МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра Информатики и систем управления

Лабораторная работа №3 «Рекурсия. Графы. Деревья.»

ОТЧЕТ по лабораторной работе № 3

по дисциплине

Технологии программирования

РУКОВОДИТЕЛЬ:

_____ Капранов С.Н.
(подпись) (фамилия, и.,о.)

СТУДЕНТ:

_____ Куликова Е.А.
(подпись) (фамилия, и.,о.)

18-ИСТ-4
(шифр группы)

Работа защищена «___» _____

С оценкой _____

Нижний Новгород

2020

Содержание

Введение

Рекурсия – определение, описание, изображение какого–либо объекта или процесса внутри самого этого объекта или процесса, то есть ситуация, когда объект является частью самого себя. Термин «рекурсия» используется в различных специальных областях знаний – от лингвистики до логики, но наиболее широкое применение находит в математике и информатике.

Граф – множество V вершин и набор E неупорядоченных и упорядоченных пар вершин; обозначается Г. через G(V, E). Неупорядоченная пара вершин называется ребром, упорядоченная пара – дугой Г., содержащий только рёбра, называется неориентированным; Г., содержащий только дуги – ориентированным. Пара вершин может соединяться двумя или более рёбрами (дугами одного направления), такие рёбра (дуги) называются кратными. Дуга (или ребро) может начинаться и кончаться в одной и той же вершине, такая дуга (ребро) называются петлей. Вершины, соединённые ребром или дугой, называются смежными. Рёбра, имеющие общую вершину, также называются смежными. Ребро (дуга) и любая из его двух вершин называется инцидентными. Говорят, что ребро (u, v) соединяет вершины к и v, а дуга (u, v) начинается в вершине и кончается в вершине v.

Дерево – это связный ациклический граф. Связность означает наличие путей между любой парой вершин, ацикличность – отсутствие циклов и то, что между парами вершин имеется только по одному пути.

В рамках третьей лабораторной работы (вариант 10) необходимо создать программу, которая должна менять местами каждую вершину с чётным значением на сына с чётным значением в бинарном дереве.

1. Цель работы

Создать программу, соответствующую правилам третьей лабораторной работы, где на вход приходит количество значений в дереве и сами значения, а на выходе дерево, в котором выполнена поставленная задача – каждая вершина с чётным номером поменяется местами с сыном, имеющим чётный номер.

2. Задачи

Поставленные задачи:

1. Разработать алгоритм, по которому будет выполняться программа.
2. Написать код, реализующий задание.
3. Протестировать, чтобы убедиться в правильности решения.

3. Описание алгоритма

Вход: Количество значений в дереве и сами значения;

Выход: Дерево в префиксной форме, где каждая вершина с чётным значением поменяна местами с сыном, имеющим чётное значение;

Начало

Рекурсивно в дерево добавляются вершины:

Если текущий узел пуст, то в него сохраняется полученное значение;

Иначе:

Если полученное значение меньше текущего, то выполняем рекурсивное добавление в левого сына;

Иначе выполняем рекурсивное добавление в правого сына;

Возвращаем дерево с добавленной вершиной;

Вывод бинарного дерева в префиксной форме;

Выполнение задачи:

Если вершина не является листом, то:

Если в вершине чётное значение, то:

Если у левого сына чётное значение, то обмениваемся с ним значениями;

Иначе:

Если у правого сына чётное значение, то обмениваемся с ним значениями;

Выполняем данные действия рекурсивно для каждого потомка;

Вывод дерева, где каждая вершина с чётным значением поменяна местами с сыном, имеющим чётное значение;

Конец.

