Учреждение образования

«БЕЛОРУССКИЙГОСУДАРСТВЕННЫЙТЕХНОЛОГИЧЕСКИЙУНИВЕРСИТЕТ»

Н. Н. Пустовалова, Н. В. Пацей

# ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

**в 2-х частях**

**Часть 2**

**Лабораторный практикум**

**Лабораторная работа № 11. Бинарные деревья**

**Лабораторная работа № 11. Бинарные деревья**

|  |  |
| --- | --- |
| **Задание** | **Программа** |
| 1. Изучить работу с *бинарным деревом*, выполнив программу, записанную в правой части.  В программе осуществляется добавление элемента в дерево и вывод дерева на экран с поворотом на 90 градусов влево. |  |
| 2. В правой части приведена программа, которая осуществляет построение бинарного дерева, каждый элемент которого состоит из ключа (целое число) и слова размером не более 4 символов. В программе осуществляется добавление элемента, удаление элемента, поиск элемента по ключу, вывод дерева на экран, очистка дерева.  При вводе информации признак окончания − ввод отрицательного ключа.  Выполнить программу и написать комментарии к операторам. |  |

#include <iostream>

using namespace std;

struct Tree //дерево

{

int key; //ключ

char text[5]; //текст - не более 4 букв

Tree\* Left, \* Right;

};

Tree\* makeTree(Tree\* Root); //Создание дерева

Tree\* list(int i, char\* s); //Создание нового элемента

Tree\* insertElem(Tree\* Root, int key, char\* s); //Добавление нового элемента

Tree\* search(Tree\* n, int key); //Поиск элемента по ключу

Tree\* delet(Tree\* Root, int key); //Удаление элемента по ключу

int view(Tree\* t, int level); //Вывод дерева

int count(Tree\* t, char letter); //Подсчет количества слов

bool delAll(Tree\* t); //Очистка дерева

int c = 0; //количество слов

Tree\* Root = NULL; //указатель на корень

int main()

{

setlocale(0, "Russian");

int key, choice, n;

Tree\* rc;

char s[5], letter;

for (;;)

{

cout << "1 - создание дерева\n";

cout << "2 - добавление элемента\n";

cout << "3 - поиск по ключу\n";

cout << "4 - удаление элемента\n";

cout << "5 - вывод дерева\n";

cout << "6 - подсчет количества букв\n";

cout << "7 - очистка дерева\n";

cout << "8 - выход\n";

cout << "ваш выбор?\n";

cin >> choice;

cout << "\n";

switch (choice)

{

case 1:

Root = makeTree(Root);

break;

case 2:

cout << "\nВведите ключ: ";

cin >> key;

cout << "Введите слово: ";

cin >> s;

insertElem(Root, key, s);

break;

case 3:

cout << "\nВведите ключ: ";

cin >> key;

rc = search(Root, key);

cout << "Найденное слово= ";

puts(rc->text);

break;

case 4:

cout << "\nВведите удаляемый ключ: ";

cin >> key;

Root = delet(Root, key);

break;

case 5:

if (Root->key >= 0)

{

cout << "Дерево повернуто на 90 град. влево" << endl;

view(Root, 0);

}

else

{

cout << "Дерево пустое\n";

}

break;

case 6:

cout << "\nВведите букву: ";

cin >> letter;

n = count(Root, letter);

cout << "Количество слов, начинающихся с буквы " << letter;

cout << " равно " << n << endl;

break;

case 7:

delAll(Root);

break;

case 8:

exit(0);

}

}

return 0;

}

Tree\* makeTree(Tree\* Root) //Создание дерева

{

int key;

char s[5];

cout << "Конец ввода - отрицательное число\n\n";

if (Root == NULL) // если дерево не создано

{

cout << "Введите ключ корня: ";

cin >> key;

cout << "Введите слово корня: ";

cin >> s;

Root = list(key, s); // установка указателя на корень

}

while (1) //добавление элементов

{

cout << "\nВведите ключ: ";

cin >> key;

if (key < 0)

{

break; //признак выхода (ключ < 0)

}

cout << "Введите слово: ";

cin >> s;

insertElem(Root, key, s);

}

return Root;

}

Tree\* list(int i, char\* s) //Создание нового элемента

{

Tree\* t = new Tree[sizeof(Tree)];

t->key = i;

for (i = 0; i < 5; i++)

{

\*((t->text) + i) = \*(s + i);

}

t->Left = t->Right = NULL;

return t;

}

Tree\* insertElem(Tree\* t, int key, char\* s) //Добавление нового элемента

{

Tree\* Prev = t; // Prev - элемент перед текущим

int find = 0; // признак поиска

while (t && !find)

{

Prev = t;

if (key == t->key)

{

find = 1; //ключи должны быть уникальны

}

else

{

if (key < t->key)

{

t = t->Left;

}

else

{

t = t->Right;

