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: Рандомизированные бинарные деревья поиска

Студент гр. 7383 _____ Лосев М. Л.

Преподаватель _____ Размочаева Н. В.

Санкт-Петербург
2018

СОДЕРЖАНИЕ

1. ЦЕЛЬ РАБОТЫ

Цель работы: познакомиться с рандомизированными бинарными деревьями поиска, реализовать их на языке C++, получить соответствующие навыки и знания.

Формулировка задачи: по заданному файлу F (file of elem), все элементы которого различны, построить рандомизированное бинарное дерево поиска. Записать в файл элементы построенного БДП в порядке их возрастания; вывести построенное БДП на экран.

2. РЕАЛИЗАЦИЯ ЗАДАЧИ

2.1 ИСПОЛЬЗУЕМЫЕ СТРУКТУРЫ ДАННЫХ

Рекурсивное определение рандомизированного бинарного дерева поиска:

1. Если T пусто, то оно является рандомизированным бинарным деревом поиска.
2. Если T непусто (содержит n вершин, n>0), то T — рандомизированное бинарное дерево поиска тогда и только тогда, когда его левое и правое поддеревья (L и R) оба являются RBST, а также выполняется соотношение P[size(L)=i]=1n,i=1..n.

class BST– структура, представляющая узел БДП, содержит в себе поля int key для хранения ключа, int size для хранения данного дерева с корнем в данном узле, BST* left, right для хранения указателей на правое и левое поддерево.

2.2 ИСПОЛЬЗУЕМЫЕ ФУНКЦИИ

BST rotateright (node* p)* – функция, делающая правый поворот вокруг узла p.

BST rotateleft(node* p)* – функция, делающая левый поворот вокруг узла p.

BST insertroot(BST* p, int k)* – вставка узла с ключом k в корень дерева.

BST insertrandom(BST* p, int k)* – функция, добавляющая узел с ключом k. Если ключ k больше (меньше) ключа рассматриваемого узла, то он вставляется вправо (влево) от этого узла, при надобности делается правый или левый поворот.

BST join(node *p, node *q)* – функция, получающая на вход два дерева, которые она объединяет.

BST remove(node* p, int k)* – функция получает на вход дерево и ключ узла. Удаляет узел с заданным ключом, объединяя его правое и левое

поддерево с помощью функции merge.

*BST *build_tree(string str)* – функция получает строку элементов входной последовательности, строит RBST, возвращает указатель на его корень.

*void printtree(BST <int> *treenode, int l)* – функция получает указатель на корень RBST, ключи которого целые числа, и отрисовывает его.

*void ascend_write_node(ofstream *outfile, BST <int> *tree)* – функция получает указатель на поток файлового вывода и выводит в него узел дерева.

*int ascend_write_tree(string filename, BST <int> *tree)* – функция имя файла вывода и выводит в него RBST.

void rand_init() – функция ничего не получает, инициализирует рандомайзер временем.

3. ТЕСТИРОВАНИЕ

3.1 ПРОЦЕСС ТЕСТИРОВАНИЯ

Программа собрана в операционной системе Ubuntu 18.04 компилятором GCC. В других ОС тестирование не проводилось.

3.1 Результаты тестирования

Тестовые случаи представлены в приложении Б. Ошибки выявить не удалось. По результатам тестирования было показано, что задача выполнена.

4 . ВЫВОД

В ходе работы была написана программа на языке C++, вычисляющая значение выражения, записанного в префиксной форме. Был получен опыт в использовании стека в ссылочной реализации.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл MakeFile:

```
all: main.o functions.o
	g++ main.o functions.o -o main
main.o: main.cpp bst_interf.h functions.h
	g++ -c main.cpp
functions.o: functions.cpp bst_interf.h functions.h
	g++ -c functions.cpp
```

