PES UNIVERSITY

Department of Computer science and Engineering

PES UNIVERSITY

UE19CS202: Data Structures and its Applications (4-0-0-4-4)

GRAPH TRAVERSAL

Abstract

Traversals, Breadth first search, Depth first search, Implementation, Advantages, Disadvantages and Applications

Dr.Sandesh and Saritha

Sandesh_bj@pes.edu

Saritha.k@pes.edu

Graph traversal:

Many graph algorithms requires a systematic way to examine all the nodes and edges of a graph. Traversing is a process of visiting each and every node of a graph.

Defining a traversal that relates to the structure of a graph is more complex than for a list or tree due to following reasons

1. A graph has no first node or root node. Therefore the graph can be traversed from any node as starting node and once the starting node is selected there may be few nodes which are not reachable from the starting node. Traversal algorithm faces the problem of selecting another starting point.

2. In a graph to reach a particular node multiple paths may be available.

3. There is no natural order among the successors of particular node. Thus there is no prior order in which the successors of a particular node should be visited.

4. A graph may have more than one predecessor and therefore there is a possibility for a node to be visited before one of its predecessors. For example if node A is a successor of nodes B and C, A may be visited after B but before C.

There are two standard algorithms for traversal of graphs. They are:

• Breadth first search (BFS): The BFS will use a queue as an auxiliary structure to hold nodes for future processing

• Depth first Search (DFS): DFS algorithm traverses a graph in a depth-ward motion and uses a stack to remember to get the next vertex to start a search, when dead end occurs in any iteration.

Depth First Search:

In Depth first search method, an arbitrary node of a graph is taken as starting node and is marked as visited. On each iteration, the algorithm proceeds to an unvisited vertex which is adjacent to the one which is currently in. Process continues until a vertex with no adjacent unvisted vertex is found. When the dead end is reached the control returns to the previous node and continues to visit all the unvisited vertices. The algorithm ends after backing up to the starting vertex, when the latter being a dead end. The algorithm has to restart at an arbitrary vertex, if there still remain unvisited vertices.

This process is implemented using stack. A vertex is inserted into the stack when the vertex is visited for the first time and deletes a vertex when the visit of the vertex ends.

Iterative procedure to traverse a graph is shown below:

1. Select a node u as a start vertex, push u onto the stack and mark it as visited.

2. While stack is not empty

    For vertex u on top of the stack, find the next immediate adjacent vertex

    If v is adjacent

        If a vertex v is not visited then

            Push it on to stack and mark it as visited

        else

            Ignore the vertex

        End if

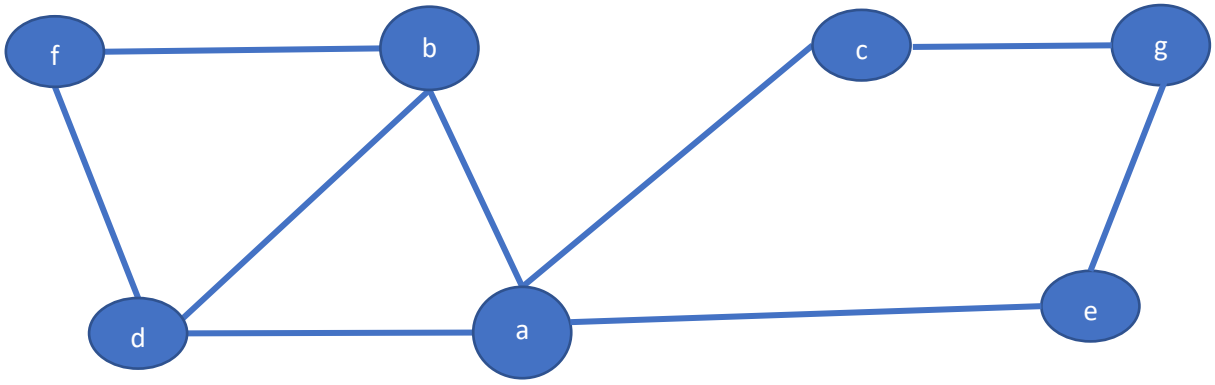
    Else

        Remove the vertex from the stack

    End if

   End while

3. Repeat step1 and step2 until all the vertices in the graph are visited.



	Stack	**V(adjacent vertex)**	**Nodes visited(S)**	**Pop(stack)**
Initial step	a	-	a	
	a	b	a,b	-
	a,b	d	a,b,d	-
	a,b,d	f	a,b,d,f	-
	a,b,d,f	-	a,b,d,f	f
	a,b,d	-	a,b,d,f	d
	a,b	-	a,b,d,f	b
	a	c	a,b,d,f,c	-
	a,c	g	a,b,d,f,c,g	-
	a,c,g	e	a,b,d,f,c,g,e	-
	a,c,g,e	-	a,b,d,f,c,g,e	e
	a,c,g	-	a,b,d,f,c,g,e	g
	a,c	-	a,b,d,f,c,g,e	c
	a	-	a,b,d,f,c,g,e	a

