Министерство образования Республики Беларусь

Учреждение образования

“Брестский государственный технический университет”

**Лабораторная работа №5**

**По дисциплине АиСД за 2 семестр**  
**Тема: «Бинарные деревья поиска и хеш-таблицы»**

**Вариант 4**

**Выполнил:**

Студент группы ПО-6(1)  
 1-го курса

Мартынович Даниил

**Проверила:**

Гирель Т. Н.

Цель работы: провести экспериментальное исследование эффективности бинарных

деревьев поиска и хеш-таблиц.

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).

Код:

**bstree.h**

#pragma once

#include <iostream>

struct bstree{

char\* key;

int volume;

bstree\* l;

bstree\* r;

};

bstree\* bstree\_create(char\* key, int volume); //создание элемента

void bstree\_add(bstree\* tree, char\* key, int volume); //добавление элемента (средний случай O(log n),худший случай O(n))

bstree\* bstree\_lookup(bstree\* tree, int volume);

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

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

//поиск элемента (средний случай O(log n), худший случай O(n))

**bstree.cpp**

#include "bstree.h"

bstree\* bstree\_create(char\* key, int volume) {

bstree\* node;

node = new bstree;

if (node != NULL) {

node->key = key;

node->volume = volume;

node->l = NULL;

node->r = NULL;

}

return node;

}

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

bstree\* parent, \*node;

if (tree == NULL) return;

for (parent = tree; tree != NULL; ) {

parent = tree;

if (strcmp(key, tree->key) < 0) tree = tree->l;

else if (strcmp(key, tree->key) > 0) tree = tree->r;

else return;

}

node = bstree\_create(key, volume);

if ((strcmp(key, parent->key) < 0)) parent->l = node;

else parent->r = node;

}

bstree\* bstree\_lookup(bstree\* tree, int volume) {

while (tree != NULL) {

if (strcmp(volume, tree->key) == 0) return tree;

else if (strcmp(volume, tree->key) < 0) tree = tree->l;

else tree = tree->r;

}

return tree;

}

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

if (tree == NULL) return NULL;

while (tree->l != NULL) tree = tree->l;

return tree;

}

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

if (tree == NULL) return NULL;

while (tree->r != NULL) tree = tree->r;

return tree;

}

**hashtab.h**

#pragma once

#define HASHTAB\_SIZE 128

#define HASHTAB\_MUL 31

#include <iostream>

struct lnode {

char\* key;

int volume;

lnode\* next;

};

lnode\* hashtab[HASHTAB\_SIZE];

unsigned int hashtab\_hash(char\* key); //хеш-функция (О(|key|))

void hashtab\_init(lnode\*\* hashtab); //инициализация хеш-таблицы (O(n))

void hashtab\_add(lnode\*\* hashtab, char\* key, int volume); //добавление элемента в хеш-таблицу

lnode\* hashtab\_lookup(lnode\*\* hashtab, char\* key); //поиск

void hashtab\_delete(lnode\*\* hashtab, char\* key); //удаление

// средний случай O(1), худший случай O(n)

**hashtab.cpp**

#include "hashtab.h"

unsigned int hashtab\_hash(char\* key) {

unsigned int h = 0;

char\* p;

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

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

}

return h % HASHTAB\_SIZE;

}

void hashtab\_init(lnode\*\* hashtab) {

int i;

for (i = 0; i < HASHTAB\_SIZE; i++) hashtab[i] = NULL;

}

void hashtab\_add(lnode\*\* hashtab, char\* key, int volume) {

lnode\* node;

int index = hashtab\_hash(key);

node = new lnode;

if (node != NULL) {

node->key = key;

node->volume = volume;

node->next = hashtab[index];

hashtab[index] = node;

}

}

lnode\* hashtab\_lookup(lnode\*\* hashtab, char\* key) {

int index;

lnode\* node;

index = hashtab\_hash(key);

for (node = hashtab[index]; node != NULL; node = node->next)

if (strcmp(node->key, key) == 0) return node;

return NULL;

}

void hashtab\_delete(lnode\*\* hashtab, char\* key) {

int index;

lnode\* p, \*prev = NULL;

index = hashtab\_hash(key);

for (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;

