Swaroop Prakash Jadhav, International Journal of Computer Science and Mobile Applications,
Vol.10 Issue. 7, July- 2022, pg. 1-7 ISSN: 2321-8363
Impact Factor: 6.308
(An Open Accessible, Fully Refereed and Peer Reviewed Journal)

GRAPH TRAVERSALS AND ITS APPLICATIONS

Swaroop Prakash Jadhav
Department of Computer Applications,
Bharati Vidyapeeth Institute of Management Kolhapur, Maharashtra, India
swarupjadhav66@gmail.com

DOI: 10.5281/zenodo.6838779

ABSTRACT

In this paper we are going to focuses on graph traversal algorithm BFS and DFS. 1) BFS (Breadth First Search) or sometimes it is also known as level order traversal and 2) DFS (Depth First Search). This research paper provides a study of graph and its traversal, based on BFS and DFS briefly. And also defines its applications where these traversals are helpful in graph theory.

KEYWORDS – BFS, DFS, Graphs, and applications of BFS and DFS

INTRODUCTION

Graphs are one of the unifying themes of computer science-an abstract representation that describes the organization of transport systems, human interactions and telecommunication networks.
Data structures are simply ways of a organizing the data. They can be either linear of nonlinear. Array, QUEUE, Stack, linked list is the examples of the linear data structure. Graphs in a data structure are non-linear data structure consisting of nodes and edges. Graphs are normally defined as a pair of sets (VE). V is set of arbitrary objects call vertices or nodes, and E is set of pairs of vertices, which we call edges or arcs. Tree is a also special kind of graph where there are never many paths that for all possible combinations of A and B, there is always only one way to get from A to B. A graph System that has many possible paths to an arbitrary point A to another arbitrary point B.
But here in this paper it is all about looking into nonlinear data structure: Graph traversal.
Graphs is good in modeling real world problems for examples we all peoples using social media like Facebook, Instagram and so on, for example Facebook uses graphs to model relationships between nodes. We can apply graph algorithms to suggest friends to people, calculate no of mutual friends etc. other examples of graph include result of web crawl for a website or for the entire world wide web, city routes etc.
Graph traversal is a technique of visiting each node of the graph. It is also used to calculate the sequence of vertices in the traverse process. We visit all nodes starting from a connected node without going into loop. Therefore, we keep a record of each vertex to protect it from this infinite loop position. For ex- if visitors visit the vertex, the value will be zero, if not then one. Basically, in a graph it can happen that visitors can visit the

node more than once. Therefore, visitors likely to go into an infinite loop. A graph operations are present in large number, such as binary operations, depth first search, breadth first search, shortest path algorithm etc. graph data structure have a different applications in a different problem domains like 1] Road network 2] Blockchains 3] Bitcoin Transaction 4] Knowledge graphs 5] Product recommendation graphs etc. There is two most widely used algorithm for graph traversing techniques 1. BFS- Breadth First Search and 2. DFS- Depth First Search. BFS- Breadth first search graph traversal algorithm techniques use a QUEUE data structure as an auxiliary data structure to store nodes for further processing the size of the QUEUE will be the maximum total number of vertices in the graph. BFS begins at the root node and it inspects all the neighboring nodes. It inspects all their neighbor nodes which were unvisited. The DFS- Depth first search algorithm is one of the graphs that use stack data structure. In DFS Traversal go as deep as possible of the graph and then backtrack once reached a vertex that has all its adjacent vertices already visited. This paper presents a new algorithm for traversing a graph using BFS and DFS.

GRAPH TRAVERSAL ALGORITHMS

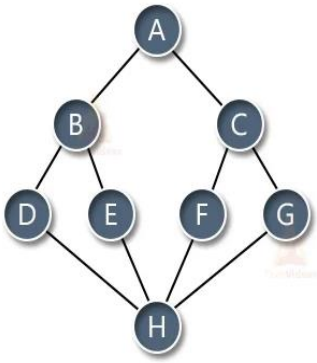
To visit each node or vertex which is a connected component, tree-based algorithms are used. We can do this easily by iterating through all the vertices of the graph. Performing the algorithm on each vertex that is still unvisited when examined.
Two algorithms are generally used for the traversal for the graph: Breadth First Search and Depth First Search

