**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

**отчет**

**по лабораторной работе №4**

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

**Тема: «Бинарное дерево поиска»**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 0322 |  | Павлова А.Н. |
| Преподаватель |  | Пестерев Д.О. |

Санкт-Петербург

2022

1. Постановка задачи

Реализовать структуру данных двоичное дерево поиска и следующие методы: нахождение минимума, нахождение максимума, прямой (preorder), центрированный (inorder) и обратный обход (postorder) по дереву, поиск элемента, нахождение следующего и предыдущего элемента, удаление элемента, обход в ширину. Реализовать визуализацию дерева. Указать теоретическую временную сложность для всех операций.

С помощью реализованной структуры данных написать программу, позволяющую преобразовать запись из префиксной/инфиксной/постфиксной нотации в префиксную/инфиксную/постфиксную нотацию.

1. Описание реализуемого класса и методов

Реализуемые методы:

Структуры:

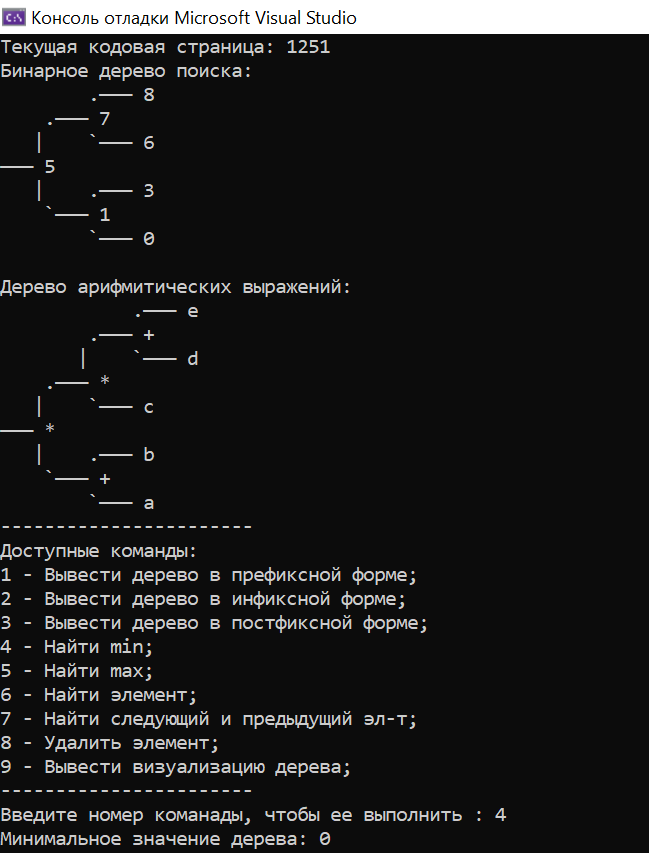
struct Node – структура данных для хранения узла бинарного дерева

struct Trее – структура данных для хранения узла дерева арифмитических выражений

* void min – функция для нахождение минимального значения
* void mах– функция для нахождение максимального значения
* Node\* find– функция для поиска
* float findPre – функция для поиска предыдущего значения
* float findSled – функция для поиска следующего значения
* Node\* DeleteNode- функция для удаления элемента
* void showTrunks, void printTree, void printTrees - функция для отрисовки дерева
* bool isOperator – функция для проверки операторов
* void preorder – префиксная форма
* void postorder – постфиксная форма
* void inorder – инфиксная форма
* Tree\* construct – функция для построения дерева выражений из заданного выражения

