ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Старший преподаватель |  |  |  | Рогачев С. А. |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №6 |
| «АВЛ - ДЕРЕВЬЯ ПОИСКА» |
| по курсу: Структуры и алгоритмы обработки данных |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | М011 |  |  |  | Борисов С. И. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2021

**Цель работы**

Целью работы является изучение деревьев поиска и получение практических навыков их использования.

**Задание на лабораторную**

Разработать на языке программирования высокого уровня программу, которая должна выполнять следующие функции:

− добавлять элементы в сбалансированное дерево поиска;

− удалять элементы из сбалансированного дерева поиска;

− искать элементы в дереве поиска с выводом количества шагов, за которое осуществляется поиск;

− выводить дерево на экран (любым способом доступным для восприятия);

− выводить список, соответствующий обходу вершин, в соответствии с вариантом задания;

− осуществлять операцию, заданную в таблице 6.

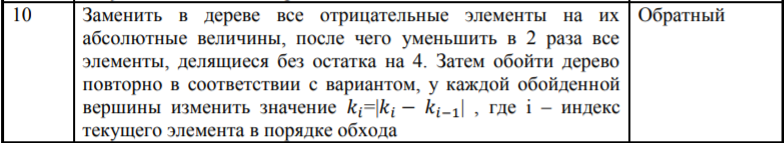


Таблица 6.

**Листинг программы, реализующей алгоритм**

// Борисов С.И. ЛБ-6 Вариант 10.

#include <iostream>

using namespace std;

int tabs = 0;

struct Node // структура для представления узлов дерева

{

int key;

unsigned char height;

Node\* left;

Node\* right;

Node(int k) { key = k; left = right = 0; height = 1; }

};

int height(Node\* p) // высота

{

return p ? p->height : 0;

}

int bfactor(Node\* p) //разность высот левого и правого поддеревьев

{

return height(p->right) - height(p->left);

}

void fixheight(Node\* p) // восстановка корректного поля высоты заданного узла

{

int hl = height(p->left);

int hr = height(p->right);

p->height = (hl > hr ? hl : hr) + 1;

}

Node\* rotate\_right(Node\* p) // правый поворот вокруг p

{

Node\* q = p->left;

p->left = q->right;

q->right = p;

fixheight(p);

fixheight(q);

return q;

}

Node\* rotate\_left(Node\* q) // левый поворот вокруг q

{

Node\* p = q->right;

q->right = p->left;

p->left = q;

fixheight(q);

fixheight(p);

return p;

}

Node\* balance(Node\* p) // балансировка

{

fixheight(p);

if (bfactor(p) == 2)

{

if (bfactor(p->right) < 0)

p->right = rotate\_right(p->right);

return rotate\_left(p);

}

if (bfactor(p) == -2)

{

if (bfactor(p->left) > 0)

p->left = rotate\_left(p->left);

return rotate\_right(p);

}

return p;

}

Node\* insert(Node\* p, int k) // вставка ключа k в дерево с корнем p

{

if (p == NULL) return new Node(k);

if (k < p->key)

p->left = insert(p->left, k);

else

p->right = insert(p->right, k);

return balance(p);

}

Node\* find\_min(Node\* p) // поиск узла с минимальным ключом в дереве p

{

return p->left ? find\_min(p->left) : p;

}

Node\* remove\_min(Node\* p) // удаление узла с минимальным ключом из дерева p

{

if (p->left == 0)

return p->right;

p->left = remove\_min(p->left);

return balance(p);

}

Node\* remove(Node\* p, int k) // удаление ключа k из дерева p

{

if (!p) return 0;

if (k < p->key)

p->left = remove(p->left, k);

else if (k > p->key)

p->right = remove(p->right, k);

else

{

Node\* q = p->left;

Node\* r = p->right;

delete p;

if (!r) return q;

Node\* min = find\_min(r);

min->right = remove\_min(r);

min->left = q;

return balance(min);

}

return balance(p);

}

void tree\_print(Node\* p)

{

if (p != NULL)

{

tree\_print(p->left);

tree\_print(p->right);

cout << p->key << endl;

}

}

void tree\_print\_beautiful(Node\* p)

{

if (p != NULL)

{

tabs += 5;

tree\_print\_beautiful(p->right);

for (int i = 0; i < tabs; i++) cout << " ";

cout << p->key << endl;

tree\_print\_beautiful(p->left);

tabs -= 5;

}

}

Node\* treeabs(Node\* p)

{

if (p != NULL)

{

treeabs(p->left);

treeabs(p->right);

p->key = abs(p->key);

return balance(p);

}

}

Node\* tree4(Node\* p)

{

if (p != NULL)

{

if (p->key % 4 == 0)

{

p->key /= 2;

}

tree4(p->left);

tree4(p->right);

return balance(p);

}

}

struct Node\_list

{

Node\* val;

Node\_list\* pnext;

Node\_list(Node\* v, Node\_list\* n)

