DSA Lab

Mr. ALEEM AHMAD

Bahria University

Lab # 10
Graph(BFS &DFS) Implementation

LAB Journal

Asim Ali (01-131232-015)

Lab 10: Graph（BFS &DFS）Implementation

TASK:

BFS and DFS

Lab Task GitHub Link:

Link

OUTPUT:

```
Binary Tree Menu:
1. Create Tree
2. Perform BFS
3. Perform DFS
4. Exit
Enter your choice: 1
Enter the value for the root node: 5
Enter left child of 5 (-1 for no child): 4
Enter right child of 5 (-1 for no child): 6
Enter left child of 4 (-1 for no child): -1
Enter right child of 4 (-1 for no child): -1
Enter left child of 6 (-1 for no child): -1
Enter right child of 6 (-1 for no child): -1
Tree created successfully.
```

```
Binary Tree Menu:
1. Create Tree
2. Perform BFS
3. Perform DFS
4. Exit
Enter your choice: 2
Breadth-First Search (BFS): 5 4 6
```

```
Binary Tree Menu:
1. Create Tree
2. Perform BFS
3. Perform DFS
4. Exit
Enter your choice: 3
Depth-First Search (DFS): 5 4 6
```

CODE:

```
#include <iostream>
#include <queue>
#include <stack>

using namespace std;

class Node {
public:
    int data;
    Node* left;
    Node* right;

    Node(int key) {
        data = key;
        left = right = nullptr;
    }
};

// Breadth-First Search
void BFS(Node* root) {
    if (root == nullptr) {
        return;
    }
    queue<Node*> q;
    q.push(root);
    while (!q.empty()) {
        Node* node = q.front();
        q.pop();
        cout << node->data << " ";
        if (node->left != nullptr) {
            q.push(node->left);
        }
        if (node->right != nullptr) {
            q.push(node->right);
        }
    }
    cout << endl;
}

// Depth-First Search
void DFS(Node* root) {
    if (root == nullptr) {
        return;
    }
    stack<Node*> s;
    s.push(root);
    while (!s.empty()) {
        Node* node = s.top();
        s.pop();
        cout << node->data << " ";
        // Push right child first so that left child is processed first
        if (node->right != nullptr) {
            s.push(node->right);
        }
        if (node->left != nullptr) {
            s.push(node->left);
```

```
            }
        }
        cout << endl;
    }

    Node* createTree() {
        cout << "Enter the value for the root node: ";
        int rootData;
        cin >> rootData;
        Node* root = new Node(rootData);

        queue<Node*> q;
        q.push(root);

        while (!q.empty()) {
            Node* current = q.front();
            q.pop();

            cout << "Enter left child of " << current->data << " (-1 for no child): ";
            int leftData;
            cin >> leftData;
            if (leftData != -1) {
                current->left = new Node(leftData);
                q.push(current->left);
            }

            cout << "Enter right child of " << current->data << " (-1 for no child): ";
            int rightData;
            cin >> rightData;
            if (rightData != -1) {
                current->right = new Node(rightData);
                q.push(current->right);
            }
        }

        return root;
    }

    int main() {
        Node* root = nullptr;
        int choice;

        do {
            cout << "\nBinary Tree Menu:\n";
            cout << "1. Create Tree\n";
            cout << "2. Perform BFS\n";
            cout << "3. Perform DFS\n";
            cout << "4. Exit\n";
            cout << "Enter your choice: ";
            cin >> choice;

            switch (choice) {
            case 1: {
                root = createTree();
                cout << "Tree created successfully.\n";
                break;
            }
            case 2:
```

```
            if (root == nullptr) {
                cout << "Tree is empty. Create the tree first.\n";
            }
            else {
                cout << "Breadth-First Search (BFS): ";
                BFS(root);
            }
            break;
        case 3:
            if (root == nullptr) {
                cout << "Tree is empty. Create the tree first.\n";
            }
            else {
                cout << "Depth-First Search (DFS): ";
                DFS(root);
            }
            break;
        case 4:
            cout << "Exiting program.\n";
            break;
        default:
            cout << "Invalid choice! Please try again.\n";
        }
    } while (choice != 4);

    return 0;
}
```