniques like BFS and DFS provide are discussed in this work. Graphs, trees, and other graph algorithms are addressed here as a concept. In order to traverse a graph efficiently, it is essential to keep track of the graph's time and space complexity. The author will get some understanding of various traversals after having a solid understanding of complexity.

REFERENCES

[1] Graphs :Representation and Exploration by Julian Mestre.

[2] Seymour Lipschutz and G A Vijayalakshmi Pai, Data Structures, Tata McGraw Hill Publishing Company Limited, 2005.

[3] Ankit Malhotra, Dipit Malhotra, Saksham Kashyap "Optimized Proposed Algorithm for Graph Traversal "International Journal of Computer Applications (0975 – 8887) Volume 104 – No.9, October 2014.

[4] Breadth First Search Wikipedia.

[5] Graph Theory with Applications by J.A.Bondy and U.S.R. Murty.

[6] Depth First Search Wikipedia.

[7] Complexity Based on Traversal of Graphs by shameek mukhodhyay(2016).