Chahat Monga *et al*, International Journal of Computer Science and Mobile Applications,
Vol.6 Issue. 1, January- 2018, pg. 38-42 **ISSN: 2321-8363**
Impact Factor: 5.515

GRAPH TRAVERSALS AND ITS APPLICATIONS IN GRAPH THEORY

Chahat Monga[1], Richa[2]

[1]Guru Nanak College, Department of Computer Science and Applications, Ferozepur, Punjab, India
[2]Punjabi University, Department of Computer Science, Patiala, Punjab, India

ABSTRACT

This paper focuses on Graphs Traversal Algorithms Breadth First Search (BFS) and Depth First Search (DFS) used in data structure and also gives an idea of complexity. The time complexity and space complexity are discussed here along with the O-notation. This research paper provides a study of graph, and its traversal based on BFS and DFS briefly and also define its applications where these traversals are helpful in graph theory.

Keywords— BFS, DFS, complexity, Graphs

INTRODUCTION

Data structure plays an important role in computing and graphs are one of the most interesting data structures in computer science. BFS and DFS are the two most common approaches to graph traversal. Graphs and the trees are somewhat similar by their structure. In fact, tree is derived from the graph data structure. Traversal of the graph is used to perform tasks such as searching for a certain node. It can also be slightly modified to search for a path between two nodes, check if the graph is connected, and check if it contains loops, and so on. Graphs are good in modelling real world problems like representing cities which are connected by roads and finding the paths between cities, modelling air traffic controller system, etc. Based on the traversal, different complexities arise.

Graphs are the commonly used data structures that describe a set of objects as nodes and the connections between them as edges. A large number of graph operations are present, such as Bellman ford minimum spanning tree, breadth-first search, shortest path etc., having applications in different problem domains like data mining[1], and network analysis[2], route finding[3] , game theory[4]. With the development of computer and information system, the research on graph algorithm is wide opened. The two most widely used algorithms used for traversing a graph are Breadth First Search and Depth First Search [5]. The BFS begins at a root node and inspects all the neighbouring nodes. Then for each of those neighbour nodes in turn, it inspects their neighbour nodes which were unvisited [6], and so on, while the DFS remove a vertex from a container, prints it and finds all the adjacent vertices. On finding the adjacent vertices it marks them visited and enter them into a stack. The algorithm is repeated as long as the stack is not empty. So far, many different variants of BFS and DFS algorithm have been implemented sequentially as well as in parallel manner. In all parallel implementations, the unvisited nodes of the root node are visited, but in our implementation, the node. With the maximum number of edges if processed first and then its neighbouring nodes are processed. This paper presents a new algorithm for traversing a graph and then compares the traversal result of a directed and undirected graph using BFS, DFS and the new algorithm. The traversal path obtained by employing all the techniques is shown and then a comparison of the time and space complexity of all the algorithms is done.

GRAPHS TRAVERSAL ALGORITHMS

BFS (Breadth First Traversal): BFS is a graph traversal algorithm and traversal method which visits all successors of a visited node before visiting any successors of any of those successors. BFS tends to create very wide and short trees. BFS is implemented using a queue, representing the fact that the first node visited is the first node whose successors are visited. BFS algorithm inserts a node into a queue, which we assume is initially empty. Every entry in the array mark is assumed to be unvisited. If the graph is not connected, BFS must be called on a node of each connected component. In BFS we must mark a node visited before inserting it into the queue, so as to avoid placing it on the queue more than once. The algorithm terminates when the queue becomes empty. A BFS spanning tree does not have any forward edges, since all nodes adjacent to a visited node have already been visited or are spanning tree sons of node. For a directed graph, all cross edges within the same tree are to nodes on the same or higher levels of the tree. For an undirected graph a BFS spanning contains no back edges since every back edge is also a forward edge. The predecessor sub graph of BFS forms a tree.