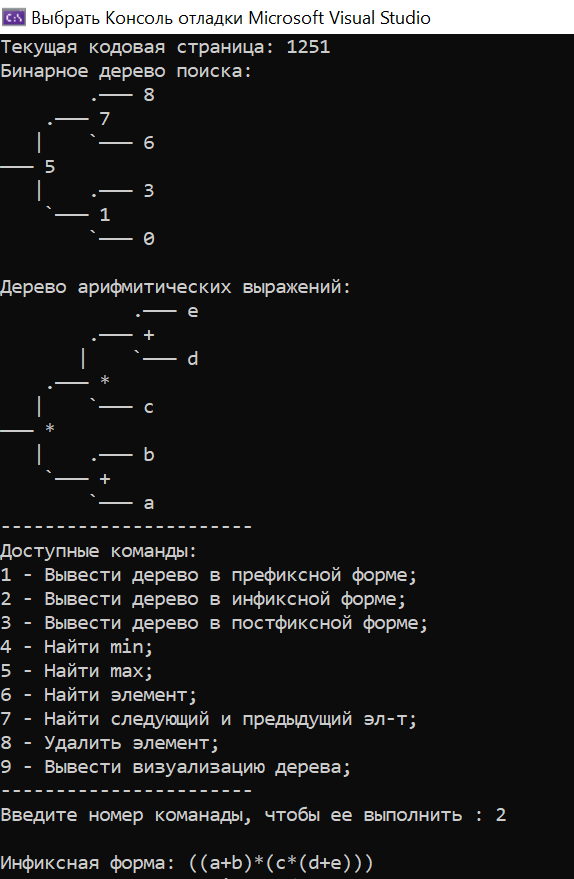
1. Пример работы программы

Первый запуск. Программа выводит оба массива. И предлагает выбор команд. Выбираем команду 4: найти минимальное значение в бинарном дерева. Смотрим на схему дерева, видим, что это 0. Ожидаем, что программа выведет 0.



Видим, что программа работает верно.

Попробуем воспользоваться командой вывести в инфиксной форме для дерева арифметических выражений.



Видим верный результат.

1. Листинг

***Laba4.cpp***

#include <iostream>

#include <string>

#include <stack>

using namespace std;

struct Node // Структура данных для хранения узла бинарного дерева

{

int data;

Node\* left, \* right;

Node(int data)

{

this->data = data;

this->left = this->right = nullptr;

}

};

struct Trunk

{

Trunk\* prev;

string str;

Trunk(Trunk\* prev, string str)

{

this->prev = prev;

this->str = str;

}

};

void min(Node\* root) {

int a = 10;

while (root->left != nullptr)

root = root->left;

a = root->data;

cout << "Минимальное значение дерева: " << a;

}

void max(Node\* root) {

int a = -1;

while (root->right != nullptr)

root = root->right;

a = root->data;

cout << "Максимальное значение дерева: " << a;

}

Node\* find(Node\* r, float d)

{

if (r == nullptr)

{

cout << "Число не найдено( ";

return nullptr; // не найден

}

if (r->data == d)

{

cout << "Найденное число: " << r->data;

return r; // нашли!!!

}

if (d <= r->data)

{

// left

if (r->left != nullptr)

return find(r->left, d); // рекурсивный поиск влево

else

{

cout << "Число не найдено( ";

return nullptr; // не найден

}

}

else

{

//right

if (r->right)

return find(r->right, d);// рекурсивный поиск вправо

else

{

cout << "Число не найдено( ";

return nullptr; // не найден

}

}

}

float findPre(Node\* r, float d, Node\* k) {

//float a;

//d = d - 1;

// Node\* k = r;

if (r == nullptr)

{

cout << "Число не найдено( ";

return 0; // не найден

}

if (r->data == d)

{

return d; // нашли!!!

}

if (d <= r->data)

{

// left

if (r->left != nullptr)

return findPre(r->left, d, k); // рекурсивный поиск влево

else

{

d--;

//cout << "Число не найдено( ";

return findPre(k, d, k); // не найден

}

}

else

{

//right

if (r->right != nullptr)

return findPre(r->right, d, k);// рекурсивный поиск вправо

else

{

d--;

//cout << "Число не найдено( ";

return findPre(k, d, k); // не найден

}

}

}

float findSled(Node\* r, float d, Node\* k) {

//float b;

// d = d + 1;

//Node\* k = r;

if (r->data == d)

{

return d;

