

МИНОБРАЗОВАНИЯ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» им.В.И.УЛЬЯНОВА (ЛЕНИНА)

Кафедра вычислительной техники

Отчет по лабораторной работе № 3
по дисциплине «Алгоритмы и структуры данных, часть 1»
Тема: «Деревья»

Студенты гр. 9306

Евдокимов О.В. Кныш С.А. Павельев М.С.

Преподаватель

Манерагена Валентина

Санкт-Петербург
2020

Содержание

Цель.....	3
Задание.....	3
Обоснование выбора способа представления деревьев.....	3
Тестовый пример.....	4
Результаты прогона программы с генерацией случайного дерева.....	4
Оценки временной сложности.....	5
Создание дерева:.....	5
Обработка дерева:.....	5
Вывод дерева:.....	5
Выводы о результатах испытания алгоритмов обхода деревьев.....	5
Код программы.....	5
Main.cpp.....	5
Tree.h.....	6
Tree.cpp.....	6

Цель

Исследование алгоритмов для работы с троичным деревом.

Задание

Вид дерева — троичное.

Разметка — прямая.

Способ обхода — в ширину.

Надо вычислить количество вершин, имеющих предков.

Обоснование выбора способа представления деревьев.

В качестве способа представления были выбраны списки, т. к. заранее не известен размер деревьев, не нужно переаллоцировать ресурсы при добавлении одной вершины в дерево, также данный способ упрощает работу с деревом, т. к. Каждая вершина является объектом имеющим ссылки на своих детей, данный способ хранения легко изобразить на диаграммах, при использовании любого другого способа представления система станет лишь сложнее.

Тестовый пример

```
(Lable, depth, parent, side)
Node (a, 0, 0, M)1/0: 1
Node (b, 1, a, L)1/0: 1
Node (c, 2, b, L)1/0: 0
Node (c, 2, b, M)1/0: 1
Node (d, 3, c, L)1/0: 0
Node (d, 3, c, M)1/0: 1
Node (e, 4, d, L)1/0: 0
Node (e, 4, d, M)1/0: 0
Node (e, 4, d, R)1/0: 0
Node (e, 3, c, R)1/0: 0
Node (e, 2, b, R)1/0: 1
Node (f, 3, e, L)1/0: 0
Node (f, 3, e, M)1/0: 0
Node (f, 3, e, R)1/0: 0
Node (f, 1, a, M)1/0: 1
Node (g, 2, f, L)1/0: 0
Node (g, 2, f, M)1/0: 0
Node (g, 2, f, R)1/0: 0
Node (g, 1, a, R)1/0: 1
Node (h, 2, g, L)1/0: 0
Node (h, 2, g, M)1/0: 0
Node (h, 2, g, R)1/0: 1
Node (i, 3, h, L)1/0: 1
Node (j, 4, i, L)1/0: 0
Node (j, 4, i, M)1/0: 0
Node (j, 4, i, R)1/0: 0
Node (j, 3, h, M)1/0: 1
Node (k, 4, j, L)1/0: 0
Node (k, 4, j, M)1/0: 0
Node (k, 4, j, R)1/0: 0
Node (k, 3, h, R)1/0: 0
```

```
.....a.....
.....b.....f.....g.....
.....c.....e.....h.....
.....d.....i.....j.....
.....
.....
.....
```

Результаты прогона программы с генерацией случайного дерева

```
(Lable, depth, parent, side)
```

```
.....a.....
.....b.....
.....c.....
.....d.....e.....
.....
.....
.....
a_b_c_d_e_
Result is : 4
```

```
(Lable, depth, parent, side)
```

```
.....a.....  
.....b.....f.....n.....  
.....c.....g.....l.....  
.....d.....h.....j.....m.....  
.....e.....i.....k.....  
.....  
a_b_f_n_c_g_l_d_h_j_m_e_i_k_  
Result is : 13
```

Оценки временной сложности

Создание дерева:

Будем считать наиболее трудоёмкой операцией операцию выделения памяти (оператор `new`). Для каждой вершины дерева она вызывается один раз, следовательно временная сложность $O(n)$, где n - кол-во вершин в дереве.

Обработка дерева:

Будем считать трудоёмкой операцией добавления вершины в очередь. Для каждой вершины дерева она вызывается один раз, следовательно временная сложность $O(n)$, где n - кол-во вершин в дереве.

Вывод дерева:

Самым трудоёмким процессом является выставление меток на диаграмму. В нем операция установки значения элементу двумерного массива, которая вызывается для каждой вершины, следовательно временная сложность $O(n)$, где n - кол-во вершин в дереве.

Выводы о результатах испытания алгоритмов обхода деревьев.

Нам не пришлось долго искать способ обхода дерева который поможет для решения нашей задачи. Наш выбор пал на алгоритм обхода в ширину, т. к. он даже в своей идее уже практически решает нашу задачу, обойти всех детей, а потом уже детей детей, а нам нужно было лишь добавить счётчик.

