МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «Полоцкий государственный университет»

Факультет информационных технологий

Кафедра технологий программирования

**Отчёт по лабораторной работе № 5 по курсу «Алгоритмы и структуры данных»**

«Деревья. Двоичное дерево. Применение деревьев при разработке приложений: код Хаффмана.»

ВЫПОЛНИЛ студент группы 21-ИТ-1

Макеёнок Д.И.

ПРОВЕРИЛ преподаватель

Виноградова А.Д.

Полоцк, 2022 г.

**Цель работы:** ознакомиться с основными понятиями «Деревья», «Двоичное дерево», «код Хаффмана» и алгоритмами их обработки, научиться применять полученные знания на практике.

**Теоретические сведения:**

1. Определение понятия двоичное дерево (далее ДД).

Двоичное дерево – древовидная структура данных, в которой у родительских узлов не может быть больше двух детей.

1. Перечислите типы двоичных деревьев.

* Полное двоичное дерево
* Совершенное двоичное дерево
* Законченное двоичное дерево
* Вырожденное двоичное дерево
* Скошенное вырожденное дерево
* Сбалансированное двоичное дерево

1. Разница совершенного ДД и законченного ДД.

**Совершенное двоичное дерево** — особый тип бинарного дерева, в котором у каждого внутреннего узла по два ребенка, а листовые вершины находятся на одном уровне.

**Законченное двоичное дерево** похоже на совершенное, но есть три большие отличия.

1. Все уровни должны быть заполнены.

2. Все листовые вершины склоняются влево.

3. У последней листовой вершины может не быть правого собрата. Это значит, что завершенное дерево необязательно полное.

1. Определение понятия обход дерева.

Обход дерева – это способ последовательного посещения узлов дерева, при котором каждый узел посещается только один раз.

1. Какие типы обходов существуют?

* В ширину
* В глубину

1. Какие характеристики имеет дерево Хаффмана?

* Для того же набора весов доступное дерево Хаффмана не обязательно уникально.
* Левое и правое поддеревья дерева Хаффмана можно поменять местами, поскольку это не влияет на взвешенную длину пути дерева.
* Все узлы с весами - это конечные узлы, а узлы без весов - все корневые узлы суббинарного дерева.
* Узлы с большими весами находятся ближе к корневому узлу дерева Хаффмана, а узлы с меньшими весами находятся дальше от корневого узла дерева Хаффмана.
* В дереве Хаффмана есть только листовые узлы и узлы со степенью 2, и нет узлов со степенью 1.
* Дерево Хаффмана с n листовыми узлами имеет 2n-1 узлов.

1. Этапы построения дерева Хаффмана.
2. Рассмотрим заданные n весов как n бинарных деревьев только с корневыми узлами (без левого и правого потомков), чтобы сформировать множество HT. Вес каждого дерева - это вес узла.
3. Выберите два двоичных дерева с наименьшими весами из набора HT, чтобы сформировать новое двоичное дерево, вес которого является суммой весов двух двоичных деревьев.
4. Удалите два двоичных дерева, выбранных на шаге 2, из набора HT, и добавьте вновь полученное двоичное дерево на шаге 2 в набор HT.
5. Повторяйте шаги 2 и 3 до тех пор, пока набор HT не будет содержать только одно дерево, которое является деревом Хаффмана.

**Код программы:**

#include <iostream>

#include <vector>

#include <Windows.h>

using namespace std;

struct node

{

int info; //Информационное поле

node\* l;

node\* r; //Левая и Правая часть дерева

};

node\* create(int a) {

node\* tree = new node;

tree->info = a;

tree->l = tree->r = NULL;

return tree;

}

node\* push(int a, node\* pdrev)

{

if (pdrev == NULL) {

return create(a);

}

else if (a > pdrev->info) {

pdrev->r = push(a, pdrev->r);

}

else {

pdrev->l = push(a, pdrev->l);

}

return pdrev;

}

bool prover(node\* t) {

return (t == NULL ? true : false);

}

void print(node\* t)

{

if (t) {

print(t->l);

cout << t->info << " ";

print(t->r);

}

}

void detour(node\* tree, vector<int>& arr) {

if (tree != NULL) {

detour(tree->l, arr);

arr.push\_back(tree->info);

detour(tree->r, arr);

}

}

void obhod(node\* t) {

if (t != NULL) {

cout << " " << t->info;

obhod(t->l);

obhod(t->r);

}

}

void delet(node\* t) {

if (t != NULL) {

delet(t->l);

delet(t->r);

delete (t);

}

}

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

node\* tmp = new node;

if (nod == NULL) {

return nod;

}

if (val == nod->info) {

if (nod->r == NULL)

node\* tmp = nod->l;

else {

node\* ptr = nod->r;

if (ptr->l == NULL) {

ptr->l = nod->l;

tmp = ptr;

}

else {

node\* pmin = ptr->l;

while (pmin->l != NULL) {

ptr = pmin;

pmin = ptr->l;

}

ptr->l = pmin->r;

pmin->l = nod->l;

pmin->r = nod->r;

tmp = pmin;

}

}

delete nod;

return tmp;

}

else if (val < nod->info) {

nod->l = DeleteNode(nod->l, val);

}

else {

nod->r = DeleteNode(nod->r, val);

}

return nod;

}

int main() {

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

int n = 0, k, y, q = 0;

int val;

cin >> y;

vector <int> a;

node\* tree = create(y);

while (n != -1) {

cout << "1-Добавить узел\n2-Удалить узел\n3-Вывести\n4-Есть ли одинаковые\n5-обход\n6-Удалить дерево\n";

std::cin >> n;

switch (n) {

case 1:

cout << "Добавляемый узел ";

cin >> k;

push(k, tree);

break;

case 2:

cin >> k;

DeleteNode(tree, k);

cout << endl;

break;

case 3:

if (prover(tree) == 0)

print(tree);

else cout << "Пусто";

cout << endl;

break;

case 4:

detour(tree, a);

for (int i = 0; i < a.size() - 1; i++) {

if (a[i] == a[i + 1]) q++;

}

if (q == 0)cout << "NO\n";

else cout << "YES\n";

cout << endl;

break;

case 5:

obhod(tree);

cout << endl;

break;

case 6:

delet(tree);

cout << endl;

break;

default:

break;

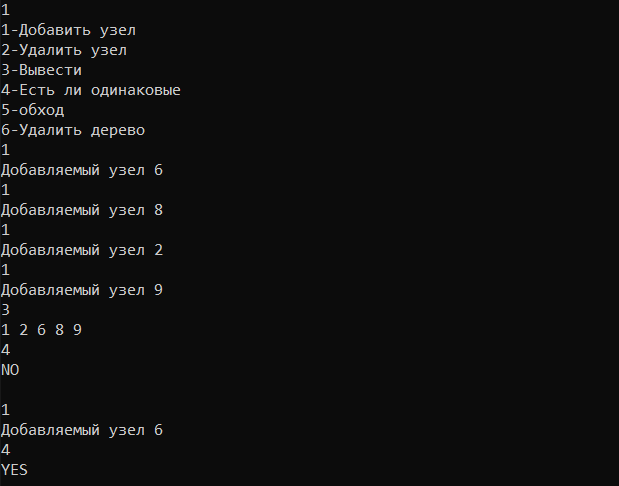
}

}

return 0;

}

**Результат работы программы:**



**Вывод:** Я ознакомился с основными понятиями «Деревья», «Двоичное дерево», «код Хаффмана», алгоритмами их обработки и научился применять полученные знания на практике.