BFS

Breadth-First-Search.cpp - DAA-practical - Visual Studio Code

Practical-8 > Breadth-First-Search.cpp > main()

```cpp
//  Author: Anish Tilloo
//  Roll No. : 34
//  Program: Breadth First Search

#include <bits/stdc++.h>
using namespace std;


// Creating a graph using Adjencency List(Array of List)
class Graph{
    public:
    int V;
    // pointer to an array of list
    list<int> *l;

    // we are creating a graph constructor
    // where we will take how many vertices
    // this graph will have
    Graph(int v){
        V = v;
        // Dynamically creating a new List of size Vertices
        // creating Vertices no of linked list
```

Breadth-First-Search.cpp - DAA-practical - Visual Studio Code

Practical-8 > Breadth-First-Search.cpp > main()

```cpp
    Graph(int v){
        V = v;
        // Dynamically creating a new List of size Vertices
        // creating Vertices no of linked list
        l = new list<int>[V];
    }

    void addEdge(int ver1, int ver2, bool undirected = true){
        l[ver1].push_back(ver2);
        if (undirected)
        {
            l[ver2].push_back(ver1);
        }
    }

    void printGraph(){
        // Iterating over all rows
        for (int i = 0; i < V; i++)
        {
            cout << i << "->";
            // Iterating over every element in the linked list
            for(auto node : l[i]){
```

Breadth-First-Search.cpp - DAA-practical - Visual Studio Code

Practical-8 > Breadth-First-Search.cpp > main()

```cpp

void printGraph(){
    // Iterating over all rows
    for (int i = 0; i < V; i++)
    {
        cout << i << "->";
        // Iterating over every element in the linked list
        for(auto node : l[i]){
            cout << node << ",";
        }
        cout << endl;
    }

}

void bfs(int source){
    queue<int> q;
    bool *visited = new bool[V]{0};

    // We are pushing the source node inside queue
    // and it is also visited
    q.push(source);
```

Breadth-First-Search.cpp - DAA-practical - Visual Studio Code

Practical-8 > Breadth-First-Search.cpp > main()

```cpp
void bfs(int source){
    queue<int> q;
    bool *visited = new bool[V]{0};

    // We are pushing the source node inside queue
    // and it is also visited
    q.push(source);
    visited[source] = true;

    // if the queue is not empty then pop out the node and print it
    // and insert its neighbours into the queue
    while (!q.empty())
    {
        int f = q.front();
        cout << f << " ";
        q.pop();

        // push the neighbours of the current node if they are not already visited
        for(auto nbr : l[f]){
            if (!visited[nbr]){
                q.push(nbr);
                visited[nbr] = true;
```

Breadth-First-Search.cpp - DAA-practical - Visual Studio Code

Practical-8 > Breadth-First-Search.cpp > main()

```cpp
                q.push(nbr);
                visited[nbr] = true;
            }
        }
    }
}
};


int main(){
    int n;
    cout << "Enter the number of vertices" << endl;
    cin >> n;
    Graph g(n);
    int m;
    cout << "Enter the no of edges" << endl;
    cin >> m;
    cout << endl;

    cout <<  "Edges " << endl;
    for (int i = 0; i < m; i++)
    {
```

Breadth-First-Search.cpp - DAA-practical - Visual Studio Code

Practical-8 > Breadth-First-Search.cpp > main()

```cpp
    cout <<  "Edges " << endl;
    for (int i = 0; i < m; i++)
    {
        int vex1, vex2;
        cout << "Vertex One: ";
        cin >> vex1;
        cout << "vertex Two: ";
        cin >> vex2;
        cout << endl;

        g.addEdge(vex1, vex2);
    }
    cout << endl;

    g.printGraph();

    int s;
    cout << "Enter the source Node " << endl;
    cin >> s;
    cout << "BFS" <<endl;
    g.bfs(s);
```

```
Enter the number of vertices
7
Enter the no of edges
6

Edges
Vertex One: 0
vertex Two: 1

Vertex One: 1
vertex Two: 2

Vertex One: 3
vertex Two: 5

Vertex One: 5
vertex Two: 6

Vertex One: 4
vertex Two: 5

Vertex One: 0
vertex Two: 4


0->1,4,
1->0,2,
2->1,
3->5,
```

```
Vertex One: 4
vertex Two: 5

Vertex One: 0
vertex Two: 4


0->1,4,
1->0,2,
2->1,
3->5,
4->5,0,
5->3,6,4,
6->5,
Enter the source Node
0
BFS
0 1 4 2 5 3 6
PS E:\College-Work\Sixth-Semester\DAA-practical\Practical-8>
```

DFS

Depth-First-Search-Algo.cpp - DAA-practical - Visual Studio Code

Practical-8 > Depth-First-Search-Algo.cpp > main()

```cpp
//  Author: Anish Tilloo
//  Roll No. : 34
//  Program: Depth First Search

#include <bits/stdc++.h>
using namespace std;


// Creating a graph using Adjencency List(Array of List)
class Graph{
    public:
    int V;
    // pointer to an array of list
    list<int> *l;

    // we are creating a graph constructor
    // where we will take how many vertices
    // this graph will have
    Graph(int v){
        V = v;
        // Dynamically creating a new List of size Vertices
        // creating Vertices no of linked list
```