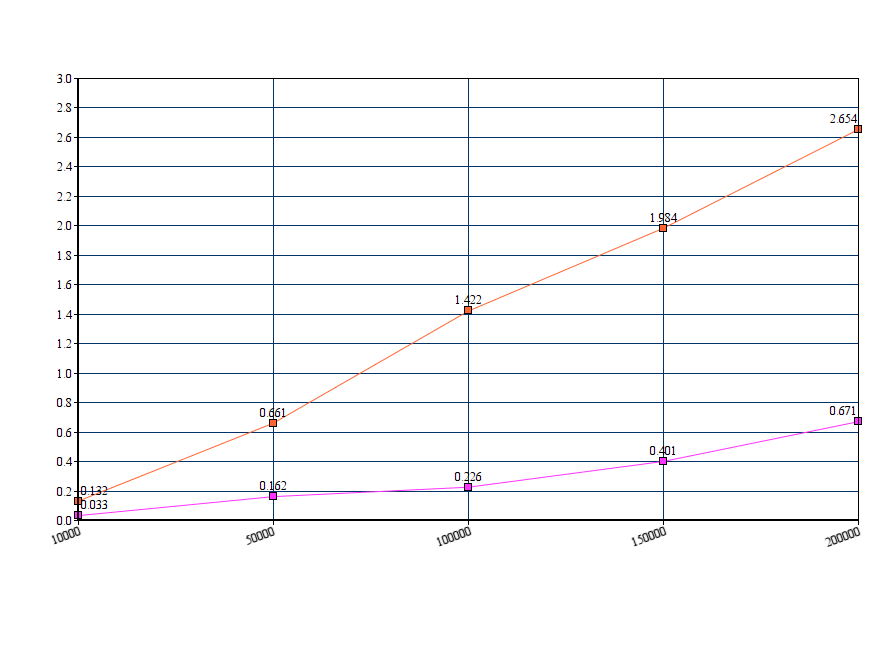
}

}

**Эксперимент 1. Сравнение эффективности поиска элементов в бинарном**

**дереве поиска и хеш-таблице в среднем случае (average case)**

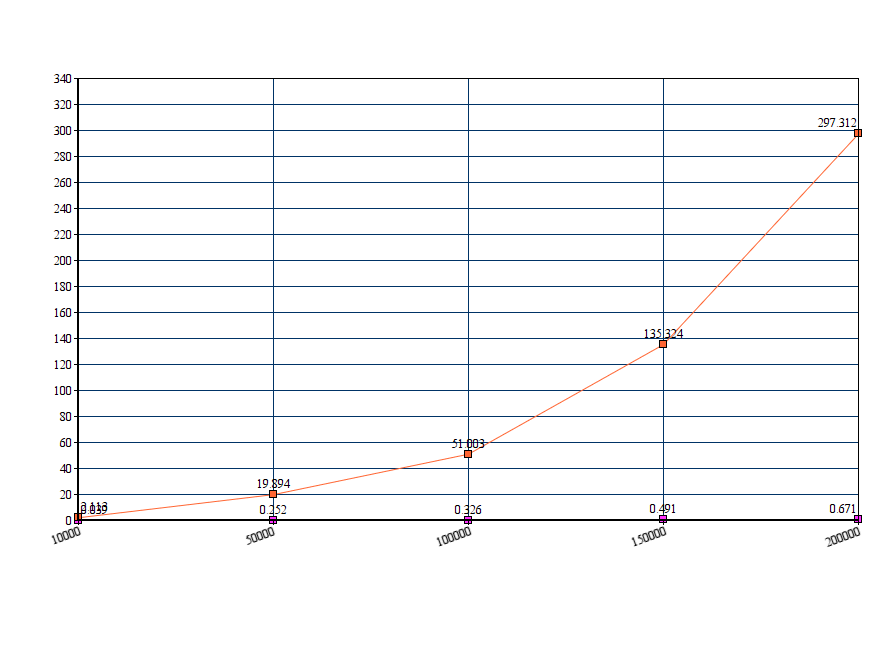
|  |  |  |  |
| --- | --- | --- | --- |
| № | Количество элементов в словаре | Время выполнения функции bstree\_lookup, с | Время выполнения функции hashtab\_lookup, с |
| 1 | 10 000 | 0,132 | 0,033 |
| 2 | 50 000 | 0,661 | 0,162 |
| 3 | 100 000 | 1,422 | 0,226 |
| 4 | 150 000 | 1,984 | 0,401 |
| 5 | 200 000 | 2,654 | 0,671 |