Файл bst_interf.h:

```
#pragma once

#include <iostream>
#include <vector>
#include <cstring>
#include <fstream>

namespace losev_BST
{
	template <typename base>
	class BST{
	private:
		base key;
		int size;
		BST* right;
		BST* left;
	 public:
		BST(base k)
		{
			key = k; left = nullptr;
			right = nullptr;
			size = 1;
		}

		base Root(BST* b)
		{
			if (b == NULL)
				exit(1);
			else
				return b->key;
		}

		BST* Left(BST* b)
		{
			if (b == NULL) { exit(1); }
			else return b->left;
		}

		BST* Right(BST* b)
		{
			if (b == NULL) { exit(1); }
			else return b->right;
		}
```

```
            int getsize(BST* p) // îáðòêà äëÿ ïîëÿ size, ðàáîòàåò ñ ïóñòûìè
äåðåâüÿìè (t=NULL)
            {
                  if(!p)
                        return 0;
                  return p->size;
            }

            void fixsize(BST* p) // óñòàíîâëåíèå êîððåêòíîãî ðàçìåðà äåðåâà
            {
                  p->size = getsize(Left(p))+getsize(Right(p))+1;
            }

            BST* rotateright(BST* p) // ïðàâûé ïîâîðîò âîêðóã óçëà p
            {
                  BST* q = p->left;
                  if( !q ) return p;
                  p->left = q->right;
                  q->right = p;
                  q->size = p->size;
                  fixsize(p);
                  return q;
            }
            BST* rotateleft(BST* q) // ëåâûé ïîâîðîò âîêðóã óçëà q
            {
                  BST* p = q->right;
                  if( !p ) return q;
                  q->right = p->left;
                  p->left = q;
                  p->size = q->size;
                  fixsize(q);
                  return p;
            }

            BST* insertroot(BST* p, base k) // âñòàâêà íîâîãî óçëà ñ êëþ÷îì k â
êîðåíü äåðåâà p
            {
                  if (!p)
                        return new BST(k);
                  if (k < p->key)
                  {
                        p->left = insertroot(p->left, k);
                        return rotateright(p);
                  }
                  else
                  {
                        p->right = insertroot(p->right, k);
                        return rotateleft(p);
                  }
            }

            BST* insertrandom(BST* p, base k) // ðàíäîìèçèðîâàííàÿ âñòàâêà íîâîãî
óçëà ñ êëþ÷îì k â äåðåâî p
            {
                  if (!p) return new BST(k);
                  if (rand() % (getsize(p)+1) == getsize(p)){
                        //  cout<<"COMBS: "<<rand()%(getsize(p)+1) <<endl;
                        return insertroot(p, k);
                  }
                  if (p->key > k)
                        p->left = insertrandom(Left(p), k);
```

```
        else
                p->right = insertrandom(Right(p), k);
        fixsize(p);
        return p;
    }

BST* join(BST* p, BST* q) // îáúåäèíåíèå äâóõ äåðåâüåâ
{
        if( !p ) return q;
        if( !q ) return p;
        if( rand()%(p->size+q->size) < p->size )
        {
                p->right = join(p->right,q);
                fixsize(p);
                return p;
        }
        else
        {
                q->left = join(p,q->left);
                fixsize(q);
                return q;
        }
}

        BST* remove(BST* p, int k) // óäàëåíèå èç äåðåâà p ïåðâîãî íàéäåííîãî
óçëà ñ êëþ÷îì k
{
        if( !p ) return p;
        if( p->key==k )
        {
                BST* q = join(p->left,p->right);
                p=q;
                return p;
        }
        else if( k<p->key )
                p->left = remove(p->left,k);
        else
                p->right = remove(p->right,k);
        return p;
}

BST* Delete(BST* p)
{
        if (left)
                delete p->left;
        if (right)
                delete p->right;
        delete p;
        return p = NULL;
}

BST* find(BST* tree, base key)
{
        if (!tree)
                return NULL;
        if (key == tree->key)
                return tree;
        if (key < tree->key)
                return find(tree->left, key);
        else
                return find(tree->right, key);
```

```
		}
	};

}
```

