

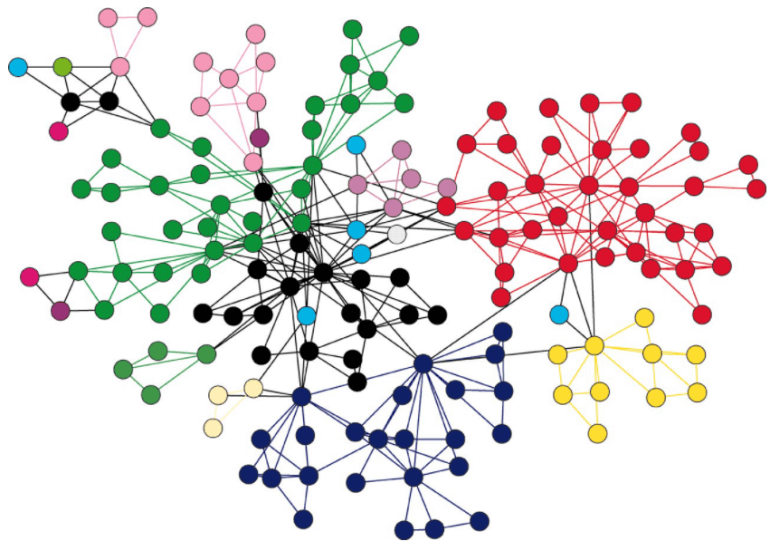
ASSIGNMENT: GRAPH DATA STRUCTURES, ITS PRACTICAL IMPLEMETATION IN ANY PROGRAMING LANGUAGE AND GRAPH TRANVERSAL ALGORITHMS AND THEIR APLICATION TO REAL LIFE PROBLEM

|  |  |
| --- | --- |
| NAME OF STUDENTS | PERCENTAGE OF WORK DONE |
| ESAMBE GEORGE KOGE(LEADER) | 30% |
| MITCHO BLAIR TATONE KUATE | 20% |
| MENDI BLAISE NDAH | 20% |
| MOHMBEH TANYU PATRICK | 20% |
| MENDOUA MINLO ARMEL B | 10% |

**TEACHER’S NAME: TEKO PALMER**

**What is a Graph?**

A graph is a unique data structure in programming that consists of finite sets of nodes or vertices and a set of edges that connect these vertices to them. At this moment, adjacent vertices can be called those vertices that are connected to the same edge with each other. In simple terms, a graph is a visual representation of vertices and edges sharing some connection or relationship. Although there are plenty of graph algorithms that you might have been familiar with, only some of them are put to use. The reason for this is simple as the standard graph algorithms are designed in such a way to solve millions of problems with just a few lines of logically coded technique. To some extent, one perfect algorithm is solely optimized to achieve such efficient results.



**Types of Graphs**

There are various types of graph algorithms that you would be looking at in this article but before that, let's look at some types of terms to imply the fundamental variations between them.

**Order**: Order defines the total number of vertices present in the graph.

**Size**: Size defines the number of edges present in the graph.

**Self-loop**: It is the edges that are connected from a vertex to itself.

**Isolated vertex**: It is the vertex that is not connected to any other vertices in the graph.

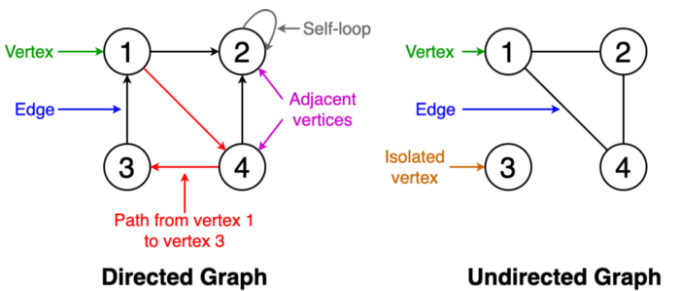
**Vertex degree**: It is defined as the number of edges incident to a vertex in a graph.

**Weighted graph**: A graph having value or weight of vertices.

**Unweighted graph**: A graph having no value or weight of vertices.

**Directed graph**: A graph having a direction indicator.

**Undirected graph**: A graph where no directions are defined.



**GRAPH IMPLEMENTATIONS**

1. **Adjacency Matrix**: An adjacency matrix is a two-dimensional array that records the relationship between each node and all other nodes in the graph.