The algorithm for BFS is as below:

```
1. Mark v visited, print v, and insert v into queue. (v: head node)
2. Repeat steps 3 to 5, while queue is not empty.
3. Delete an item from the queue and assign it to v.
4. Assign the address of adjacent list of v to adj. [adj is a pointer variable of node type]
5. Repeat steps a to c, while adj ≠NULL.
        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]
6. Exit
```

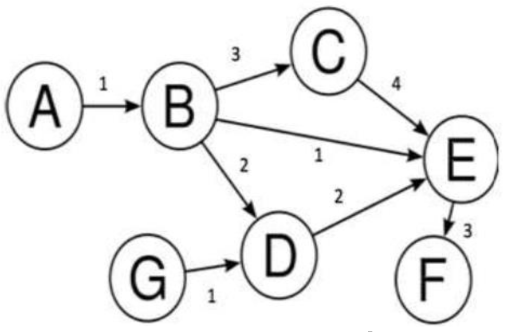

Fig: BFS Traversal

Visit the top node A first. In the next step visit the adjacent unvisited node of A which is B. Then look for adjacent unvisited node of B which are C, E, and D. Check for adjacent nodes of C and found out that there is no unvisited node left. Then check for E and found F as the unvisited adjacent node of E. F is marked as visited. Move on to D and found that it also has an unvisited node adjacent to it which is G and thus G is marked as visited. Now it is realised that all the nodes have been visited. So the output of the above tree traversal is A, B, C, E, D, F, G.

DFS (Depth First Traversal): A depth first search of a graph differs from a breadth first search in that the exploration of a vertex v is suspended as soon as a new vertex is reached. At this time, exploration of the new vertex u begins. When this new vertex has been explored, the exploration of u continues. The search terminates when all reached vertices have been fully explored. This search process is best described recursively. DFS uses a strategy that searches "deeper" in the graph whenever possible. The predecessor sub graph produced by DFS may be composed of several trees, because the search may be repeated from several sources. This predecessor sub graph forms a depth-first forest E composed of several depth-first trees and the edges in E are called tree edges.

The algorithm for DFS is as below

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

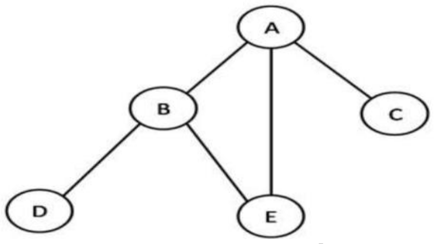

Fig: DFS Traversal

First visit the top node A and mark it as visited. Then look for the adjacent unvisited nodes of A and B and C are found. B is visited and is marked as visited. Then look for adjacent nodes of B, D is found and is marked as visited, and then E is visited next. Then the unvisited adjacent node of A is visited which is C. So the output of the above tree traversal is A, B, D, E, C. DFS is use to determine acyclic graphs. In both directed and undirected graphs, a cycle exists if a back edge exists in a DF forest. A DFS traversal can be used to produce a reverse topological ordering of the nodes.

***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) [n node visits and n^2 possible successors examinations].

If the adjacency list representation is used traversing all successors of all nodes is O(e), where e is the number of edges in the graph. Assuming that the graph nodes are organized as an array or a link list , visiting all n nodes is O(n), so that the efficiency of DFS traversal using adjacency lists is O(n+e).

The efficiency of BFS is the same as that of DFS: each node is visited once and all arcs emanating from every node are considered. Thus its efficiency is O (n^2) for the adjacency matrix graph representation and O (n+e) for the adjacency list graph representation.

APPLICATIONS OF BFS

1. Shortest Path and Minimum Spanning Tree for unweighted graph: In unweighted graph, the shortest path is the path with least number of edges. With Breadth First, we always reach a vertex from given source using minimum number of edges. Also, in case of unweighted graphs, any spanning tree is Minimum Spanning Tree and we can use either Depth or Breadth first traversal for finding a spanning tree.
***2. Peer to Peer Networks*:** In Peer to Peer Networks like BitTorrent, Breadth First Search is used to find all neighbour nodes.
3. Crawlers in Search Engines: Crawlers build index using Breadth First. The idea is to start from source page and follow all links from source and keep doing same. Depth First Traversal can also be used for crawlers, but the advantage with Breadth First Traversal is, depth or levels of built tree can be limited.
4. Social Networking Websites: In social networks, we can find people within a given distance ‘k’ from a person using Breadth First Search till ‘k’ levels.
5. GPS Navigation systems: Breadth First Search is used to find all neighbouring locations.

