Министерство образования Республики Беларусь

Учреждение образования

Брестский государственный технический университет

Кафедра ИИТ

Лабораторная работа №5

По дисциплине: АиСД

Тема: “Бинарные деревья поиска и хеш-таблицы ”

Выполнил:

Студент 1 курса

Группы ПО-7

Лобан К.Ю.

Проверила:

Глущенко Т.А.

Брест 2021

**Цель работы** — провести экспериментальное исследование эффективности бинарных деревьев поиска и хеш-таблиц.

**Постановка задачи:**

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). Функции для работы с бинарным деревом поиска должны быть помещены в файлы bstree.c (реализация функций) и bstree.h (объявление функций).

**Бинарное дерево:**

**bstree.h:**

#pragma once

struct bstree {

char\* key;

int value;

bstree\* left, \*right;

};

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

Создание элемента(O(1))

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

Добавление элемента(Худший случай–O(n), Средний случай–O(log n))

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

Поиск элемента(Худший случай–O(n), Средний случай–O(log n))

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

Поиск элемента с минимальным значением(Худший случай–O(n), Средний случай–O(log n))

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

Поиск элемента с максимальным значением(Худший случай–O(n), Средний случай–O(log n))

void show(bstree\*&);Обход дерева(O(n)

void del(bstree\*&);

Удаление дерева(Худший случай–O(n), Средний случай–O(log n))

**bstree.cpp:**

#include "bstree.h"

#include <iostream>

using namespace std;

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

bstree\* node;

node = new bstree;

if (node) {

node->key = \_strdup(key);

node ->value = value;

node->left = 0;

node->right = 0;

}

return node;

}

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

bstree\* root;

bstree\* node;

if (!tree) return;

root = tree;

while (tree) {

root = tree;

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, root->key) < 0))

root->left = node;

else

root->right = node;

}

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

if (!tree)return 0;

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

return tree;

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

if (tree->left)

bstree\_lookup(tree->left, key);

else return 0;

}

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

if (tree->right)

bstree\_lookup(tree->right, key);

else return 0;

}

}

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

if (tree->left)

bstree\_min(tree->left);

else return tree;

}

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

if (tree->right)

bstree\_max(tree->right);

else return tree;

}

void show(bstree\*& tree) {

if (tree) {

show(tree->left);

cout << tree->key << " - " << tree->value << endl;

show(tree->right);

}

}

void del(bstree\*& tree) {

if (tree) {

del(tree->left);

del(tree->right);

delete tree;

tree = 0;

}}

**main.cpp**

#include "bstree.h"

#include <iostream>

#include <fstream>

using namespace std;

int main() {

char key[30] = "root";

bstree\* tree = bstree\_create(key, strlen(key));

for (int i = 0; i < 10; i++) {

cout << "input >>> ";

cin.getline(key, 30);

bstree\_add(tree, key, strlen(key));

}

cout << "TREE:\n\n";

show(tree);

cout << endl;

bstree\* temp;

temp = bstree\_min(tree);

cout << "Min element >>> " << temp->key << " - " << temp->value << endl;

temp = bstree\_max(tree);

cout << "Max element >>> " << temp->key << " - " << temp->value << endl;

cout << "Search for >>> ";

char find[30];

cin.getline(find, 30);

temp = bstree\_lookup(temp, find);

cout << "Searched element >>> " << temp->key << " - " << temp->value << endl;

del(tree);

}

**Результат:**



**Хеш-таблица:**

**hashtab.h:**

#pragma once

#define HASHTAB\_SIZE 120

#define HASHTAB\_MUL 40

#include <iostream>

using namespace std;

struct listnode {

char\* key;

int value;

struct listnode\* next;

};

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

void hashtab\_init(listnode\*\*); инициализация таблицы(О(n))

void hashtab\_add(listnode\*\*, char\*, int); добавление элемента в таблицу(средний – O(1), худший – O(n))

listnode\* hashtab\_lookup(listnode\*\*, char\*); поиск элемента(средний – O(1), худший – O(n))

void hashtab\_delete(listnode\*\*, char\*); удаление элемента(средний – O(1), худший – 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(listnode\*\* hashtab) {

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

hashtab[i] = 0;

}

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

listnode\* node = new listnode;

int index = hashtab\_hash(key);

if (node) {

node->key = key;

node->value = value;

node->next = hashtab[index];

hashtab[index] = node;

}

}

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

listnode\* node;

int index = hashtab\_hash(key);

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

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

return node;

}

return 0;

}

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

int index = hashtab\_hash(key);

listnode\* node, \* prev = 0;