Код программы

Main.cpp

```
#include <iostream>  
#include "Tree.h"  
#include <time.h>  
  
using namespace std;  
int main()  
{  
    srand(time(0));  
  
    printf("(Lable, depth, parent, side)\n");  
    Tree t('a', 'd', 7);  
    t.MakeTree();  
}
```

```

        if (t.exist())
        {
            t.OutTree();
            int res = t.BFS();
            std::cout << "\nResult is : " << res;
        }
        else
            std::cout << "Tree is empty";

        return 0;
    }

```

Tree.h

```
#pragma once
```

```

class Node {
    char d;
    Node* lft;
    Node* mdl;
    Node* rgt;
public:
    Node() : d(0) ,lft(nullptr), mdl(nullptr), rgt(nullptr) { }
    ~Node() {
        if (lft) delete lft;
        if (mdl) delete mdl;
        if (rgt) delete rgt;
    }
    friend class Tree;
};

```

```

class Tree
{
    Node* root;
    char num, maxnum;
    int maxrow, offset;
    char** SCREEN;
    void clrscr();
    Node* MakeNode(char,char ,int depth);
    void OutNodes(Node* v, int r, int c);
    Tree(const Tree&) = delete;
    Tree(Tree&&) = delete;
    Tree operator = (const Tree&) const = delete;
    Tree operator = (Tree&&) const = delete;

public:
    Tree(char num, char maxnum, int maxrow);
    ~Tree();
    void MakeTree()
    {
        root = MakeNode('0','M',0);
    }
    bool exist() { return root != nullptr; }
    int BFS();
    void OutTree();
};

```

Tree.cpp

```

#include "Tree.h"
#include <iostream>

using namespace std;

```

```

Tree::Tree(char nm, char mnm, int mxr) :
    num(nm), maxnum(mnm), maxrow(mxr), offset(40), root(nullptr),
    SCREEN(new char* [maxrow])
{
    for (int i = 0; i < maxrow; i++) SCREEN[i] = new char[80];
}

Tree::~~Tree() {
    for (int i = 0; i < maxrow; i++) delete[] SCREEN[i];
    delete[] SCREEN; delete root;
}

Node* Tree::MakeNode(char parent ,char side, int depth)
{
    Node* v = nullptr;
    int Y = (depth < rand() % 5 + 1) && (num <= maxnum);
    //cout << "Node (" << num << ", " << depth << ", " << parent << ",
    " << side << ")1/0: "; cin >> Y; // fill via keyboard
    if (Y) {
        v = new Node;
        v->d = num++;
        v->lft = MakeNode(v->d, 'L', depth + 1);
        v->mdl = MakeNode(v->d, 'M', depth + 1);
        v->rgt = MakeNode(v->d, 'R', depth + 1);
    }
    return v;
}

void Tree::OutTree()
{
    clrscr();
    OutNodes(root, 1, offset);
    for (int i = 0; i < maxrow; i++)
    {
        SCREEN[i][79] = '\\0';
        cout << '\\n' << SCREEN[i];
    }
    cout << '\\n';
}

void Tree::clrscr()
{
    for (int i = 0; i < maxrow; i++)
        memset(SCREEN[i], '.', 80);
}

void Tree::OutNodes(Node* v, int r, int c)
{
    if (r && c && (c < 80)) SCREEN[r - 1][c - 1] = v->d; // print
metku    if (r < maxrow) {
        if (v->lft) OutNodes(v->lft, r + 1, c - (offset >> r));
//left child
        if (v->mdl) OutNodes(v->mdl, r + 1, c); // - mid child
        if (v->rgt) OutNodes(v->rgt, r + 1, c + (offset >> r));
//right child
    }
    std::cout << "F";
}

template <class Item> class QUEUE
{
    Item* Q;
    int h, t, N;
public:
    QUEUE(int maxQ)

```

```

: h(0), t(0), N(maxQ), Q(new Item[maxQ + 1]) { }
int empty() const {
    return (h % N) == t;
}
void push(Item item)
{
    Q[t++] = item;
    t %= N;
}
Item pop()
{
    h %= N;
    return Q[h++];
}
~QUEUE()
{
    delete[] Q;
}
};

int Tree::BFS()
{
    const int MaxQ = 20; //максимальный размер очереди
    int count = -1;
    QUEUE < Node* > Q(MaxQ); //создание очереди указателей на узлы
    Q.push(root); // QUEUE <- root поместить в очередь корень дерева
    while (!Q.empty()) //пока очередь не пуста
    {
        Node* v = Q.pop(); // взять из очереди,
        cout << v->d << '_'; count++; // выдать тег, счёт узлов
        if (v->lft) Q.push(v->lft); // QUEUE <- (левый сын)
        if (v->mdl) Q.push(v->mdl);
        if (v->rgt) Q.push(v->rgt); // QUEUE <- (правый сын)
    }
    return count;
}

```