**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: Бинарные деревья

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

Санкт-Петербург

2020

**Цель работы**:

Познакомиться с одной из часто используемых на практике нелинейных

конструкций, способами её организации и рекурсивной обработки, получить

навыки решения задач обработки бинарных деревьев, как с использованием базовых функций их рекурсивной обработки, так и без использования рекурсии.

**Основные теоритические положения**:

Дерево – конечное множество Т, состоящее из одного или более узлов,

таких, что

а) имеется один специально обозначенный узел, называемый корнем

данного дерева;

б) остальные узлы (исключая корень) содержатся в m  0 попарно не

пересекающихся множествах Т1 , Т2 , ..., Тm , каждое из которых, в свою

очередь, является деревом. Деревья Т1 , Т2 , ..., Тm называются поддеревьями данного дерева.

**Задание:**

Бинарное дерево называется бинарным деревом поиска, если для

каждого его узла справедливо: все элементы правого поддерева больше этого

узла, а все элементы левого поддерева – меньше этого узла.

Бинарное дерево называется пирамидой, если для каждого его узла

справедливо: значения всех потомков этого узла не больше, чем значение

узла.

Для заданного бинарного дерева с числовым типом элементов

определить, является ли оно бинарным деревом поиска и является ли оно

пирамидой.

**Выполнение работы:**

Бинарное дерево:

template <class Elem>

class Tree {

public:

struct node {

Elem info;

Tree \*lt;

Tree \*rt;

…

};

private:

node \*Node = nullptr;

…

}

info – данные узла

lt, rt – указатели на левое и правое поддерево

Реализованные методы:

*Очистка*

Сигнатура: void Clear()

*Получение указателя на левое/правое поддерево*

Сигнатура: Tree \*Left() / Tree \*Right()

*Получение указателя на узел*

Сигнатура: node \*NodePtr()

*Получение информации узла*

Сигнатура: Elem \*GetNode()

*Проверка на бинарное дерево поиска*

Сигнатура: bool CheckSearchTree(float min, float max, int lvl)

Аргументы:

* min, max –текущая левая и правая граница
* lvl – уровень рекурсии

Возвращаемое значение:

bool – является ли дерево BST (Да/Нет)

Алгоритм: (обход КЛП)

* Сравнить элемент текущего корня с min и max, если неравенство неверно – дерево не соответствует дереву поиска
* Запустить проверку левого поддерева (max становится равен элементу текущего корня)
* Запустить проверку правого поддерева (min становится равен элементу текущего корня)
* Вернуть конъюнкцию этих проверок

В общем случае алгоритм запускается с min = -inf, max = inf

*Проверка на бинарное дерево-пирамиду*

Сигнатура: bool CheckPyramidTree()

Аргументы: **-**

Возвращаемое значение:

bool - является ли дерево пирамидой (Да/Нет)

Алгоритм: (итеративный обход в ширину с использованием очереди)

* Положить корень дерева в очередь
* Пока очередь не пуста:
  + Получить первый элемент из очереди
  + Сравнить с дочерними элементами, если элемент оказался меньше дочерних, дерево не является пирамидой
  + Положить дочерние элементы в конец очереди (если они есть)
* Если все неравенства в цикле оказались верны, значит дерево является пирамидой

**Тестирование:**

/ - конец поддерева

Любой другой символ (,) считается разделителем чисел

|  |  |  |
| --- | --- | --- |
| **№** | Входные данные | Результат |
| 1 | 6,5,3,/4,///8,7,//12,11,//15,13,/// | Search: true  Pyramid: false |
| 2 | 10,9,8,/7,///6,5,//4,3,//2,1,/// | Search: false  Pyramid: true |
| 3 | 1,// | Search: true  Pyramid: true |
| 4 | 0,0,//0,// | Search: true  Pyramid: true |
| 5 | 1,2,3,4,5,////////// | Search: false  Pyramid: false |
| 6 | 1,/2,/3,/4,/5,///// | Search: true  Pyramid: false |
| 7 | / | Wrong input |

**Вывод:**

В результате выполнения работы были изучены принципы обработки списков а также структура бинарных деревьев. Были реализованы функции определения дерева и поиска и дерева-пирмиды.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

**tree.h**

#ifndef \_\_TREE\_H

#define \_\_TREE\_H

#include <cstdlib>

#include <iostream>

#include <fstream>

#include <queue>

template <class Elem>

class Tree {

public:

struct node {

Elem info;

Tree \*lt;

Tree \*rt;

node() {

lt = nullptr;

rt = nullptr;

}

node(const Elem &x, Tree \*lst, Tree \*rst) {

info = x;

lt = lst;

rt = rst;

}

};

private:

node \*Node = nullptr;

public:

Tree() {}

Tree(node \*N) {

Node = N;

}

void Clear() {

if (Node != nullptr) {

if (Node->lt != nullptr) Node->lt->Clear();

if (Node->rt != nullptr) Node->rt->Clear();

delete Node;

Node = nullptr;

}

}

~Tree() { Clear(); }

Tree \*Left() {

if (Node == nullptr) {

std::cout << "Error: Left(null) \n";

exit(1);

