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 при-

ложений: код Хаффмана

Вариант №4

ВЫПОЛНИЛ студент группы 21-ИТ-1

Шиковец Е.А.

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 — особый тип бинарных деревьев, в котором у каждого узла либо 0 потомков, либо 2.

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.

Вырожденное двоичное дерево — дерево, в котором на каждый уровень

приходится по одной вершине.

Скошенное вырожденное дерево — вырожденное дерево, в котором есть

либо только левые, либо только правые узлы. Таким образом, есть два типа

скошенных деревьев — скошенные влево вырожденные деревья и скошенные вправо вырожденные деревья.

Сбалансированное двоичное дерево — тип бинарного дерева, в котором у

каждой вершины количество вершин в левом и правом поддереве различаются либо на 0, либо на 1.

1. Разница совершенного ДД и законченного ДД.

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?

Существует три способа обхода дерева в глубину:

1. прямой (корень-лево-право), Pre\_order

2. поперечный (лево-корень-право), In\_order

3. обратный (лево-право-корень), Post\_order.

1. Какие характеристики имеет дерево Хаффмана?

В зависимости от количества узлов и весов форма дерева Хаффмана также

различна: дерево Хаффмана имеет следующие характеристики:

• 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.

Этапы построения дерева Хаффмана следующие:

1. 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.

2. Выберите два двоичных дерева с наименьшими весами из набора HT,

чтобы сформировать новое двоичное дерево, вес которого является суммой весов

двух двоичных деревьев.

3. Удалите два двоичных дерева, выбранных на шаге 2, из набора HT, и

добавьте вновь полученное двоичное дерево на шаге 2 в набор HT.

4. Повторяйте шаги 2 и 3 до тех пор, пока набор HT не будет содержать

только одно дерево, которое является деревом Хаффмана.

**ОПИСАНИЕ ПРОДЕЛАННОЙ РАБОТЫ:**

При выполнении работы я пользовался IDE Visual Studio Code. В результате выполнения работы был написан следующий код:

1 Часть:

#include <iostream>

#include <stdlib.h>

#include <cstdlib>

#include <vector>

using namespace std;

struct BinaryTree

{

int Data; // поле данных

BinaryTree \*Left; // указатель на левый потомок

BinaryTree \*Right; // указатель на правый потомок

};

//создание бинарного дерева

void create(BinaryTree \*\*Node, int n)

{

BinaryTree \*\*ptr; //вспомогательный указатель

srand(time(NULL) \* 1000);

while (n > 0)

{

ptr = Node;

while (\*ptr != NULL)

{

if ((double)rand() / RAND\_MAX < 0.5)

ptr = &((\*ptr)->Left);

else

ptr = &((\*ptr)->Right);

}

(\*ptr) = new BinaryTree();

cout << "0. Input data: ";

cin >> (\*ptr)->Data;

n--;

}

}

//печать бинарного дерева

void print(BinaryTree \*Node, int l)

{

int i;

if (Node != NULL)

{

print(Node->Right, l + 1);

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

cout << " ";

printf("%4ld", Node->Data);

print(Node->Left, l + 1);

}

else

cout << endl;

}

//прямой обход бинарного дерева

void pre\_order(BinaryTree \*Node)

{

if (Node != NULL)

{

printf("%3ld", Node->Data);

pre\_order(Node->Left);

pre\_order(Node->Right);

}

}

//обратный обход бинарного дерева

void post\_order(BinaryTree \*Node)

{

if (Node != NULL)

{

post\_order(Node->Left);

post\_order(Node->Right);

printf("%3ld", Node->Data);

}

}

//симметричный обход бинарного дерева

void symmetric\_order(BinaryTree \*Node)

{

if (Node != NULL)

{

post\_order(Node->Left);

printf("%3ld", Node->Data);

post\_order(Node->Right);

}

}

vector<int> array;

void task\_data(BinaryTree \*Node)

{

if (Node != NULL)

{

task\_data(Node->Left);

task\_data(Node->Right);

array.push\_back(Node->Data);

}

}

bool task(BinaryTree \*Node)

{

task\_data(Node);

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

{

for (int j = i + 1; j < array.size(); j++)

{

if (array[i] == array[j])

return 1;

}

}

return 0;

}

//вставка вершины в бинарное дерево

void insert(BinaryTree \*\*Node, int Data)

{

BinaryTree \*New\_Node = new BinaryTree;

New\_Node->Data = Data;

New\_Node->Left = NULL;

New\_Node->Right = NULL;

BinaryTree \*\*ptr = Node; //вспомогательный указатель

srand(time(NULL) \* 1000);

while (\*ptr != NULL)

{

double q = (double)rand() / RAND\_MAX;

if (q < 1 / 3.0)

ptr = &((\*ptr)->Left);

else if (q > 2 / 3.0)

ptr = &((\*ptr)->Right);

else

break;

}

if (\*ptr != NULL)

{

if ((double)rand() / RAND\_MAX < 0.5)

New\_Node->Left = \*ptr;

else

New\_Node->Right = \*ptr;

