**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №3**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: Деревья

|  |  |  |
| --- | --- | --- |
| Студент гр. 9382 |  | Докукин В.М. |
| Преподаватель |  | Фирсов М.А. |

Санкт-Петербург

2020

**Цель работы.**

Изучить такую структуру данных, как дерево, её разновидности, а также область применения и методы работы с ней.

**Основные теоретические положения.**

**Дерево** — одна из наиболее широко распространённых структур данных, эмулирующая древовидную структуру в виде набора связанных узлов. Является связным графом, не содержащим циклы. Дерево может быть реализовано как при помощи связного списка(когда каждый узел содержит указатель на левого и правого сыновей), так и на базе вектора(т.е. массива; в этом случае узел хранит внутри себя индексы левого и правого сыновей).

**Задание.**

19в. Бинарное дерево называется идеально сбалансированным, если для каждой его вершины количества вершин в левом и правом поддереве различаются не более чем на 1.Бинарное дерево называется АВЛ-деревом, если для каждой его вершины высо́ты её двух поддеревьев различаются не более чем на 1.Для заданного бинарного дерева определить, является ли оно идеально сбалансированным и является ли оно АВЛ-деревом.

**Ход работы.**

1. Были написаны 2 класса, составляющие основу структуры дерева: класс Node – узел дерева, содержащий индексы сыновей и родителя, а также ключ key, и класс BinTree, хранящий вектор(массив) nodes элементов Node, высоту дерева height и реализующий некоторые операции над деревом.

2. Для работы с деревом определены следующие функции:

1) int count(BinTree\* b, int i = 0) – для заданного дерева b подсчитывает количество сыновей у узла с индексом i. Алгоритм подсчёта реализован рекурсивно.

2) int height(BinTree\* b, int i = 0) – для заданного дерева b подсчитывает высоту узла с индексом i. Алгоритм подсчёта реализован рекурсивно.

3) bool isBalanced(BinTree\* b, int i = 0) – определяет, является ли заданное бинарное дерево идеально сбалансированным. Алгоритм работы функции таков: для каждого узла, начиная с узла с индексом i, проверяется, удовлетворяют ли условию сбалансированности его сыновья и он сам. Таким образом, дерево обходится по правилу ЛПК.

4) bool isAVL(BinTree\* b, int i = 0) – определяет, является ли заданное дерево АВЛ-деревом. Алгоритм работы схож с алгоритмом работы функции isBalanced(), отличие лишь в проверяемом условии(проверяется разность высот, а не количества сыновей).

3. Написана функция func(), в которую вынесен код пользовательского интерфейса из main(). В ней осуществляется выбор режима ввода и ручной ввод дерева.

4. Определена функция process(), в которой происходит создание дерева и работа с ним, а также вывод результата.

5. В функции main() расположен блок try-catch, внутри которого вызывается функция func().

**Пример работы программы.**

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| 1234567 | The tree you've inserted:  1 2 3 4 5 6 7  4 node height: 0  5 node height: 0  2 node height: 1  6 node height: 0  7 node height: 0  3 node height: 1  1 node height: 2  Tree height: 3  4 node children count: 0  5 node children count: 0  2 node children count: 2  6 node children count: 0  7 node children count: 0  3 node children count: 2  4 node children count: 0  5 node children count: 0  6 node children count: 0  7 node children count: 0  This tree is a balanced tree.  4 node height: 0  5 node height: 0  2 node height: 1  6 node height: 0  7 node height: 0  3 node height: 1  4 node height: 0  5 node height: 0  6 node height: 0  7 node height: 0  This tree is an AVL-tree. |

**Тестирование.**

Результаты тестирования представлены в таблице ниже.

|  |  |  |  |
| --- | --- | --- | --- |
| № теста | Входные данные | Выходные данные | Комментарий |
| 1 | #12 | The tree you've inserted:  # 1 2  Incorrect tree format. | Проверка на обработку «неправильных» деревьев. |
| 2 | jfpdglvc####nsx############qu## | The tree you've inserted:  j f p d g l v c # # # # n s x # # # # # # # # # # # # q u # #  *//промежуточная информация об узлах*  Tree height: 5  *//промежуточная информация об узлах*  This tree isn’t a balanced tree.  *//промежуточная информация об узлах*  This tree is an AVL-tree. | Пример АВЛ-дерева, не являющегося сбалансированным. |
| 3 | a#b##cd | The tree you've inserted:  a # b # # c d  c node height: 0  d node height: 0  b node height: 1  a node height: 2  Tree height: 3  c node children count: 0  d node children count: 0  b node children count: 2  c node children count: 0  d node children count: 0  This tree isn't a balanced tree.  c node height: 0  d node height: 0  b node height: 1  c node height: 0  d node height: 0  This tree isn't an AVL-tree. | Пример дерева, не являющегося ни АВЛ-деревом, ни сбалансированным. |

**Выводы.**

В результате выполнения лабораторной работы:

1. Была изучена такая структура данных, как дерево; были изучены методы работы с ней, область применения и разновидности.

2. Написана программа, решающая поставленную задачу.

3. Написана серия тестов, позволяющих качественно оценить работу программы (тесты находятся в файле tests.txt).

Код программы размещён в Приложении 1.