Файл functions.cpp:

```
/* Функции Лосева */
#include "bst_interf.h"
#include "functions.h"

using namespace std;
using namespace losev_BST;

namespace losev_functions
{
	BST <int> *build_tree(string str)
	{
		int n;
		char* tok;
		BST <int> *b = NULL;

		char* arr = new char[str.size()+1];
		strcpy(arr, str.c_str());

		tok = strtok(arr, " ");

		while(tok){
			n = atoi(tok);
			if(b->find(b, n)){
				tok = strtok(NULL, " ");
			} // не добавляем число в дерево, если оно там уже есть
			if (tok){
				b = b->insertrandom(b, atoi(tok));
				tok = strtok(NULL, " ");
			} // проверка нужна, т.к. если число последовательности было
пропущено, то после него может идти конец строки.
		}

		delete[] arr;

		return b;
	}

	void printtree(BST <int> *treenode, int l){
		if(treenode==NULL){
		   for(int i = 0;i<l;++i)
				cout<<"\t";
			cout<<'#'<<endl;
			return;
		}
		printtree(treenode->Right(treenode), l+1);
		for(int i = 0; i < l; i++)
			cout << "\t";
		cout << treenode->Root(treenode) << endl;
		printtree(treenode->Left(treenode),l+1);
	}
```

```
	void ascend_write_node(ofstream *outfile, BST <int> *tree)
	{
		if (tree == NULL)
			return;
		ascend_write_node(outfile, tree->Left(tree));
		(*outfile) << tree->Root(tree) << ' ';
		ascend_write_node(outfile, tree->Right(tree));
	}


	int ascend_write_tree(string filename, BST <int> *tree)
	{
		if (tree == NULL){
			return 2;
		}

		ofstream fout;
		fout.open(filename, ofstream::trunc);
		if (fout.is_open()){
			ascend_write_node(&fout, tree);
			fout.close();
		}
		else {
			cerr << "Ошибка: файл вывода не открывается" << endl;
			return 1;
		}
		return 0;
	}
	void rand_init()
	{
		srand(time(NULL));
		// rand надо инициализировать, поскольку иначе он будет выдавать
		// одну и ту же последовательность псевдослучайных чисел при каждом
		// запуске программы. rand инициализируется временем, потому что
		// время в каждый момент времени различно, и поэтому последовательность
		// псевдослучайных чисел будет всегда различная.
	}
}
```

Файл functions.h:

```
#pragma once


#include <iostream>
#include <vector>
#include <cstring>
#include <fstream>
#include "bst_interf.h"


using namespace std;
using namespace losev_BST;
```

```
namespace losev_functions
{
	BST <int> *build_tree(string str);
	void printtree(BST <int> *treenode, int l);
	void ascend_write_node(ofstream *outfile, BST <int> *tree);
	int ascend_write_tree(string filename, BST <int> *tree);
	void rand_init();
}
```

Файл main.cpp:

```
#include <iostream>
#include <vector>
#include <cstring>
#include <fstream>
#include <cstdlib> // чтобы юзать rand
#include "bst_interf.h"
#include "functions.h"


using namespace std;
using namespace losev_BST;
using namespace losev_functions;


 int main()
 {
	int a = 0;
	string str;
	BST <int> *b = NULL;

	rand_init();

	cout << "Введите ключи: " << endl;
	getline(cin, str);

	if (!tok)
		cout << "Введенная строка пуста." << endl;
	else {
		cout << "Построение дерева... " << endl;
		b = build_tree(str);
		if (b)
			cout << "Дерево построено." << endl;
		else
			cout << "Дерево не построено." << endl;
	}


	cout << "Отрисовка дерева:" << endl;
	printtree(b,0);

	cout << "Запись в файл..." << endl;
	int f = ascend_write_tree("out.txt", b);
	if (f == 0) cout << "Запись завершена." << endl;
	else cout << "Запись не сделана." << endl;
```

```
        b = b->Delete(b);
        str.clear();

        return 0;
}
```

ПРИЛОЖЕНИЕ Б
ТЕСТОВЫЕ СЛУЧАИ

Тестовые случаи представлены на рисунках 1 – 3.