\*ptr = New\_Node;

}

else

{

\*ptr = New\_Node;

}

}

//удаление вершины из бинарного дерева

void delete\_node(BinaryTree \*\*Node, int Data)

{

if ((\*Node) != NULL)

{

if ((\*Node)->Data == Data)

{

BinaryTree \*ptr = (\*Node);

if ((\*Node)->Left == NULL && (\*Node)->Right == NULL)

(\*Node) = NULL;

else if ((\*Node)->Left == NULL)

(\*Node) = ptr->Right;

else if ((\*Node)->Right == NULL)

(\*Node) = ptr->Left;

else

{

(\*Node) = ptr->Right;

BinaryTree \*\*ptr1;

ptr1 = Node;

while (\*ptr1 != NULL)

ptr1 = &((\*ptr1)->Left);

(\*ptr1) = ptr->Left;

}

delete (ptr);

delete\_node(Node, Data);

}

else

{

delete\_node(&((\*Node)->Left), Data);

delete\_node(&((\*Node)->Right), Data);

}

}

}

//проверка пустоты бинарного дерева

bool is\_tree\_empty(BinaryTree \*Node)

{

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

}

//освобождение памяти, выделенной под бинарное дерево

void delete\_tree(BinaryTree \*Node)

{

if (Node != NULL)

{

delete\_tree(Node->Left);

delete\_tree(Node->Right);

delete (Node);

}

}

int main()

{

BinaryTree \*bt = NULL;

system("CLS");

system("Color 09");

int n = -1, amount, data;

while (n != 0)

{

cout << "~-~ Main menu ~-~\n";

cout << "1. Insert node\n";

cout << "2. Delete node\n";

cout << "3. Print tree\n";

cout << "4. Is empty\n";

cout << "5. Order\n";

cout << "6. Task\n";

cout << "0. Exit\n";

cin >> n;

switch (n)

{

case 1: // создание дерева или вставка элемента

system("CLS");

cout << "Amount of nodes: ";

cin >> amount;

if (amount <= 0) // проверка на якушева

{

system("CLS");

cout << "!\* Non correct input\n";

break;

}

if (bt == NULL) // проверка на наличие дерева

{

create(&bt, 1);

for (int i = 1; i < amount; i++)

{

cout << i << ". Input node: ";

cin >> data;

insert(&bt, data);

}

}

else

{

for (int i = 1; i < amount + 1; i++)

{

cout << i << ". Input node: ";

cin >> data;

insert(&bt, data);

}

}

system("CLS");

cout << "Tree was modyfied\n";

break;

case 2:

if (bt != NULL)

{

system("CLS");

cout << "Input data: ";

cin >> data;

delete\_node(&bt, data);

system("CLS");

cout << "Tree was modyfied\n";

}

else

{

system("CLS");

cout << "!\* Binary tree is empty\n";

}

break;

case 3:

if (bt != NULL)

{

system("CLS");

print(bt, 1);

}

else

{

system("CLS");

cout << "!\* Binary tree is empty\n";

}

break;

case 4:

system("CLS");

if (is\_tree\_empty(bt))

cout << "Tree is empty\n";

else

cout << "Tree is not empty\n";

break;

case 5:

system("CLS");

if (is\_tree\_empty(bt))

{

cout << "!\* Binary tree is empty\n";

}

else

{

cout << "Symmetric order:";

symmetric\_order(bt);

cout << endl

<< "Post order: \t";

post\_order(bt);

cout << endl

<< "Pre order: \t";

pre\_order(bt);

cout << endl;

}

break;

case 6:

system("CLS");

if (bt != NULL)

{

if (task(bt))

cout << "There are equal elements\n";

else

cout << "There are no equal elements\n";

}

else

{

cout << "!\* Binary tree is empty\n";

}

break;

case 0:

system("Color 07");

system("CLS");

delete\_tree(bt);

return 0;

break;

default:

system("CLS");

cout << "!\* Check your input\n";

break;

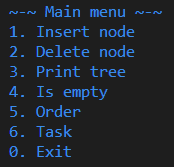
}

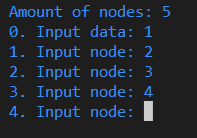
}

return 0;

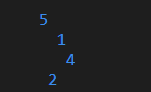
}

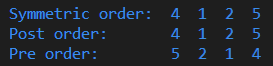
**РЕЗУЛЬТАТЫ ВЫПОЛЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ**:

Главное меню. Через него выполняются все операции, прописанные в задании, осуществляемые с бинарным деревом. А именно: вставка элемента, создание дерева, удаление элемента, печать дерева, проверка на пустоту и вывод задания (проверка на совпадающие элементы).

****Вставка элемента выглядит следующим образом. Сначала задается количество вставляемых элементов, потом проверяется создано ли дерево, в зависимости от этого дерево либо дополняется, либо создается и дополняется.

****Удаление элемента.

****Вывод двоичного дерева.

Обход дерева.

**Вывод:** в результате выполнения лабораторной работы было реализовано двоичное дерево. Также я научился работать с деревьями на ЯП C++.