МИНОБРНАУКИ РОССИИ

ФЕДЕРЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГУ»)

Факультет прикладной математики, информатики и механики

Кафедра математического обеспечения ЭВМ

**ТЕМА**

Курсовая работа

по направлению: 02.03.02 «Фундаментальная информатика   
и информационные технологии»

Подготовили: студенты 2 курса 61 группы

Жарковский Никита, Поминова Алена,

Козлова Анна, Малежина Ирина,

Попов Евгений, Целовальников Евгений

Воронеж 2017

**Постановка задачи**

Реализовать хеш-таблицу с хеш-функцией, вычисляемой методом умножения, а так же основные функции работы с хеш-таблицей:

* Инициализация
* Вставка
* Поиск по ключу
* Удаление элемента
* Очистка таблицы
* Вывод всех элементов на экран

**Разработка программы**

**Модель и метод решения задачи**

Реализация хеш-таблицы заключается в использовании одномерного массива списков.

Хеш-функция вычисляется методом умножения.   
Формула для вычисления h (k) = где k – ключ,   
m = 2p (размер хеш-таблицы), d = .

Для разрешения коллизий используется метод цепочек. После вычисления значения хеш-функции = h(k) задача сводится к последовательному поиску в -м списке.

**Алгоритм решения задачи**

Инициализируется исходная хеш-таблица путем обнуления указателей в массиве.

Выводится список команд пользователю. И поочередно считываются и выполняются необходимые команды.

При добавлении элемента проверяется наличие такого же элемента в хеш-таблице. Если элемент не найден, то создается новый элемент и заносится в начало списка по соответствующему хешу.

Поиск работает следующим образом: вычисляется хеш-функция, и в соответствующем списке ищется требуемый элемент.

При удалении элемента из хеш-таблицы сначала вычисляется хеш-функция. После проверяется 3 случая. Если в указанном списке 0 элементов, то искомого для удаления элемента нет в таблице. Если в первый элемент совпадает с удаляемым, то он удаляется, и указатель на начало списка указывает на второй элемент списка или обнуляется, если в списке был единственный элемент. Если элемент находится не в начале списка, то он удаляется как удаляется элемент из однонаправленного списка.

При демонстрации списка на экране, соответствующая функция проходится по массиву списков хеш-таблицы и последовательно выводит все элементы (в порядке значения хеш-функции данного элемента).

При полном удалении хеш-таблицы удаляются все ее элементы по отдельности.

В случае если пользователь хочет завершить программу, происходит вызов return из main().

**Руководство пользователя**

1) Выбрать нужную команду

1 - добавить элемент в таблицу

Ввести число, которое нужно добавить в таблицу.

2 - найти элемент

Ввести число, которое ищется в таблице. Программа вернет число, если

оно найдено, и значение его хеш-функции.

3 - удалить элемент

Ввести число, которое нужно удалить из таблицы.

4 - показать всю таблицу

Программа выведет все элементы хеш-таблицы.

5 - очистить таблицу

Программа очистит хеш-таблицу

6 - завершить программу

Программа прекратит свою работу.

2) Вернуться к пункту 1)

**Руководство программиста**

Функция unsigned int hashtab\_hash(int key) вычисляет значение хеш-функции. На вход подается сам ключ, возвращается значение, вычисляемое по формуле h (k) = , описанной в разделе “Модель и метод решения задачи”.

Функция void hashtab\_init(struct listnode \*\*hashtab) получает на вход указатель на массив и обнуляет все его элементы.

Функция bool hashtab\_lookup(struct listnode \*\*hashtab, int key) сначала вычисляет значение хеш-функции, после производит по нему поиск элемента. Возвращает true в случае удачного поиска, false – иначе.

Функция bool hashtab\_add(struct listnode \*\*hashtab, int key) осуществляет вставку элемента в хеш-таблицу в качестве головного элемента списка. Возвращает true в случае удачной вставки, false – если не удалось выделить память под элемент или он уже существует в хеш-таблице.

bool hashtab\_delete(struct listnode \*\*hashtab, int key) осуществляет удаление элемента из хеш-таблицы по принципу, описанному в разделе “Алгоритм решения задачи”. Возвращает true, если элемент удален, false – иначе.

void Clear(struct listnode \*\*hashtab) очищает всю хеш-таблицу.

bool Show(struct listnode \*\*hashtab) выводит результат на экран.