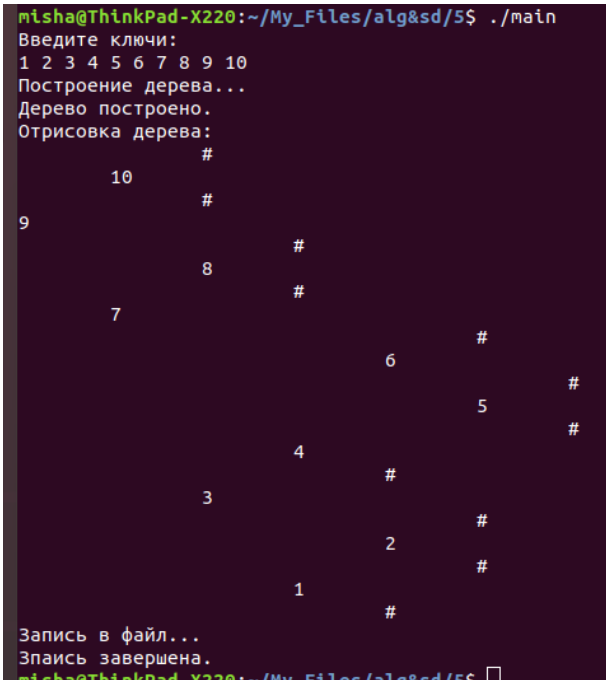

Рисунок 1.1. Тест 1. Консоль

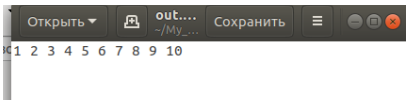

Рисунок 1.2. Тест 1. Файл вывода

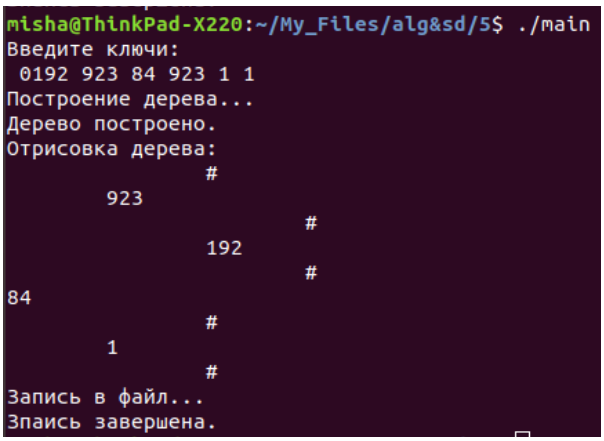

Рисунок 2.1. Тест 1. Консоль

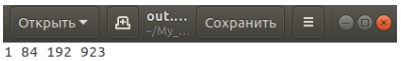

Рисунок 2.2. Тест 1. Файл вывода

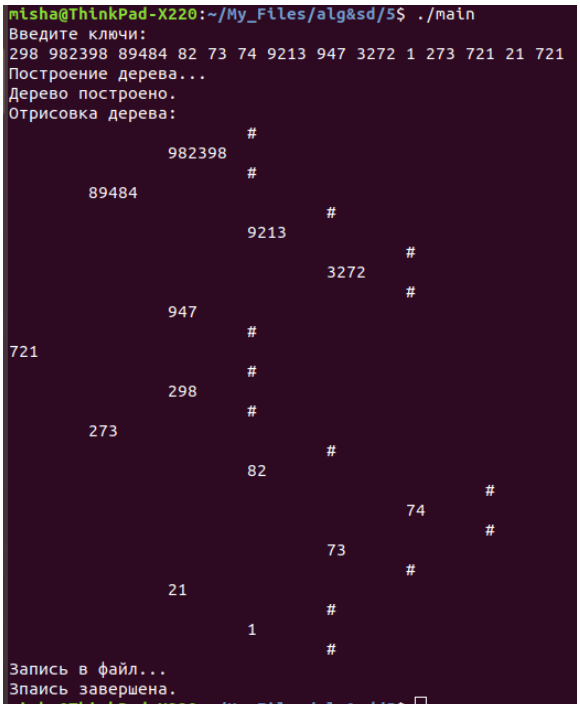

Рисунок 3.1. Тест 1. Консоль

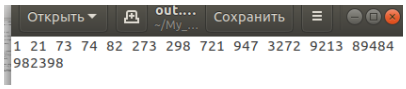

Рисунок 3.2. Тест 1. Файл вывода