**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Математического обеспечения и применения ЭВМ**

**ОТЧЕТ**

**по Лабораторной работе №5  
По курсу алгоритмы и структуры данных**

Тема: **Случайные бинарные деревья поиска**

**Вариант 10**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 6383 |  | Гомонова А.А. |
| Преподаватель |  | Шолохова О.М. |

Санкт-Петербург

2017

**Цель работы.**

Знакомство с моделью случайных бинарных деревьев поиска. Построение алгоритмов по его реализации и обработке.

# Постановка задачи.

Необходимо задать случайное бинарное дерево поиска путем вставки и исключения случайных чисел, генерируемых в программе или вводимых пользователем с клавиатуры и продемонстрировать поиск в полученном бинарном дереве заданного числа.

Действия:

* По заданному файлу F (типа file of Elem), все элементы которого различны, построить БДП определенного типа;
* Для построенного БДП проверить, входит ли в него элемент е типа Elem, и если входит, то удалить элемент е из дерева поиска.

# Ход работы.

## **Анализ.**

В исходных данных имеем последовательность ключей узлов. Мы должны, беря последовательно эти ключи добавлять их в наше дерево. Вставка происходит следующим образом: первый элемент последовательности записывается в корень. Последующий элемент сравниваем со значением корня, если данный элемент больше значения корня, записываем его в правое поддерево, если меньше- в левое поддерево. Последующие числа сравниваем со значением предыдущего узла. Действия совершаются до последнего элемента.

Далее мы можем выполнять действия вставки и исключения. По построенному случайному дереву можно проверить, входит ли в него заданный элемент, если не входит, можно добавить, если входит, можно удалить.

## **Формальная постановка задачи.**

### **Исходные данные.**

Последовательность ключей узлов случайного бинарного дерева поиска.

### **Результирующие данные.**

Построенное случайное бинарного дерева поиска. Возможность удаления и вставки элемента.

## **Алгоритм работы программы.**

Программа выполняет следующую последовательность действий:

* Запрашивает входную последовательность

Предусмотрен выбор ввода последовательности:

-с клавиатуры

-с файла

-ввод произвольных чисел

* Обрабатывает входную последовательность и строит результирующее дерево
* Выводит конечное дерево на экран
* Выполняет выбранные действия: вставка и исключение элемента.

## **Спецификация программы.**

### **Входные данные.**

1. Ограничения на входные данные:

Каждый из ключей должен быть представлен числом, входящим в пределы переменной *int*.

1. Место и форма предоставления входных данных

По выбору пользователя ввод данных можно осуществлять с консоли, с файла или это могут быть случайные числа, подобранные программой.

* 1. **Выходные данные.**

Результат работы выводиться в программе в виде дерева.

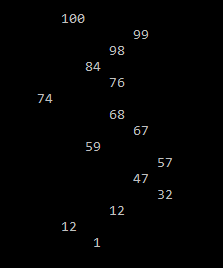


Рисунок 1. Пример построенного случайного бинарного дерева поиска для последовательности «12 1 12 32 47 57 59 67 68 74 76 84 98 99 100»

## **Реализация бинарного дерева.**

Для хранения данных используются следующая структура *Node*:

|  |  |  |
| --- | --- | --- |
| **Имя переменной** | **Тип переменной** | **Назначение** |
| *value* | *Int* | Ключ узла |
| *Size* | *Int* | Количество узлов в поддеревьях узла |
| *Left* | *Node\** | Указатель на левое поддерево |
| *Right* | *Node\** | Указатель на правое поддерево |

