Министерство науки и высшего образования Российской Федерации
Федеральное государственное
бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Фундаментальные Науки»

КАФЕДРА ФН-12 «Математическое моделирование»

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ НА ТЕМУ:

Обход графовых структур

Студент: _____ Мациевский И. М.

дата, подпись Ф.И.О.

Преподаватель: _____ Волкова Л. Л.

дата, подпись Ф.И.О.

Москва, 2023

Содержание

Введение **2**

1 Аналитическая часть **3**

2 Конструкторская часть **4**

3 Технологическая часть **7**

Заключение **13**

Список используемых источников **14**

Введение

Цель лабораторной работы: реализовать алгоритмы обхода графовых структур. Для достижения поставленной цели требуется решить следующие **задачи**.

1. Описать граф и бинарное дерево.
2. Реализовать графовую структуру.
3. Реализовать возможность пользователю самостоятельно заполнять графовую структуру.
4. Реализовать обход графовой структуры в ширину и в глубину.
5. Выполнить тестирование реализации разработанного алгоритма.

Согласно варианту, требуется разработать бинарное дерево.

1 Аналитическая часть

Граф — это абстрактная структура данных, представляющая собой набор вершин (узлов) и рёбер, соединяющих их.

Граф с петлями — граф, в котором рёбра могут соединять вершину с самой собой, а также быть кратными, то есть соединять одни и те же вершины несколько раз.

Бинарное дерево — это иерархическая структура данных в виде дерева, в которой каждый узел может иметь не более двух дочерних узлов: левый и правый. У дерева есть две главные характеристики:

1. Корень — верхний узел дерева, от которого начинаются все другие узлы. У бинарного дерева может быть только один корень.

2. Листья — узлы без дочерних узлов называются листьями. Листья находятся на самом нижнем уровне дерева. **Обход в глубину** — алгоритм обхода графа или дерева, начиная с выбранной вершины и продвигаясь максимально вглубь, прежде чем возвращаться к предыдущей вершине. В процессе обхода отмечаются посещенные вершины. **Обход в ширину** — алгоритм обхода графа или дерева, начиная с выбранной вершины и постепенно расширяясь на смежные вершины одного уровня перед переходом к следующему уровню. В процессе обхода отмечаются посещенные вершины.

2 Конструкторская часть

Бинарное дерево

1. **Создание корня дерева:** Пользователь вызывает функцию $addElementToTree()$. Вводится значение для корня дерева, которое затем становится корнем нового узла.

2. **Метод** $addElementToTree()$ добавляет элемент в дерево: Создается узел с введенным значением и устанавливается как корень дерева. Вызов вспомогательной функции addElement(root) для добавления левого и правого потомка корня.

3. **Метод** *addElement* Запрос у пользователя для добавления левого узла к текущему узлу. Если ответ "да то вводится значение для левого узла, создается новый узел и устанавливается в качестве левого потомка текущего узла. Затем рекурсивно вызывается *addElement* для левого потомка. Аналогичные шаги для правого узла.

4. **Вывод результатов:** Вызываются методы $printBreadthTree()$ и $printDepthFirstTree()$ для вывода элементов дерева в ширину и в глубину соответственно.

Обход в ширину $BreadthTree$**:**

1. Проверяет, является ли переданный узел нулевым. Если да, то возвращается.

2. Иначе, создается очередь *nodesQueue*, и корень дерева помещается в очередь.

3. В цикле, пока очередь не пуста, извлекается передний узел очереди.

4. Выводится значение текущего узла.

5. Если у текущего узла есть левый потомок, он добавляется в очередь.

6. Если у текущего узла есть правый потомок, он также добавляется в очередь.

Блок-схема представлена на рис. 1.

Рис. 1 – Блок-схема обхода в ширину

Обход в глубину $DepthTree$**:**

1. Проверяет, является ли переданный узел нулевым. Если да, то возвращается.
2. Иначе, создается стек (nodesStack), и корень дерева помещается в стек.
3. В цикле, пока стек не пуст, извлекается верхний узел стека.
4. Выводится значение текущего узла.
5. Если у текущего узла есть правый потомок, он добавляется в стек.
6. Если у текущего узла есть левый потомок, он добавляется в стек.

