Graph traversal methods

- Narges BabaAhmadi
- Niloufar BabaAhmadi
- Zahra Khatibi
- Mahyar Mohammadi
- Niko Rokni

Teacher : Dr.M.Nori
TA : Sh.Peyghambari

Summer 2020

Introduction

Graph traversal is a technique used for a searching vertex in a graph. The graph traversal is also used to decide the order of vertices is visited in the search process. A graph traversal finds the edges to be used in the search process without creating loops. That means using graph traversal we visit all the vertices of the graph without getting into looping path.

There are two graph traversal techniques and they are as follows :

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

DFS

Depth-first search

DFS:

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 . Here, the word backtrack means that when you are moving forward and there are 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.

Time and space :

The time and space analysis of DFS differs according to its

application area. In theoretical computer science, DFS is typically used to traverse an entire graph, and takes time $O(|v| + |E|)$, linear in

the size of the graph. In these applications it also uses space $O(|v|)$ in

the worst case to store the stack of vertices on the current search path

as well as the set of already-visited vertices.

Usage :

For applications of DFS in relation to specific domains, such as searching for solutions in artificial intelligence or web-crawling, the graph to be traversed is often either too large to visit in its entirety or infinite (DFS may suffer from non-termination).

In such cases, search is only performed to a limited depth ; due to limited resources, such as memory or disk space, one typically does not use data structures to keep track of the set of all previously visited vertices.

When search is performed to a limited depth, the time is still linear in terms of the number of expanded vertices and edges (although this number is not the same as the size of the entire graph because some vertices may be searched more than once and others not at all) .

But the space complexity of this variant of DFS is only proportional to the depth limit, and as a result, is much smaller than the space needed for searching to the same depth using breadth-first search.

DFS algorithm :

A standard DFS implementation puts each vertex of the graph into one of two categories:

1. Visited
2. Not Visited

The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

The DFS algorithm works as follows:

1. Start by putting any one of the graph's vertices on top of a stack.

2. Take the top item of the stack and add it to the visited list.

3. Create a list of that vertex adjacent nodes. Add the ones which aren't in the visited list to the top of the stack.

Keep repeating steps 2 and 3 until the stack is empty.

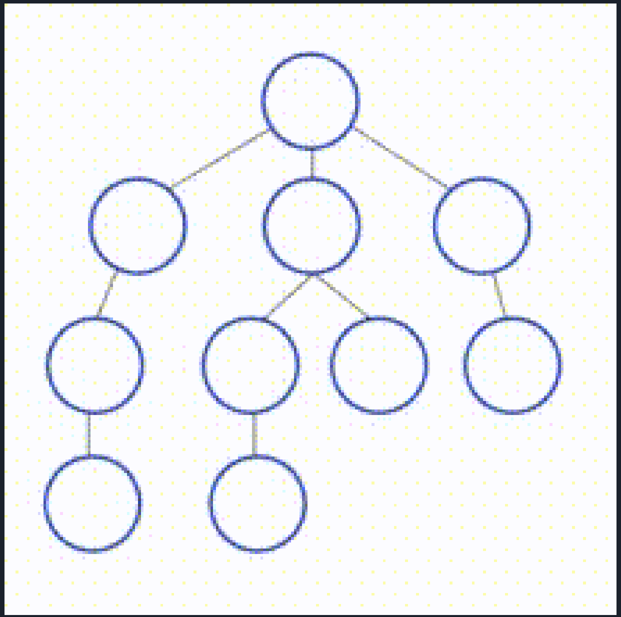
Here's a code that we prepare that calculate all ways between two vertex:

(http://s13.picofile.com/file/8403847168/DFS.cpp.html)

Here's a code that we prepare that calculate all ways between two vertex:

(http://s13.picofile.com/file/8403847168/DFS.cpp.html)

Example of DFS



DFS Algorithm Applications:

1-For finding the path (in a tree)

2-To test if the graph is bipartite

3-For detecting cycles in a graph

4-Solving a maze

BFS

Breadth-first Search

BFS:

one of the simplest and most basic algorithms for navigating and searching the graph is "first-level search algorithm (first search) or BFS".

This algorithm has lots of usage. We can also state that many other algorithms for different kinds of problems , like finding the shortest way ,use the similar algorithm .

the reason of its name is that a boundary is drawn between the traversed vertices, and the traversed vertices are uniformly drawn along the crossing boundary level. In other words, you will visit all vertices with distance k from vertex x before vertices with distance K + 1 from vertex x.

from the name of the representative algorithm, we can understand that layers are visited one by one ,and each vertex goes to queue after getting visited.

