Data Structures Project

Submitted by

Team 5

III-CSE

Jyothy Institute of Technology

Course Outcomes

CO1: To describe different types of data structures and their applications.
CO2: To apply pattern matching, sorting, searching algorithms for problem solving.

CO3: Implement all data structures in a high level language for problem solving.

Project Statement

AIR TRAVEL GRAPH CREATOR AND ROUTE FINDER USING BFS TECHNIQUE.

The program must be able to find all the possible routes that the flight can take from the source in a particular graph using the BFS technique.

Abstract

The **Breadth First Search (BFS)** traversal technique is used in finding the route as well as in creating the graph for a given air travel. Breadth-first search (BFS) is an important **graph search algorithm** that is used to solve many problems including finding the shortest path in a graph and solving puzzle games.

Many problems in Computer Science can be thought of in terms of graphs. For example, analyzing networks, mapping routes, and scheduling are graph problems. Graph search algorithms like Breadth First Search are

useful for analyzing and solving graph problems.

Thus, our aim is to show the implementation of the BFS technique for determining the routes of airlines for the smooth commute of flights such that there is no interference in the flight's route, i.e. , sets a predestined calculated path for the airline to follow in order to reach its destination in the shortest time.

Keywords

Breadth First Search: It uses Queue data structure for finding the shortest path. BFS can be used to find a single source shortest path in an unweighted graph,because in BFS, we reach a vertex with a minimum number of edges from a source vertex.

Graph: A graph G consists of two sets;

- a finite, non empty set of vertices V(G)
- a finite, possible empty set of edges E(G) then G(V,E) represents a graph.

Queue: A queue is a collection of items in which only the earliest added items may be accessed. It is an ordered list in which insertions and deletions take place at different ends.it follows the First-in-First-Out(FIFO) structure.The first object into a queue is the first object to leave the queue.

- Enqueue: Adding elements to the queue.
- Dequeue: Removing elements from the queue.

Adjacency Matrix: Adjacency Matrix is a square matrix used to represent a finite graph. The elements of the matrix indicate whether the pair of vertices are adjacent or not.

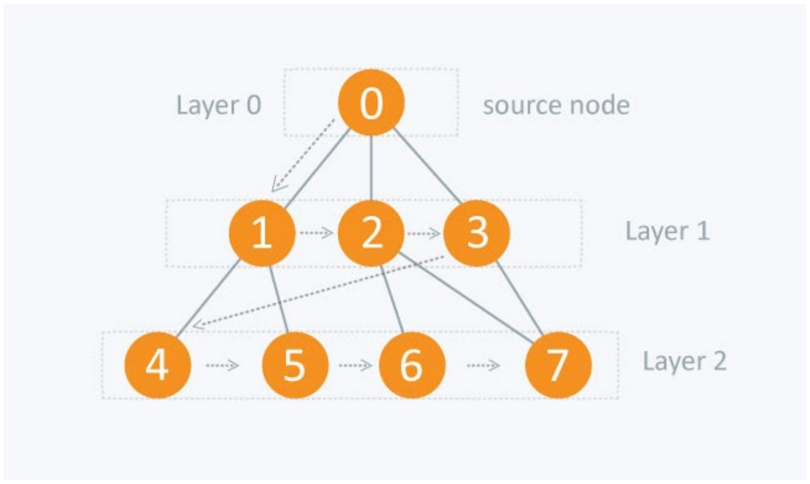
Introduction

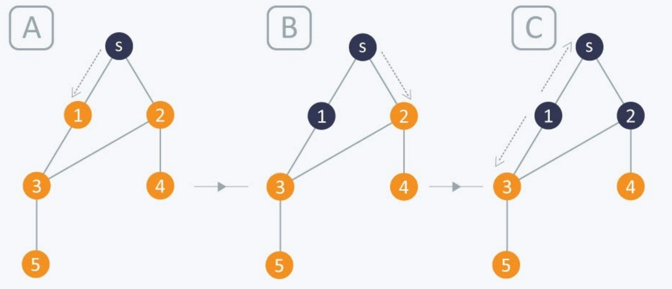
Breadth First Search

BFS is a graph search algorithm that begins with a root node and explores all the neighbouring nodes. Then for each of those nearest nodes, the algorithm explores their unexplored neighbour nodes, and so on, until it finds the goal. As the name BFS suggests, you are required to traverse the graph breadthwise as follows:

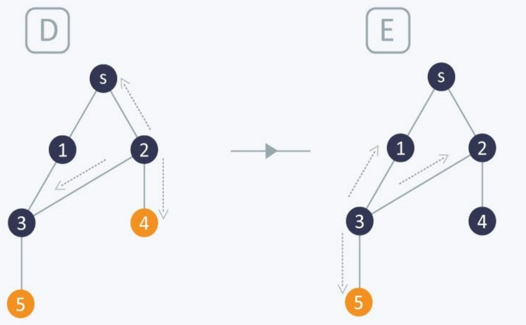
1. First move horizontally and visit all the nodes of the current layer,
2. Move to the next layer.

Consider this example:



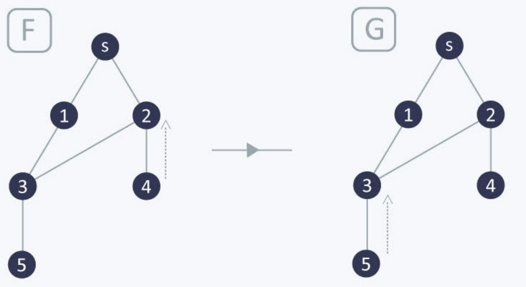


Here s is already marked, so it will be ignored



Here s and 3 are already marked, so they will be ignored

Here 1 & 2 are already marked so they will be ignored



Here 2 is already marked, so it will be ignored

Here 3 is already marked, so it will be ignored

Advantages of BFS

There are numerous reasons to utilize the BFS Algorithm to use as searching for your dataset. Some of the most vital aspects that make this algorithm your first choice are:

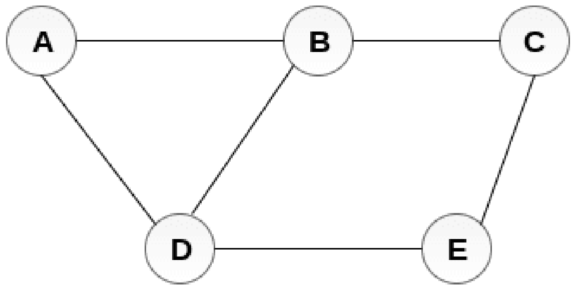
- BFS is useful for analyzing the nodes in a graph and constructing the shortest path of traversing through these.
- BFS can traverse through a graph in the smallest number of iterations.
- The architecture of the BFS algorithm is simple and robust.
- The result of the BFS algorithm holds a high level of accuracy in comparison to other algorithms.
- BFS iterations are seamless, and there is no possibility of this algorithm getting caught up in an infinite loop problem.

Adjacency Matrix

An adjacency matrix is a square matrix used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices are adjacent or not in the graph. In the special case of a finite simple graph, the adjacency matrix is a (0,1)-matrix with zeros on its diagonal.

If the graph is directed, then the matrix is symmetric, otherwise(undirected graph) the matrix will be asymmetric.

Example for an undirected graph:



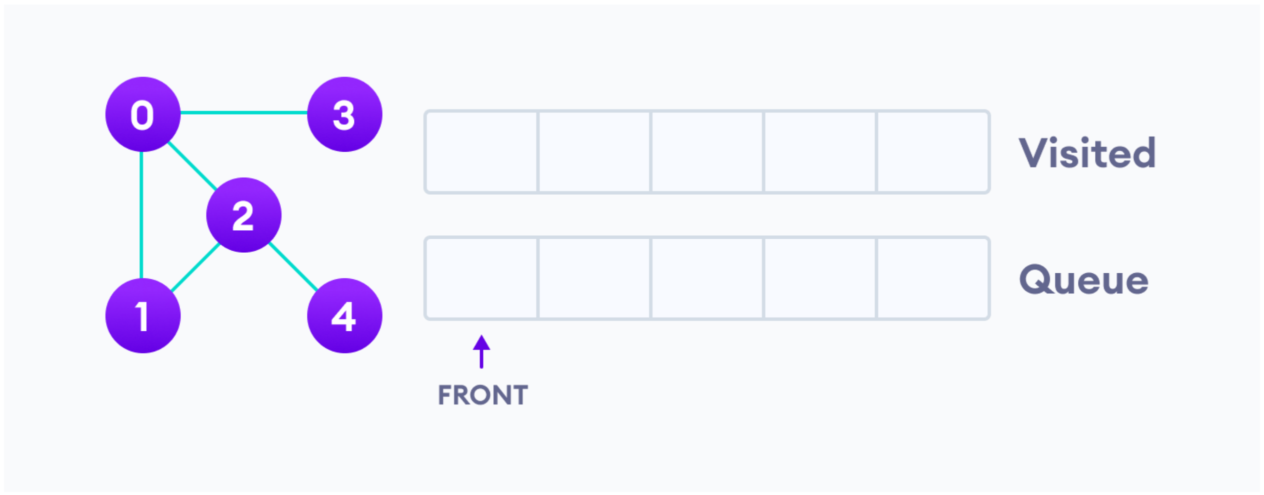
Undirected Graph

$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 1 & 0 & 1 & 0 \\ B & 1 & 0 & 1 & 1 & 0 \\ C & 0 & 1 & 0 & 0 & 1 \\ D & 1 & 1 & 0 & 0 & 1 \\ E & 0 & 0 & 1 & 1 & 0 \end{array}$$