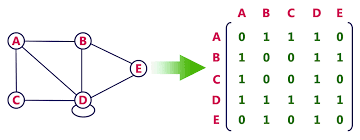
**-** Observe that the data in an adjacency matrix for an unweighted graph is binary and symmetric.

-The data in an adjacency matrix for a directed graph is not symmetric.

-For weighted graphs, the values of the weights are stored in the adjacency matrix (rather than just 1,1 or 0,0). If an edge does not exist between two nodes, a very large number (or often the infinity symbol, ∞) is set.

-Space requirement O(|V|^2).

-Adjacency matrix is acceptable if the graph is dense.

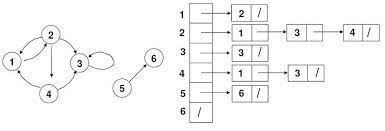


2. **Adjacency List**; An adjacency list records the existence of an edge between each node and all of its neighbours.

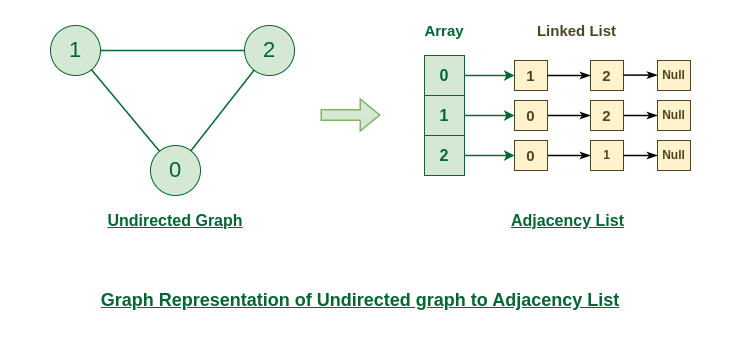
-For directed graphs, the data is stored in an adjacency list in the same way as for undirected graphs, but the data must be interpreted carefully. The adjacent neighbours are those for which there exists a directed edge.

-When a graph is weighted, the adjacency list is used to record connections between two nodes, and the corresponding weights.

**Adjacency list for Directed graph;**



**Adjacency list for Directed graph;**

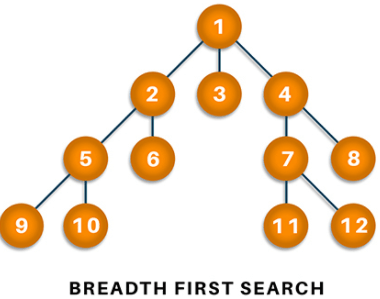


**Graph Transversal Algorithm**;

Graph traversal algorithms are used to visit all the nodes in a graph, typically in a systematic way. There are different types of graph traversal algorithms, such as depth-first search (DFS) and breadth-first search (BFS).

1. **Breadth-First Search** (**BFS**);

Traversing or searching is one of the most used operations that are undertaken while working on graphs. Therefore, in breadth-first-search (BFS), you start at a particular vertex, and the algorithm tries to visit all the neighbors at the given depth before moving on to the next level of traversal of vertices. Unlike trees, graphs may contain cyclic paths where the first and last vertices are remarkably the same always. Thus, in BFS, you need to keep note of all the track of the vertices you are visiting. To implement such an order, you use a queue data structure which First-in, First-out approach. To understand this, see the image given below.



**Graph Algorithms of Breadth First Search**;

- Start putting anyone vertices from the graph at the back of the queue.

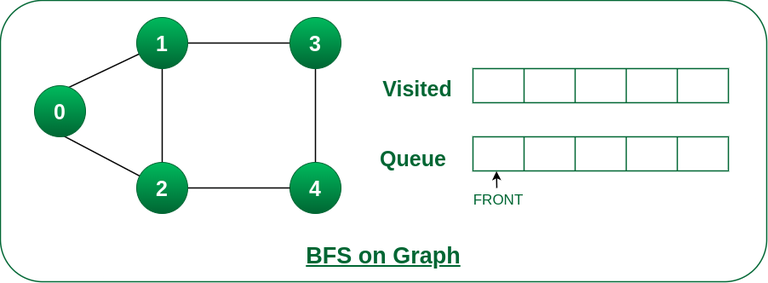
-First, move the front queue item and add it to the list of the visited nod

-Next, create nodes of the adjacent vertex of that list and add them which have not been visited yet.

