МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет: Информационных технологий

Кафедра: Программной инженерии

Выполнила: студентка 1 курса 5 группы

специальности ПОИТ Дзивнель М.А.

Проверил: преподаватель

Белодед Николай Иванович

**РЕФЕРАТ**

По дисциплине “Основы алгоритмизации и программирования”

На тему “Бинарные деревья”

Минск

2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ ……………………………………………………………………….2

Вставка элемента….………………………………………………………………2

Обход…………………………………...………………………………………….4

Удаление……………………………..…………………………………………….4

ВЫВОД………………………………………………….…………………………7

**Введение**

Бинарное дерево – иерархическая структура данных, в которой каждый узел имеет значение и ссылки на левого и правого потомка. Потомков не может быть больше 2 (отсюда и название «бинарное»). Узел, находящийся на самом верхнем уровне и не имеющий потомков – корень. Узлы, у которых нет потомков (правый и левый потомок == NULL) – листья.

Есть несколько подвидов бинарных деревьев. Один из самых часто встречающихся – бинарное дерево поиска. Суть этого вида деревьев в том, что данные находятся в отсортированном виде. Хранение организовано следующим образом:

1. Элементы, которые меньше значения родительского узла располагаются в левом поддереве.
2. Элементы, которые больше (либо равны) значения родительского узла – в правом.
3. Все дочерние узлы также являются бинарными деревьями поиска.

При таком способе хранения данных поиск элемента в дереве занимает О(logN), что значительно меньше, если хранить элементы в списках.

**Реализация. Функция вставки элемента**

Реализовать бинарные деревья можно на структурах:

#include <iostream>

using namespace std;

struct Node {

int data;//данные, хранящиеся в узле

Node\* left;//указатель на левого потомка

Node\* right;//указатель на правого потомка

Node(int data)//конструктор нового элемента

{

this->data = data;

left = nullptr;

right = nullptr;

}

};

struct Tree {

Node\* root;

Tree()//конструктор дерева, задающий корню в начале значение NULL

{

this->root= NULL;

}

};

Где Node – структура, реализующая элемент дерева (узел), Tree – структура дерева.

Вставка элемента в бинарное дерево поиска должна сохранять дерево в отсортированном виде. Для этого необходимо сравнивать значение элемента со значением узла и идти либо к левому, либо к правому потомку. Подход к реализации может быть рекурсивным или через циклы. Функция вставки, реализованная на основе циклов:

void pushElem(int data, Node\* &root)

{

Node\* node = new Node(data);//Создаем новый элемент типа Node\*, который будет хранить добавляемые данные

Node\* temp = NULL;//временная переменная

if (root == nullptr)//если в дереве нет элементов

{

root = node;//значению корня дерева прсваиваем добавляемое значение

return;

}

temp = root;//врем переменной присваиваем значение корня (начинаем проходить по элементам дерева начиная с корня)

while (temp)

{

if (node->data < temp->data)//доб значение меньше значения корня

{

if (temp->left == NULL)//нет левого потомка

{

temp->left = node;//добавляем значение на место левого потомка

return;

}

else {

temp = temp->left;//узел == левый потомок

continue;

}

}

if (node->data > temp->data)//доб значение больше значения корня

{

if (temp->right == NULL)//нет правого потомка

{

temp->right = node;//добавляем значение на место правого потомка

return;

}

else {

temp = temp->right;//узел == правый потомок

continue;

}

}

}

}

**Функция обхода дерева**

Обход дерева – последовательное посещение всех узлов дерева. Пример прямого обхода, реализованного с помощью рекурсии:

void show(Node\* Node)

{

if (Node == NULL)

return; //Если дерева нет, выходим

cout << Node->data << " "; //Посетили узел

show(Node->left); //Обошли левое поддерево

show(Node->right); //Обошли правое поддерево

}

Выполнение двух функций (добавления элемента и обхода):

int main() {

Tree tree;

pushElem(5, tree.root);

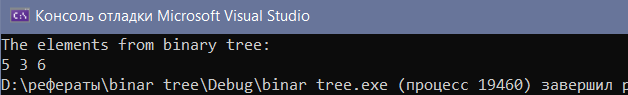
pushElem(6, tree.root);

pushElem(3, tree.root);

cout << "The elements from binary tree: " << endl;

show(tree.root);

}



**Функция удаления элемента из дерева**

При удалении элемента могут возникнуть 3 ситуации: звено не имеет потомков, имеет одного потомка или имеет двух потомков. Реализация функции удаления звена с помощью рекурсии:

// Вспомогательная функция для поиска узла мин значения в поддереве с корнем `curr`

Node\* getMinimumEl(Node\* curr)

{

while (curr->left != nullptr) {

curr = curr->left;

}

return curr;

}

void searchEl(Node\*& curr, int data, Node\*& parent)

{

while (curr != nullptr && curr->data != data)

{

// обновить родителя до текущего узла

parent = curr;

// если заданное значение меньше текущего узла, переходим в левое поддерево;

// иначе идем в правое поддерево

if (data < curr->data) {

curr = curr->left;

}

else {

curr = curr->right;

}

}

}

// Функция для удаления узла из дерева

void deleteNode(Node\*& root, int data)

{

// указатель для хранения родителя текущего узла

Node\* parent = nullptr;

// начинаем с корневого узла

Node\* curr = root;

// поиск значения в дереве и установка его родительского указателя

searchEl(curr, data, parent);

// возвращаем, если значение не найдено в дереве

if (curr == nullptr) {

return;

}

// Случай 1: удаляемый узел не имеет дочерних элементов, т. е. является листом

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

{

// если удаляемый узел не является корневым узлом, то устанавливаем его

// родительский левый/правый дочерний элемент в null

if (curr != root)

{

if (parent->left == curr) {

parent->left = nullptr;

}

else {

parent->right = nullptr;

}

}

// если дерево имеет только корневой узел, устанавливаем его в null

else {

root = nullptr;

}

delete curr;

}

// Случай 2: удаляемый узел имеет двух потомков

else if (curr->left && curr->right)

{

// найти его неупорядоченный узел-преемник

Node\* successor = getMinimumEl(curr->right);

// сохраняем последующее значение

int val = successor->data;

// рекурсивно удаляем преемника

deleteNode(root, successor->data);

// копируем значение преемника в текущий узел

curr->data = val;

}

// Случай 3: удаляемый узел имеет только одного потомка

else {

// выбираем дочерний узел

Node\* child = (curr->left) ? curr->left : curr->right;

// если удаляемый узел не является корневым узлом, устанавливаем его родителя

// своему потомку

if (curr != root)

{

if (curr == parent->left) {

parent->left = child;

}

else {

parent->right = child;

}

}

// если удаляемый узел является корневым узлом, то установить корень дочернему

else {

root = child;

}

delete curr;

}

}

Результат выполнения:

int main() {

Tree tree;

pushElem(5, tree.root);

pushElem(6, tree.root);

pushElem(3, tree.root);

cout << "The elements from binary tree: " << endl;

show(tree.root);

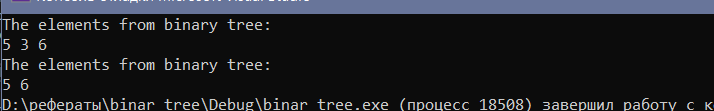
cout << endl;

cout << "The elements from binary tree: " << endl;

deleteNode(tree.root,3);

show(tree.root);

}



**Вывод**

Бинарное дерево поиска – удобная организация данных, которую используют не только для поиска, но и как основу других структур данных (например кучи и словари). Для работы с деревьями есть функции вставки, поиска, обхода и удаления элементов.