Depth-First-Search-Algo.cpp - DAA-practical - Visual Studio Code

Practical-8 > Depth-First-Search-Algo.cpp > main()

```cpp
    Graph(int v){
        V = v;
        // Dynamically creating a new List of size Vertices
        // creating Vertices no of linked list
        l = new list<int>[V];
    }

    void addEdge(int ver1, int ver2, bool undirected = true){
        l[ver1].push_back(ver2);
        if (undirected)
        {
            l[ver2].push_back(ver1);
        }
    }

    void printGraph(){
        // Iterating over all rows
        for (int i = 0; i < V; i++)
        {
            cout << i << "->";
            // Iterating over every element in the linked list
            for(auto node : l[i]){
```

File Edit Selection View Go Run Terminal Help Depth-First-Search-Algo.cpp - DAA-practical - Visual Studio Code

Depth-First-Search-Algo.cpp

Practical-8 > Depth-First-Search-Algo.cpp > main()

```cpp
    void printGraph(){
        // Iterating over all rows
        for (int i = 0; i < V; i++)
        {
            cout << i << "->";
            // Iterating over every element in the linked list
            for(auto node : l[i]){
                cout << node << ",";
            }
            cout << endl;
        }

    }

    void dfsHelper(int node, bool *visited){
      visited[node] = true;
      cout << node << ",";
      // make a dfs call on all its unvisited neighbours
      for(auto nbr : l[node]){
        if (!visited[nbr])
        {
            dfsHelper(nbr, visited);
```

Run Testcases 0 0 Live Share Screen Reader Optimized Ln 98, Col 26 Spaces: 4 UTF-8 CRLF C++ Go Live Spell Win32 Prettier

File Edit Selection View Go Run Terminal Help Depth-First-Search-Algo.cpp - DAA-practical - Visual Studio Code

Depth-First-Search-Algo.cpp

Practical-8 > Depth-First-Search-Algo.cpp > main()

```cpp

    void dfsHelper(int node, bool *visited){
      visited[node] = true;
      cout << node << ",";
      // make a dfs call on all its unvisited neighbours
      for(auto nbr : l[node]){
        if (!visited[nbr])
        {
            dfsHelper(nbr, visited);
        }
      }
      return;
    }

    void dfs(int source){
        bool *visited = new bool [V]{0};
        dfsHelper(source, visited);
    }

};


int main(){
```

Run Testcases 0 0 Live Share Screen Reader Optimized Ln 98, Col 26 Spaces: 4 UTF-8 CRLF C++ Go Live Spell Win32 Prettier

Depth-First-Search-Algo.cpp - DAA-practical - Visual Studio Code

Practical-8 > Depth-First-Search-Algo.cpp > main()

```cpp
int main(){
    int n;
    cout << "Enter the number of vertices" << endl;
    cin >> n;
    Graph g(n);
    int m;
    cout << "Enter the no of edges" << endl;
    cin >> m;
    cout << endl;

    cout <<  "Edges " << endl;
    for (int i = 0; i < m; i++)
    {
        int vex1, vex2;
        cout << "Vertex One: ";
        cin >> vex1;
        cout << "vertex Two: ";
        cin >> vex2;
        cout << endl;

        g.addEdge(vex1, vex2);
    }
```

Depth-First-Search-Algo.cpp - DAA-practical - Visual Studio Code

Practical-8 > Depth-First-Search-Algo.cpp > main()

```cpp
        cin >> vex1;
        cout << "vertex Two: ";
        cin >> vex2;
        cout << endl;

        g.addEdge(vex1, vex2);
    }
    cout << endl;

    g.printGraph();

    int s;
    cout << "Enter the source Node " << endl;
    cin >> s;
    cout << "DFS" <<endl;
    g.dfs(s);

    return 0;
}
```

```
($?) { g++ Depth-First-Search-Algo.cpp -o Depth-First-Search-Algo } ; if ($?) { .\Depth-First-Search-Algo }
Enter the number of vertices
7
Enter the no of edges
7

Edges
Vertex One: 0
vertex Two: 1

Vertex One: 1
vertex Two: 2

Vertex One: 3
vertex Two: 5

Vertex One: 5
vertex Two: 6

Vertex One: 4
vertex Two: 5

Vertex One: 0
vertex Two: 4

Vertex One: 3
vertex Two: 4
```

```
Vertex One: 4
vertex Two: 5

Vertex One: 0
vertex Two: 4

Vertex One: 3
vertex Two: 4


0->1,4,
1->0,2,
2->1,
3->5,4,
4->5,0,3,
5->3,6,4,
6->5,
Enter the source Node
1
DFS
1,0,4,5,3,6,2,
PS E:\College-Work\Sixth-Semester\DAA-practical\Practical-8>
```