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

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

if (!prev) hashtab[index] = node->next;

else prev->next = node->next;

free(node);

return;

}

prev = node;

}

}

**main.cpp**

#include "hashtab.h"

int main() {

listnode\* hashtab[HASHTAB\_SIZE];

hashtab\_init(hashtab);

listnode\* node;

char key[30];

for (int i = 0; i < 10; i++) {

cout << "input: ";

cin.getline(key, 30);

hashtab\_add(hashtab, key, strlen(key));

}

cout << "Input searched word >>> ";

cin.getline(key, 30);

node = hashtab\_lookup(hashtab, key);

if (node)

cout << node->key << " - " << node->value << endl;

else

cout << "Not found" << endl;

hashtab\_delete(hashtab, key);

node = hashtab\_lookup(hashtab, key);

if (node)

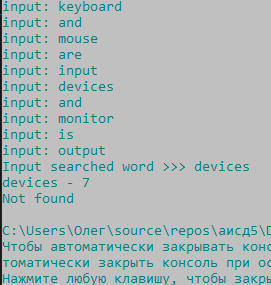
cout << node->key << " - " << node->value << endl;

else

cout << "Not found" << endl;

}

**Результат:**



Сравнение эффективности поиска элементов в бинарном дереве

|  |  |  |  |
| --- | --- | --- | --- |
| № | N элементов | T(bstree\_lookup) | T(hashtab\_lookup) |
| 1 | 10000 | 0.074 | 0,014 |
| 2 | 30000 | 0,219 | 0,030 |
| 3 | 50000 | 0.356 | 0,051 |
| 4 | 70000 | 0.692 | 0,082 |
| 5 | 100000 | 0.805 | 0,121 |
| 6 | 150000 | 1,956 | 0.218 |
| 7 | 200000 | 2,605 | 0.41 |

поиска и хеш-таблице в среднем случае:

Сравнение эффективности добавления элементов в бинарное дерево и хеш-таблицу:

|  |  |  |  |
| --- | --- | --- | --- |
| № | N элементов | T(bstree\_add)(c) | T(hashtab\_add)(c) |
| 1 | 10000 | 1,032 | 0.013 |
| 2 | 30000 | 3,582 | 0.045 |
| 3 | 50000 | 7,230 | 0.123 |
| 4 | 70000 | 11,465 | 0.251 |
| 5 | 100000 | 19,983 | 0.458 |
| 6 | 150000 | 52,201 | 0.669 |
| 7 | 200000 | 110,429 | 0.871 |

Сравнение эффективности поиска элемеентов в бинарном дереве и хеш-таблице в худшем случае

|  |  |  |  |
| --- | --- | --- | --- |
| № | N элементов | T(bstree\_lookup) | T(hashtab\_lookup) |
| 1 | 10000 | 0.18 | 0,002 |
| 2 | 30000 | 0,24 | 0,017 |
| 3 | 50000 | 0.32 | 0,032 |
| 4 | 70000 | 0.48 | 0,065 |
| 5 | 100000 | 0.87 | 0,097 |
| 6 | 150000 | 1,256 | 0.121 |
| 7 | 200000 | 2,737 | 0.348 |

Исследование эффективности поиска минимального элемента в бинарном дереве в среднем и худшем случаях

|  |  |  |  |
| --- | --- | --- | --- |
| № | N элементов | T(средний) | T(худший) |
| 1 | 10000 | 0,001 | 0.075 |
| 2 | 30000 | 0,001 | 0.176 |
| 3 | 50000 | 0.001 | 0.213 |
| 4 | 70000 | 0.002 | 0.345 |
| 5 | 100000 | 0.002 | 0.563 |
| 6 | 150000 | 0.003 | 0.871 |
| 7 | 200000 | 0.005 | 1,21 |
|  |  |  |  |

Исследование эффективности поиска максимального элемента в бинарном дереве в среднем и худшем случаях

|  |  |  |  |
| --- | --- | --- | --- |
| № | N элементов | T(средний) | T(худший) |
| 1 | 10000 | 0,001 | 0.012 |
| 2 | 30000 | 0,001 | 0.072 |
| 3 | 50000 | 0,002 | 0.213 |
| 4 | 70000 | 0.003 | 0.326 |
| 5 | 100000 | 0.004 | 0,523 |
| 6 | 150000 | 0.005 | 0.798 |
| 7 | 200000 | 0.01 | 1,219 |

**Вывод:** В ходе выполнения данной лабораторной работы я изучил такие структуры данных как бинарное дерево поиска и хеш-таблица, а также сравнил их эффективность в выполнении различных операций, в результате которых выяснилось, что хеш-таблицы работают быстрее.