**Эксперимент 2. Сравнение эффективности добавления элементов в бинарное**

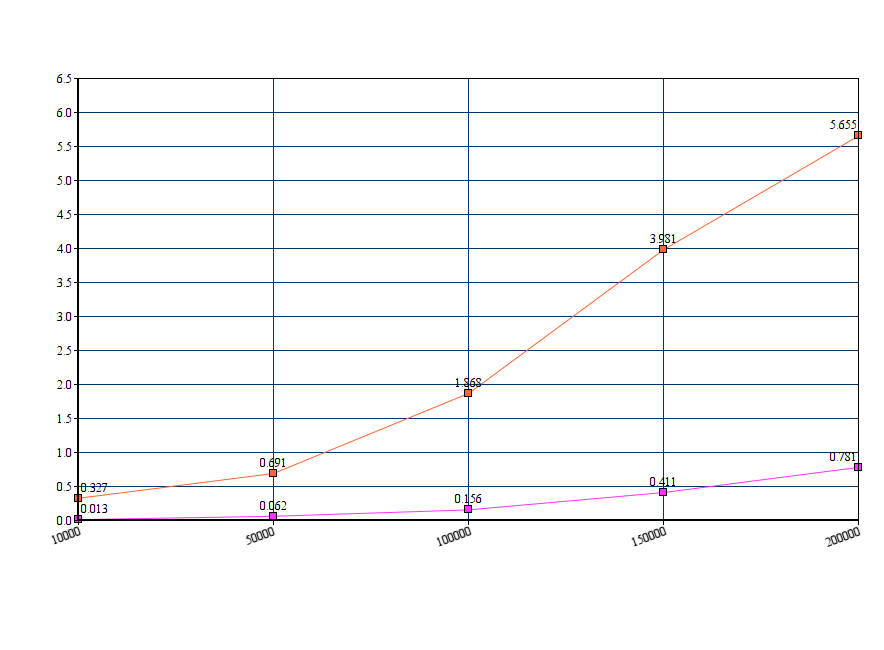
**дерево поиска и хеш-таблицу**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Количество элементов в словаре | Время выполнения  функции  bstree\_add, с | Время выполнения  функции  hashtab\_add, с |
| 1 | 10 000 | 2.113 | 0.039 |
| 2 | 50 000 | 19.894 | 0.252 |
| 3 | 100 000 | 51.003 | 0.326 |
| 4 | 150 000 | 135.324 | 0.491 |
| 5 | 200 000 | 297.312 | 0.671 |

****

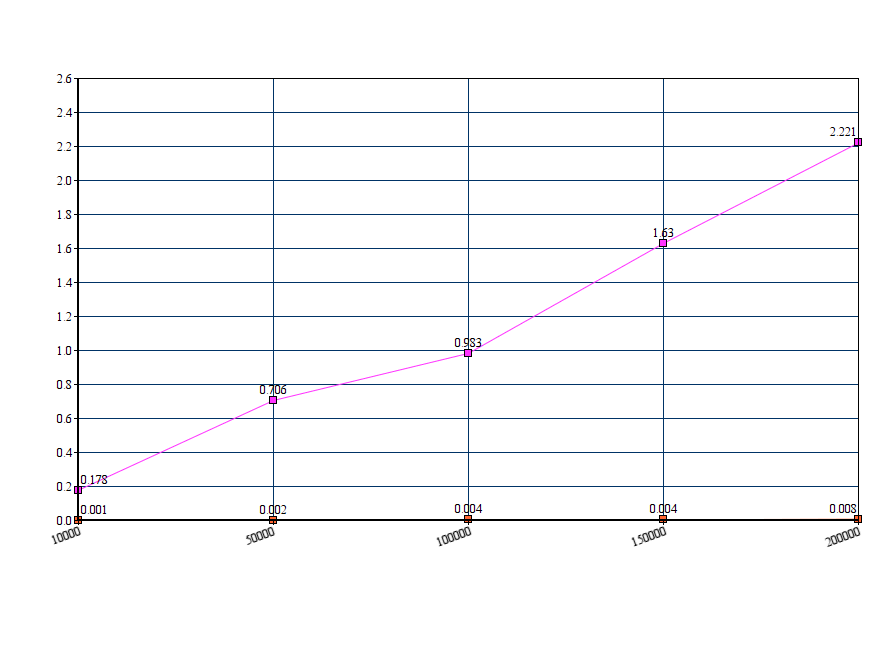
**Эксперимент 3. Сравнение эффективности поиска элементов в бинарном дереве поиска и хеш-таблице в худшем случае (worst case)**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Количество элементов в словаре | Время выполнения  функции  bstree\_lookup, с | Время выполнения  функции  hashtab\_lookup, с |
| 1 | 10 000 | 0.327 | 0.013 |
| 2 | 50 000 | 0.691 | 0.062 |
| 3 | 100 000 | 1.868 | 0.156 |
| 4 | 150 000 | 3.981 | 0.411 |
| 5 | 200 000 | 5.655 | 0.781 |