Блок-схема представлена на рис. 2.

Рис. 2 – Блок-схема обхода в глубину

3 Технологическая часть

Для реализации выбран язык C++. На листинге 1 представлена реализация программы (Реализация 1)

Листинг 1 – Исходный код

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

using namespace std;

template <typename T>
class BinaryTreeNode {
public:
    T data;
    BinaryTreeNode<T>* left;
    BinaryTreeNode<T>* right;

    BinaryTreeNode(T value) : data(value), left(nullptr), right(nullptr
   ) {}
};

template <typename T>
class BinaryTree {
private:
    BinaryTreeNode<T>* root;

    void addElement(BinaryTreeNode<T>* node) {
        string choice;

        cout << "Добавить значение в левый узел для узла " << node->data <<
   "? данет(/): ";
        cin >> choice;

        if (choice == "да") {
            T value;
            cout << "Введите значение для левого узла: ";
            cin >> value;
            node->left = new BinaryTreeNode<T>(value);
            addElement(node->left);
```

```
    }
    else if (choice == "нет") {
        node->left = nullptr;
    }

    cout << "Добавить значение в правый узел для узла " << node->data <<
"? данет(/): ";
    cin >> choice;

    if (choice == "да") {
        T value;
        cout << "Введите значение для правого узла: ";
        cin >> value;
        node->right = new BinaryTreeNode<T>(value);
        addElement(node->right);
    }
    else if (choice == "нет") {
        node->right = nullptr;
    }
}

void BreadthTree(BinaryTreeNode<T>* node) {
    if (node == nullptr) {
        return;
    }

    queue<BinaryTreeNode<T>*> nodesQueue;
    nodesQueue.push(node);

    while (!nodesQueue.empty()) {
        BinaryTreeNode<T>* current = nodesQueue.front();
        nodesQueue.pop();
        cout << current->data << " ";

        if (current->left != nullptr) {
            nodesQueue.push(current->left);
        }
        if (current->right != nullptr) {
            nodesQueue.push(current->right);
        }
    }
```

```
    }

    void DepthTree(BinaryTreeNode<T>* node) {
        if (node == nullptr) {
            return;
        }

        stack<BinaryTreeNode<T>*> nodesStack;
        nodesStack.push(node);

        while (!nodesStack.empty()) {
            BinaryTreeNode<T>* current = nodesStack.top();
            nodesStack.pop();
            std::cout << current->data << " ";

            if (current->right != nullptr) {
                nodesStack.push(current->right);
            }
            if (current->left != nullptr) {
                nodesStack.push(current->left);
            }
        }
    }

public:
    BinaryTree() : root(nullptr) {}

    void addElementToTree() {
        T value;
        cout << "Введите значение для корня дерева: ";
        cin >> value;
        root = new BinaryTreeNode<T>(value);
        addElement(root);
    }

    void printBreadthTree() {
        cout << "Обход дерева в ширину: ";
        BreadthTree(root);
        cout << endl;
    }
```

```
    void printDepthFirstTree() {
        cout << "Обход дерева в глубину: ";
        DepthTree(root);
        cout << endl;
    }
};

int main() {
    setlocale(0, "");
    BinaryTree<int> binaryTree;
    binaryTree.addElementToTree();
    cout << endl;
    binaryTree.printBreadthTree();
    cout << endl;
    binaryTree.printDepthFirstTree();
    return 0;
}
```

Примеры работы

На рисунках примеры работы программы.

1. Входные файлы: граф, состоящий из переменных типа *char*. Результат приведён на рис. 3.