Таблица 1 — Описание переменных структуры *Node*

## **Описание функций для работы бинарным деревом.**

1. *int getsize(node\* p);*

Возвращает *size* узла *node\* p*

1. *void fixsize(node\* p);*

Пересчитывает size узла *node\* p*

1. *node\* rotateright(node\* p);*

Совершает правый поворот бинарного дерева *node\* p*

1. *node\* rotateleft(node\* p);*

Совершает левый поворот бинарного дерева *node\* p*

1. *node\* insertroot(node\* p, int k);*

Совершает вставку k узла в корень дерева *node\* p*

1. *node\* insertrandom (node\* p, int k, int &t);*

Строит случайное бинарное дерево *node\* p* поиска с рандомизацией

1. *bool isNull(binTree);*

Проверяет узел *binTree* на пустоту

1. *int RootBT (binTree);*

Возвращает ключ корня, передаваемого дерева *binTree*

1. *binTree Left (binTree);*

Возвращает левое поддерево текущего корня *binTree*

1. *binTree Right (binTree);*

Возвращает правое поддерево текущего корня *binTree*

1. *int CountTreeEl(binTree b);*

Возвращает количество ненулевых элементов текущего корня *binTree*

1. *void displayBT(binTree, int);*

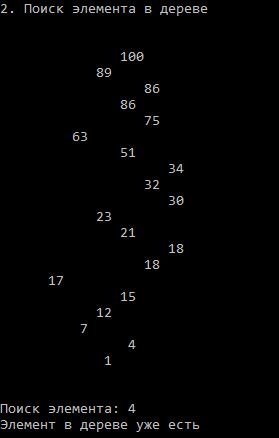
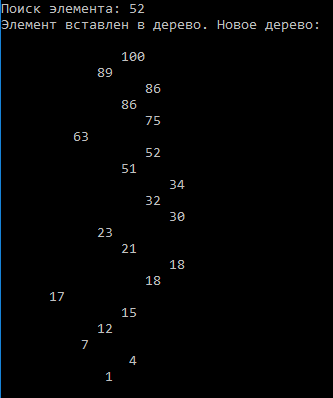
Для вывода бинарного дерева на консоль

1. *bool searchAndElement(binTree, int);*

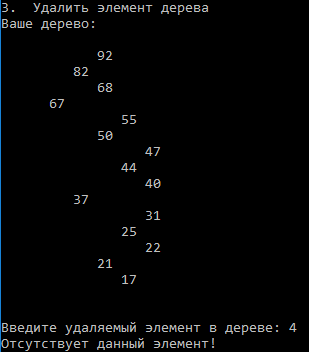
Поиск введенного элемента дерева *binTree*

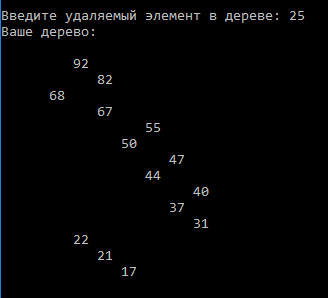
## **Примеры работы программы.**

1. Пример №1 – Вставка элемента

Входные данные: «63 23 7 1 30 21 89 12 17 86 86 100 34 18 51 32 18 4 75 15»

1. Пример №2 – Исключение элемента

Входные данные: «67 50 21 37 17 47 55 68 44 25 22 40 31 82 92»



# Вывод.

В ходе выполнения данной лабораторной работы была изучена модель случайного бинарного дерева поиска. Также были построены алгоритмы по его реализации и обработке.

# ПРИЛОЖЕНИЕ А

**ИСХОДНЫЙ КОД BINTREE.H**

#include <iostream>

namespace binTree\_modul

{

struct node

{

int value;

int size;

node\* left;

node\* right;

node(int k) { value = k; left = right = 0; size = 1; }

};

typedef node \*binTree;

int getsize(node\* p);

void fixsize(node\* p);

node\* rotateright(node\* p);

node\* rotateleft(node\* q);

node\* insertroot(node\* p, int k);

node\* insertrandom(node\* p, int k, int &t);

bool isNull(binTree);

int RootBT(binTree);

binTree Left(binTree);

binTree Right(binTree);

int CountTreeEl(binTree);

void displayBT(binTree, int);

bool searchAndElement(binTree, int);

}

# ПРИЛОЖЕНИЕ Б

**ИСХОДНЫЙ КОД BINTREE.CPP**

#include "bintree.h"

#include <iostream>

using namespace std;

namespace binTree\_modul

{

int getsize(node\* p)

{

if (!p) return 0;

return p->size;

}

void fixsize(node\* p)

{

p->size = getsize(p->left) + getsize(p->right) + 1;

}

node\* rotateright(node\* p)

{

node\* q = p->left;

if (!q) return p;

p->left = q->right;

q->right = p;

q->size = p->size;

fixsize(p);

return q;

}

node\* rotateleft(node\* q)

{

node\* p = q->right;

if (!p) return q;

q->right = p->left;

p->left = q;

p->size = q->size;

fixsize(q);

return p;

