**Write a program to traverse a graph using BFS method.**

```
#include <stdio.h>
#define MAX_VERTICES 10

int n, i, j, visited[MAX_VERTICES], queue[MAX_VERTICES], front = 0, rear = 0;
int adj[MAX_VERTICES][MAX_VERTICES];

void bfs(int v) {
    visited[v] = 1;
    queue[rear++] = v;

    while (front < rear) {
        int current = queue[front++];
        printf("%d\t", current);

        for (int i = 0; i < n; i++) {  // Corrected loop condition
            if (adj[current][i] && !visited[i]) {
                visited[i] = 1;  // Corrected setting visited flag
                queue[rear++] = i;
            }
        }
    }
}

int main() {
    int v;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);

    for (i = 0; i < n; i++) {  // Corrected loop condition
        visited[i] = 0;  // Initialize all nodes as unvisited
    }

    printf("Enter graph data in matrix form:\n");
    for (i = 0; i < n; i++)  // Corrected loop condition
        for (j = 0; j < n; j++)  // Corrected loop condition
```

```
            scanf("%d", &adj[i][j]);

    printf("Enter the starting vertex: ");
    scanf("%d", &v);
    bfs(v);

    for (i = 0; i < n; i++) {  // Corrected loop condition
        if (!visited[i]) {
            printf("\nBFS is not possible. Not all nodes are reachable.\n");
            return 0;
        }
    }

    return 0;
}
```

```
Enter the number of vertices: 7
Enter graph data in matrix form:
0 1 0 1 0 0 0
1 0 1 1 0 1 1
0 1 0 1 1 1 0
1 1 1 0 1 0 0
0 0 1 1 0 0 1
0 1 1 0 0 0 0
0 1 0 0 1 0 0
Enter the starting vertex: 2
2       1       3       4       5       0       6
```

**Write a program to check whether given graph is connected or not using DFS method**

```
#include <stdio.h>
#include <stdbool.h>

#define MAX_VERTICES 10

void dfs(int graph[MAX_VERTICES][MAX_VERTICES], int num_vertices, bool
visited[MAX_VERTICES], int vertex) {
    visited[vertex] = true;
    int i;
    for ( i = 0; i < num_vertices; ++i) {
        if (graph[vertex][i] == 1 && !visited[i]) {
            dfs(graph, num_vertices, visited, i);
        }
    }
}

bool is_connected(int graph[MAX_VERTICES][MAX_VERTICES], int
num_vertices) {
    bool visited[MAX_VERTICES] = {false};

    // Perform DFS starting from vertex 0
    dfs(graph, num_vertices, visited, 0);
    int i;

    // Check if all vertices were visited
    for ( i = 0; i < num_vertices; ++i) {
        if (!visited[i]) {
            return false;
        }
    }
    return true;
}

int main() {
```

```
    int num_vertices;
    printf("Enter the number of vertices: ");
    scanf("%d", &num_vertices);
    int i,j;
    int graph[MAX_VERTICES][MAX_VERTICES];
    printf("Enter the adjacency matrix:\n");
    for ( i = 0; i < num_vertices; ++i) {
        for (j = 0; j < num_vertices; ++j) {
            scanf("%d", &graph[i][j]);
        }
    }

    if (is_connected(graph, num_vertices)) {
        printf("The graph is connected.\n");
    } else {
        printf("The graph is not connected.\n");
    }

    return 0;
}
```

```
Enter the number of vertices: 7
Enter the adjacency matrix:
0 1 0 1 0 0 0
1 0 1 1 0 1 1
0 1 0 1 1 1 0
1 1 1 0 1 0 0
0 0 1 1 0 0 1
0 1 1 0 0 0 0
0 1 0 0 1 0 0
The graph is connected.
```