EXPOSEE ON FUNDAMENTALS OF ALGORITHM

DEPARTMENT: SOFTWARE ENGINEERING

GROUP 15

TOPIC: GRAPH

DATA STRUCTURE

LECTURER/SUPERVISOR:

MR. TEKOH PALMA

INTRODUCTION TO GRAPH DATA STRUCTURE

Let’s try to understand this through a very common example. On Facebook, everything is a node. That includes Users, Photo, Album, Story, Video, Link, Notes and anything that has data is a node. Every relationship is an edge from one node to another. Whether you post photo, join a group, like a page, etc., a new edge is created for that relationship. All of Facebook is then a collection of these nodes and edges. This is because Facebook uses a graph data structure to store its data.

Illustrating a graph data structure

Therefore A graph data structure is a collection of nodes that have data and are connected to other nodes. A graph is a data structure that consists of **vertices (V)** and **Edges (E).** The Vertices are also referred to as **Nodes** and the Edges are lines or arcs that connect any two nodes in a graph.

COMPONENETS OF A GRAPH

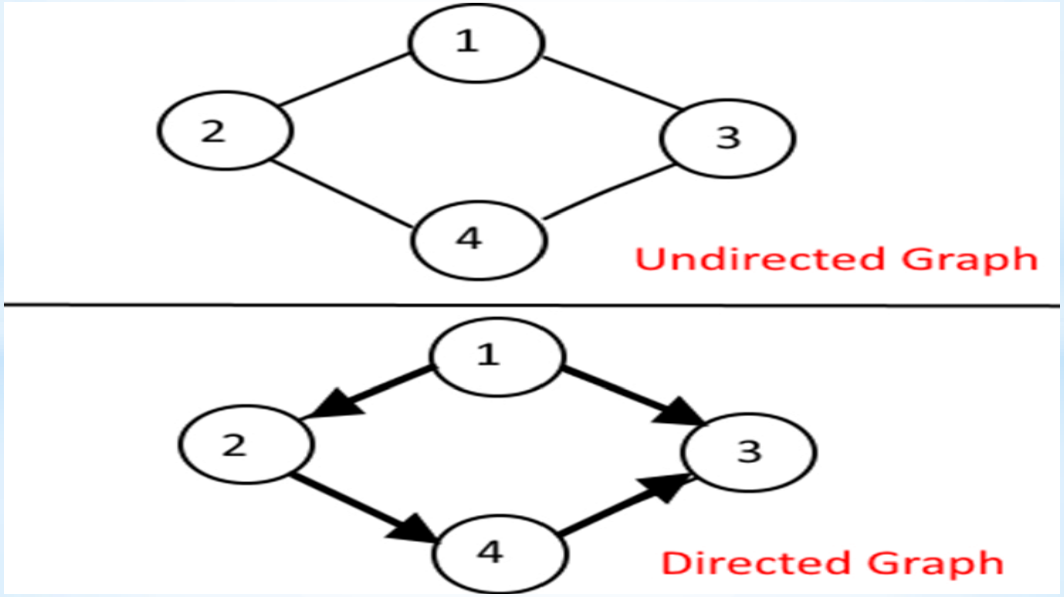
- **Vertices (V):** A vertex is an items or objects that have data, such as names, numbers, colors, etc. it is also called a **node**. Every vertex can be labeled or unlabeled
- **Edges (E):** Edges are the connections or the relationships between the vertices, such as distances, directions, similarities, etc. They can be labeled or unlabeled

NB: You can think of a graph as a collection of dots (vertices) and lines (edges) that link some of the dots together.

Types of graphs in algorithm

There are two types of graphs

- ✓ **Directed graph** – In a directed graph, edges have direction, i.e., edges go from one vertex to another.
- ✓ **Undirected graph** – In an undirected graph, edges have no direction

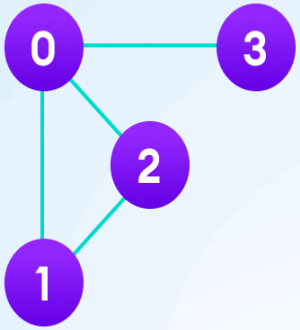


GRAPH REPRESENTATION

Graphs are commonly represented in two ways:

1. Adjacency Matrix

An Adjacency matrix is a square matrix used to represent connections between nodes in a graph. Each row and column corresponds to a node. The entry in the matrix indicates whether there is an edge between the respective nodes. If there is a connection the entry is usually 1. If there is no the entry it is 0.



