**BFS algorithm**

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

1. Visited
2. Not Visited

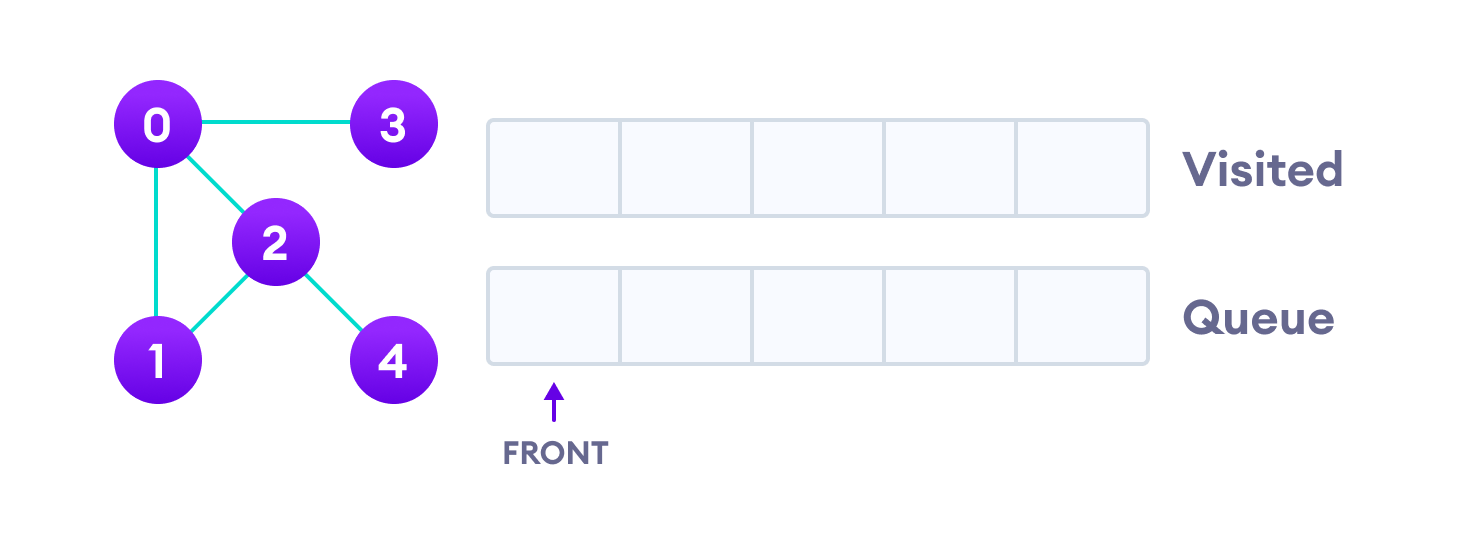
The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

The algorithm works as follows:

1. Start by putting any one of the graph's vertices at the back of a queue.
2. Take the front item of the queue and add it to the visited list.
3. 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.
4. Keep repeating steps 2 and 3 until the queue is empty.
5. 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