}

if (d <= r->data)

{

// left

if (r->left != nullptr)

return findSled(r->left, d, k); // рекурсивный поиск влево

else

{

d++;

//cout << "Число не найдено( ";

return findSled(k, d, k); // не найден

}

}

else

{

//right

if (r->right != nullptr)

return findSled(r->right, d, k);// рекурсивный поиск вправо

else

{

d++;

//cout << "Число не найдено( ";

return findSled(k, d, k); // не найден

}

}

}

/\*Если удаляемый элемент находится в корне текущего поддерева и имеет два дочерних узла,

то нужно заменить его минимальным элементом из правого поддерева

и рекурсивно удалить минимальный элемент из правого поддерева.\*/

Node\* DeleteNode(Node \* node, int val) {

if (node == NULL)

return node;

if (val == node->data) {

Node\* tmp;

if (node->right == NULL)

tmp = node->left;

else {

Node\* ptr = node->right;

if (ptr->left == NULL) {

ptr->left = node->left;

tmp = ptr;

}

else {

Node\* pmin = ptr->left;

while (pmin->left != NULL) {

ptr = pmin;

pmin = ptr->left;

}

ptr->left = pmin->right;

pmin->left = node->left;

pmin->right = node->right;

tmp = pmin;

}

}

delete node;

return tmp;

}

else if (val < node->data)

node->left = DeleteNode(node->left, val);

else

node->right = DeleteNode(node->right, val);

return node;

}

// Вспомогательная функция для печати ветвей бинарного дерева

void showTrunks(Trunk\* p)

{

if (p == nullptr) {

return;

}

showTrunks(p->prev);

cout << p->str;

}

void printTree(Node\* root, Trunk\* prev, bool isLeft)

{

if (root == nullptr) {

return;

}

string prev\_str = " ";

Trunk\* trunk = new Trunk(prev, prev\_str);

printTree(root->right, trunk, true);

if (!prev) {

trunk->str = "———";

}

else if (isLeft)

{

trunk->str = ".———";

prev\_str = " |";

}

else {

trunk->str = "`———";

prev->str = prev\_str;

}

showTrunks(trunk);

cout << " " << root->data << endl;

if (prev) {

prev->str = prev\_str;

}

trunk->str = " |";

printTree(root->left, trunk, false);

}

// Структура данных для хранения узла дерева арифмитических выражений

struct Tree

{

char data;

Tree\* left, \* right;

Tree(char data)

{

this->data = data;

this->left = this->right = nullptr;

};

Tree(char data, Tree\* left, Tree\* right)

{

this->data = data;

this->left = left;

this->right = right;

};

};

void printTrees(Tree\* root, Trunk\* prev, bool isLeft)

{

if (root == nullptr) {

return;

}

string prev\_str = " ";

Trunk\* trunk = new Trunk(prev, prev\_str);

printTrees(root->right, trunk, true);

if (!prev) {

trunk->str = "———";

}

else if (isLeft)

{

trunk->str = ".———";

prev\_str = " |";

}

else {

trunk->str = "`———";

prev->str = prev\_str;

}

showTrunks(trunk);

cout << " " << root->data << endl;

if (prev) {

prev->str = prev\_str;

}

trunk->str = " |";

printTrees(root->left, trunk, false);

}

// Функция для проверки, является ли данный токен оператором

bool isOperator(char c) {

return (c == '+' || c == '-' || c == '\*' || c == '/' || c == '^');

}

void preorder(Tree\* tree) { // префиксная форма

if (tree != nullptr) { //Пока не встретится пустой узел

cout << tree->data; //Отображаем корень дерева

preorder(tree->left); //Рекурсивная функция для левого поддерева

preorder(tree->right); //Рекурсивная функция для правого поддерева

}

}

// Печатаем постфиксное выражение для дерева выражений

void postorder(Tree\* root)

{

if (root == nullptr) {

return;

}

postorder(root->left);

postorder(root->right);

cout << root->data;