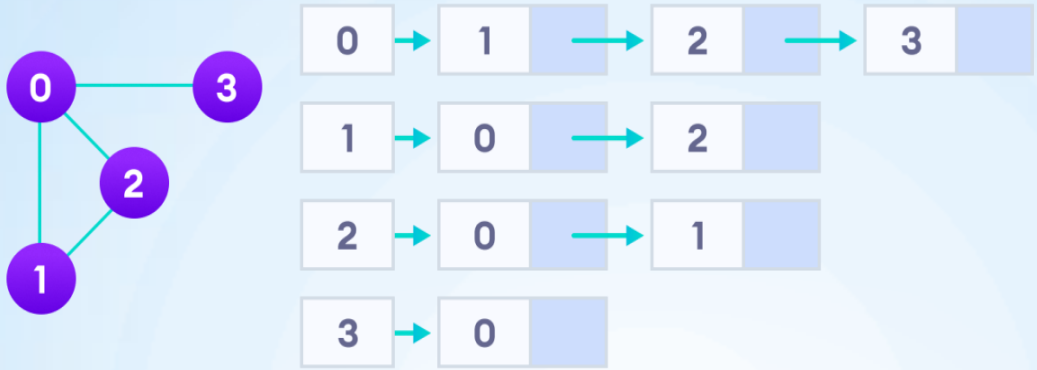
	0	1	2	3
0	0	1	1	1
1	1	0	1	0
2	1	1	0	0
3	1	0	0	0

Graph adjacency matrix

- ✓ NB: The graph above is an undirected graph hence no directions indicated. i.e if there is an edge between 0 and 1 it means there is an edge between 1 and 0.

2. Adjacency List

An Adjacency List is a data structure that is used to represent connections between vertices in a graph. It stores a list of adjacent vertices (neighbors) for each vertex, that is; the list represents all the other vertices that form an edge with each vertex in the graph. An Adjacency list uses pointers in other to store vertices of a graph



Adjacency List Representation

NOTE

- ✓ The adjacency matrix is easy to represent and to handle while the adjacency list is not easy for beginners to handle since it requires knowledge on link list and pointers
- ✓ The adjacency matrix consumes a lot of memory hence, not fit for sparse graphs while adjacency list uses low space hence can handle sparse graphs.
- ✓ The adjacency matrix is good for dense graphs due to low space requirement compared to adjacency list which only need pointers

GRAPH TRANSVERSAL ALGORITHMS

Graph Transversal is technic used to visit all the vertices and edges of a graph. Graph transversal specifically refers to the process of traversing or visiting all the vertices of a graph. Common algorithms for graph transversal include; **Breadth First Transversal or Search Algorithm (BFS)** and **Depth First Transversal and Search Algorithm (DFS)**

1. Breadth First Transversal or Search Algorithm (BFS)

- ✓ The Breadth First Search (BFS) algorithm is used to search a graph data structure for a node that meets a set of criteria. It starts at the root of the graph and visits all nodes at the current depth level before moving on to the nodes at the next depth level.
- ✓ Breadth-First Traversal or Search for a graph is similar to the Breadth-First Traversal of a tree. The only catch here is, that, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we divide the vertices into two categories:
- **Visited and**
- **Not visited.**

A Boolean visited array is used to mark the visited vertices. For simplicity, it is assumed that all vertices are reachable from the starting vertex. BFS uses a queue data structure for traversal. The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

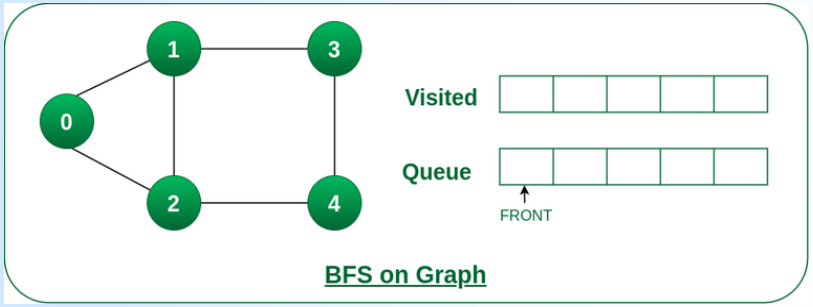
The algorithm works as follows:

- ✓ Start by putting any one of the graph's vertices at the back of a queue.
- ✓ Take the front item of the queue and add it to the visited list.
- ✓ Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the back of the queue.
- ✓ Keep repeating steps 2 and 3 until the queue is empty.
- ✓ The graph might have two different disconnected parts so to make sure that we cover every vertex, we can also run the BFS algorithm on every node