BFS (Breadth First Search): Breadth-first search is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighboring nodes. BFS involves in iterative loop over a queue of vertices. Exploration of graph components is achieved by dequeuing a vertex to discover unexplored vertices which are subsequently enqueued. It selects the nearest node and explores all the unexplored nodes. While using bfs for traversal, any node in the graph can be considered as the root node. It is recursive algorithm to search all the vertices of a tree or graph data structure. Starting at the first vertex, the graph is traversed as much possible; then it will go to the next vertex that has not been visited. It can be stated as all vertices that are adjacent to the current vertex are traversed first. A BFS spanning tree does not have any forward edges, all nodes adjacent to a visited node have already been visited. For a direct graph all cross edges in the same tree are nodes on some or higher levels of the tree. For undirect graph a BFS spanning contains no back edges since every back edge is also forward edge.

ALGORITHM FOR BFS

1. Choose any one node randomly, to start traversing.
2. Visit its adjacent unvisited node.
3. Mark it as visited in the Boolean array and display it.
4. Insert the visited node into the queue.
5. If there is no adjacent node, remove the first node from the queue.
6. Repeat the above steps until the queue is empty.
7. Exit.

Swaroop Prakash Jadhav, International Journal of Computer Science and Mobile Applications,
Vol.10 Issue. 7, July- 2022, pg. 1-7 ISSN: 2321-8363
Impact Factor: 6.308
(An Open Accessible, Fully Refereed and Peer Reviewed Journal)



Initially, the queue will be empty.

Start traversing the graph from node A. once visit node A, mark it as visited and also place it inside the queue.

The next is to traverse its adjacent nodes B and C and place them inside the queue. When we place the adjacent node of A in the queue, we will remove A from the queue and display it in the output.

Next, we don't have any more adjacent nodes for A. therefore, we remove node B from the queue and place its adjacent nodes into the queue. Similarly, we will traverse the nodes of C and put them into the queue.

The next adjacent node is H. thus, we will traverse that as well and place it in the queue. Once all the nodes have entered the queue, we will start removing them from the queue and putting them into the output array.

Slowly the queue starts decreasing in size and the output array will be full.

In this way, we get the output of BFS is

A→ B→C→D→E→F→G→H.

Pseudo-code for BFS:

```
Procedure BFS(Graph, root)is
Let root = explored
Queue.enqueue(root)
while(Queue is not empty):
V := Queue.dequeue()
If v == goal:
Return v
for all edges from v to w in Graph.adjacentEdges(v):
if w is not labeled as explored:
Label w as explored;
Queue.enqueue(w)
END for
END while
END BFS
```


Swaroop Prakash Jadhav, International Journal of Computer Science and Mobile Applications,
Vol.10 Issue. 7, July- 2022, pg. 1-7 ISSN: 2321-8363
Impact Factor: 6.308
(An Open Accessible, Fully Refereed and Peer Reviewed Journal)

Applications of Breadth First Search:

1. To find the shortest path between two edges when the path length is equivalent to the number of edges.
2. To copy garbage collection by Cheney's algorithm
3. To from peer-to-peer network connections
4. To broadcast packets in a network
5. To detect cycle in an undirected graph
6. Used in unweighted graphs to find the minimum cost spanning tree
7. To find neighboring locations in the GPS navigation system
8. To find all the nodes within one connected component in an other wise disconnected graph.
9. To check whether a graph is bipartite or not
10. In social networks, we can find people within a given distance 'k' from a person using BFS till 'k' levels.

Time complexity: Since we are visiting all the nodes exactly once, therefore, the time complexity is **O(V+E).** Here, O(E) may vary between O(1) and O(V2). In the worst case, the time complexity of BFS is **O(V2).**

Space complexity: The space complexity of the BFS algorithm is **O(V)** where V denotes vertices/nodes of the graph.

DFS (Depth First Traversal): Depth-first search(DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node(selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking. The DFS algorithm is a recursive algorithm that uses the idea of backtracking. It involves exhaustive searches of all the nodes by going ahead, if possible, else by backtracking. Backtrack means when you moving forward and there is no more nodes along the current path, you move backwards on the same path to find nodes to traverse. All the nodes will be visited on the current path till all the unvisited nodes have been traversed after which the next path will be selected. When we implementing DFS it is important to ensure that the nodes are visited and marked. This will prevent us from visiting the same node more than once if there is one or more cycles. If we do not mark the nodes that are visited we may end up in an infinite loop.