}

node\* insertroot(node\* p, int k)

{

if (!p)

return new node(k);

if (k<p->value)

{

p->left = insertroot(p->left, k);

return rotateright(p);

}

else

{

p->right = insertroot(p->right, k);

return rotateleft(p);

}

}

node\* insertrandom(node\* p, int k, int &t)

{

if (!p)

{

return new node(k);

t = 0;

}

int random = rand() % 10;

if (!random)

{

t = 1;

return insertroot(p, k);

}

if (p->value>k)

{

p->left = insertrandom(p->left, k, t);

t = 0;

}

else

{

p->right = insertrandom(p->right, k, t);

t = 0;

}

fixsize(p);

return p;

}

bool isNull(binTree b)

{

return (b == NULL);

}

int RootBT(binTree b)

{

if (b == NULL) { exit(1); }

else return b->value;

}

binTree Left(binTree b)

{

if (b == NULL) { exit(1); }

else return b->left;

}

binTree Right(binTree b)

{

if (b == NULL) { exit(1); }

else return b->right;

}

int CountTreeEl(binTree b)

{

int c = 1;

if (b == NULL)

return 0;

else

{

c += CountTreeEl(Left(b));

c += CountTreeEl(Right(b));

return c;

}

}

void displayBT(binTree b, int n)

{

if (b != NULL) {

displayBT(Right(b), n + 1);

for (int i = n + 1; i > 0; i--)

printf(" ");

printf("%2.d\n", b->value);

displayBT(Left(b), n + 1);

}

}

bool searchAndElement(binTree b, int x)

{

if (x == b->value)

return true;

if (x < b->value)

{

if (Left(b) == NULL)

{

b->left = new node(x);

return false;

}

return searchAndElement(Left(b), x);

}

if (x > b->value)

{

if (Right(b) == NULL)

{

b->right = new node(x);

return false;

}

return (searchAndElement(Right(b), x));

}

}

}

# 

# ПРИЛОЖЕНИЕ В

**ИСХОДНЫЙ КОД main.cpp**

#include "bintree.h"

#include <stdlib.h>

#include <locale.h>

#include <fstream>

#include <vector>

#include <sstream>

#include <iostream>

#include <time.h>

using namespace std;

using namespace binTree\_modul;

binTree b;

vector<int> numbers;

void readFromFileBinTree()

{

char path[256];

cout << "Введите пункт к файлу: ";

cin >> path;

ifstream file(path);

if (!file.is\_open())

{

cout << "Ошибка открытия файла!\n";

return;

}

else

{

cout << "\nВведенные значения:\n";

numbers.clear();

while (!file.eof())

{

char value[80];

file >> value;

cout << value << " ";

numbers.push\_back(atoi(value));

}

cout << endl;

}

}

void readFromKeyboardBinTree()

{

int size;

cout << "Введите количество чисел: ";

cin >> size;

numbers.clear();

for (int i = 0; i < size; i++)

{

cout << "Введите " << i + 1 << " число: ";

int value;

cin >> value;

numbers.push\_back(value);

}

cout << "\nВведенные значения:\n";

for (int i = 0; i < numbers.size(); i++)

{

cout << numbers[i] << " ";

}

cout << endl;

}

void generateRandomNumbers()

{

int size;

cout << "Введите количество чисел: ";

cin >> size;

numbers.clear();

cout << "\nВведенные значения:\n";

for (int i = 0; i < size; i++)

{

numbers.push\_back(rand() % 100 + 1);

}

for (int i = 0; i < numbers.size(); i++)

{

cout << numbers[i] << " ";

}

cout << endl;

}

void generateBinTree()

{

b = NULL;

int t = 0;

for (int i = 0; i<numbers.size(); i++)

{

b = insertrandom(b, numbers[i], t);

}

}

void generateDataBinTree()

{

int decision1 = 1;

while (decision1 != 4)

{

system("cls");

cout << " #################################################" << endl;

cout << " # #" << endl;

cout << " # Меню #" << endl;

cout << " # #" << endl;

cout << " # 1. Ввести данные из файла #" << endl;

cout << " # 2. Ввести данные с клавиатуры #" << endl;

cout << " # 3. Создать произвольное дерево #" << endl;

cout << " # 4. Вернуться к предыдущему пункту #" << endl;

cout << " # #" << endl;

cout << " #################################################" << endl << endl;

cout << " Выберите: ";

cin >> decision1;

switch (decision1)

