МИНОБРНАУКИ РОССИИ

ФГБОУ ВО Череповецкий государственный университет

Институт информационных технологий

Кафедра: МПО ЭВМ

Дисциплина: Структуры и алгоритмы обработки данных

ЛАБОРАТОРНАЯ РАБОТА № 2

Бинарные деревья

Выполнила: студентка гр. 1ПИб-01-1оп-21

Пикалова А.С. Проверил: Пышницкий Константин Михайлович

Череповец, 2023 г.

Задание: написать функцию формирования бинарного дерева, состоящего из целых чисел. Для представления дерева использовать динамические структуры данных. Количество элементов дерева, а также его вид задаются случайным образом. Произвести вывод элементов дерева тремя видами обхода. Используя информацию о выведенном дереве изобразить структуру одного из построенных деревьев в отчете. Выполнить по вариантам следующие задания:

4. Напишите рекурсивную функцию, которая удаляет из дерева все четные элементы.

**Теория**:

Дерево – структура данных, представляющая собой древовидную структуру в виде набора связанных узлов.

Бинарное дерево — это конечное множество элементов, которое либо пусто, либо содержит элемент (корень), связанный с двумя различными бинарными деревьями, называемыми левым и правым поддеревьями. Каждый элемент бинарного дерева называется узлом. Связи между узлами дерева называются его ветвями.

Максимальный уровень какого-либо элемента дерева называется его **глубиной** или **высотой.**

Если элемент не имеет потомков, он называется **листом** или **терминальным узлом дерева.**

Остальные элементы – **внутренние узлы** (узлы ветвления).

Число потомков внутреннего узла называется его **степенью**. Максимальная степень всех узлов есть **степень дерева**.

Число ветвей, которое нужно пройти от корня к узлу x, называется **длиной пути к x**. Корень имеет длину пути равную 0; узел на уровне i имеет длину пути равную i.

Бинарное дерево применяется в тех случаях, когда в каждой точке вычислительного процесса должно быть принято **одно из двух возможных решений**.

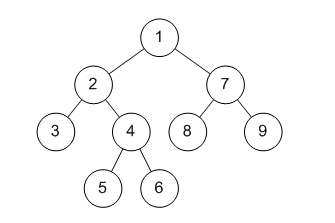
**Бинарное (двоичное) дерево поиска** – это бинарное дерево, для которого выполняются следующие дополнительные условия (свойства дерева поиска):

* оба поддерева – левое и правое, являются двоичными деревьями поиска;
* у всех узлов левого поддерева произвольного узла X значения ключей данных меньше, чем значение ключа данных самого узла X;
* у всех узлов правого поддерева произвольного узла X значения ключей данных не меньше, чем значение ключа данных узла X.

Данные в каждом узле должны обладать ключами, на которых определена операция сравнения меньше.

Как правило, информация, представляющая каждый узел, является записью, а не единственным полем данных.

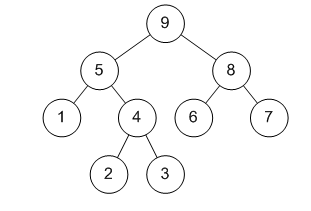
Существует три основных способа обхода в глубину.

1. Прямой (pre-order)

Посетить корень

Обойти левое поддерево

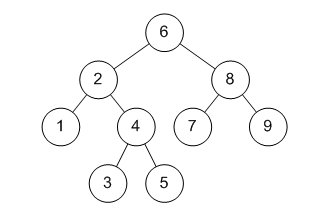
Обойти правое поддерево

1. В обратном порядке (post-order)