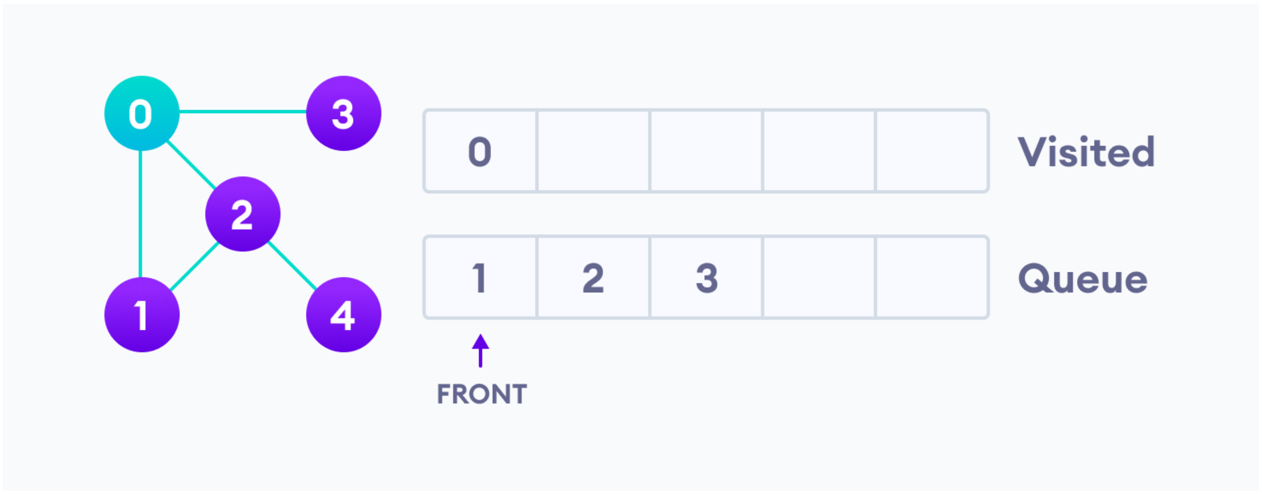
Adjacency Matrix

Proposed Solution

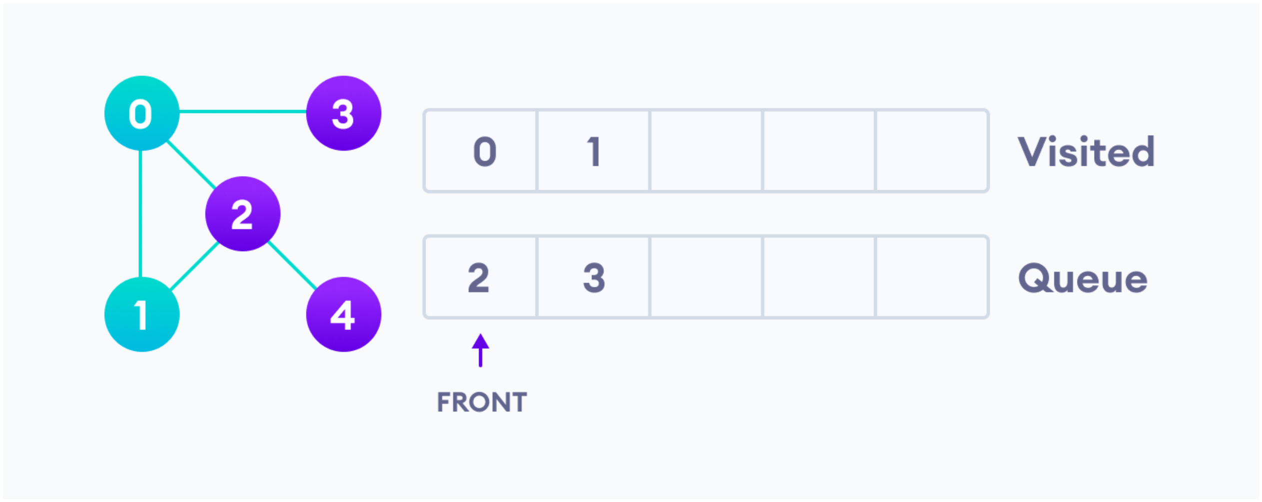
Let's see how the Breadth First Search algorithm works with an example. We use an undirected graph with 5 vertices.



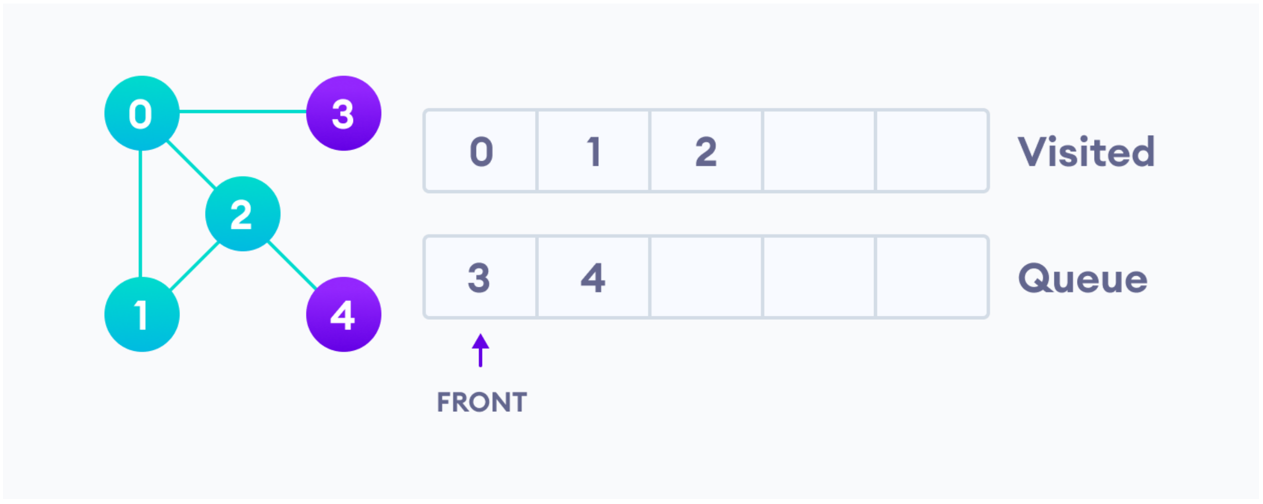
We start from vertex 0, the BFS algorithm starts by putting it in the Visited list and putting all its adjacent vertices in the stack.