How BFS works :

The algorithm starts at the root [in graphs and trees without roots, the arbitrary vertex is selected as the root] and sets it to zero. At each stage, it puts all unvisited neighbors of the lastly visited vertex to the next level. This process ends when all neighbors of the vertices of the last level have been seen.

To investigate this, we can use a binary array .we first create an array with the same size as the number of vertices of the graph and set all its elements to zero. By scrolling and visiting each vertex, we change its value from zero to one in our array .The algorithm continues until the array contains no zero.

In some cases, solving the problem requires finding a specific vertex. In this case, the algorithm visits all the neighbors of one vertex each time and then goes to the next vertex, and thus the graph is scrolled level by level. This search continues until the specific vertex is found.

A queue is used to implement this algorithm.

First, the root is placed in the queue, then at each step, the initial element of the queue is pulled out, its neighbors are checked, and any neighbor that has not been visited yet , is added to the queue (this can be done with the help of an array Examined the aforementioned binary).

For better visualization, the BFS algorithm can be explained as the spread of fire in a graph. In this example, the first vertex catches fire first, and at each stage it burns all neighbors of lastly visited vertex on the next floor.

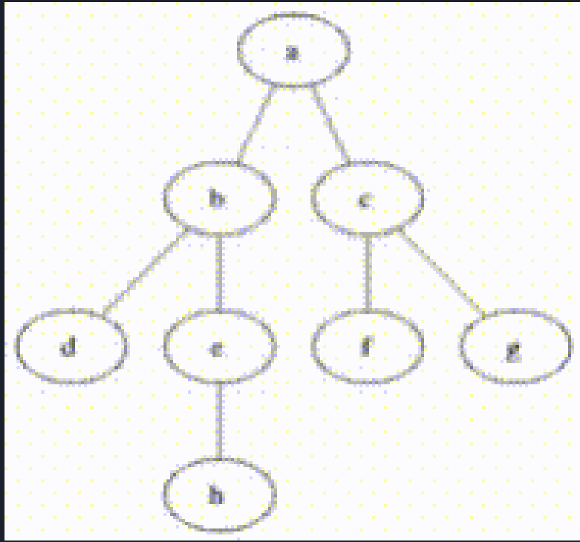
Temporal complexity :

A vertex enters the queue if it is not visited, so each vertically accessible vertex enters the queue exactly once. Each time a vertex enters the queue, $O(1)$ is performed, and assuming that the graph is connected, the queue operations will take $O(n)$. Also, each edge in the directionless graph, is traversed exactly twice, and each edge in a directional graph is traversed exactly once .If the graph is not connected, the time complexity of the order is $O(ms + ns)$, where ns and ms are the number of vertices and the number of edges that are accessible from s.

Here's a code that we prepare that calculate shortest way between two vertex :

(http://s12.picofile.com/file/8403846950/BFS.cpp.html)

Example of BFS



Difference between BFS and DFS:

BFS	DFS
BFS finds the shortest path to the destination.	DFS goes to the bottom of a subtree,then backtracks.
The full form of BFS is Breadth-First Search.	The full form of DFS is Depth First Search.
It uses a queue to keep track of the next location to visit	It uses a stack to keep track of the next location to visit
BFS traverses according to tree level.	DFS traverses according to tree depth.
It is implemented using FIFO list.	It is implemented using LIFO list.
It requires more memory as compared to DFS.	It requires less memory as compared to BFS.
This algorithm gives the shallowest path solution.	This algorithm doesn't guarantee the shallowest path solution.
There is no need for backtracking in BFS.	There is a need for backtracking in DFS.
You can never be trapped into finite loops.	You can be trapped into infinite loops.
If you do not find any goal, you may need to expand many nodes before the solution is found.	If you do not find any goal, the leaf node backtracking may occur.

Applications of BFS:

1. Un-weighted Graphs
2. P2P Networks
3. Web Crawlers
4. Network Broadcasting

Applications of DFS:

1. Weighted Graph
2. Path Finding
3. Detecting a Cycle in a Graph
4. Searching Strongly Connected Components of a Graph
5. Topological Sorting
6. Solving Puzzles with Only One Solution

at last, we answer this problem in codeforces about Graph that solved with

recursive DFS: http://s12.picofile.com/file/8403847200/FinalCode.cpp.html