Обойти левое поддерево

Обойти правое поддерево

Посетить корень

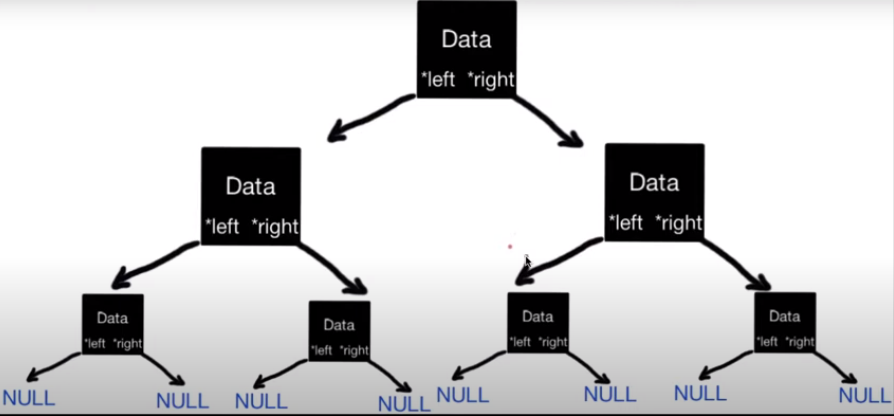


1. Симметричный или поперечный (in-order)

Обойти левое поддерево

Посетить корень

Обойти правое поддерево



Алгоритм:

Класс Tree:

Структура Node

Хранит поле данных и указатель на левый и правый потомок

Переменная структурного типа root, отвечающая за текущий элемент

Функция show\_tree\_Pr

* + - 1. Функция принимает значение node
      2. Если node!=NULL
         1. вывод значения node->data на экран
         2. Вызов функции show\_tree\_Pr для значения node->left
         3. Вызов функции show\_tree\_Pr для значения node->right

Функция show\_tree\_Obr

* + - 1. Функция принимает значение node
      2. Если node!=NULL
         1. Вызов функции show\_tree\_Obr для значения node->left
         2. Вызов функции show\_tree\_Obr для значения node->right
         3. вывод значения node->data на экран

Функция show\_tree\_Sim

* + - 1. Функция принимает значение node
      2. Если node!=NULL
         1. Вызов функции show\_tree\_Sim для значения node->left
         2. вывод значения node->data на экран
         3. Вызов функции show\_tree\_Sim для значения node->right

Функция Tree

root присвоить NULL

Функция add

* + - 1. Функция получает на вход значение value
      2. Если root = NULL
         1. root присвоить new Node
         2. Переменной root->data присвоить value
         3. Переменной root->left присвоить NULL
         4. Переменной root->right присвоить NULL
      3. Иначе
         1. Если rand()%2 == 1

вызов функции add для значений root->left и value

* + - 1. Иначе
         1. вызов функции add для значений root->right и value

Функция add

Вызов функции add и передача узла и значения value

Функция show\_Pr

Вызов функции show\_tree\_Pr для значения root

Функция show\_Obr

1. Вызов функции show\_tree\_Obr для значения root

Функция show\_Sim

1. Вызов функции show\_tree\_Sim для значения root

Функция DeleteElement

1. Функция принимает значение node
2. Если node!=NULL
   1. Если node->left != NULL
      1. Записывает поле данных левого потомка
      2. Если DeleteElement(node->left)
         1. Присваиваем левому потомку значение NULL
   2. Иначе если node->right != NULL
      1. Записывает поле данных правого потомка
      2. DeleteElement(node->right)
         1. Присваиваем правому потомку значение NULL
   3. Иначе
      1. delete node
      2. return true
3. Возвращает флаг false;

(Флаг нужен, чтобы знать, что удалился элемент при предыдущем вызове функции и поставить вместо него в дереве null)

Функция DeleteEven

1. Функция принимает значение node
2. Если node!=NULL
   1. Если DeleteEven(node->left)
      1. node->left = NULL
   2. Если DeleteEven(node->right)
      1. node->right = NULL;
   3. Если node->data % 2 == 0
      1. Если DeleteElement(node)
         1. return true;
3. Возвращает флаг false;

Функция DeleteEven

Вызов функции DeleteEven для значения root

Функция main

Инициализация переменной TreeSize со случайным значением

Создание объекта tree класса Tree

Цикл по переменной i от 0 до TreeSize

Инициализация переменной v со случайным значением

Вызов функции add для объекта tree и значения v

Вызов функции show\_Pr() для объекта tree

Вызов функции show\_Obr() для объекта tree

Вызов функции show\_Sim() для объекта tree

Вызов функции DeleteEven()