ALGORITHM FOR DFS:

Pseudocode :

DFS -recursive(G, s):

Mark s as visited

for all neighbors w of s in Graph G:

if w is not visited:

DFS-recursive(G, w)

Swaroop Prakash Jadhav, International Journal of Computer Science and Mobile Applications,
Vol.10 Issue. 7, July- 2022, pg. 1-7 ISSN: 2321-8363
Impact Factor: 6.308
(An Open Accessible, Fully Refereed and Peer Reviewed Journal)

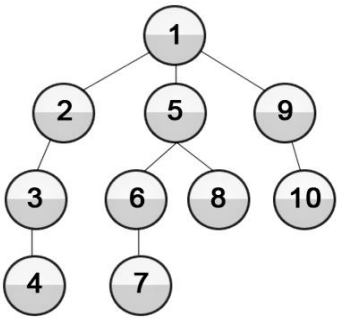


Fig: DFS Traversal

This recursive nature of DFS can be implemented using stacks.

Pick a starting node and push all its adjacent nodes into a stack.

Pop a node from stack to selected the next node to visit and push all its adjacent nodes into a stack.

Repeat this process until the stack is empty. However, ensure that the nodes that are visited are marked. This will prevent from visiting the same node more than once. If you do not mark the nodes that are visited and you visit the same node more than once, you may end up in an infinite loop.

APPLICATIONS OF DFS:

1. **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.

2. **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 recomputing formula values in spreadsheets, logic synthesis, determining the order of completion tasks to perform in make files, data serialization, and resolving symbol dependencies in linkers.

3. **To test if a graph is bipartite**

   We can augment either BFS or DFS when we first discover a new vertex, color it opposite its parents, 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.

4. **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.)

5. **For an unweighted graph** DFS traversal of the graph produces the minimum spanning tree and all pair shortest path tree.

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. **Path Finding:** We can specialize the DFS algorithm to find a path between the 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.

Time Complexity:

The best way to compute the time complexity would be to think about how many nodes you are processing and how many time you are processing each nodes. For example, If in our code a node is not visited more than once then a DFS operation would be O(n).

As for the |E| part, |E| which represents total number of edges present in the graph, will ultimately be expressed in terms of number of nodes (|V| or n). It depends on whether the graph is dense or sparse. If a graph is dense in most cases you would see that

$|E| = |V|*|V| = n \wedge 2$.

It would also depend whether you are representing the graph using Adjacency List or Adjacency Matrix and the way you are implementing them.

Any additional complexity comes from how you discover all the outgoing paths or edges for each node which, in turn, is dependent on the way your graph is implemented. If the edge leads you to node that has already been traversed, you skip it and check the next. Typical DFS implementations use hash table to maintain the list of traversed nodes so that you could find out if node has been encountered before in O(1) time (constant time).

Space Complexity:

DFS goes along a path all the way down before it backtracks and store all the nodes in the path in the recursion stack. So, the size of the stack would grow as large as the height of the longest path. The space complexity for DFS is O(h) where h is the maximum height of the tree.

Conclusion:

Breadth Search Algorithm comes with some great advantages to recommend it. One of the many applications of the BFS algorithm is to calculate the shortest path. It is also used in networking to find neighbouring nodes and can be found in social networking sites, network broadcasting, and garbage collection.

The results in this paper on methodology performance are analogous to the associated results in search theory. So as we have seen, Breadth-First Search is a very useful algorithm, which is used to traverse the Trees/Graphs and ensures that we explore the closest neighbours first, and then their neighbours, and so on. We have seen its working, its algorithm and its applications in many real-life use cases.

REFERENCES

[1] Graph theory with Applications to engineering & Computer Science NARSHIGH DEO DOVER PUBLICATIONS, INC MINEOLA, New York

[2] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. Communications of the ACM, pages 1116–1127, September 1988.

[3] 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.

[4] Jong Ho Kim, Helen Cameron, Peter Graham ―Lock-Free Red-Black Trees Using CAS‖ October 20, 2011.

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