Illustration

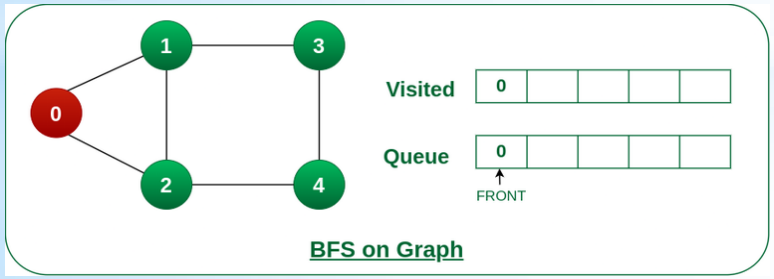
Let us understand the working of the algorithm with the help of the following example.

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



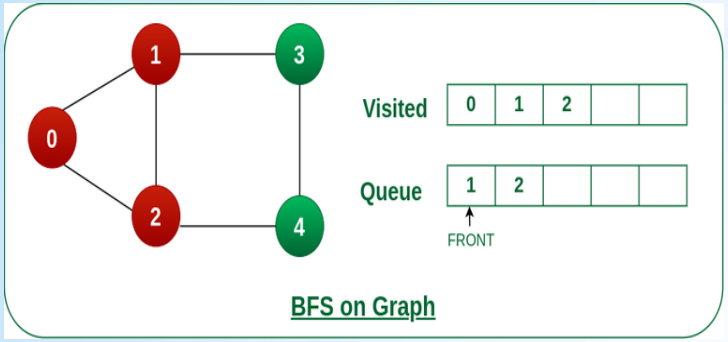
Queue and visited arrays are empty initially

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



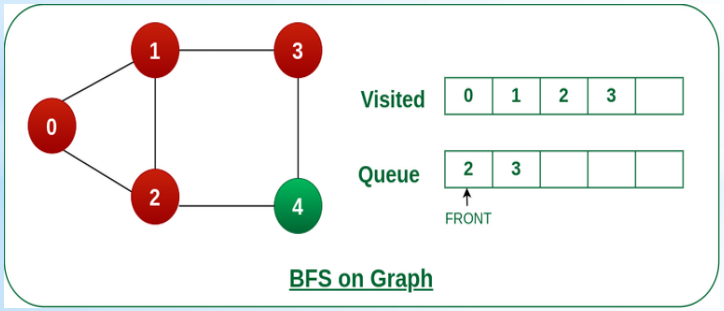
Push node 0 into queue and mark it visited.

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



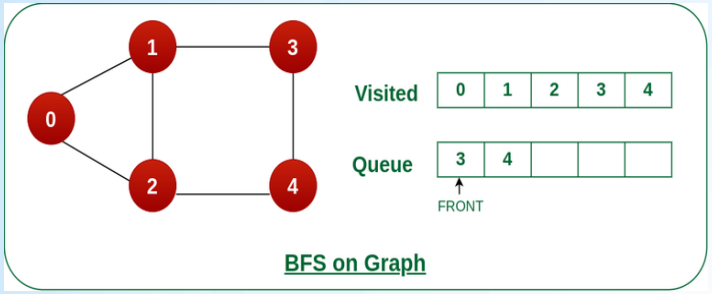
Remove node 0 from the front of queue and visited the unvisited neighbors and push into queue.

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



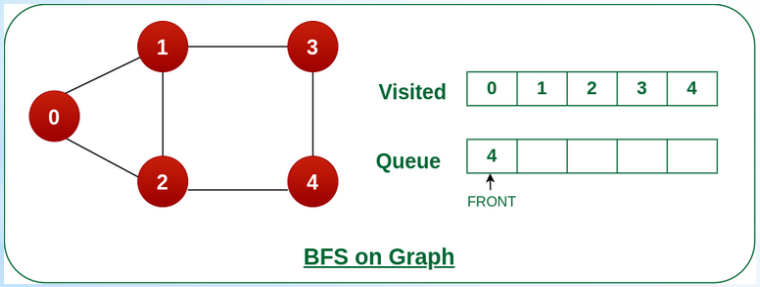
Remove node 1 from the front of queue and visited the unvisited neighbors and push