Вызов функции show\_Pr() для объекта tree

Программа:

#include <iostream>

#include <time.h>

using namespace std;

class Tree {

private:

struct Node {

int data; //поле данных

Node\* left, \* right; //указатель на левый и правый потомок

};

Node\* root; //указатель на узел

void show\_tree\_Pr(Node\* node) { //прямой обход

if (node != NULL) { //Пока не встретится пустой узел

cout << node->data << " ";//Отображаем корень дерева

show\_tree\_Pr(node->left); //Рекурсивная функция для левого поддерева

show\_tree\_Pr(node->right);//Рекурсивная функция для правого поддерева

}

}

void show\_tree\_Obr(Node\* node) {//обратный обход

if (node != NULL) {//Пока не встретится пустой узел

show\_tree\_Obr(node->left);//Рекурсивная функция для левого поддерева

show\_tree\_Obr(node->right);//Рекурсивная функция для правого поддерева

cout << node->data << " ";//Отображаем корень дерева

}

}

void show\_tree\_Sim(Node\* node) {//симметричный обход

if (node != NULL) {//Пока не встретится пустой узел

show\_tree\_Sim(node->left);//Рекурсивная функция для левого поддерева

cout << node->data << " ";//Отображаем корень дерева

show\_tree\_Sim(node->right);//Рекурсивная функция для правого поддерева

}

}

public:

Tree() {// функция

root = NULL;//root присвоить NULL

}

void add(Node\*& root, int value) {//Добавление узлов в дерево

if (root == NULL) {// Если дерева нет, то формируем корень

root = new Node;// память под узел

root->data = value;// поле данных

root->left = NULL; // ветви инициализируем пустотой

root->right = NULL; //

}

else {

if (rand() % 2) {// условие добавление левого потомка

//cout << " l";

add(root->left, value);

}

else {// условие добавление правого потомка

//cout << " r";

add(root->right, value);

}

}

}

void add(int value) {

add(root, value);

}

void show\_Pr() {

show\_tree\_Pr(root);

cout << endl;

}

void show\_Obr() {

show\_tree\_Obr(root);

cout << endl;

}

void show\_Sim() {

show\_tree\_Sim(root);

cout << endl;

}

bool DeleteElement(Node\* node) { //удаление элемента поддерева

if (node != NULL) {

if (node->left != NULL) {

node->data = node->left->data;

if (DeleteElement(node->left)) {

node->left = NULL;

}

}

else if (node->right != NULL) {

node->data = node->right->data;

if (DeleteElement(node->right)) {

node->right = NULL;

}

}

else {

delete node;

return true;

}

}

return false;

}

bool DeleteEven(Node\* node) {

if (node != NULL) {

if (DeleteEven(node->left)) {//если идем в лево - удаляем

node->left = NULL;

}

if (DeleteEven(node->right)) {//если идем в право - удаляем

node->right = NULL;

}

if (node->data % 2 == 0) {//если элемент четный, то мы его удаляем

if (DeleteElement(node)) {

return true;

}

}

}

return false;

}

void DeleteEven() {

DeleteEven(root);

}

};

void main() {

srand(time(0));

int TreeSize = rand() % 5 + 10;

Tree tree;

for (int i = 0; i < TreeSize; i++) {

int v = rand() % 1000;

cout << v;

tree.add(v);

cout << endl;

