МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

КАФЕДРА МО ЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Алгоритмы и структуры данных»

Тема: Бинарное дерево поиска

Студент гр. 7383 _____ Ласковенко Е.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2018

Оглавление

Цель работы

Познакомиться со структурой и реализацией бинарного дерева поиска, получить навыки программирования функций, использующих бинарное дерево поиска, на языке программирования C++.

Задача

Требуется:

1) По заданному файлу F (типа file of *Elem*), все элементы которого различны, построить случайное БДП с рандомизацией;
2) Для построенного БДП проверить, входит ли в него элемент *e* типа *Elem*, и если не входит, то добавить элемент *e* в дерево поиска.

Реализация задачи

Описание функций, классов и переменных. Описание работы алгоритма

Функция **int main()**:

Является головной функцией, в которой происходит считывание данных из файла (название файла вводит пользователь в консоли) или с консоли. Если файла с введенным названием нет или же файл невозможно открыть на чтение, то в консоль выводится ошибка, а программа снова требует ввести название файла на открытие. Далее объявляются необходимые переменные, описанные ниже, происходит вывод на экран содержимого файла. Если файл пуст, то выводится соответствующее предупреждение, если файл не пустой, то происходит вызов функций построения бинарного дерева поиска, вывода дерева в консоль. Далее пользователю предлагается ввести новое значение для добавления в бинарное дерево поиска в соответствии с заданием. Все вызовы находятся в цикле, выходом из которого является условие ввода пользователем пустой строки в консоли. Также в функции реализована обработка исключений — проверка наличия файла с данными и прочее.

Объявляемые переменные:

- string filename — строковая переменная, хранящая имя файла, с которого необходимо считать данные.
- string temp_str — строковая переменная, хранящая входные данные.
- filebuf file — файловый буфер, используемый при считывании входных данных с файла.
- stringbuf str_buf — строковый буфер, используемый при считывании входных данных с консоли.
- istream is_str — входной поток, в который помещаются входные данные.

- auto size — переменная, хранящая размер (количество символов) файла с входными данными.

Функция **BST<BASE>* read_stream(std::istream& is_str, BST<BASE>* root=nullptr)**:

Рекурсивная функция, принимающая на вход входной поток, с которого происходит считывание символа. Осуществляет вызов методов класса бинарного дерева для построения бинарного дерева поиска в соответствии с заданием. Возвращает корень бинарного дерева поиска.

Функция **void printBST(BST<BASE>* root, unsigned int l=0):**

Рекурсивная функция, принимающая на вход корень бинарного дерева поиска. Выводит на экран представление бинарного дерева, проходя по всем узлам. Обход — ПКЛ. В каждой строке выводятся знаки табуляций, количество которых равняется уровню узла.

Класс **class BST:**

Класс бинарного дерева поиска. Состоит из следующих полей: значения узла, значения ключа, значения размера узла, указателей на левый и правый узел. Для класса были реализованы следующие методы:

- BST() - конструктор класса. Инициализирует все поля нулевыми значениями.
- BASE value() - функция, возвращающая значение текущего узла.
- int key() - функция, возвращающая значение ключа.
- int size() - функция, возвращающая значение размера текущего узла.
- BST* left () - функция, возвращающая адрес левого узла.
- BST* right () - функция, возвращающая адрес правого узла.
- void set_lf(BST* lft) - функция, устанавливающая поле значения левого узла.

- void set_rt(BST* rgt) - функция, устанавливающая поле значения правого узла.
- void set_vl(BASE vl) - функция, устанавливающая поле значения текущего узла.
- void fix_size() - рекурсивная функция, пересчитывающая поля размеров для каждого узла дерева.
- bool find(int key, bool flag=false) — рекурсивная функция, возвращающая true, если входной ключ имеется в бинарном дереве. Иначе — false.
- void rebuild(BST<BASE>* el) — функция, перестраивающая дерево следующим образом: заменяет текущий узел входным узлом, текущий узел ставит левым/правым сыном входного узла в зависимости от значения этого узла.
- void add_rand(BASE val) — рекурсивная функция, добавляющая элемент в бинарное дерево в соответствии с заданием.
- ~BST() - декструктор класса. Рекурсивно освобождает память.