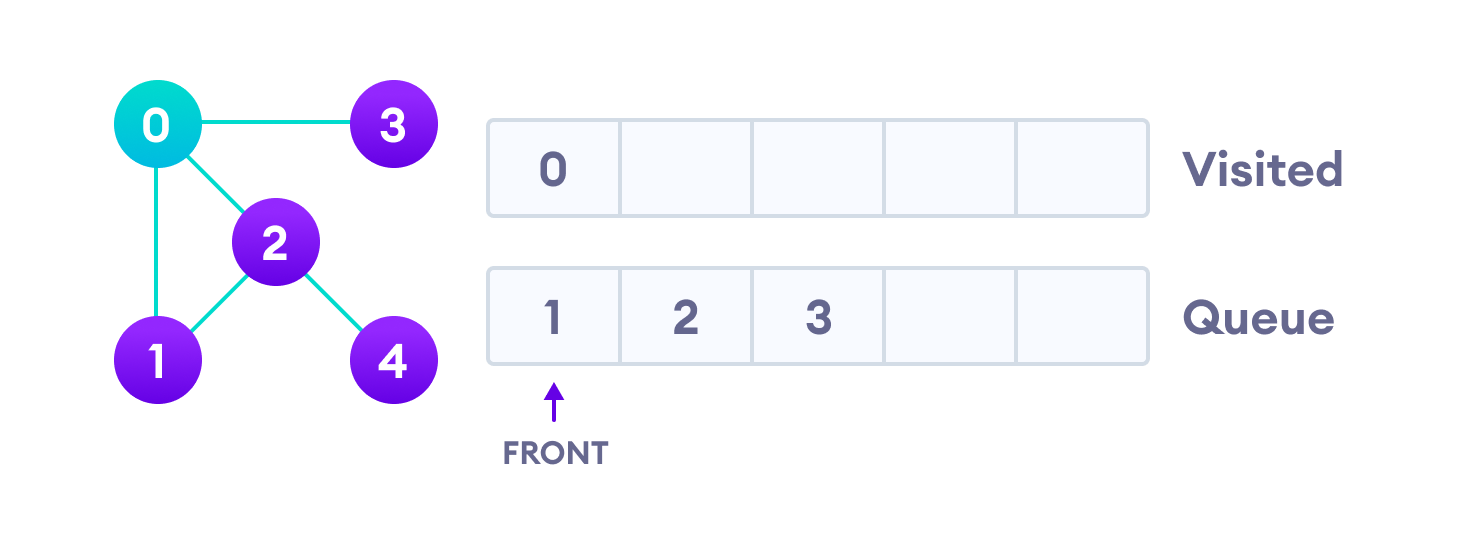
## BFS example

Let's see how the Breadth First Search algorithm works with an example. We use an undirected graph with 5 vertices.

****

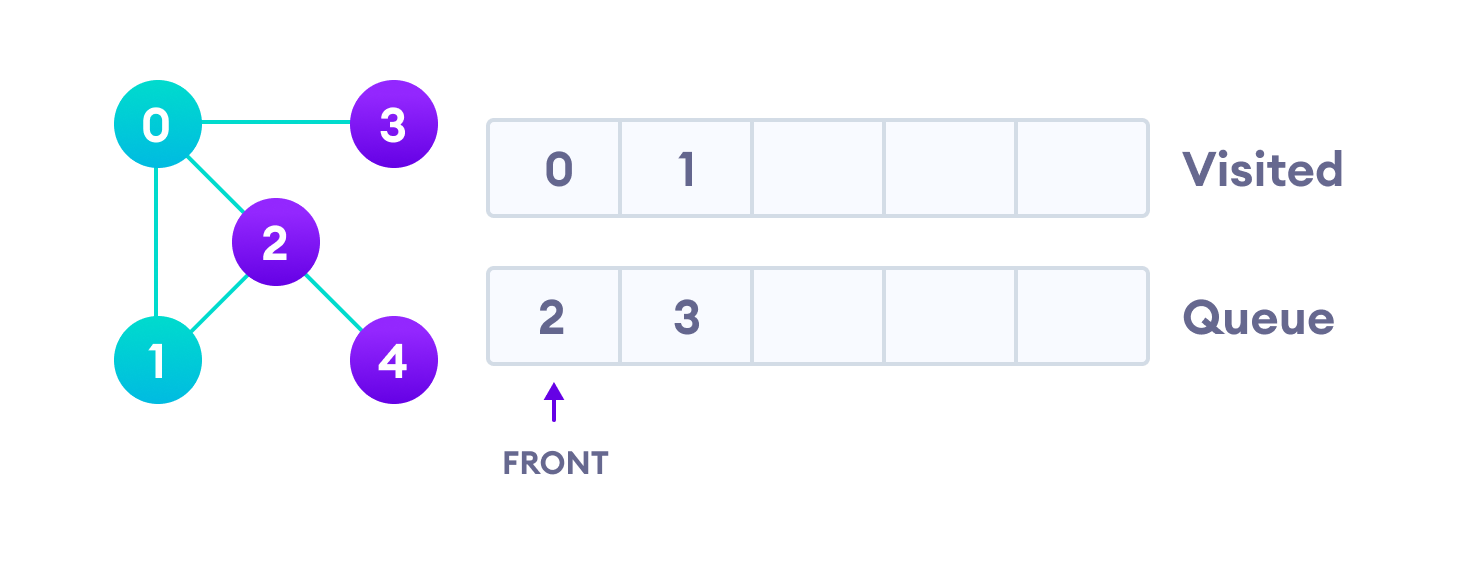
**Undirected graph with 5 vertices**

We start from vertex 0, the BFS algorithm starts by putting it in the Visited list and putting all its adjacent vertices in the stack.