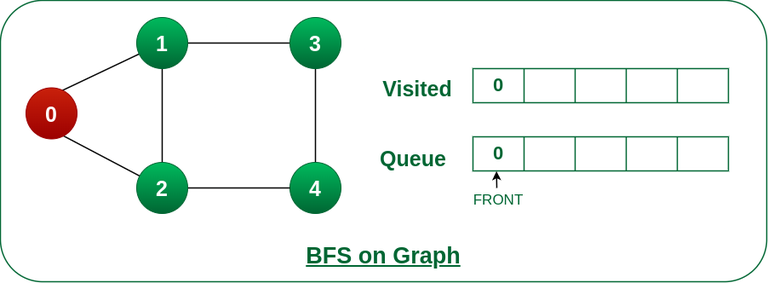
-Keep repeating steps two and three until the queue is found to be empty.

**How It Works;**

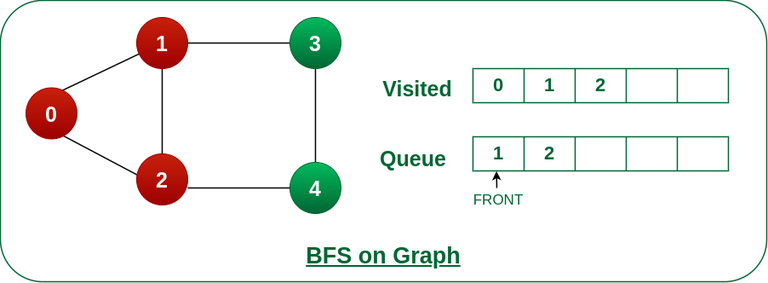
***Step1:****Initially queue and visited arrays are empty*



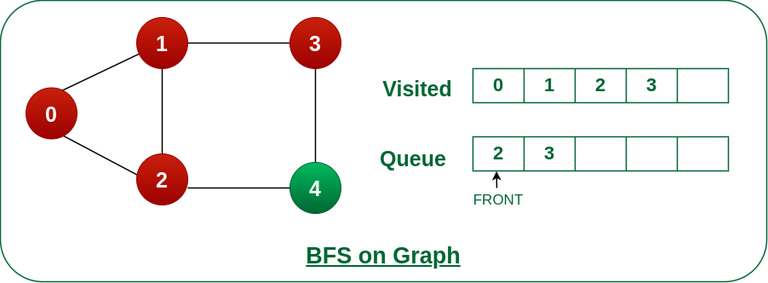
***Step2:****Push node 0 into queue and mark it visited.*



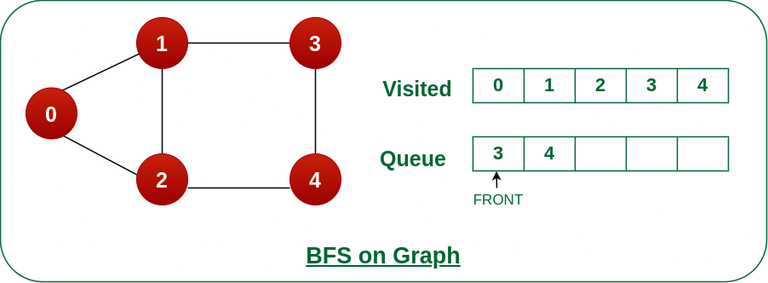
***Step 3:****Remove node 0 from the front of queue and visit the unvisited neighbours and push them into queue.*



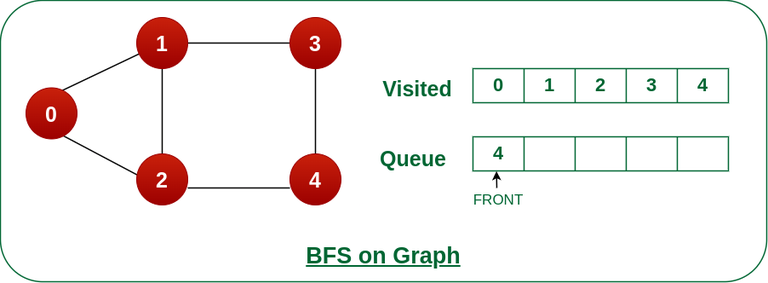
***Step 4:****Remove node 1 from the front of queue and visit the unvisited neighbours and push them into queue.*