4. Код программы

Main.cpp

```
#include <iostream>

struct Node
{
	int field; // Поле данных
	Node * left; // Левый потомок
	Node * right; // Правый потомок
};

Node * AddNode(int x, Node * tree)
{ // Добавление узлов в дерево
	if (tree == nullptr)
	{ // Если дерева нет, то формируем корень
		tree = new Node(); // Память под узел
		tree->field = x; // Поле данных
		// Ветви инициализируем пустотой
		tree->left = nullptr;
		tree->right = nullptr;
	}
	else if (x < tree->field) // Условие добавление левого потомка
		tree->left = AddNode(x, tree->left);
	else // Условие добавление правого потомка
		tree->right = AddNode(x, tree->right);
	return tree;
}

void TreePrint(Node * tree)
{// Представление дерева в префиксной форме
	if (tree != nullptr) { // Пока не встретится пустой узел
		std::cout << tree->field << " "; // Отображаем корень дерева
		TreePrint(tree->left); // Рекурсивная функция для левого поддерева
		TreePrint(tree->right); // Рекурсивная функция для правого поддерева
	}
}

Node * Task(Node * tree)
{ // Каждую вершину с четным значением поменять местами с сыном, имеющим четное значение
```

```
        if (tree != nullptr)
        {
                if (tree->field % 2 == 0)
                { // Если в узле четное значение
                        int swap = tree->field;
                        if (tree->left->field % 2 == 0)
                        { // Если левый сын четный, то обмениваемся значениями
                                tree->field = tree->left->field;
                                tree->left->field = swap;
                        }
                        else if (tree->right->field % 2 == 0)
                        {// Если правый сын четный, то обмениваемся значениями
                                tree->field = tree->right->field;
                                tree->right->field = swap;
                        }
                }
                // Выполняем рекурсивно для каждого потомка
                Task(tree->left);
                Task(tree->right);
        }
        return tree;
}

int main()
{
        setlocale(LC_ALL, "Russian");
        int n; // Количество значений
        std::cout << "Количество значений - ";
        std::cin >> n;
        std::cout << "Введите значения:" << std::endl;
        int x;
        Node * tree = nullptr;
        for (int i = 0; i < n; i++)
        {
                std::cin >> x;
                tree = AddNode(x, tree);
        }
        std::cout << "Бинарное дерево в префиксной форме:" << std::endl;
        TreePrint(tree);
        tree = Task(tree); // Выполнение задачи
        std::cout << std::endl << "Каждую вершину с четным значением поменять местами с
        сыном, имеющим четное значение:" << std::endl;
        TreePrint(tree);
        return 0;
}
```

Листинг 1 – Код программы

5. Реализация программы

Приходящее на вход дерево, используемое как пример для проверки кода, выглядит следующим образом.

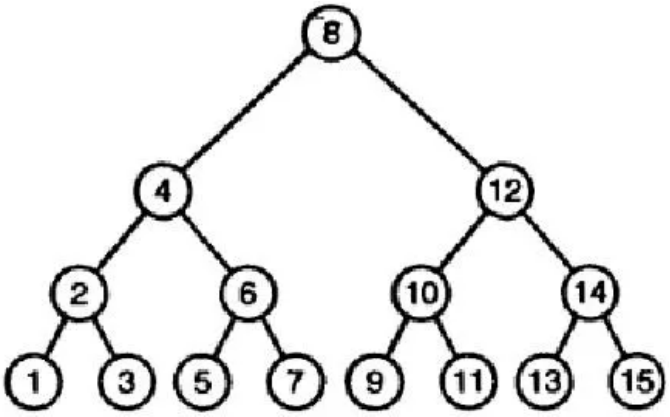

Рисунок 1 – Пример бинарного дерева

Консольный вид реализации выглядит следующим образом.

```
Количество значений - 15
Введите значения:
8 4 12 2 6 10 14 1 3 5 7 9 11 13 15
Бинарное дерево в префиксной форме:
8 4 2 1 3 6 5 7 12 10 9 11 14 13 15
Каждую вершину с четным значением поменять местами с сыном,
имеющим четное значение:
4 2 8 1 3 6 5 7 10 12 9 11 14 13 15
```

Рисунок 2 – Результат работы программы

Заключение

В ходе третьей лабораторной работы была создана программа, соответствующая правилам лабораторной работы, на вход приходит количество значений в дереве и сами значения, а на выходе дерево, в котором выполнена поставленная задача – каждая вершина с чётным номером поменяется местами с сыном, имеющим чётный номер. Также программа была протестирована необходимое количество раз для проверки на корректность.

Используемая литература

1. Дерево – https://ru.wikipedia.org/wiki/Дерево_(теория_графов)
2. Граф – https://ru.wikipedia.org/wiki/Граф_(математика)
3. Рекурсия – https://ru.wikipedia.org/wiki/Рекурсия
4. Двоичное дерево поиска – https://ru.wikipedia.org/wiki/Двоичное_дерево_поиска