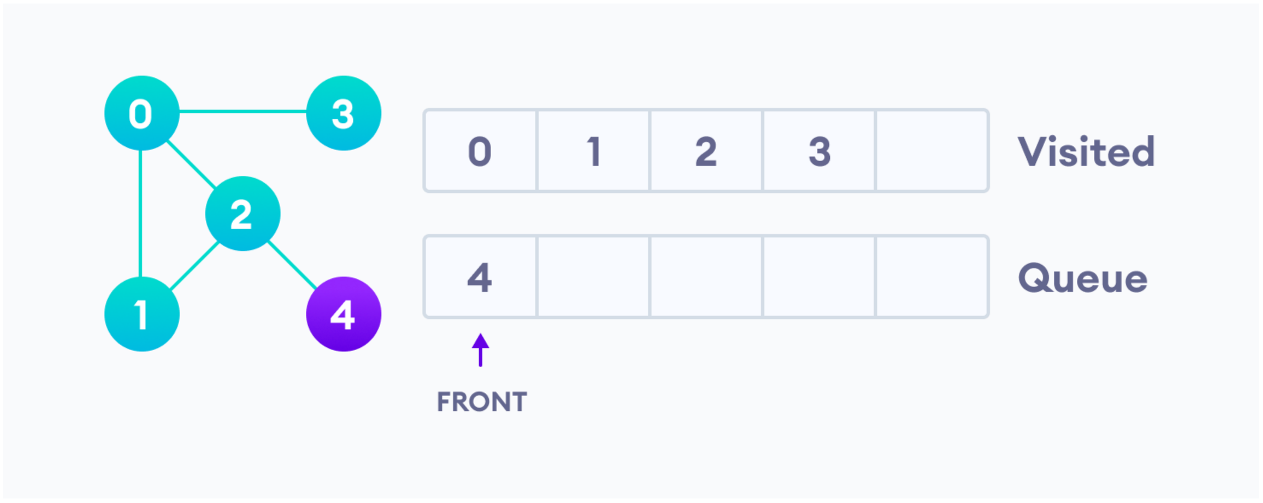


Next, we visit the element at the front of queue i.e. 1 and go to its adjacent nodes. Since 0 has already been visited, we visit 2 instead.

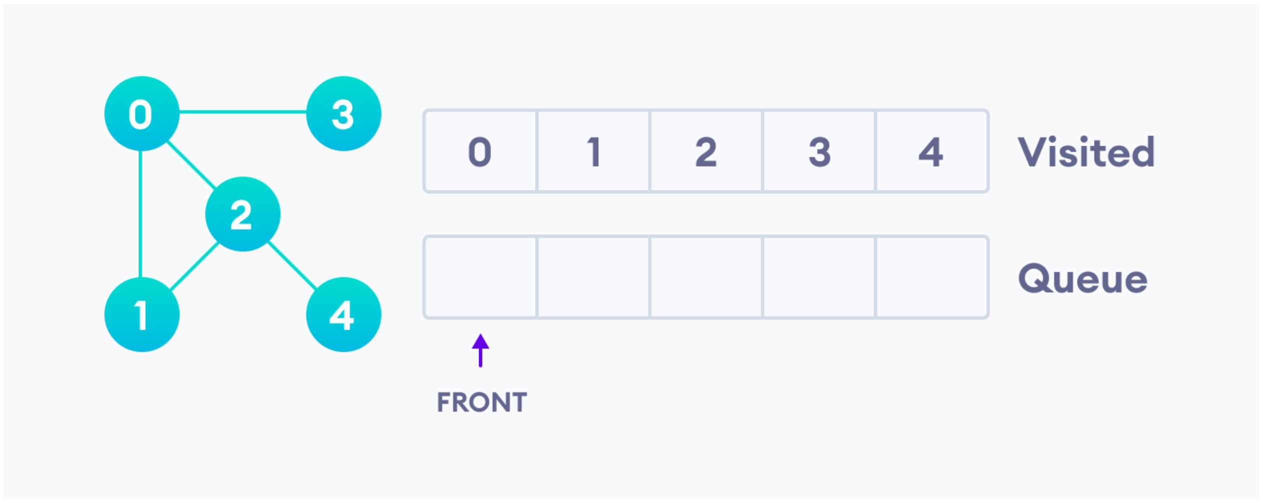


Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the back of the queue and visit 3, which is at the front of the queue.