```
Введите значение для корня дерева: A
Добавить значение в левый узел для узла A? (да/нет): да
Введите значение для левого узла: T
Добавить значение в левый узел для узла T? (да/нет): да
Введите значение для левого узла: U
Добавить значение в левый узел для узла U? (да/нет): да
Введите значение для левого узла: Q
Добавить значение в левый узел для узла Q? (да/нет): нет
Добавить значение в правый узел для узла Q? (да/нет): нет
Добавить значение в правый узел для узла U? (да/нет): да
Введите значение для правого узла: L
Добавить значение в левый узел для узла L? (да/нет): нет
Добавить значение в правый узел для узла L? (да/нет): нет
Добавить значение в правый узел для узла T? (да/нет): да
Введите значение для правого узла: N
Добавить значение в левый узел для узла N? (да/нет): нет
Добавить значение в правый узел для узла N? (да/нет): да
Введите значение для правого узла: S
Добавить значение в левый узел для узла S? (да/нет): нет
Добавить значение в правый узел для узла S? (да/нет): нет
Добавить значение в правый узел для узла A? (да/нет): да
Введите значение для правого узла: C
Добавить значение в левый узел для узла C? (да/нет): да
Введите значение для левого узла: Z
Добавить значение в левый узел для узла Z? (да/нет): да
Введите значение для левого узла: J
Добавить значение в левый узел для узла J? (да/нет): нет
Добавить значение в правый узел для узла J? (да/нет): да
Введите значение для правого узла: K
Добавить значение в левый узел для узла K? (да/нет): нет
Добавить значение в правый узел для узла K? (да/нет): нет
Добавить значение в правый узел для узла Z? (да/нет): нет
Добавить значение в правый узел для узла C? (да/нет): да
Введите значение для правого узла: O
Добавить значение в левый узел для узла O? (да/нет): нет
Добавить значение в правый узел для узла O? (да/нет): нет

Обход дерева в ширину: A T C U N Z O Q L S J K

Обход дерева в глубину: A T U Q L N S C Z J K O
```

Рис. 3 – Пример работы 1

2. Входные файлы: граф, состоящий из переменных типа *int*. Результат приведён на рис. 4.

```
Введите значение для корня дерева: 7
Добавить значение в левый узел для узла 7? (да/нет): да
Введите значение для левого узла: 2
Добавить значение в левый узел для узла 2? (да/нет): да
Введите значение для левого узла: 9
Добавить значение в левый узел для узла 9? (да/нет): да
Введите значение для левого узла: 6
Добавить значение в левый узел для узла 6? (да/нет): нет
Добавить значение в правый узел для узла 6? (да/нет): нет
Добавить значение в правый узел для узла 9? (да/нет): да
Введите значение для правого узла: 33
Добавить значение в левый узел для узла 33? (да/нет): нет
Добавить значение в правый узел для узла 33? (да/нет): нет
Добавить значение в правый узел для узла 2? (да/нет): да
Введите значение для правого узла: 1
Добавить значение в левый узел для узла 1? (да/нет): нет
Добавить значение в правый узел для узла 1? (да/нет): нет
Добавить значение в правый узел для узла 7? (да/нет): да
Введите значение для правого узла: 5
Добавить значение в левый узел для узла 5? (да/нет): да
Введите значение для левого узла: 10
Добавить значение в левый узел для узла 10? (да/нет): нет
Добавить значение в правый узел для узла 10? (да/нет): нет
Добавить значение в правый узел для узла 5? (да/нет): нет

Обход дерева в ширину: 7 2 5 9 1 10 6 33

Обход дерева в глубину: 7 2 9 6 33 1 5 10
Program ended with exit code: 0
```

Рис. 4 – Пример работы 2

Заключение

Цель лабораторной работы достигнута: реализован алгоритмы обхода бинарного дерева.

В результате выполнения лабораторной работы были выполнены все задачи.

1. Описано бинарное дерево.
2. Программно реализовано бинарное дерево.
3. Реализована возможность пользователю самостоятельно заполнять бинарное дерево.
4. Реализован обход бинарного дерева в ширину и в глубину.
5. Выполнено тестирование реализации разработанного алгоритма.

Список литературы

1. Буркатовская Ю.Б., Теория графов. Часть 1: учебное пособие Томский политехнический университет. – Томск: Изд-во Томского политехнического университета, 2014. – 200 с.