6. Broadcasting in Network**:** In networks, a broadcasted packet follows Breadth First Search to reach all nodes.
7. In Garbage Collection**:** Breadth First Search is used in copying garbage collection using Cheney's algorithm. Refer this and for details. Breadth First Search is preferred over Depth First Search because of better locality of reference:
8. Cycle detection in undirected graph: In undirected graphs, either Breadth First Search or Depth First Search can be used to detect cycle. In directed graph, only depth first search can be used.
9. Ford–Fulkerson algorithm*:* In Ford-Fulkerson algorithm, we can either use Breadth First or Depth First Traversal to find the maximum flow. Breadth First Traversal is preferred as it reduces worst case time complexity to $O(VE^2)$.
10. To test if a graph is Bipartite: We can either use Breadth First or Depth First Traversal.
11. Path Finding: We can either use Breadth First or Depth First Traversal to find if there is a path between two vertices.
12. Finding all nodes within one connected component: We can either use Breadth First or Depth First Traversal to find all nodes reachable from a given node.
Many algorithms like Prim's Minimum Spanning Tree and Dijkstra's Single Source Shortest Path use structure similar to Breadth First Search.
There can be many more applications as Breadth First Search is one of the core algorithms for Graphs.

APPLICATIONS OF DFS

1. For an unweighted graph, DFS traversal of the graph produces the minimum spanning tree and all pair shortest path tree.
2. Detecting cycle in a graph**:** A graph has cycle if and only if we see a back edge during DFS. So we can run DFS for the graph and check for back edges.
3. Path Finding*:* We can specialize the DFS algorithm to find a path between two given vertices u and z
i) Call DFS(G, u) with u as the start vertex.
ii) Use a stack S to keep track of the path between the start vertex and the current vertex.
iii) As soon as destination vertex z is encountered, return the path as the contents of the stack
4. Topological Sorting*:* Topological Sorting is mainly used for scheduling jobs from the given dependencies among jobs. In computer science, applications of this type arise in instruction scheduling, ordering of formula cell evaluation when re-computing formula values in spreadsheets, logic synthesis, determining the order of compilation tasks to perform in make files, data serialization, and resolving symbol dependencies in linkers [2].
5. To test if a graph is bipartite*:* We can augment either BFS or DFS when we first discover a new vertex, color it opposite to its parents, and for each other edge, check it doesn't link two vertices of the same color. The first vertex in any connected component can be red or black.
6. Finding Strongly Connected Components of a graph: A directed graph is called strongly connected if there is a path from each vertex in the graph to every other vertex.
7. Solving puzzles with only one solution*:* such as mazes. (DFS can be adapted to find all solutions to a maze by only including nodes on the current path in the visited set.)

CONCLUSION AND FUTURE WORK

We had offered a novel arrangement of breadth-first and depth-first search that allows a single search algorithm to acquire the matching strengths of both. Many of the ideas have the prospective to be applied to other search troubles, especially graph-search harms with large encoding sizes, for which memory-reference neighbourhood is the key to achieving good piece. Possibilities include model checking, where a large data structure that represents the current state is typically stored with each search node, and constraint-based forecast and scheduling, where a simple secular group is stored with each search node. As long as the similarities among unusual search nodes can be captured in a form that allows depth-first search to waste the state-representation locality in node expansions, the approach we have described could be ineffectual.

REFERENCES

[1] J. Abello, A. L. Buchsbaum, and J. R. Westbrook. A functional approach to external graph algorithms. *Algorithmica*, 32(3):437–458, 2002.
[2] P. Agarwal, L. Arge, M.Murali, K. Varadarajan, and J. Vitter. I/O-efficient algorithms for contour-line extraction and planar graph blocking. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 117–126, 1998.
[3] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, pages 1116–1127, September 1988.
[4] C. Gkantsidis, M. Mihail, and A. Saberi, "Random walks in peer-to-peer networks," in *Proc. of Infocom*, 2004.
[5] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou, "Walking in facebook: A case study of unbiased sampling of osns," *Infocom*, 2010.
[6] B. Krishnamurthy, P. Gill, and M. Arlitt, "A few chirps about twitter," in *Proc. of WOSN*, 2008.
[7] Jong Ho Kim, Helen Cameron, Peter Graham "Lock-Free Red-Black Trees Using CAS" October 20, 2011.

[8] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Cli_ord Stein.Introduction to Algorithms. MIT Press, 2001. Second Edition.

[9] Kurt Mehlhorn. Data Structures and Algorithms 1: Sorting and Searching. EATCS,Monographs on Theoretical Computer Science. Springer, 1984.