}

// Печатаем инфиксное выражение для дерева выражений

void inorder(Tree\* root)

{

if (root == nullptr) {

return;

}

// если текущий токен является оператором, вывести открывающую скобку

if (isOperator(root->data)) {

cout << "(";

}

inorder(root->left);

cout << root->data;

inorder(root->right);

// если текущий токен является оператором, вывести закрывающую скобку

if (isOperator(root->data)) {

cout << ")";

}

}

// Функция для построения дерева выражений из заданного постфиксного выражения

Tree\* construct(string postfix)

{

// базовый вариант

if (postfix.length() == 0) {

return nullptr;

}

// создаем пустой stack для хранения указателей дерева

stack<Tree\*> s;

// обход постфиксного выражения

for (char c : postfix)

{

// если текущий токен является оператором

if (isOperator(c))

{

// извлекаем из stack два узла `x` и `y`

Tree\* x = s.top();

s.pop();

Tree\* y = s.top();

s.pop();

// построить новое бинарное дерево, корнем которого является оператор, а

// левый и правый дочерние элементы указывают на `y` и `x` соответственно

Tree\* node = new Tree(c, y, x);

// поместить текущий узел в stack

s.push(node);

}

// если текущий токен является операндом, создаем новый узел бинарного дерева

// чей корень является операндом и помещаем его в stack

else {

s.push(new Tree(c));

}

}

// указатель на корень дерева выражений остается в stack

return s.top();

}

int main()

{

system("chcp 1251");

//int a, n;

// Построение над деревом

Node\* root = new Node(5);

root->left = new Node(1);

root->right = new Node(7);

root->left->left = new Node(0);

root->left->right = new Node(3);

root->right->right = new Node(8);

root->right->left = new Node(6);

string postfix = "ab+cde+\*\*";

Tree\* roots = construct(postfix);

// вывести построенное бинарное дерево

cout << "Бинарное дерево поиска:" << endl;

printTree(root, nullptr, false);

cout << endl;

cout << "Дерево арифмитических выражений:" << endl;

printTrees(roots, nullptr, false);

int Numb = 0;

cout << "-----------------------" << endl;

cout << "Доступные команды: " << endl;

cout << "1 - Вывести дерево в префиксной форме;" << endl;

cout << "2 - Вывести дерево в инфиксной форме;" << endl;

cout << "3 - Вывести дерево в постфиксной форме;" << endl;

cout << "4 - Найти min;" << endl;

cout << "5 - Найти max;" << endl;

cout << "6 - Найти элемент;" << endl;

cout << "7 - Найти следующий и предыдущий эл-т;" << endl;

cout << "8 - Удалить элемент;" << endl;

cout << "9 - Вывести визуализацию дерева;" << endl;

cout << "-----------------------" << endl;

cout << "Введите номер команады, чтобы ее выполнить : ";

cin >> Numb;

switch (Numb) {

case 1:

cout << "\nПрефиксная форма: ";

preorder(roots);

break;

case 2:

cout << "\nИнфиксная форма: ";

inorder(roots);

break;

case 3:

cout << "\nПостфиксная форма: ";

postorder(roots);

break;

case 4:

min(root);

break;

case 5:

max(root);

break;

case 6:

float a;

cout << "Введите число, которое хотите найти: ";

cin >> a;

find(root, a);

break;

case 7:

float b;

cout << "Введите число: ";

cin >> b;

cout << "Запрошенное число: " << b << endl;

cout << "Предыдущее число: " << findPre(root, b-1, root) << endl;

cout << "Следующее число: " << findSled(root, b+1,root);

break;

case 8:

int val;

cout << "Выберите элемент, который нужно удалить: ";

cin >> val;

DeleteNode(root, val);

cout << endl << "Элемент успешно удален"<<endl;

printTree(root, nullptr, false);

break;

case 9:

printTree(root, nullptr, false);

break;

}

return 0;

}

1. Ссылка на репозиторий

<https://github.com/typicalanya/lab4> sem5