

Program Code :

```
#include <iostream>
#include <queue>
#include <stack>
#include <omp.h>
using namespace std;

class Graph {
private:
    int vertices;
    int** adjMatrix;
public:
    Graph(int v) {
        vertices = v;
        adjMatrix = new int*[vertices];
        for (int i = 0; i < vertices; i++) {
            adjMatrix[i] = new int[vertices]();
        }
    }
    void addEdge(int u, int v) {
        adjMatrix[u][v] = 1;
        adjMatrix[v][u] = 1; // Undirected graph
    }
    void parallelBFS(int start) {
        bool* visited = new bool[vertices]();
        queue<int> q;
        q.push(start);
        visited[start] = true;
        while (!q.empty()) {
            int size = q.size();
            int* currentLevel = new int[size];

            #pragma omp parallel for shared(q, visited)
```

```

for (int i = 0; i < size; i++) {
    int node;

    #pragma omp critical
    {
        node = q.front();
        q.pop();
    }

    currentLevel[i] = node;

    #pragma omp parallel for shared(q, visited)
    for (int j = 0; j < vertices; j++) {
        if (adjMatrix[node][j] == 1 && !visited[j]) {
            visited[j] = true;
            q.push(j);
        }
    }
}

#pragma omp critical
{
    for (int i = 0; i < size; i++) {
        cout << currentLevel[i] << " ";
    }
}

delete[] currentLevel;
}

cout << endl;
delete[] visited;
}

void parallelDFS(int start) {
    bool* visited = new bool[vertices]();
    stack<int> s;
    s.push(start);
    while (!s.empty()) {
        int node;

        #pragma omp critical

```

```

    {
        node = s.top();
        s.pop();
    }
    if (!visited[node]) {
        visited[node] = true;
        cout << node << " ";
    }
    #pragma omp parallel for shared(s, visited)
    for (int j = 0; j < vertices; j++) {
        if (adjMatrix[node][j] == 1 && !visited[j]) {
            s.push(j);
        }
    }
}

cout << endl;
delete[] visited;
}

~Graph() {
    for (int i = 0; i < vertices; i++) {
        delete[] adjMatrix[i];
    }
    delete[] adjMatrix;
}
};

```

```

int main() {
    int vertices, edges, u, v, startNode;
    cout << "Enter number of vertices and edges: ";
    cin >> vertices >> edges;
    Graph g(vertices);
    cout << "Enter edges (u v):\n";
    for (int i = 0; i < edges; i++) {
        cin >> u >> v;
    }
}

```

```

        g.addEdge(u, v);
    }

    cout << "Enter start node for traversal: ";
    cin >> startNode;

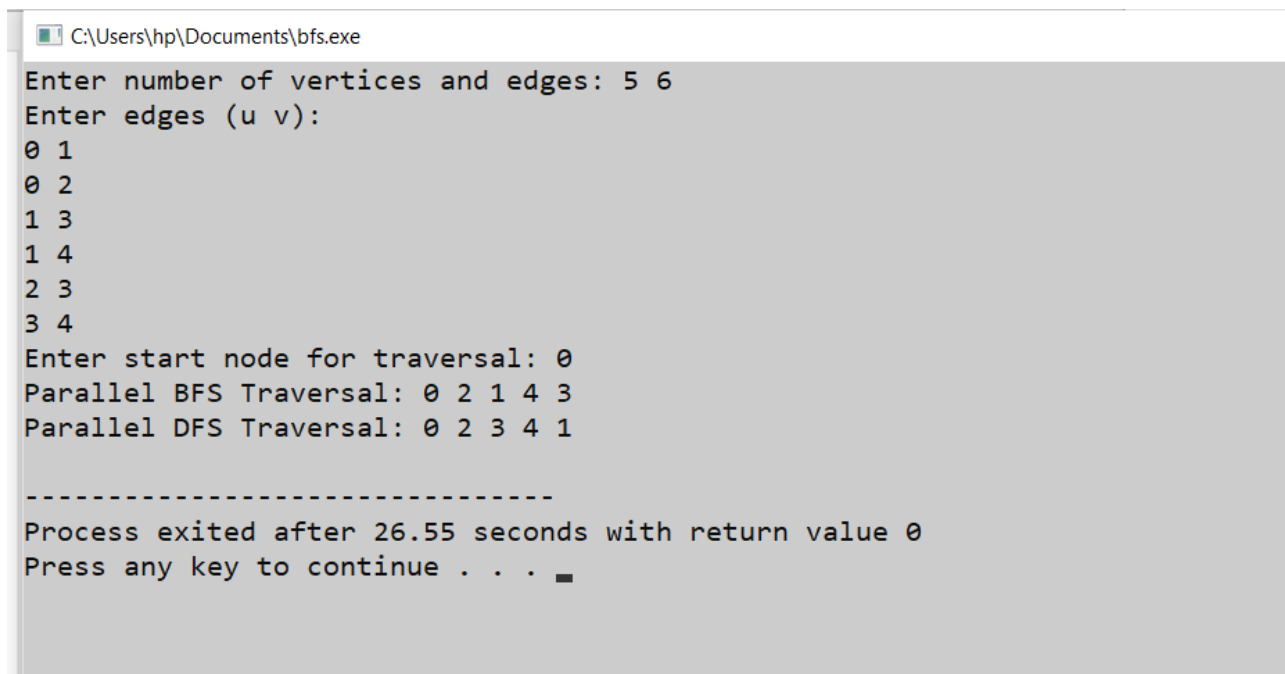
    cout << "Parallel BFS Traversal: ";
    g.parallelBFS(startNode);

    cout << "Parallel DFS Traversal: ";
    g.parallelDFS(startNode);

    return 0;
}

```

Output :



```

C:\Users\hp\Documents\bfs.exe
Enter number of vertices and edges: 5 6
Enter edges (u v):
0 1
0 2
1 3
1 4
2 3
3 4
Enter start node for traversal: 0
Parallel BFS Traversal: 0 2 1 4 3
Parallel DFS Traversal: 0 2 3 4 1

-----
Process exited after 26.55 seconds with return value 0
Press any key to continue . . .

```