{

val = v;

pnext = n;

}

};

struct List

{

Node\_list\* first = NULL;

Node\_list\* tail = NULL;

bool is\_empty()

{

return first ? 0 : 1;

};

void add(Node\* v)

{

Node\_list\* new\_elem = new Node\_list(v, NULL);

if (is\_empty())

{

first = new\_elem;

tail = new\_elem;

}

else

{

tail->pnext = new\_elem;

tail = new\_elem;

}

}

void minus()

{

Node\_list\* tmp1 = first;

Node\_list\* tmp2 = first->pnext;

do

{

tmp2->val->key -= tmp1->val->key;

tmp1 = tmp1->pnext;

tmp2 = tmp2->pnext;

} while (tmp2 != NULL);

}

};

Node\* trees(Node\* p, List\* n)

{

if (p != NULL)

{

trees(p->left, n);

trees(p->right, n);

n->add(p);

return balance(p);

}

}

int main()

{

setlocale(LC\_ALL, "rus");

system("color 0F");

Node\* p = NULL;

char x[100];

int choice = -1;

while (choice != 0)

{

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

cout << "|1. Показать дерево. |\n";

cout << "|2. Добавить вершину. |\n";

cout << "|3. Удалить вершину. |\n";

cout << "|4. Делящиеся на 4 без остатка уменьшить в 2.|\n";

cout << "|5. От текущей вершины отнять предыдущую. |\n";

cout << "|6. Красивое дерево. |\n";

cout << "|0. Выход. |\n";

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

cout << "Введите пункт меню: ";

do

{

cin >> x;

} while (!isdigit(x[0]));

choice = atof(x);

cout << "\n";

switch (choice)

{

case 1:

{

tree\_print(p);

break;

}

case 2:

{

int elem;

cout << "Введите значение: ";

cin >> elem;

p = insert(p, elem);

break;

}

case 3:

{

int elem;

cout << "Введите значение: ";

cin >> elem;

p = remove(p, elem);

break;

}

case 4:

{

treeabs(p);

tree4(p);

cout << "Новое дерево: \n";

tree\_print(p);

break;

}

case 5:

{

List\* n = new List;

trees(p, n);

n->minus();

cout << "Новое дерево: \n";

tree\_print(p);

break;

}

case 6:

{

tree\_print\_beautiful(p);

break;

}

case 0:

{

return 0;

break;

}

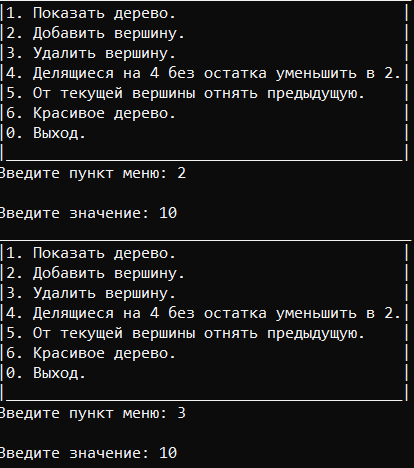
}

}

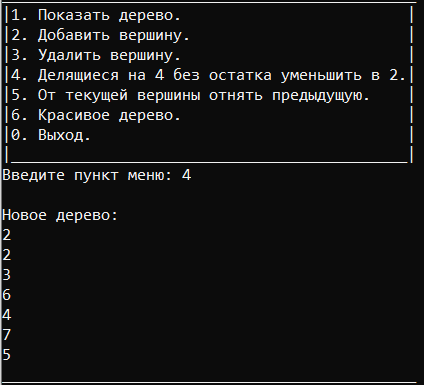
}

**Тестирование алгоритма:**

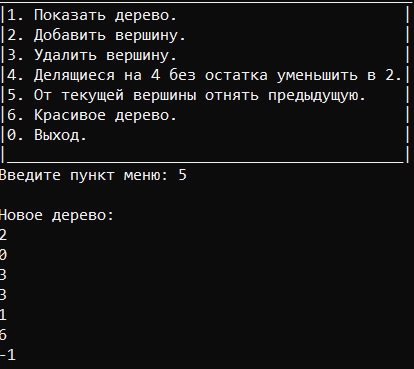
Добавление и удаление элемента:



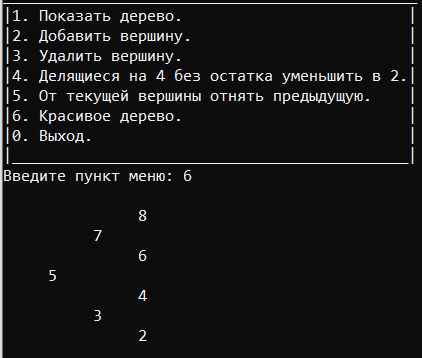
Пункт задания 1.



Пункт задания 2:



Вывод в консоль дерева:



**Выводы**

На основе этих расчетов можно сделать вывод, что был разработан алгоритм, характеристики которого соответствуют поставленному заданию.