****

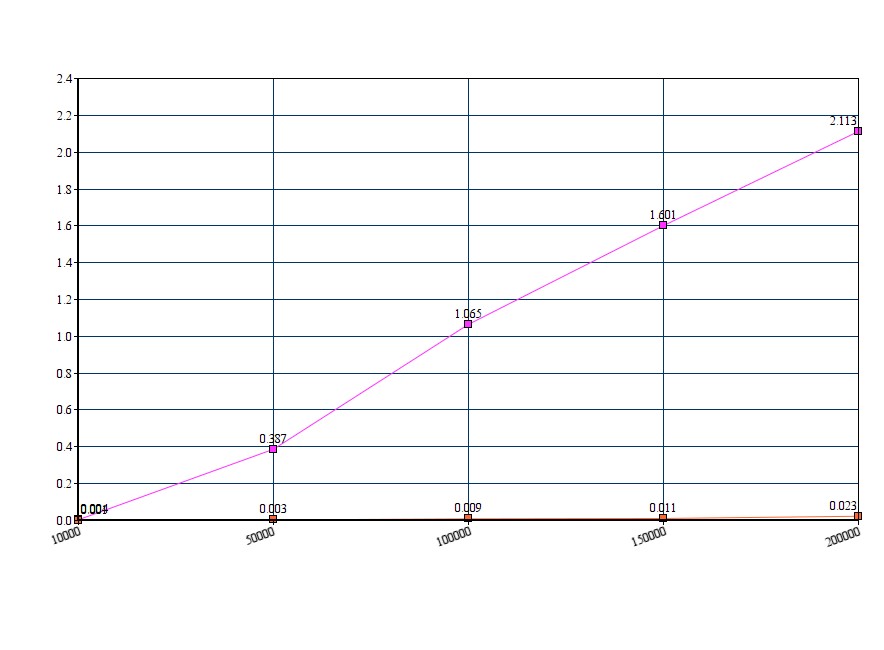
**Эксперимент 4. Исследование эффективности поиска минимального элемента в бинарном дереве поиска в худшем и среднем случаях.**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Количество элементов в словаре | Время выполнения  функции  bstree\_min, с | Время выполнения  функции  hashtab\_min, с |
| 1 | 10 000 | 0.001 | 0.178 |
| 2 | 50 000 | 0.002 | 0.706 |
| 3 | 100 000 | 0.004 | 0.983 |
| 4 | 150 000 | 0.004 | 1.630 |
| 5 | 200 000 | 0.008 | 2.221 |

****

**Эксперимент 5. Исследование эффективности поиска максимального элемента в бинарном дереве поиска в худшем и среднем случаях.**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Количество элементов в словаре | Время выполнения  функции  bstree\_max, с | Время выполнения  функции  hashtab\_max, с |
| 1 | 10 000 | 0.001 | 0.004 |
| 2 | 50 000 | 0.003 | 0.387 |
| 3 | 100 000 | 0.009 | 1.065 |
| 4 | 150 000 | 0.011 | 1.601 |
| 5 | 200 000 | 0.023 | 2.113 |

****

Вывод: Путем исследования эффективности бинарных деревьев поиска и хеш-таблиц, я сделал вывод, что хеш-таблицы работают быстрее и эффективнее, чем бинарные деревья.