**Шараев Владислав Николаевич 020602**

**Лабораторная работа №16**

**Вариант 14**

***Условие:*** Определить количество узлов в левой ветви дерева.

***Текст программы:***

#define \_CRT\_SECURE\_NO\_WARNINGS // Предпроцессор для отключения предупреждений при работе со scanf и printf

#include <iostream>

#include <cstdio>

#include <ctime>

using namespace std;

struct Tree // Структура элемента очереди

{

int value; // Значение элемента

Tree\* left = NULL; // Указатель на левый элемент

Tree\* right = NULL; // Указатель на правый элемент

};

Tree\* BuildTree(int\*, int, int, Tree\*); // Построение дерева из массива данных

Tree\* NewNode(int); // Создание нового узла

Tree\* AddNode(Tree\*, int); // Добавление узла в дерево

Tree\* DeleteNode(Tree\*, int); // Удаление узла из дерева

Tree\* FindNode(Tree\*, int); // Поиск узла в дереве

Tree\* DeleteTree(Tree\*); // Удаление всего дерева

Tree\* MinValueNode(Tree\*); // Поиск минимального значения в дереве

void StraightPrint(Tree\*); // Прямой обход дерева

void ReversePrint(Tree\*); // Обратный обход дерева

void IncPrint(Tree\*); // Обход в порядке возрастания элементов

void quickSort(int\*, int, int);

int CountNodes(Tree\*); // Посчитать количество узлов

int main()

{

setlocale(LC\_ALL, "Russian"); // Поддержка кириллицы

srand(time(0)); // Инициализация псевдослучайных чисел

int n;

Tree\* root = 0; // Корень дерева

printf("Введите количество элементов: "); // Чтение количества элементов

scanf("%d", &n);

int\* a = new int[n];

for (int i = 0; i < n; i++) // Заполнение списка элементами

{

int temp = rand() % 101 - 50; // Генерация чисел в диапазоне [-50;50]

a[i] = temp;

}

quickSort(a, 0, n - 1); // Сортировка массива элементов

printf("Значения добавляемых узлов: ");

root = BuildTree(a, 0, n - 1, root); // Построение сбалансированного дерева

printf("\n");

printf("\nВсе узлы дерева (прямой обход):\n");

StraightPrint(root); // Прямой обход дерева

printf("\n");

printf("\nВсе узлы дерева (в порядке возрастания):\n");

IncPrint(root); // Обход в порядке возрастания элементов

printf("\n");

printf("\nВсе узлы дерева (обратный обход):\n");

ReversePrint(root); // Обратный обход дерева

printf("\n");

int toDelete, toFind; // Значения элементов для поиска и удаления

printf("\nВведите значения элементов для поиска и удаления: ");

scanf("%d%d", &toFind, &toDelete);

if (FindNode(root, toFind) != 0) // Поиск узла по значению

{

printf("\nУзел %d существует\n", toFind);

}

else

{

printf("\nУзел %d не существует\n", toFind);

}

printf("\nУдаление элемента %d из дерева\n", toDelete);

root = DeleteNode(root, toDelete); // Удаление элемента по значению

printf("Все узлы дерева после удаления %d:\n", toDelete);

StraightPrint(root); // Прямой обход дерева

printf("\n");

printf("\nКоличество узлов в левой ветви дерева: %d\n", CountNodes(root->left)); // Подсчет узлов в левой ветви

root = DeleteTree(root);

printf("\nДерево удалено");

return 0;

}

Tree\* BuildTree(int\* arr, int start, int end, Tree\* root) // Построение взвешенного дерева

{

if (start > end) return 0;

int mid = (start + end) / 2; // Индекс элемента в середине массива

printf("%d ", arr[mid]);

root = AddNode(root, arr[mid]); // Добавляем корневой узел

root->left = BuildTree(arr, start, mid - 1, root->left); // Добавляем узел в левую ветвь

root->right = BuildTree(arr, mid + 1, end, root->right); // Добавляем узел в правую ветвь

return root;

}

Tree\* NewNode(int value) // Создание нового узла

{

Tree\* node = new Tree; // Создание нового узла

node->value = value; // Задаем значение узлу

node->left = node->right = 0; // Обнуляем указатели на дочерние элементы

return node; // Возвращаем узел

}

Tree\* AddNode(Tree\* node, int value) // Добавление узла в дерево

{

if (node == 0) return NewNode(value); // Если узла нет, то создаем его

if (value < node->value) // Если значение меньше, чем у узла, то идем рекурсивно влево

node->left = AddNode(node->left, value);

else // Иначе вправо

node->right = AddNode(node->right, value);

return node; // Возвращаем корень дерева

}

Tree\* DeleteNode(Tree\* root, int value) // Удаление узла из дерева

{

if (root == 0) return root; // Если узла нет, то ничего не делаем

if (value < root->value) // Если искомое значение меньше, то идем влево

root->left = DeleteNode(root->left, value);

else

{

if (value > root->value) // Иначе идем вправо

{

root->right = DeleteNode(root->right, value);

}

else // Если нашли узел по значению

{

if (root->left == 0) // Есть нет поддерева слева, то правое поддерево перемещаем на место узла

{

Tree\* temp = root->right;

delete root;

return temp;

}

else if (root->right == 0) // Если нет поддерева справа, то левое поддерево перемещаем на место узла