**Тестирование программы**

**Основной тест:**

Enter the command: 1

Enter the number: 12345

Your key is added

Enter the command: 1

Enter the number: 54321

Your key is added

Enter the command: 4

54321

12345

Enter the command: 3

Enter number: 12345

Your key is deleted

Enter the command: 2

Enter the number: 54321

Key: 54321

Hash: 17

Enter the command: 2

Enter the number: 12345

Not found

**Результаты работы программы и их анализ**

Работа работает корректно в случае основного тестирования. Экстренных тестов не обнаружено. Корректные данные – числа в диапазоне, поддерживаемом типом int.

**Выводы**

Программа, основанная на хеш-таблице с методом умножения, успешно справляется с поставленной задачей. Эффективность программы зависит от выбранных m и d для формулы метода умножения.

**Приложение**

- Листинг программы

#include<iostream>

#include<cstdlib>

#include "cmath"

#define HASHTAB\_SIZE 128

using namespace std;

struct listnode {

int key;

struct listnode \*next;

};

struct listnode \*hashtab[HASHTAB\_SIZE];

unsigned int hashtab\_hash(int key) //хэш-функция

{

float A = 0.618033;

int hashCode = HASHTAB\_SIZE \* fmod (key \* A, 1);

return hashCode;

}

void hashtab\_init(struct listnode \*\*hashtab) //инициализация хэш-таблицы

{

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

hashtab[i] = NULL;

}

bool hashtab\_lookup(struct listnode \*\*hashtab, int key) //поиск значения в хэш-таблице

{

int index = hashtab\_hash(key);

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

if (node->key == key)

return true;

return false;

}

bool hashtab\_add(struct listnode \*\*hashtab, int key) //добавление в хэш-таблицу

{

struct listnode \*node = new listnode;

int index = hashtab\_hash(key);

// добавление в начало списка

if (node != NULL && !hashtab\_lookup(hashtab, key) )

{

node->key = key;

node->next = hashtab[index];

hashtab[index] = node;

return true;

}

return false;

}

bool hashtab\_delete(struct listnode \*\*hashtab, int key) //удаление хэш-таблицы

{

int index = hashtab\_hash(key);

struct listnode \*node = hashtab[index];

//case 0 element

if(node == NULL) return false;

//case 1 element

if(node->key == key)

{

struct listnode \*temp = node;

node = node->next;

hashtab[index] = node;

temp->next = NULL;

delete temp;

return true;

}

//case n elements

while(node)

{

if(node->next->key == key)

{

struct listnode \*temp = node->next;

node->next = (node)->next->next;

temp->next = NULL;

delete temp;

return true;

}

node = node->next;

}

return false;

}

void Clear(struct listnode \*\*hashtab) //очистка хэш-таблицы

{

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

{

listnode \*head = hashtab[i];

if (head != NULL)

{

listnode \*prev = NULL;

while (head)

{

prev = head;

head = head->next;

prev->next = NULL;

delete prev;

}

}

hashtab[i] = head;

}

}

bool Show(struct listnode \*\*hashtab) //элементы хеш-таблицы

{

bool flag=false;

listnode \*node;

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

if(hashtab[i]!=NULL)

{

flag=true;

node=hashtab[i];

while(node)

{

cout<<node->key<<endl;

node=node->next;

}

}

return flag;

}

int main()

{

hashtab\_init(hashtab);

bool flag = true;

cout << "Commands:" << endl

<< "1.Add number to the hash" << endl

<< "2.Lookup all information"<<endl

<< "3.Delete number" << endl

<< "4.Show all numbers(are ordered on a hash)" << endl

<< "5.Clear hashtable" << endl

<< "6.End the program" << endl;

int command;

while(1)

{

cout << endl << "Enter the command: ";

cin >> command;

if(command<1 || command>6)

cout<<"Command isn't found";

switch(command)

{

case 1:{

cout << "Enter the number: ";

int key;

cin >> key;

flag = hashtab\_add(hashtab, key);

if(!flag)

{

cout << "This number already exist" << endl;

}

else cout << "Your key is added" << endl;

break;

}

case 2:{

cout << "Enter the number: ";

int key;

cin >> key;

flag = hashtab\_lookup(hashtab, key);

if(flag)

{

cout << " Key: " << key << endl;

cout << " Hash: " << hashtab\_hash(key) << endl;

}

else cout << "Not found" << endl;

break;

}

case 3:{

cout << "Enter number: ";

int key;

cin >> key;

flag = hashtab\_delete(hashtab,key);

if (flag)

cout << "Your key is deleted" << endl;

else

cout << "Your key isn`t founded" << endl;

break;

}

case 4:{

if(!(Show(hashtab))) cout<<"Hash-table is empty";

break;

}

case 5:{

Clear(hashtab);

break;

}

case 6:{

return 0;

break;

}

}

}

Clear(hashtab);

return 0;

}