Step 1: Insert a source vertex *a* onto the stack and add *a* to node visited(S)
Step 2: Push the node adjacent to *a* that is *b* onto the stack (alphabetical order) and add to S if not present in S (Also note that in DFS only one adjacent vertex which is not visited is considered).
Step 3: push the node adjacent to *b*, that is *d* onto stack and add to S if not present.
Step 4: push the node adjacent to *d* that is *f* onto the stack and add to S if not visited.
Step 5: Since the node adjacent to *f* is already visited. Pop *f* from the stack.
Step 6: Since the node adjacent to *d* is already visited, pop *d* from the stack.
Step 7: Pop *b* from the stack.
Step 8: Since there is adjacent node from *a* to *c* we add c to stack and S
Step 9: Push the node adjacent to *c* that is *g* to stack and add to S.
Step 10: push the node adjacent to *g* that is e to stack and add to S.
Step 11: Since the node adjacent to *e* is visited, pop *e* from the stack.
Step 12: Since the node adjacent to *g* is visited, pop *g* from stack
Step 13: Since the node adjacent to *c* is visited, pop *c* from stack.
Step 14. Pop *a* from stack
Thus nodes reachable from *a* are *a, b, c, d, e, f, g.*

Advantages of Depth First Search:

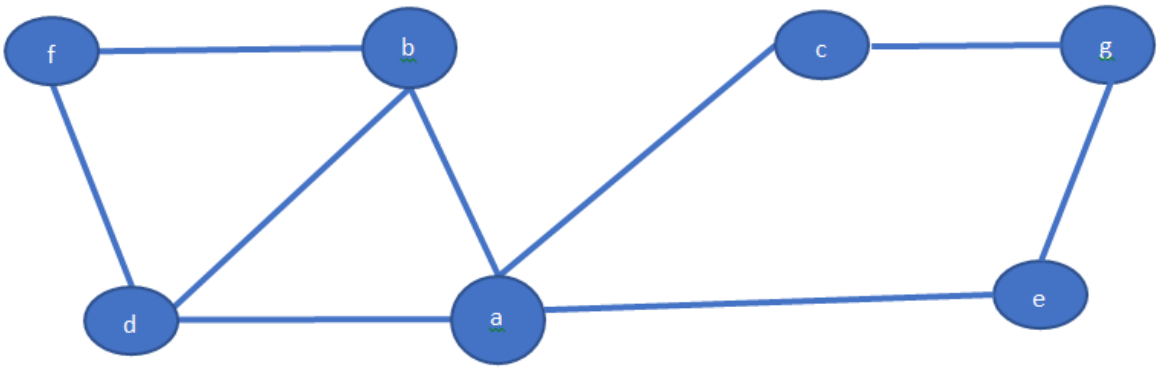
1. Consumes less memory
2. Finds the larger distant element from source vertex in less time.

Disadvantages:

1. It works very fine when search graphs are trees or lattices, but can get struck in an infinite loop on graphs. This is because depth first search can travel around a cycle in the graph forever. To eliminate this we keep a list of states previously visited, and never permit search to return to any of them.
2. We cannot come up with shortest solution to the problem.

Breadth First Search: Breadth first search uses the queue data structure for traversing vertices of the graph. Any node of the graph can act as the starting node. Using the starting node, all other nodes of the graph are traversed. To prevent repeated visit to the same node, an array is maintained which keeps track of visited node.

In breadth first search algorithm the queue is initialized with the starting vertex and is marked as visited. On each iteration, the algorithm identifies all the unvisited vertices adjacent to the front vertex, marks them as visited and adds them to the queue; after that ,the front vertex is removed from the queue.



	U(deleted from queue)	V(adjacent vertex)	Nodes visited(s)	Queue
Initial node			a	a
	a	b,c,d,e	a,b,c,d,e	b,c,d,e
	b	a,d,f	a,b,c,d,e,f	c,d,e,f
	c	a,g	a,b,c,d,e,f,g	d,e,f,g
	d	a,b,f	a,b,c,d,e,f,g	e,f,,g,
	e	a,g	a,b,c,d,e,f,g	f,g
	f	b,d	a,b,c,d,e,f,g	g
	g	c,e	a,b,c,d,e,f,g	empty

Step 1: Insert a source vertex *a* into the queue and add *a* to node visited(S).

Step 2: Delete an element a from queue and find all the adjacent nodes of .The nodes adjacent to *a* are *b, c, d* and *e*. Add these nodes to S only if it is not present in S. So we add *b, c, d* and *e* to S

Step 3: Delete b from queue and find all the adjacent nodes to *b* i.e. *a, d, f* and add to S only if it not present in S, so only *f* is added to S.

Step 4: Delete *c* from queue, Find all the adjacent nodes to *c*, and add those nodes to S if it is not present in S. So add only *g* to S.

Step 5: Delete *d* from queue, Find all the adjacent nodes of *d* and add those to S if not present. All the adjacent nodes of *d* is already in S.

Step 6: Delete *e* from queue, find all the adjacent nodes of *e*. i.e *a* and *g* .Since *g* is not in S we add *g* to S.

Step7: Delete *f* from queue; find all the nodes adjacent to *f.* add to S if not present in S

Step 8: Delete *g* from queue, find all the adjacent nodes to *g*. Since all the adjacent nodes of *g* are already present in S we don't add to S.

And the queue is empty so the nodes reachable from Source *a*: *a, b, c, d, e, f, g*

Pseudo code:

```
Insert source u to queue
Mark u as visited
While queue is not empty
        Delete a vertex u from queue
        Find the vertices v adjacent to u
        If v is not visited
            Print v
            Mark v as visited
            Insert v to queue
        End if
```

End while

Advantages of Breadth First Search:

1. BFS algorithm is used find the shortest path between vertices
2. Always finds optimal solutions.

Disadvantages of BFS:

All of the connected vertices must be stored in memory. So consumes more memory