Тестирование программы

Процесс тестирования

Программа собрана в операционной системе Linux Mint 19, с использованием компилятора G++. В других ОС и компиляторах тестирование не проводилось.

Результаты тестирования

Тестовые случаи представлены в Приложении А.

По результатам тестирования было показано, что поставленная задача была выполнена.

Выводы

В ходе лабораторной работы были получены навыки работы с бинарным деревом поиска. В работе был реализован класс, который представляет собой бинарное дерево поиска на базе ссылок. Бинарное дерево поиска удобно использовать, при использовании рекурсивных функций.

ПРИЛОЖЕНИЕ А.

Тестовые случаи

Входные данные	Вывод	Верно?
cbdae c	Binary tree: # e # d # c # b # a # Enter an element: c # e # d # c # b # a # 	Да

cbdae h	Binary tree: # e # a # d # b # c # Enter an element: h # e # a # d # b # c # h #	Да

ПРИЛОЖЕНИЕ Б.

Код программы

1. Код **main.cpp:**

```
#include <iostream>
#include <istream>
#include <fstream>
#include <sstream>
#include <string>
#include <exception>

#include "bst.h"
#include "api.h"
#include "myexception.h"

using namespace std;

int main()
{
    string filename;
    string temp_str;
    filebuf file;
    stringbuf str_buf;
    istream is_str(&str_buf);

    int int_t;
    string t_str;
    char temp_ch;
    long size;

    while(true)
    {
        try
        {
            cout << "Enter \"1\" to input data from file." << endl
                 << "Enter \"2\" to input data from console."<< endl
                 << "Press ENTER to exit." << endl;
            getline(cin, t_str);
            if(t_str.empty())
                break;
            int_t = stoi(t_str);
            t_str.clear();

            switch(int_t)
            {
            case 0:
                break;
            case 1:
                cout << "Enter a name of the data-file:" << endl;
                getline(cin, filename);
                if(filename.empty())
                    throw new my_ex("Empty filename.");

                if(!file.open(filename, ios::in))
                {
                    file.close();
                    throw new my_ex("Incorrect filename.");
                }

                size = file.in_avail();
```

```
            cout << "File contains: ";
            for(auto c_size = 0; c_size<size; c_size++)
            {
                temp_ch=file.sbumpc();
                cout << temp_ch;
                if(temp_ch != ' ' && temp_ch != '\n')
                    temp_str.append(1, temp_ch);
            }
            cout << endl;
            file.close();
            if(!temp_str.size())
                throw "Input file is empty.";

            break;
        case 2:
            cout << "Enter an expression:" << endl;
            getline(cin, temp_str);
            if(temp_str.empty())
                throw new my_ex("Empty.");

            break;
        default:
            throw new my_ex("Wrong integer.");
        }

        str_buf.str(temp_str);
        temp_str.clear();

        //Building binary tree:
        BST<char>* root;
        root = read_stream<char>(is_str);

        is_str.clear();

        cout << "Binary tree:" << endl;
        printBST(root);
        cout << endl;

        //Adding extra element:
        cout << "Enter an element:" << endl;
        char ch;
        cin >> ch;
        cin.ignore();
        srand(ch);
        if(!root->find(rand()))
            root->add_rand(ch);
        printBST(root);
        cout << endl;

        delete root;
    }
    catch (exception* ex)
    {
        cout << ex->what() << endl;
        temp_str.clear();
        is_str.clear();
    }
    catch (...)
    {
        cout << "It isn't an integer!" << endl;
        continue;
    }
```

```
    }

    return 0;
}
```