{

Tree\* temp = root->left;

delete root;

return temp;

}

// Если есть оба поддерева, то ищем минимальное значение в правом поддереве, чтобы не нарушить структуру дерева

Tree\* temp = MinValueNode(root->right); // Получаем указатель на искомый узел

root->value = temp->value; // Перемещаем его на место удаляемого

root->right = DeleteNode(root->right, temp->value); // Удаляем перенесенный узел

}

}

return root; // Возвращаем корень дерева

}

Tree\* FindNode(Tree\* node, int value) // Поиск узла в дереве

{

if (node == NULL) return 0; // Если узла нет, то возвращаем 0

if (node->value == value) return node; // Если нашли, то возвращаем узел

if (value < node->value) // Если искомое значение меньше значения узла, то идем влево

{

FindNode(node->left, value);

}

else // Иначе идем вправо

{

FindNode(node->right, value);

}

}

Tree\* DeleteTree(Tree\* root) // Удаление всего дерева

{

if (root == 0) return 0; // Если нет узла, то ничего не делаем

DeleteTree(root->left); // Рекурсивно удаляем поддеревья, а затем сам узел

DeleteTree(root->right);

delete root;

return 0;

}

Tree\* MinValueNode(Tree\* root) // Поиск минимального значения в дереве

{

Tree\* node = root; // Итератор для перемещения по дереву

while (node && node->left) // Идем по левой стороне, пока не дойдем до листа

node = node->left;

return node;

}

void StraightPrint(Tree\* node) // Прямой обход дерева

{

if (node != 0) // Если узел существует, то сначала выводим его, затем левое поддерево и наконец правое поддерево

{

printf("%d ", node->value);

StraightPrint(node->left);

StraightPrint(node->right);

}

}

void ReversePrint(Tree\* node) // Обратный обход дерева

{

if (node != 0) // Если узел существует, то сначала выводим левое поддерево, затем правое поддерево и наконец его

{

ReversePrint(node->left);

ReversePrint(node->right);

printf("%d ", node->value);

}

}

void IncPrint(Tree\* node) // Обход в порядке возрастания элементов

{

if (node != 0) // Если узел существует, то сначала выводим левое поддерево, затем его и наконец правое поддерево

{

IncPrint(node->left);

printf("%d ", node->value);

IncPrint(node->right);

}

}

void quickSort(int\* arr, int first, int last) // Быстрая сортировка

{

if (first < last)

{

int left = first, right = last, middle = arr[(left + right) / 2];

do

{

while (arr[left] < middle) left++;

while (arr[right] > middle) right--;

if (left <= right)

{

swap(arr[left], arr[right]);

left++;

right--;

}

} while (left <= right);

quickSort(arr, first, right);

quickSort(arr, left, last);

}

}

int CountNodes(Tree\* node) // Посчитать количество узлов

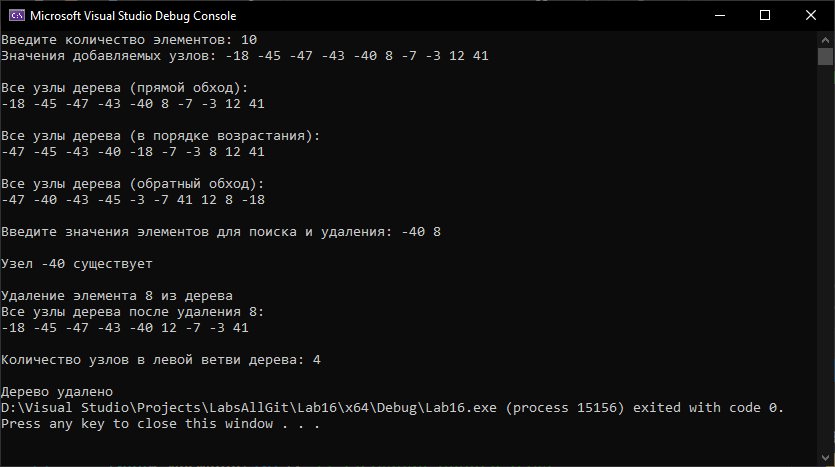
{

if (!node) return 0; // Если узла нет, то возвращаем 0

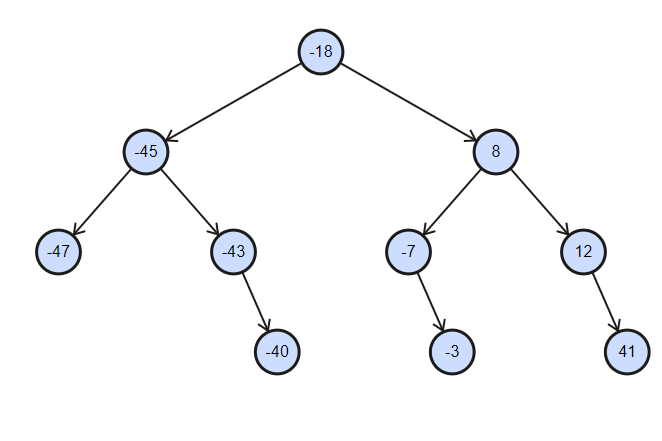
return CountNodes(node->left) + CountNodes(node->right) + 1; // Рекурсивно считаем узлы в поддеревьях + сам узел

}

***Результат работы программы:***



До удаления



После удаления