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



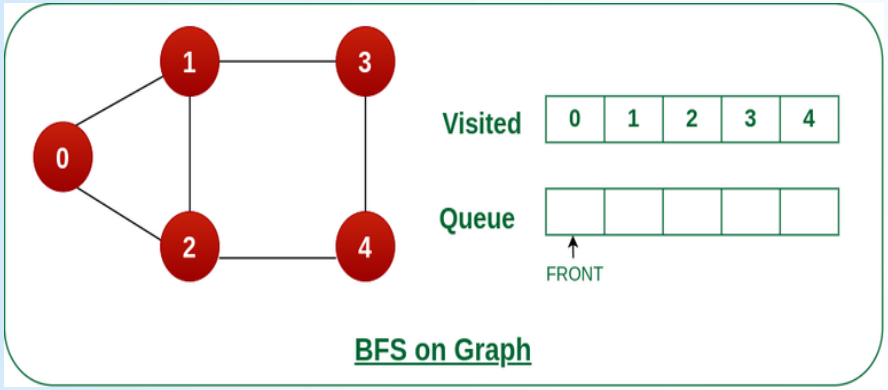
Remove node 2 from the front of queue and visit the unvisited neighbors and push them into queue.

✓ **Step 6:** Remove node 3 from the front of queue and visit the unvisited neighbors and push them into queue.



As we can see that every neighbors 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 neighbors and push them into queue. As we can see that every neighbors 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.

- **Time Complexity:** O(V+E), where V is the number of nodes and E is the number of edges.
- **Auxiliary Space:** O(V)

Applications of Breadth First Search

- **Shortest Path for unweight graph:** In an unweight graph, the shortest path is the path with the least number of edges. With Breadth First, we always reach a vertex from a given source using the minimum number of edges.
- **Peer-to-Peer Networks:** In Peer-to-Peer Networks like Bit Torrent, Breadth First Search is used to find all neighbor nodes.
- **Crawlers in Search Engines:** Crawlers build an index using Breadth First. The idea is to start from the source page and follow all links from the source and keep doing the same. Depth First Traversal can also be used for crawlers, but the advantage of Breadth First Traversal is, the depth or levels of the built tree can be limited.

- **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.
- **GPS Navigation systems:** Breadth First Search is used to find all neighboring locations.
- **Broadcasting in Network:** In networks, a broadcasted packet follows Breadth First Search to reach all nodes.
- **In Garbage Collection:** Breadth First Search is used in copying garbage collection using Cheney’s algorithm. Breadth First Search is preferred over Depth First Search because of a better locality of reference.

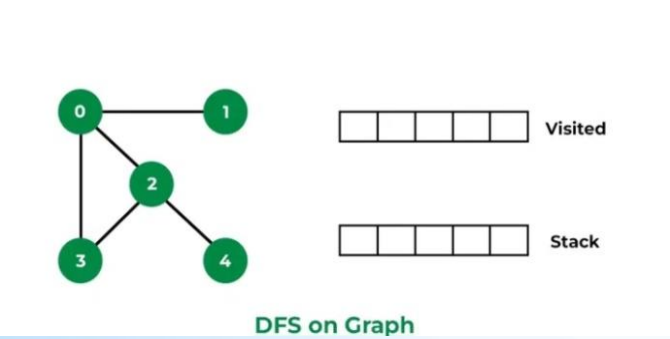
DEPTH FIRST TRANSVERSAL OR SEARCH ALGORITHM (DFS)

Depth-first search 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.

How does DFS work?

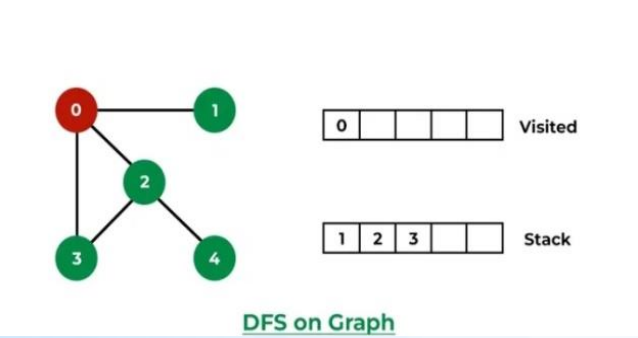
Let us understand the working of Depth First Search with the help of the following illustration:

Step1: Initially stack and visited arrays are empty.