2. Код **bst.h:**

```
#include <iostream>
#include <cstdlib>

#ifndef BST_H
#define BST_H

template<typename BASE>
class BST
{
private:
    BASE val;
    int k;
    int sz;
    BST* lf;
    BST* rt;

    void rebuild(BST<BASE>* el)
    {
        BASE tmp_vl = this->value();
        this->set_vl(el->value());
        el->set_vl(tmp_vl);
        el->set_lf(this->left());
        el->set_rt(this->right());
        if(tmp_vl < this->value())
        {
            this->set_lf(el);
            this->set_rt(nullptr);
        }
        else
        {
            this->set_rt(el);
            this->set_lf(nullptr);
        }
    }
public:
    BST()
    {
        val = 0;
        k = -1;
        sz = 1;
        lf = nullptr;
        rt = nullptr;
    }

    BASE value()
    { return val; }

    int key()
    { return k; }

    int size()
    { return sz; }

    BST* left()
    { return lf; }
```

```
BST* right()
{ return rt; }

void set_lf(BST* lft)
{ this->lf = lft; }

void set_rt(BST* rgt)
{ this->rt = rgt; }

void set_vl(BASE vl)
{
    val = vl;
    srand(val);
    k = std::rand();
}

void fix_size()
{
    this->sz = 1;
    if(this->left())
    {
        this->left()->fix_size();
        this->sz += this->left()->size();
    }
    if(this->right())
    {
        this->right()->fix_size();
        this->sz += this->right()->size();
    }
}

bool find(int key)
{
    bool flag=false;
    if(this)
    {
        if(key == this->key())
            return true;

        flag += this->left()->find(key);
        flag += this->right()->find(key);
    }
    else
        return flag;
}

void add_rand(BASE val)
{
    BST<BASE>* bst_el = new BST<BASE>;
    bst_el->set_vl(val);

    srand(val);
    if(!this->find(rand()))
    {
        this->fix_size();
        srand(time(nullptr));
        if(rand()%(this->size()+1) == this->size())
                this->rebuild(bst_el);
        else
        {
            if(this->value() > bst_el->value())
```

```
                {
                    if(!this->left())
                        this->set_lf(bst_el);
                    this->left()->add_rand(val);
                }
                else
                {
                    if(!this->right())
                        this->set_rt(bst_el);
                    this->right()->add_rand(val);
                }
            }
        }
    }

    ~BST()
    {
        if(this->left()!=nullptr)
            delete lf;
        if(this->right()!=nullptr)
            delete rt;
    }
};

#endif // BST_H
```

3. Код **api.h:**

```
#include <iostream>
#include <istream>
#include <ctime>
#include "bst.h"

#ifndef API_H
#define API_H

template<typename BASE>
BST<BASE>* read_stream(std::istream& is_str, BST<BASE>* root=nullptr)
{
    BASE cur_el;

    if(is_str >> cur_el)
    {
        if(!root)
        {
            root = new BST<BASE>;
            root->set_vl(cur_el);
        }
        else
        {
            srand(cur_el);
            if(!root->find(rand()))
                root->add_rand(cur_el);
        }
        root->fix_size();
        root = read_stream<char>(is_str, root);
    }

    return root;
}

template<typename BASE>
```

```
void printBST(BST<BASE>* root, unsigned int l=0)
{
   if(!root)
   {
       for(unsigned int i=0; i<l; ++i)
           std::cout << '\t';
       std::cout << '#' << std::endl;
       return;
   }

   printBST(root->right(), l+1);

   for(unsigned int i=0; i<l; i++)
       std::cout << '\t';
   std::cout << root->value() << std::endl;

   printBST(root->left(), l+1);
}

#endif // API_H
```

4. Код **myexception.h:**

```
#include <exception>


#ifndef MYEXCEPTION_H
#define MYEXCEPTION_H


class my_ex : public std::exception
{
public:
    my_ex(const char* exp) noexcept
    {
        msg = exp;
    }
    virtual const char* what() const noexcept
    {
        return msg;
    }
    virtual ~my_ex(){}
private:
    const char* msg;
};


#endif // MYEXCEPTION_H
```

5. Код **Makefile:**

```
all: main.cpp bst.h api.h myexception.h
      g++ main.cpp -o start
```

```
clean:
	rm start
```