**747МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе № 5**

**по дисциплине «Программирование»**

**Тема:**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6383 |  | Быков И. В. |
| Преподаватель |  | Шолохова О.М. |

Санкт-Петербург

2017

1. **Цель работы.**

Познакомиться с такой часто используемой на практике, особенно при решении задач кодирования и поиска, нелинейной структурой данных, как бинарное дерево, способами её представления и реализации, получить навыки решения задач обработки бинарных деревьев.

1. **Постановка задачи.**

Вариант 1б.

Задано бинарное дерево b типа BT с типом элементов Elem. Для введенной пользователем величины E (var E: Elem) определить число вхождений элемента Е в дерево b.

1. **Основные теоретические положения.**

Бинарное дерево － это конечное множество узлов, которое либо пусто, либо состоит из корня и двух непересекающихся бинарных деревьев, называемых правым поддеревом и левым поддеревом.

Скобочное представление бинарного дерева (БД):

< БД > ::= < пусто > | < непустое БД >,

< пусто > ::= Λ,

< непустое БД > ::= ( < корень > < БД > < БД > ).

Возможна ссылочная реализация абстрактного типа данных (АТД) «бинарное дерево» как в связанной памяти, так и на базе вектора. Каждый узел бинарного дерево (BT) рассматривается как корень поддерева, которому сопоставляется запись из трех полей: одно из них содержит значение узла, а два других являются указателями на левое и правое поддеревья.

Базовые операции бинарного дерева задаются набором функций:

NullBT (t: BinT): Boolean;

RootBT (t: BinT): Elem;

LeftBT (t: BinT): BinT;

RightBT (t: BinT): BinT;

ConsBT (e: Elem; LS, RS: BinT): BinT.

1. **Спецификация программы.**

Программа предназначена для нахождения количества вхождений элемента в заданное дерево.

Программа написана на языке C++ с использованием компилятора mingw32- g++. Входными данными для программы являются:

строка – прямая запись бинарного дерева (задается в файле)

символ типа char – элемент, число вхождений которого ищем

Выходные данные:

число типа int – количество вхождений элемента.

1. **Пример диалога с пользователем.**

Enter the string: 234234234

Tree:

3

9

3

6

3

Code: 010110101101011

Decode: 234234234

Repeat? 1 - yes; 0 - no. 1

Enter the string: 22222222

Tree:

8

Code: 00000000

Decode: 22222222

Repeat? 1 - yes; 0 - no. 1

Enter the string: 78348950376786230567602264872650350734нгцшкншолва77683

Tree:

8

15

7

26

6

11

5

54

5

13

4

8

4

28

3

7

2

4

2

15

1

2

1

4

1

2

1

8

1

2

1

4

1

2

1

Code: 00010001011001001111111011011010000001000100001101001001110110010000010111

01010100011100100000101000110110110101011011000010110011010111110111010110111110

001101011011111011111001111101111100000000001100010

Decode: 78348950376786230567602264872650350734нгцшкншолва77683

Repeat? 1 - yes; 0 - no. 1

Enter the string: ф

Tree:

1

Code: 0

Decode: ф

Repeat? 1 - yes; 0 - no.

1. **Реализация.**

Структура Бинарное дерево:

struct node {

base info;

node \*lt;

node \*rt;

// constructor

node () {lt = NULL; rt = NULL;}

};

typedef node \*binTree;

Базовые функции:

bool isNull(binTree);

base RootBT (binTree);

binTree Left (binTree);

binTree Right (binTree);

binTree ConsBT(const base &x, binTree &lst, binTree &rst);

void destroy (binTree&);

binTree enterBT (std::istream &infile);

void outBT(binTree b);

unInt sizeBT (binTree b);

Функция поиска количества вхождений:

int count(binTree b, base &x)

/Функция поиска количества вхождений элемента х в дереве b

Предусловие

b – бинарное поддерево

x – элемент типа char – искомый

Возвращает cnt – количество вхождений элемента x

1. **Тестирование.**

|  |  |  |
| --- | --- | --- |
| a// | a | 1 |
| a// | b | 0 |
|  | d | 0 |
| a/a/a// | a | 3 |
| a/b/c// | a | 1 |
| a/b/c// | b | 1 |
| a/b/c// | c | 1 |
| aa/bb/db// | a | 2 |
| aa/bb/db// | b | 3 |

**Вывод****.**

В результате выполнения лабораторной работы мною были получены основные сведения о бинарных деревьях. Мною были изучены базовые функции обработки бинарных деревьев.

**Приложение А. Исходный код.**

**main.cpp**

// Быков И. В. лабораторная работа №4. Поиск количества вхождений элемента в заданном дереве.

// Дерево вводится из файла или с экрана в КЛП форме.