{

case 1:

system("cls");

cout << "1. Ввести данные из файла\n";

readFromFileBinTree();

generateBinTree();

system("pause");

break;

case 2:

system("cls");

cout << "2. Ввести данные с клавиатуры\n";

readFromKeyboardBinTree();

generateBinTree();

system("pause");

break;

case 3:

system("cls");

cout << "3. Ввести данные с клавиатуры \n";

generateRandomNumbers();

generateBinTree();

system("pause");

break;

case 4:

break;

default:

system("cls");

cout << "Нет такого варианта! Повторите попытку.\n";

system("pause");

break;

}

}

}

void find\_add()

{

while (true)

{

int x = 0;

cout << "Поиск элемента: ";

cin >> x;

if (searchAndElement(b, x))

cout << "Элемент в дереве уже есть\n\n";

else

{

cout << "Элемент вставлен в дерево. Новое дерево:\n\n";

numbers.push\_back(x);

displayBT(b, 1);

cout << endl << endl;

}

int decision = 0;

while (decision == 0)

{

cout << "Завершить? [Y/N]: ";

char c;

cin >> c;

if (c == 'Y' || c == 'y')

{

return;

}

else

if (c == 'N' || c == 'n')

{

break;

}

else

{

decision = 0;

}

}

}

}

void printBinTree()

{

if (numbers.size() == 0)

{

cout << "Данные отсутствуют! Необходимо добавить данные прежде чем выводить дерево.\n";

system("pause");

return;

}

cout << endl << endl;

displayBT(b, 1);

cout << endl << endl;

find\_add();

system("pause");

}

void deleteElemntInBinTree()

{

if (numbers.size() == 0)

{

cout << "Данные отсутствуют! Необходимо добавить данные прежде чем выводить дерево.\n";

system("pause");

return;

}

cout << "Ваше дерево:\n\n";

displayBT(b, 1);

cout << endl << endl;

cout << "Введите удаляемый элемент в дереве: ";

int x;

cin >> x;

bool flag = false;

for (int i = 0; i < numbers.size(); i++)

{

if (numbers[i] == x)

{

numbers[i] = numbers[numbers.size() - 1];

numbers.erase(numbers.begin() + numbers.size() - 1);

flag = true;

break;

}

}

if (!flag)

{

cout << "Отсутствует данный элемент!\n";

system("pause");

return;

}

b = NULL;

int t = 0;

for (int i = 0; i<numbers.size(); i++)

{

b = insertrandom(b, numbers[i], t);

}

cout << "Ваше дерево:\n\n";

displayBT(b, 1);

cout << endl << endl;

system("pause");

}

int main()

{

srand(time(NULL));

setlocale(0, "Russian");

int decision = 1;

while (decision != 5)

{

system("cls");

cout << " #################################################" << endl;

cout << " # #" << endl;

cout << " # Меню #" << endl;

cout << " # #" << endl;

cout << " # 1. Создать СБД поиска #" << endl;

cout << " # 2. Поиск элемента в дереве #" << endl;

cout << " # 3. Удалить элемент дерева #" << endl;

cout << " # 4. Очистка СБД поиска #" << endl;

cout << " # 5. Выход #" << endl;

cout << " # #" << endl;

cout << " #################################################" << endl << endl;

cout << " Выберите: ";

cin >> decision;

switch (decision)

{

case 1:

system("cls");

cout << "1. Создание случайного бинарного дерева поиска\n\n";

generateDataBinTree();

break;

case 2:

system("cls");

cout << "2. Поиск элемента в дереве \n";

printBinTree();

break;

case 3:

system("cls");

cout << "3. Удалить элемент дерева\n";

deleteElemntInBinTree();

break;

case 4:

system("cls");

cout << "4. Очистка СБД поиска \n";

numbers.clear();

b = NULL;

cout << "Выполнено успешно!\n";

system("pause");

break;

case 5:

break;

default:

system("cls");

cout << "Нет такого варианта! Повторите попытку.\n";

system("pause");

break;

}

}

return 0;

}