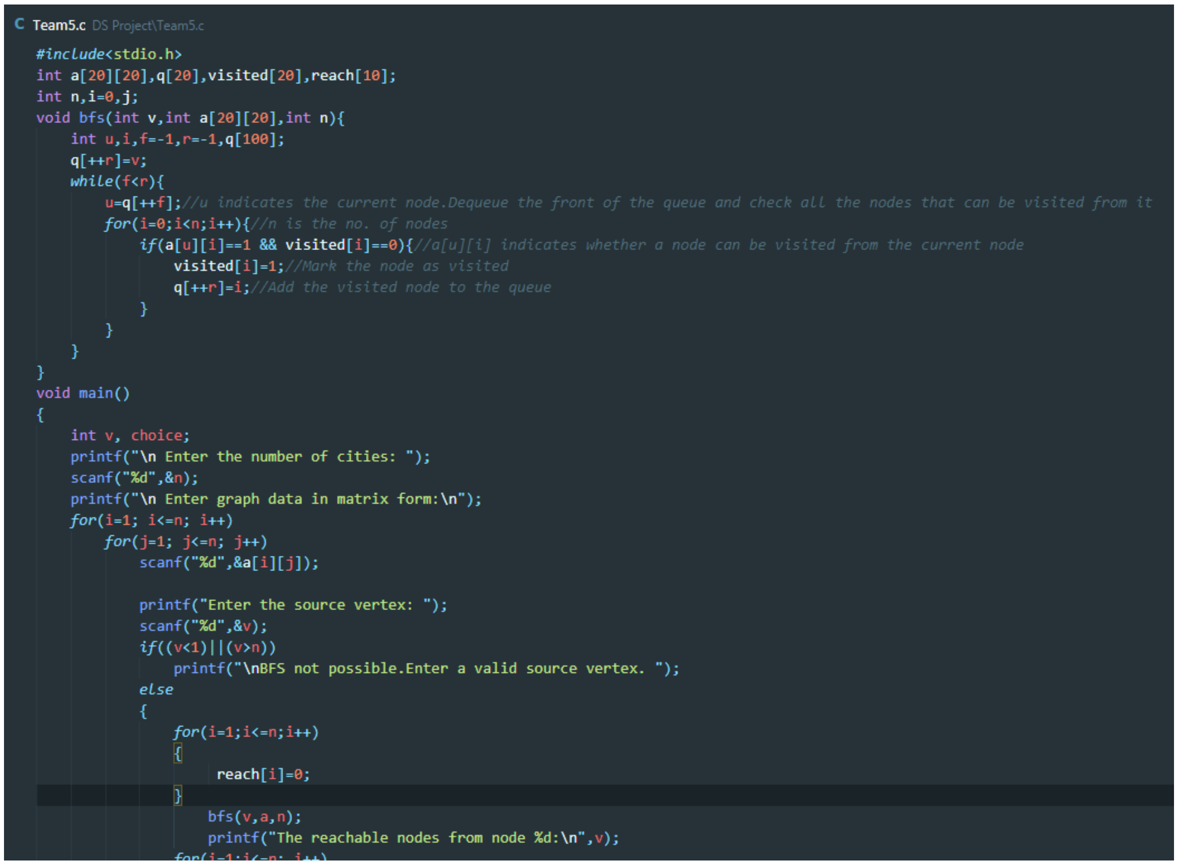
Only 4 remains in the queue since the only adjacent node of 3 i.e. 0 is already visited. We visit it.



Since the queue is empty, we have completed the Breadth First Search Technique.
Now after this there are no more unvisited nodes, hence the program terminates after it empties the queue.

Therefore, the algorithm is as follows:

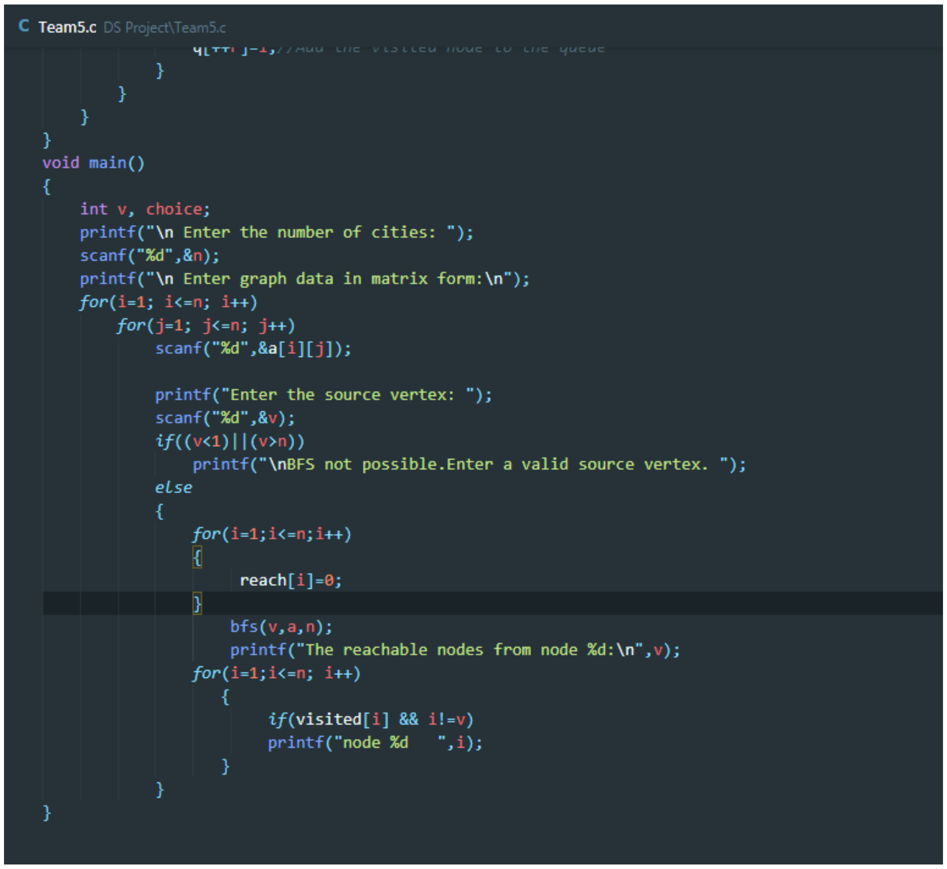
- Rule 1 – Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue.
- Rule 2 – If no adjacent vertex is found, visit the first vertex from the queue and dequeue that vertex after repeating Rule 1.
- Rule 3 – Repeat Rule 1 and Rule 2 until the queue is empty.



Team5.c DS Project\Team5.c

```c
#include<stdio.h>
int a[20][20],q[20],visited[20],reach[10];
int n,i=0,j;
void bfs(int v,int a[20][20],int n){
    int u,i,f=-1,r=-1,q[100];
    q[++r]=v;
    while(f<r){
        u=q[++f];//u indicates the current node.Dequeue the front of the queue and check all the nodes that can be visited from it
        for(i=0;i<n;i++){//n is the no. of nodes
            if(a[u][i]==1 && visited[i]==0){//a[u][i] indicates whether a node can be visited from the current node
                visited[i]=1;//Mark the node as visited
                q[++r]=i;//Add the visited node to the queue
            }
        }
    }
}
void main()
{
    int v, choice;
    printf("\n Enter the number of cities: ");
    scanf("%d",&n);
    printf("\n Enter graph data in matrix form:\n");
    for(i=1; i<=n; i++)
        for(j=1; j<=n; j++)
            scanf("%d",&a[i][j]);

            printf("Enter the source vertex: ");
            scanf("%d",&v);
            if((v<1)||(v>n))
                printf("\nBFS not possible.Enter a valid source vertex. ");
            else
            {
                for(i=1;i<=n;i++)
                {
                    reach[i]=0;
                }
                    bfs(v,a,n);
                    printf("The reachable nodes from node %d:\n",v);
                for(i=1;i<=n; i++)
```