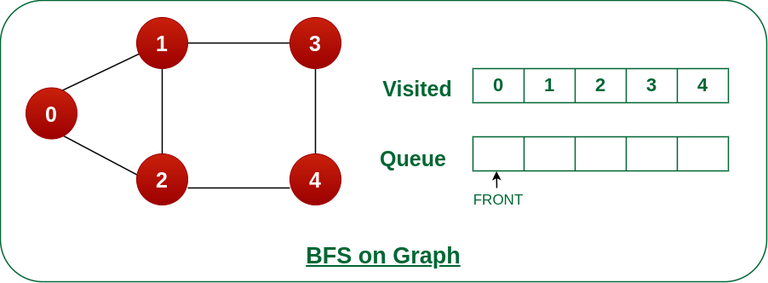
***Step 5:****Remove node 2 from the front of queue and visit the unvisited neighbours and push them into queue.*



***Step 6:****Remove node 3 from the front of queue and visit the unvisited neighbours and push them into queue.   
As we can see that every neighbours of node 3 is visited, so move to the next node that are in the front of the queue.*



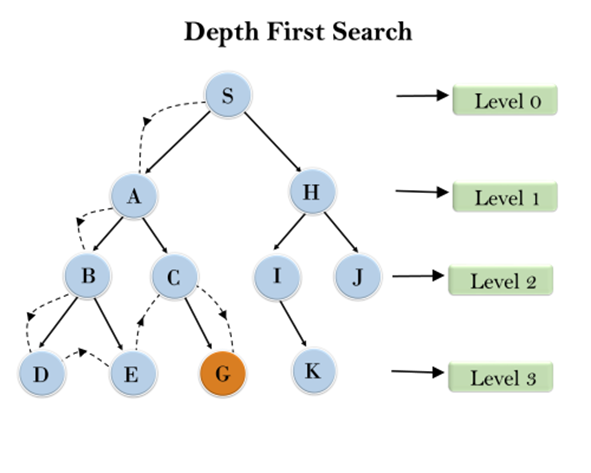
***Steps 7:****Remove node 4 from the front of queue and visit the unvisited neighbours and push them into queue.   
As we can see that every neighbours of node 4 are visited, so move to the next node that is in the front of the queue.*



Now, Queue becomes empty, So, terminate these process of iteration

**Depth-first search**;

In depth-first-search (DFS), you start by particularly from the vertex and explore as much as you along all the branches before backtracking. In DFS, it is essential to keep note of the tracks of visited nodes, and for this, you use stack data structure.



**Graph Algorithms of Depth First Search**;

-Start by putting one of the vertexes of the graph of the graph on the stack’s top.

-Put the top item of the stack and add it to the visited vertex list.

-Create a list of all the adjacent nodes of the vertex and then add those nodes to the unvisited at the top of the stack.

-Keep repeating steps 2 and 3, and the stack becomes empty.

**Application of Graphs in real life problems**

Graph traversal algorithms have numerous real-life applications, including:

1. Social Networks: DFS and BFS can be used to find connections between individuals in a social network, recommend friends, or identify communities within the network.

2. Web Crawling: Search engines use BFS to crawl and index web pages, ensuring that all pages are visited and indexed in an organized manner.

3. Routing and Navigation: These algorithms are used to find the shortest path between locations in GPS systems and map applications.

4. Network Analysis: BFS and DFS are used to analyze computer networks, detect vulnerabilities, and optimize network traffic.

5. Game Development: Graph traversal algorithms are used to find paths, analyze game maps, and make decisions for non-player characters (NPCs) in video games.

6. Puzzles and Mazes: These algorithms are used to solve puzzles like mazes, Sudoku, and other logical games.

7. Compiler Design: DFS is used in compiler design to traverse abstract syntax trees and optimize code generation.

8. Image Processing: BFS and DFS are used in image processing for tasks such as region filling, connected component analysis, and object recognition.

These are just a few examples of the many real-life applications of graph traversal algorithms. Their versatility makes them valuable tools in various fields, including computer science, engineering, social sciences, and more.