}

}

}

if (!find) //найдено место с адресом Prev

{

t = list(key, s); //создается новый узел

if (key < Prev->key) // и присоединяется либо

{

Prev->Left = t; //переход на левую ветвь,

}

else

{

Prev->Right = t; // либо на правую

}

}

return t;

}

Tree\* delet(Tree\* Root, int key) //Удаление элемента по ключу

{ // Del, Prev\_Del - удаляемый элемент и его предыдущий;

// R, Prev\_R - элемент, на который заменяется удаленный, и его родитель;

Tree\* Del, \* Prev\_Del, \* R, \* Prev\_R;

Del = Root;

Prev\_Del = NULL;

while (Del != NULL && Del->key != key)//поиск элемента и его родителя

{

Prev\_Del = Del;

if (Del->key > key)

{

Del = Del->Left;

}

else

{

Del = Del->Right;

}

}

if (Del == NULL) // элемент не найден

{

puts("\nНет такого ключа");

return Root;

}

if (Del->Right == NULL) // поиск элемента R для замены

{

R = Del->Left;

}

else

{

if (Del->Left == NULL)

{

R = Del->Right;

}

else

{

Prev\_R = Del; //поиск самого правого элемента в левом поддереве

R = Del->Left;

while (R->Right != NULL)

{

Prev\_R = R;

R = R->Right;

}

if (Prev\_R == Del) // найден элемент для замены R и его родителя Prev\_R

{

R->Right = Del->Right;

}

else

{

R->Right = Del->Right;

Prev\_R->Right = R->Left;

R->Left = Prev\_R;

}

}

}

if (Del == Root)

{

Root = R; //удаление корня и замена его на R

}

else

{

// поддерево R присоединяется к родителю удаляемого узла

if (Del->key < Prev\_Del->key)

{

Prev\_Del->Left = R; //на левую ветвь

}

else

{

Prev\_Del->Right = R; //на правую ветвь

}

}

int tmp = Del->key;

cout << "\nУдален элемент с ключом " << tmp << endl;

delete Del;

return Root;

}

Tree\* search(Tree\* n, int key) //Поиск элемента по ключу

{

Tree\* rc = n;

if (rc != NULL)

{

if (key < (key, n->key))

{

rc = search(n->Left, key);

}

else

{

if (key > (key, n->key))

{

rc = search(n->Right, key);

}

}

}

else

{

cout << "Нет такого элемента\n";

}

return rc;

}

int count(Tree\* t, char letter) //Подсчет количества слов

{

if (t)

{

count(t->Right, letter);

if (\*(t->text) == letter)

{

c++;

}

count(t->Left, letter);

}

return c;

}

int view(Tree\* t, int level) //Вывод дерева

{

if (t)

{

view(t->Right, level + 1); //вывод правого поддерева

for (int i = 0; i < level; i++)

{

cout << " ";

}

int tm = t->key;

cout << tm << ' ';

puts(t->text);

view(t->Left, level + 1); //вывод левого поддерева

return 0;

}

return 1;

}

bool delAll(Tree\* t) //Очистка дерева

{

if (t != NULL)

{

delAll(t->Left);

delAll(t->Right);

delete t;

return true;

}

return false;

}

|  |  |
| --- | --- |
| 3. Разработать программу работы с ***бинарным деревом поиска***, в которую включить основные функции манипуляции данными и функцию в соответствии со своим вариантом из таблицы, представленной ниже.  **Вар.1:**  Вершина бинарного дерева содержит ключ, три целых числа и два указателя на потомков. Написать функцию удаления вершины с минимальной суммой трех целых значений узла. |  |

#include <iostream>

using namespace std;

struct Tree

{

int key;

int parm[3];

Tree\* left, \* right;

};

Tree\* root = new Tree;

void printLine()

{

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

}

void printMenu()

{

printLine();

cout << "Меню:\n";

cout << "1 - добавить элемент\n";

cout << "2 - поиск по ключу\n";

cout << "3 - удаление элемента по ключу\n";

cout << "4 - вывод дерева\n";

cout << "5 - очистка дерева\n";

cout << "6 - удаление вершины с минимальной суммой 3 значений узла\n";

cout << "7 - вывод меню\n";

cout << "8 - выход\n";

printLine();

}

Tree\* insertElement(Tree\* ptr, int key)

{

if (ptr->key < key && ptr->right != nullptr)

{

ptr = insertElement(ptr->right, key);

}

else if (ptr->key > key && ptr->left != nullptr)

{

ptr = insertElement(ptr->left, key);

}

return ptr;

}

void addElement()

{

Tree\* branch = new Tree;

Tree\* ptr = root;

branch->left = nullptr;

branch->right = nullptr;

cout << "Введите ключ: ";

cin >> branch->key;

cout << "Введите 3 целых числа:\n";

for (int i = 0; i < 3; i++)

{

cout << "Введите " << i + 1 << " целое число: ";

cin >> branch->parm[i];