#include "BTree.h"

using namespace binTree\_modul;

int readAction();

int main ()

{

setlocale(LC\_ALL, "Russian");

int flag, format, cnt;

ifstream in("KLP.txt");

binTree b;

base elem;

cout << "Выберите формат работы:\n1 - чтение из файла.\n2 - работа в консоли\nother - выход.\n" << endl;

cin >> format;

switch (format)

{

case 1:

if (!in) cerr << "Не найден файл input.txt\n";

cout << "Чтение из input.txt..." << endl;

b = enterBT(in);

cout << "Бинарное дерево в КЛП-представлении: " << endl;

outBT(b);

cout << endl << "Введите элемент: ";

cin >> elem;

cnt = count(b, elem);

cout << "Элемент " << elem << " встречается в дереве " << cnt << " раз.\n";

destroy(b);

break;

case 2:

while (true)

{

switch (readAction())

{

case 1:

cout << "Введите дерево: ";

b = enterBT(cin);

break;

case 2:

cout << "Введите элемент: ";

cin >> elem;

cnt = count(b, elem);

cout << "Элемент " << elem << " встречается в дереве " << cnt << " раз.\n";

break;

case 3:

destroy(b);

break;

case 4:

cout << "Бинарное дерево в КЛП-представлении: " << endl;

outBT(b);

cout << endl;

break;

case 5:

return 0;

}

cout << "Продолжить? 1 - да; 0 - нет. ";

cin >> flag;

cout << endl;

if (flag == 0)

return 0;

}

break;

}

destroy (b);

return 0;

}

int readAction()

{

while (true)

{

cout <<"Выберите действие:\n1 - ввод дерева;\n2 - посчитать количество вхождений элемента;\n3 - удалить дерево;\n4 - вывод дерева\n5 -выход.\n";

int action;

cin >> action;

if (action > 5 || action < 1)

{

printf("Ощибка!\n");

continue;

}

return action;

}

}

**BTree.cpp**

#include "BTree.h"

namespace binTree\_modul

{

//-------------------------------------

binTree Create()

{ return NULL;

}

//-------------------------------------

bool isNull(binTree b)

{ return (b == NULL);

}

//-------------------------------------

base RootBT (binTree b) // для непустого бин.дерева

{ if (b == NULL) { cerr << "Error: RootBT(null) \n"; exit(1); }

else return b->info;

}

//-------------------------------------

binTree Left (binTree b) // для непустого бин.дерева

{ if (b == NULL) { cerr << "Error: Left(null) \n"; exit(1); }

else return b ->lt;

}

//-------------------------------------

binTree Right (binTree b) // для непустого бин.дерева

{ if (b == NULL) { cerr << "Error: Right(null) \n"; exit(1); }

else return b->rt;

}

//-------------------------------------

binTree ConsBT(const base &x, binTree &lst, binTree &rst)

{ binTree p;

p = new node;

if ( p != NULL) {

p ->info = x;

p ->lt = lst;

p ->rt = rst;

return p;

}

else {cerr << "Memory not enough\n"; exit(1);}

}

//-------------------------------------

void destroy (binTree &b)

{ if (b != NULL) {

destroy (b->lt);

destroy (b->rt);

delete b;

b = NULL;

}

}

//---------------------------------------

binTree enterBT (istream &infile)

{

if (infile.eof())

{

cerr << "Error!\n";

exit(1);

}

char ch;

binTree p, q;

infile >> ch;

if (ch=='/') return NULL;

else {p = enterBT(infile); q = enterBT(infile); return ConsBT(ch, p, q);}

}

//---------------------------------------

void outBT(binTree b)

{

if (b!=NULL) {

cout << RootBT(b);

outBT(Left(b));

outBT(Right(b));

}

else cout << '/';

}

//---------------------------------------

void displayBT (binTree b, int n)

{

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

cout << " ";

cout << "-" << RootBT(b) << endl;

if (!isNull(Left(b)))

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

if (!isNull(Right(b)))

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

/\* if (b!=NULL) {

cout << ' ' << RootBT(b);

if(!isNull(Right(b))) {displayBT (Right(b),n+1);}

else cout << endl; // вниз

if(!isNull(Left(b))) {

for (int i=1;i<=n;i++) cout << " "; // вправо

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

}\*/

// else {};

}

//---------------------------------------

unInt hBT (binTree b)

{

if (isNull(b)) return 0;

else return max (hBT (Left(b)), hBT(Right(b))) + 1;

}

//---------------------------------------

unInt sizeBT (binTree b)

{

if (isNull(b)) return 0;

else return sizeBT (Left(b))+ sizeBT(Right(b)) + 1;

}

//---------------------------------------

void printKLP (binTree b)

{

if (!isNull(b)) {

cout << RootBT(b);

printKLP (Left(b));

printKLP (Right(b));

}

}

//---------------------------------------

int count(binTree b, base x)

{

int cnt = 0;

if(!isNull(b))

{

if (RootBT(b) == x)

cnt++;

cnt = cnt + count(Left(b), x) + count(Right(b), x);

}

return cnt;

}

BTree.h

#ifndef BTREE\_H\_INCLUDED

#define BTREE\_H\_INCLUDED

#include <iostream>

#include <cstdlib>

#include <fstream>

typedef unsigned int unInt;

using namespace std ;

namespace binTree\_modul

{

//-------------------------------------

typedef char base;

struct node {

base info;

node \*lt;

node \*rt;

// constructor

node () {lt = NULL; rt = NULL;}

};

typedef node \*binTree;

binTree Create(void);

bool isNull(binTree);

base RootBT (binTree);

binTree Left (binTree);

binTree Right (binTree);

binTree ConsBT(const base &x, binTree &lst, binTree &rst);

void destroy (binTree&);

binTree enterBT (std::istream &infile);

void outBT(binTree b);

void displayBT (binTree b, int n);

unInt hBT (binTree b);

unInt sizeBT (binTree b);

void printKLP (binTree b);

int count(binTree b, base x);

int path\_length(binTree b, base x);

} // end of namespace binTree\_modul

#endif // BTREE\_H\_INCLUDED