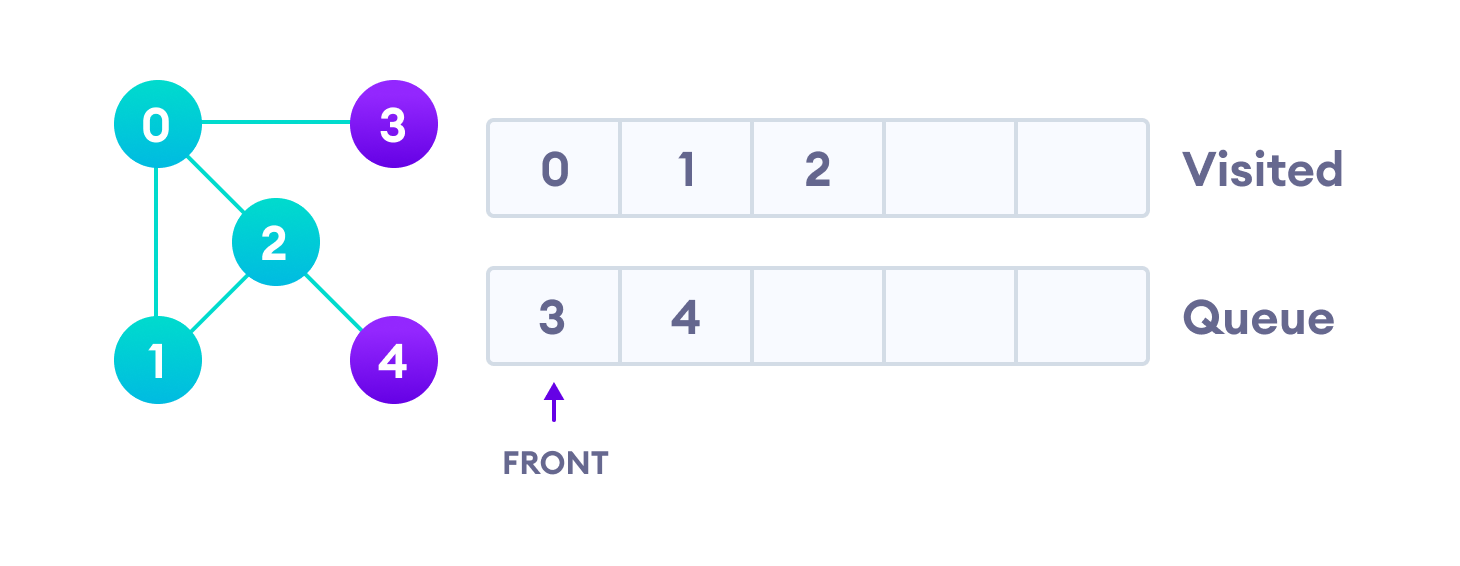
Visit start vertex and add its adjacent vertices to queue

Next, we visit the element at the front of queue i.e. 1 and go to its adjacent nodes. Since 0 has already been visited, we visit 2 instead.

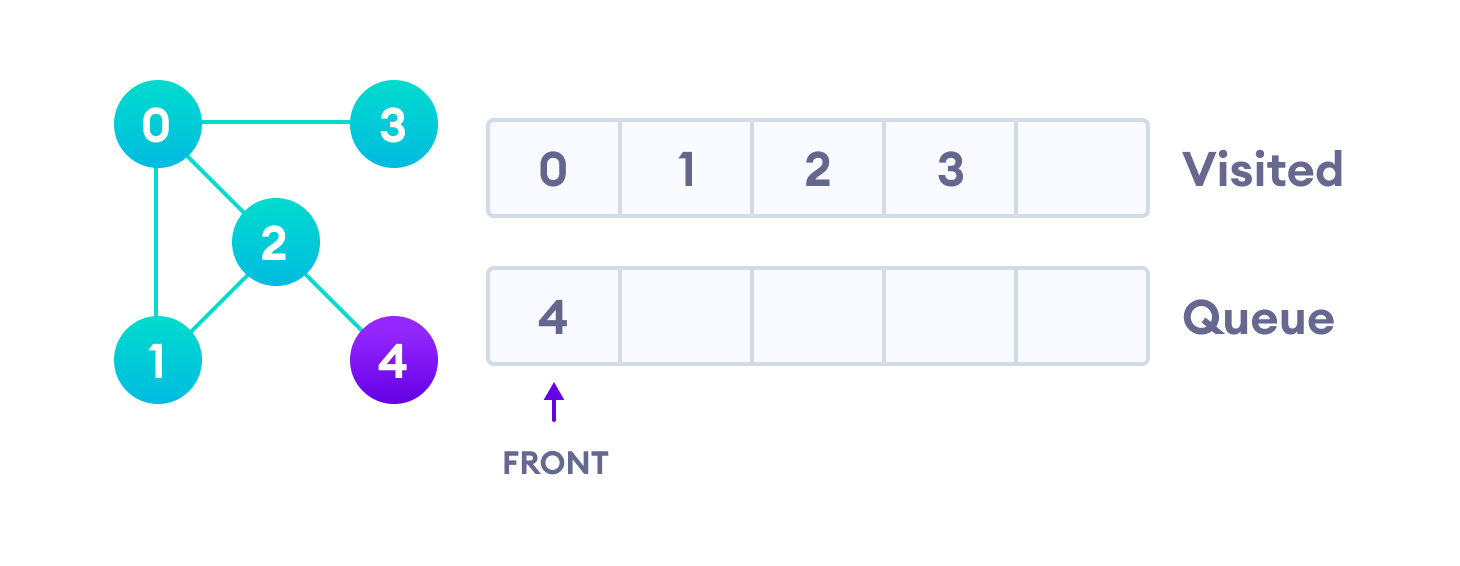


Visit start vertex and add its adjacent vertices to queue

Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the back of the queue and visit 3, which is at the front of the queue.