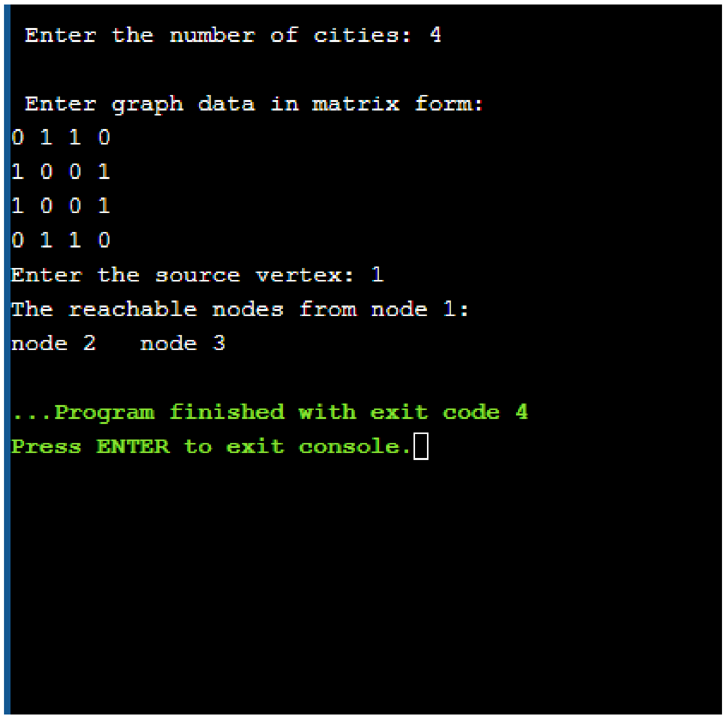


Team5.c DS Project\Team5.c

```c
                }
            }
        }
}
void main()
{
    int v, choice;
    printf("\n Enter the number of cities: ");
    scanf("%d",&n);
    printf("\n Enter graph data in matrix form:\n");
    for(i=1; i<=n; i++)
        for(j=1; j<=n; j++)
            scanf("%d",&a[i][j]);

            printf("Enter the source vertex: ");
            scanf("%d",&v);
            if((v<1)||(v>n))
                printf("\nBFS not possible.Enter a valid source vertex. ");
            else
            {
                for(i=1;i<=n;i++)
                {
                    reach[i]=0;
                }
                    bfs(v,a,n);
                    printf("The reachable nodes from node %d:\n",v);
                for(i=1;i<=n; i++)
                    {
                        if(visited[i] && i!=v)
                        printf("node %d   ",i);
                    }
            }
}
```

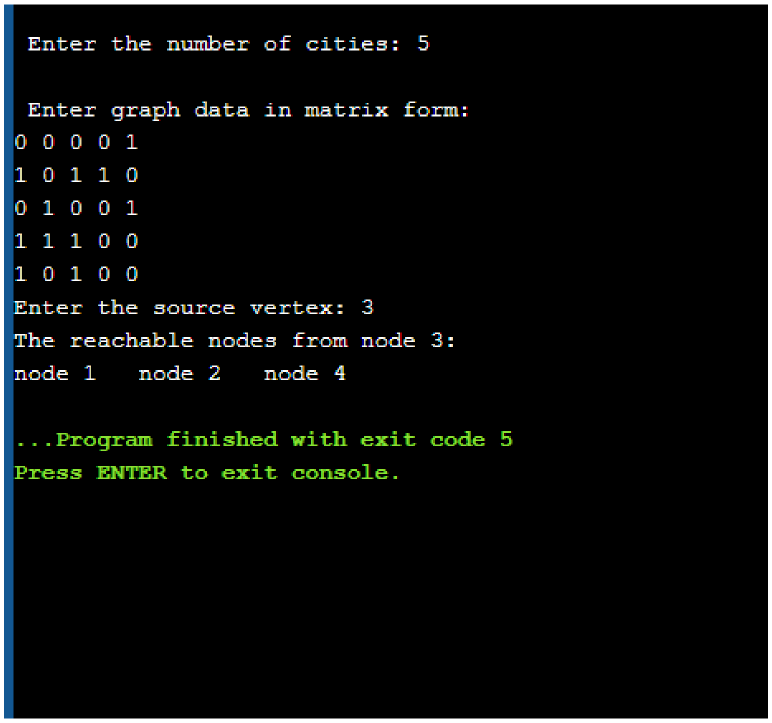
Output



```
 Enter the number of cities: 4

 Enter graph data in matrix form:
0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0
Enter the source vertex: 1
The reachable nodes from node 1:
node 2   node 3

...Program finished with exit code 4
Press ENTER to exit console.
```



```
 Enter the number of cities: 5

 Enter graph data in matrix form:
0 0 0 0 1
1 0 1 1 0
0 1 0 0 1
1 1 1 0 0
1 0 1 0 0
Enter the source vertex: 3
The reachable nodes from node 3:
node 1   node 2   node 4

...Program finished with exit code 5
Press ENTER to exit console.
```