} else

return Node->lt;

}

Tree \*Right() {

if (Node == nullptr) {

std::cout << "Error: Right(null) \n";

exit(1);

} else

return Node->rt;

}

node \*NodePtr() {

if (Node == nullptr) {

std::cout << "Error: RootBT(null) \n";

exit(1);

} else

return Node;

}

Elem GetNode() {

if (Node == nullptr) {

std::cout << "Error: RootBT(null) \n";

exit(1);

} else

return Node->info;

}

// Checking if the tree is a binary search tree

bool CheckSearchTree(float min, float max, int lvl) {

bool L = true, R = true;

Elem cur = GetNode();

std::cout << "\n";

for (int i = 0; i < lvl; i++) std::cout << " ";

std::cout << "ELEMENT: " << cur << "\n";

for (int i = 0; i < lvl; i++) std::cout << " ";

std::cout << min << "(min)" << " <= " << cur << " <= " << max << "(max)" << " ? ";

// Check current node

if (cur > max || cur < min) {

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

return false;

}

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

// Check left subtree

if (Left() != nullptr) {

for (int i = 0; i < lvl; i++) std::cout << " ";

std::cout << "Check left: (max -> " << cur << ")\n";

// max -> cur

L = Left()->CheckSearchTree(min, cur, lvl + 1);

}

if (!L)

return false;

// Check right subtree

if (Right() != nullptr) {

for (int i = 0; i < lvl; i++) std::cout << " ";

std::cout << "Check right: (min -> " << cur << ")\n";

// min -> cur

R = Right()->CheckSearchTree(cur, max, lvl + 1);

}

if (!R)

return false;

return true;

}

// Checking if the tree is a pyramid tree

bool CheckPyramidTree() {

std::queue<Tree<Elem> \*> q;

// Push root

q.push(this);

while (!q.empty()) {

//Print queue

std ::cout << " Queue: ";

std::queue<Tree<Elem> \*> t = q;

while (!t.empty()) {

std::cout << t.front()->GetNode() << " ";

t.pop();

}

std ::cout << "\n";

// Check current node

Tree<Elem> \*cur = q.front();

q.pop();

std::cout << " Element:" << cur->GetNode() << "\n";

bool L = false, R = false;

// Check left

if (cur->Left() != nullptr) {

std::cout << " Left: " << cur->GetNode()

<< " >= " << cur->Left()->GetNode()

<< " ? ";

L = cur->GetNode() >= cur->Left()->GetNode();

std::cout << (L ? "true" : "false") << "\n";

if (!L)

return false;

q.push(cur->Left());

}

// Check right

if (cur->Right() != nullptr) {

std::cout << " Right: " << cur->GetNode()

<< " >= " << cur->Right()->GetNode() << " ? ";

R = cur->GetNode() >= cur->Right()->GetNode();

std::cout << (R ? "true" : "false") << "\n";

if (!R) return false;

q.push(cur->Right());

}

std::cout << "\n";

}

return true;

}

};

#endif // \_\_TREE\_H

**main.cpp**

/\* Кодуков Александр 9382, в. 18д

\*

\* Бинарное дерево называется бинарным деревом поиска,

\* если для каждого его узла справедливо

\* : все элементы правого поддерева больше этого узла,

\* а все элементы левого поддерева – меньше этого узла.Бинарное дерево

\* называется пирамидой,

\* если для каждого его узла справедливо

\* : значения всех потомков этого узла не больше,

\* чем значение узла.Для заданного бинарного дерева с числовым типом

\* элементов определить,

\* является ли оно бинарным деревом поиска и является ли оно пирамидой.

\*/

#include "tree.h"

template <typename Elem>

Tree<Elem> \*Read(std::ifstream &f) {

char ch;

Elem e = 0;

Tree<Elem> \*p, \*q;

f >> ch;

int d = 0;

while (ch >= '0' && ch <= '9') {

e = e \* pow(10, d++) + ch - '0';

f >> ch;

}

if (ch == '/')

return NULL;

else {

p = Read<Elem>(f);

q = Read<Elem>(f);

typename Tree<Elem>::node \*N = new typename Tree<Elem>::node(e, p, q);

return new Tree<Elem>(N);

}

}

template <typename Elem>

void Print(Tree<Elem> \*q, long n) {

long i;

if (q != nullptr) {

Print<Elem>(q->Right(), n + 5);

for (i = 0; i < n; i++)

std::cout << " ";

std::cout << q->GetNode() << "\n";

Print<Elem>(q->Left(), n + 5);

}

}

int main() {

Tree<int> \*t;

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

t = Read<int>(f);

if (t != nullptr) {

f.close();

Print(t, 0);

std::cout << "Check search:\n";

bool Search = t->CheckSearchTree(-INFINITY, INFINITY, 1);

std::cout << "Search: " << (Search ? "true" : "false") << "\n\n";

std::cout << "Check pyramid:\n";

bool Pyramid = t->CheckPyramidTree();

std::cout << "Pyramid: " << (Pyramid ? "true" : "false") << "\n";

t->Clear();

} else

std::cout << "Wrong input";

}