}

cout << "Pre-order" << endl;

tree.show\_Pr();//вывод прямого обхода

cout << "Post-order" << endl;

tree.show\_Obr();//вывод обратного обхода

cout << "In-order" << endl;

tree.show\_Sim();//вывод симметричного обхода

tree.DeleteEven();//удаление четных элементов

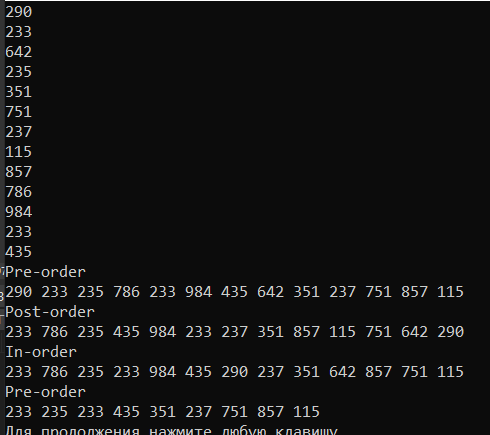
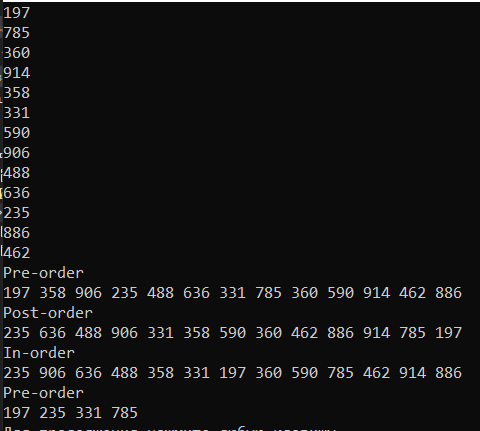
cout << "Pre-order" << endl;

tree.show\_Pr();//вывод нового дерева

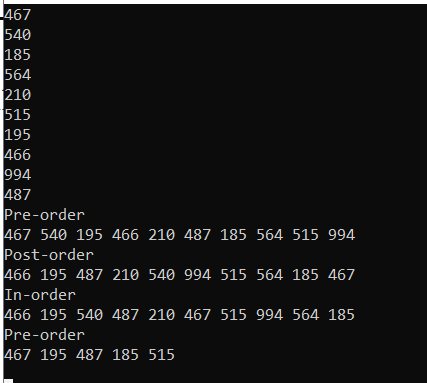
system("pause");

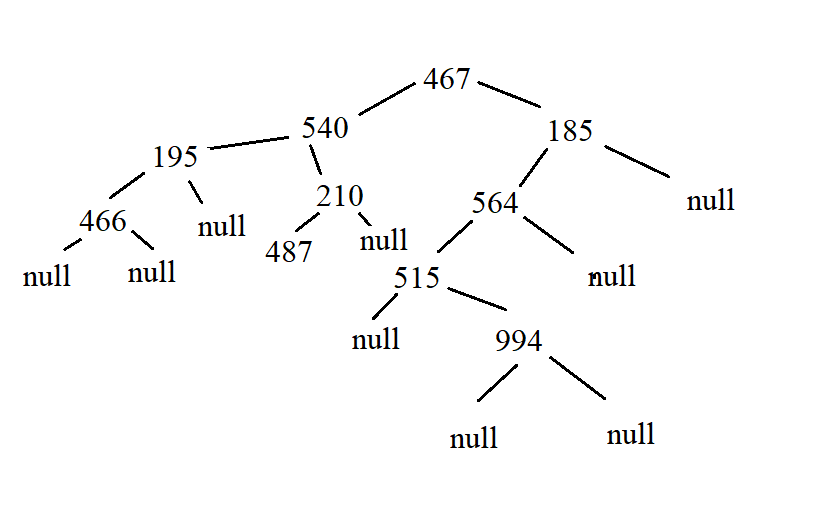
}

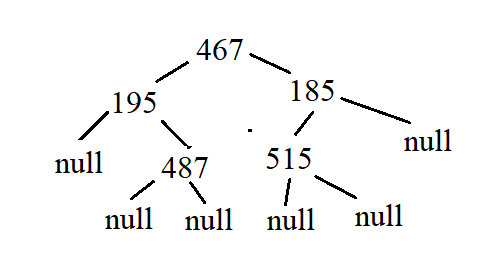
Тест:

Дерево:





- дерево с удаленными элементами

Вывод: на основе своего варианта было построено бинарное дерево, с помощью рекурсивной функции удалены четные элементы дерева и построено новое дерево с нечетными элементами.