МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

**КАФЕДРА ИИТ**

ОТЧЁТ

по лабораторной работе №4

**«Бинарные деревья поиска и хеш-таблицы»**

Выполнил:

Студент 1 курса

группы ПО-9

Харитонович Захар Сергеевич

Проверила:

Войцехович О. Ю.

Брест 2022

**Цель работы:** провести экспериментальное исследование эффективности бинарных

деревьев поиска и хеш-таблиц.

**Двоичное дерево поиска –** это двоичное дерево, в котором каждый узел имеет не более дочерних узлов и содержит ключ и значение. Ключи всех узлов левого поддерева меньше значения его ключа. Ключи всех узлов правого поддерева узла больше значения его ключа.



**Хеш-таблица** – это структура данных для хранения пар «ключ – значение», в которой доступ к элементам осуществляется по ключу. Ключи могут быть строками, числами, указателями. Хеш-таблицы позволяют в среднем за О(1) выполнять добавление, поиск и удаление элемента.



**Порядок выполнения работы**

*Задание*

Требуется реализовать на языке C две библиотеки для работы с бинарным деревом  
поиска (Binary search tree) и хеш-таблицей (Hash table). Ключом в обоих случаях  
является строка (char []), а значением целое число (int).

**Реализация**

*btree.h*

#ifndef TASK\_BSTREE\_H

#define TASK\_BSTREE\_H

#include <stdlib.h>

#include <string.h>

struct bstree {

char \* key;

int value;

struct bstree \*left;

struct bstree \*right;

};

struct bstree \*bstree\_create(char \* key, int value);

void bstree\_add(struct bstree \*tree, char \* key, int value);

struct bstree \*bstree\_lookup(struct bstree \*tree, char \* key);

struct bstree \*bstree\_min(struct bstree \*tree);

struct bstree \*bstree\_max(struct bstree \*tree);

#endif

*bstree.cpp*

#include "bstree.h"

struct bstree \*bstree\_create(char \* key, int value) {

struct bstree \*node;

node = (bstree \*)(malloc(sizeof(\*node)));

if (node != NULL) {

strcpy(node->key, key);

node->value = value;

node->left = NULL;

node->right = NULL;

}

}

void bstree\_add(struct bstree \*tree, char \* key, int value) {

struct bstree \*parent, \*node;

if (tree == NULL) return;

while (tree != NULL) {

if (strcmp(key, tree->key) < 0)

tree = tree->left;

else if (strcmp(key, tree->key) > 0)

tree = tree->right;

else

return;

}

node = bstree\_create(key, value);

if (strcmp(key, tree->key) < 0)

tree->left = node;

else

tree->right = node;

}

struct bstree \*bstree\_lookup(struct bstree \*tree, char \* key) {

while (tree != NULL) {

if (strcmp(key, tree->key) == 0) {

return tree;

} else if (strcmp(key, tree->key) < 0) {

tree = tree->left;

} else {

tree = tree->right;

}

}

return tree;

}

struct bstree \*bstree\_min(struct bstree \*tree) {

if (tree == NULL)

return NULL;

while (tree->left != NULL)

tree = tree->left;

return tree;

}

struct bstree \*bstree\_max(struct bstree \*tree) {

if (tree == NULL)

return NULL;

while (tree->right != NULL)

tree = tree->right;

return tree;

}

*hashtab.h*

#ifndef TASK\_HASHTAB\_H

#define TASK\_HASHTAB\_H

#include <stdlib.h>

#include <string.h>

#define HASHTAB\_SIZE 200000

#define HASHTAB\_MUL 31

struct listnode {

char \*key;

int value;

struct listnode \*next;

};

struct listnode \*hashtab[HASHTAB\_SIZE];

int hash(char \*key);

void hashtab\_init(struct listnode \*\*hashtab);

void hashtab\_add(struct listnode \*\*hashtab, char \*key, int value);

struct listnode \*hashtab\_lookup(struct listnode \*\*hashtab, char \*key);

#endif

*hashtab.cpp*

#include "hashtab.h"

int hash(char \*key) {

int h = 0;

char \*p;

for (p = key; \*p != '\0'; p++) {

h = h \* HASHTAB\_MUL + (int) \*p;

}

return h % HASHTAB\_SIZE;

}

void hashtab\_init(struct listnode \*\*hashtab) {

for (int i = 0; i < HASHTAB\_SIZE; i++) {

hashtab[i] = NULL;

}

}

void hashtab\_add(struct listnode \*\*hashtab, char \*key, int value) {

struct listnode \*node;

int index = hash(key);

node = (listnode \*) (malloc(sizeof(\*node)));

if (node != NULL) {

strcpy(node->key, key);

node->value = value;

node->next = hashtab[index];

hashtab[index] = node;

}

}

struct listnode \*hashtab\_lookup(struct listnode \*\*hashtab, char \*key) {

int index = hash(key);

for (struct listnode \*node = hashtab[index]; node != NULL; node = node->next) {

if (strcmp(node->key, key) == 0) {

return node;

}

}

return NULL;

}

void hashtab\_delete(struct listnode \*\*hashtab, char \*key) {

int index = hash(key);

struct listnode \*prev = NULL;

for (struct listnode \*p = hashtab[index]; p != NULL; p = p->next) {

if (strcmp(p->key, key) == 0) {

if (prev == NULL)

hashtab[index] = p->next;

else

prev->next = p->next;

free(p);

return;

}

prev = p;

}

}

*main.cpp*