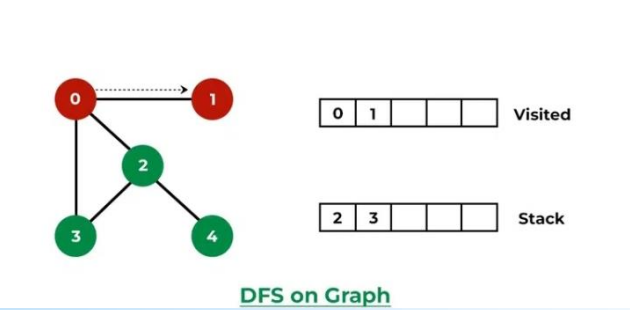
Stack and visited arrays are empty initially.

Step 2: Visit 0 and put its adjacent nodes which are not visited yet into the stack.



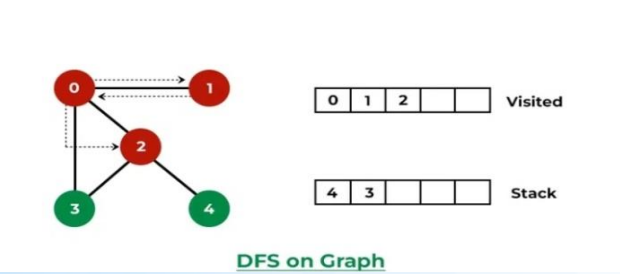
Visit node 0 and put its adjacent nodes (1, 2, 3) into the stack

Step 3: Now, Node 1 at the top of the stack, so visit node 1 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.



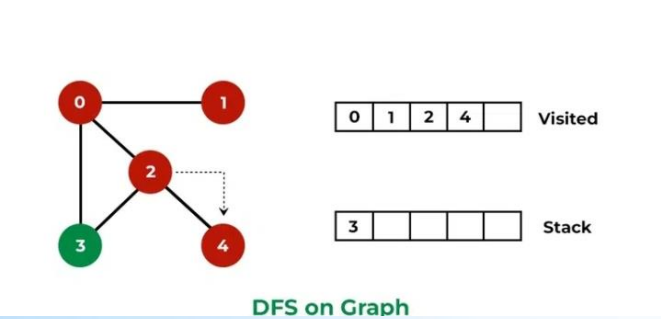
Visit node 1

Step 4: Now, Node 2 at the top of the stack, so visit node 2 and pop it from the stack and put all of its adjacent nodes which are not visited (i.e. 3, 4) in the stack.



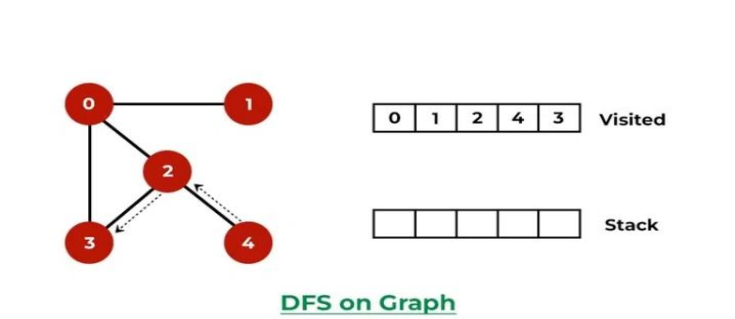
Visit node 2 and put its unvisited adjacent nodes (3, 4) into the stack

Step 5: Now, Node 4 at the top of the stack, so visit node 4 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.



Visit node 4

- **Step 6:** Now, Node 3 at the top of the stack, so visit node 3 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.



Visit node 3

Now, Stack becomes empty, which means we have visited all the nodes and our DFS traversal ends.

- **Time complexity:** O(V + E), where V is the number of vertices and E is the number of edges in the graph.
- **Auxiliary Space:** O(V + E), since an extra visited array of size V is required, And stack size for iterative call to DFS function.

Applications of Depth First Search

- **Detecting cycle in a graph:** A graph has a 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.
- **Path Finding:** We can specialize the DFS algorithm to find a path between two given vertices u and z. Call DFS (G, u) with u as the start vertex. Use a stack S to keep track of the path between the start vertex and the current vertex. As soon as destination vertex z is encountered, return the path as the contents of the stack
- **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 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.

- **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. (See this for DFS-based algo for finding Strongly Connected Components)
- **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.).
- **Web crawlers:** Depth-first search can be used in the implementation of web crawlers to explore the links on a website.
- **Maze generation:** Depth-first search can be used to generate random mazes.
- **Model checking:** Depth-first search can be used in model checking, which is the process of checking that a model of a system meets a certain set of properties.
- **Backtracking:** Depth-first search can be used in backtracking algorithms.