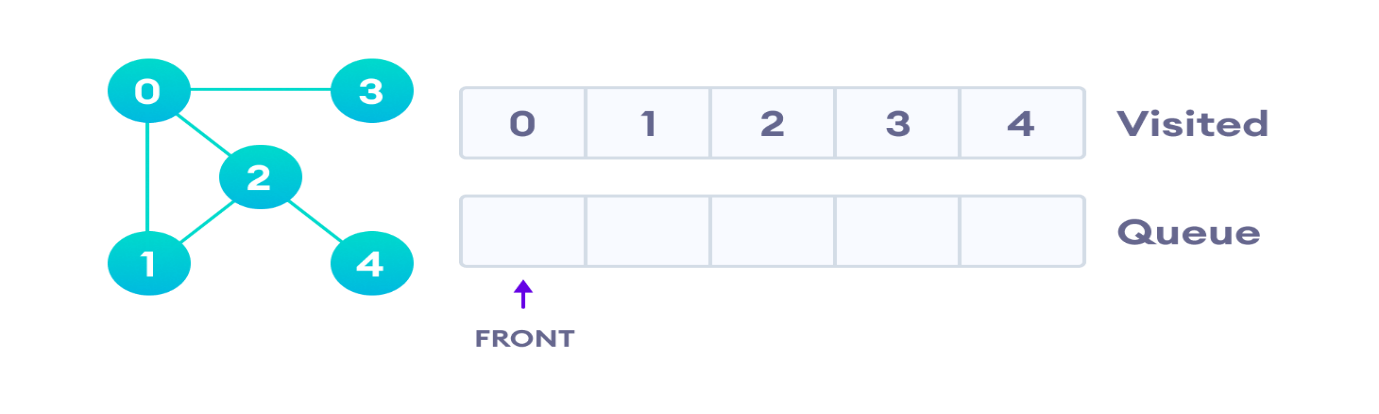


Visit 2 which was added to queue earlier to add its neighbours



4 remains in the queue

Only 4 remains in the queue since the only adjacent node of 3 i.e. 0 is already visited. We visit it.



Visit last remaining item in the queue to check if it has unvisited neighbours

Since the queue is empty, we have completed the Breadth First Traversal of the graph.

**BFS pseudocode**

create a queue Q

mark v as visited and put v into Q

while Q is non-empty

remove the head u of Q

mark and enqueue all (unvisited) neighbours of u

**// BFS algorithm in C++**

**#include <iostream>**

**#include <list>**

**using namespace std;**

**class Graph {**

**int numVertices;**

**list<int>\* adjLists;**

**bool\* visited;**

**public:**

**Graph(int vertices);**

**void addEdge(int src, int dest);**

**void BFS(int startVertex);**

**};**

**// Create a graph with given vertices,**

**// and maintain an adjacency list**

**Graph::Graph(int vertices) {**

**numVertices = vertices;**

**adjLists = new list<int>[vertices];**

**}**

**// Add edges to the graph**

**void Graph::addEdge(int src, int dest) {**

**adjLists[src].push\_back(dest);**

**adjLists[dest].push\_back(src);**

**}**

**// BFS algorithm**

**void Graph::BFS(int startVertex) {**

**visited = new bool[numVertices];**

**for (int i = 0; i < numVertices; i++)**

**visited[i] = false;**

**list<int> queue;**

**visited[startVertex] = true;**

**queue.push\_back(startVertex);**

**list<int>::iterator i;**

**while (!queue.empty()) {**

**int currVertex = queue.front();**

**cout << "Visited " << currVertex << " ";**

**queue.pop\_front();**

**for (i = adjLists[currVertex].begin(); i != adjLists[currVertex].end(); ++i) {**

**int adjVertex = \*i;**

**if (!visited[adjVertex]) {**

**visited[adjVertex] = true;**

**queue.push\_back(adjVertex);**

**}**

**}**

**}**

**}**

**int main() {**

**Graph g(4);**

**g.addEdge(0, 1);**

**g.addEdge(0, 2);**

**g.addEdge(1, 2);**

**g.addEdge(2, 0);**

**g.addEdge(2, 3);**

**g.addEdge(3, 3);**

**g.BFS(2);**

**return 0;**

**}**