}

if (ptr->key == NULL)

{

root = branch;

}

else

{

ptr = insertElement(ptr, branch->key);

if (ptr->key < branch->key)

{

ptr->right = branch;

}

else

{

ptr->left = branch;

}

}

}

void outputTree(int level, Tree\* ptr)

{

if (ptr->key != NULL)

{

if (ptr->right != nullptr)

{

outputTree(level + 1, ptr->right);

}

for (int i = 0; i < level; i++)

{

cout << " ";

}

cout << ptr->key << ": ";

for (int i = 0; i < 3; i++)

{

cout << ptr->parm[i] << ", ";

}

cout << "\n";

if (ptr->left != nullptr)

{

outputTree(level + 1, ptr->left);

}

}

else

{

cout << "Дерево не создано (для создания нажмите 1)\n";

}

}

Tree\* searchByKey(Tree\* ptr, int key)

{

if (ptr->key == key)

{

return ptr;

}

else if (ptr->key < key && ptr->right != nullptr)

{

searchByKey(ptr->right, key);

}

else if (ptr->key > key && ptr->left != nullptr)

{

searchByKey(ptr->left, key);

}

else

{

printLine();

cout << "Данного ключа нет в бинарном деревe\n";

printLine();

return nullptr;

}

}

void delByKey(Tree\* ptr, int key)

{

Tree\* temp, \* prevBranch = nullptr;

if (ptr->left == nullptr && ptr->right == nullptr)

{

printLine();

cout << "Данного ключа нет в бинарном дереве или значение является корнем\n";

printLine();

}

else if (ptr->right != nullptr && ptr->right->key == key)

{

prevBranch = ptr;

}

else if (ptr->left != nullptr && ptr->left->key == key)

{

prevBranch = ptr;

}

else

{

if (ptr->key < key && ptr->right != nullptr)

{

delByKey(ptr->right, key);

}

else if (ptr->key > key && ptr->left != nullptr)

{

delByKey(ptr->left, key);

}

}

if (prevBranch != nullptr)

{

temp = searchByKey(root, key);

if (prevBranch->right != nullptr && prevBranch->right->key == key)

{

prevBranch->right = temp->right;

Tree\* ptr1 = temp->right;

if (ptr1 != nullptr)

{

while (ptr1->left != nullptr)

{

ptr1 = ptr1->left;

}

ptr1->left = temp->left;

}

delete(temp);

}

else

{

if (temp->right != nullptr)

{

prevBranch->left = temp->right;

Tree\* ptr1 = temp->right;

while (ptr1->left != nullptr)

{

ptr1 = ptr1->left;

}

ptr1->left = temp->left;

}

else

{

prevBranch->left = temp->left;

}

delete(temp);

}

}

}

Tree\* freeCleaning(Tree\* ptr)

{

if (ptr->right != nullptr)

{

freeCleaning(ptr->right);

}

if (ptr->left != nullptr)

{

freeCleaning(ptr->left);

}

ptr->left = nullptr;

ptr->right = nullptr;

delete(ptr);

return ptr;

}

int removalMinSum(Tree\* ptr, int minSum, int key)

{

int sum=0;

if (ptr->right != nullptr)

{

key = removalMinSum(ptr->right, minSum, key);

}

if (ptr->left != nullptr)

{

key = removalMinSum(ptr->left, minSum, key);

}

for (int i = 0; i < 3; i++)

{

sum += ptr->parm[i];

}

if (sum < minSum)

{

key = ptr->key;

minSum = sum;

}

return key;

}

int main()

{

Tree\* ptr;

root->left = nullptr;

root->right = nullptr;

root->key = NULL;

int userInput = 0;

setlocale(LC\_CTYPE, "Russian");

printMenu();

while (userInput != 8)

{

cin >> userInput;

switch (userInput)

{

case 1:

addElement();

break;

case 2:

{

int key;

cout << "Введите ключ ";

cin >> key;

ptr = searchByKey(root, key);

if (ptr != nullptr)

{

printLine();

cout << "Ключ: " << ptr->key << ", значения: " << ptr->parm[0] << ", " << ptr->parm[1] << ", " << ptr->parm[2] << "\n";

printLine();

}

break;

}

case 3:

{

int key;

cout << "Введите ключ ";

cin >> key;

delByKey(root, key);

}

break;

case 4:

outputTree(0, root);

break;

case 5:

root = freeCleaning(root);

root = new Tree;

root->left = nullptr;

root->right = nullptr;

root->key = NULL;

break;

case 6:

{

int minSum = 0, key = root->key;

for (int i = 0; i < 3; i++)

{

minSum += root->parm[i];

}

key = removalMinSum(root, minSum, key);

delByKey(root, key);

}

break;

case 7:

printMenu();

break;

case 8:

break;

default:

cout << "Ошибка выбора\n";

break;

}

}

return 0;

}