ПРИЛОЖЕНИЕ 1

ИСХОДНЫЙ КОД ПРОГРАММЫ

Имя файла: main.cpp

#include <iostream>

#include <string>

#include <cmath>

#include <fstream>

class Node{

public:

int iParent;

int iLeft;

int iRight;

bool isVisited;

char key;

Node(){

this->iLeft = -1;

this->iRight = -1;

this->key = '#';

}

};

class BinTree{

public:

Node\* nodes;

int height;

bool isNull(){

return ( nodes->key == '#' );

}

void print(){

for(int i = 0; i < std::pow(2, height) - 1; i++){

std::cout<<nodes[i].key<<" ";

}

std::cout<<'\n';

}

void expand(){

Node\* tmp = new Node[(int)(std::pow(2, this->height + 1) - 1)];

for (int i = 0; i < std::pow(2, this->height) - 1; i++){

tmp[i].key = this->nodes[i].key;

(tmp+i)->iLeft = 2\*i + 1;

(tmp+i)->iRight = 2\*i + 2;

}

delete[] nodes;

this->nodes = tmp;

this->height++;

}

BinTree(std::string keys){

int height = 0;

while (std::pow(2, height) - 1 < keys.length()){

height++;

}

Node\* tmp = new Node[(int)(std::pow(2, height) - 1)];

for (int i = 0; i < std::pow(2, height) - 1; i++){

if (i < keys.length()){

if (isspace(keys[i])) tmp[i].key = '#';

else tmp[i].key = keys[i];

}

(tmp+i)->iLeft = 2\*i + 1;

(tmp+i)->iRight = 2\*i + 2;

}

this->nodes = tmp;

this->height = height;

}

bool isCorrectTree(){

for (int i = 0; i < std::pow(2, this->height - 1) - 1; i++){

if (this->nodes[i].key == '#' && (this->nodes[2\*i + 1].key != '#' || this->nodes[2\*i + 2].key != '#')) return false;

}

return true;

}

};

int count(BinTree\* b, int i = 0){

int c = 0;

int iLeft = b->nodes[i].iLeft;

int iRight = b->nodes[i].iRight;

if ((iLeft >= pow(2, b->height) - 1) || (iRight >= pow(2, b->height) - 1)) b->expand();

if (b->nodes[i].key == '#') return 0;

else{

if (b->nodes[iLeft].key != '#') c = c + count(b, b->nodes[i].iLeft);

if (b->nodes[iRight].key != '#') c = c + count(b, b->nodes[i].iRight);

std::cout<<b->nodes[i].key<<" node children count: "<<c<<'\n';

return c + 1;

}

}

int height(BinTree\* b, int i = 0){

int h1 = 0, h2 = 0;

int iLeft = b->nodes[i].iLeft;

int iRight = b->nodes[i].iRight;

if ((iLeft >= pow(2, b->height) - 1) || (iRight >= pow(2, b->height) - 1)) b->expand();

if (b->nodes[i].key == '#') return 0;

else{

if (b->nodes[iLeft].key != '#') h1 = height(b, b->nodes[i].iLeft);

if (b->nodes[iRight].key != '#') h2 = height(b, b->nodes[i].iRight);

std::cout<<b->nodes[i].key<<" node height: "<<std::max(h1, h2)<<'\n';

return std::max(h1, h2) + 1;

}

}

bool isBalanced(BinTree\* b, int i = 0){

int iLeft = b->nodes[i].iLeft;

int iRight = b->nodes[i].iRight;

int cleft = 0, cright = 0;

if (b->nodes[iLeft].key == '#' && b->nodes[iRight].key == '#') return true;

if (b->nodes[iLeft].key != '#') cleft = count(b, iLeft);

if (b->nodes[iRight].key != '#') cright = count(b, iRight);

return isBalanced(b, iLeft) && isBalanced(b, iRight) && (std::abs(cleft - cright) < 2);

}

bool isAVL(BinTree\* b, int i = 0){

int iLeft = b->nodes[i].iLeft;

int iRight = b->nodes[i].iRight;

int hleft = 0, hright = 0;

if (b->nodes[iLeft].key == '#' && b->nodes[iRight].key == '#') return true;

if (b->nodes[iLeft].key != '#') hleft = height(b, iLeft);

if (b->nodes[iRight].key != '#') hright = height(b, iRight);

return isAVL(b, iLeft) && isAVL(b, iRight) && (std::abs(hleft - hright) < 2);

}

int process(std::string str){

BinTree b(str);

std::cout<<"The tree you've inserted:\n";

b.print();

if (!b.isCorrectTree()){

std::cout<<"Incorrect tree format.\n";

std::cout<<"--------------------------------\n";

return 1;

}

std::cout<<"Tree height: "<<height(&b)<<'\n';

bool tmp = isBalanced(&b);

if (tmp == 1) std::cout<<"This tree is a balanced tree.\n";

else std::cout<<"This tree isn't a balanced tree.\n";

tmp = isAVL(&b);

if (tmp == 1) std::cout<<"This tree is an AVL-tree.\n";

else std::cout<<"This tree isn't an AVL-tree.\n";

std::cout<<"--------------------------------\n";

return 0;

}

int func(){

int a;

std::string str;

std::cout<<"Choose input option (0 - file input, 1 - console input):\n";

std::cin>>a;

if(a){

std::cin>>str;

if (str.empty()) str = "#";

process(str);

return 0;

}

std::ifstream f("tests.txt");

if(!f){

std::cout<<"Couldn't open file!\n";

return 1;

}

int testn = 1;

while(!f.eof()){

std::cout<<"Test #"<<testn<<'\n';

f>>str;

if (str.empty()) str = "#";

else process(str);

testn++;

}

return 0;

}

int main(){

try{

func();

}

catch(...){}

return 0;

}