МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ «ЛЭТИ»
ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЁТ

по лабораторной работе №5
по дисциплине «Алгоритмы и структуры данных»
Тема: Хеш-таблицы

Студент гр. 9303 _____ Эйсвальд М.И.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург
2020

Цель работы.

Изучить принцип алгоритма хеширования, реализовать хеш-таблицу самостоятельно.

Задание.

Вариант **23**: Хеш-таблица: с цепочками; действие: 1+2а (по заданной последовательности элементов *Elem* построить хеш-таблицу; для построенной структуры данных проверить, входит ли в неё элемент *e* типа *Elem*, и если входит, то в скольких экземплярах. Добавить элемент *e* в структуру данных. Предусмотреть возможность повторного выполнения с другим элементом.)

Основные теоретические положения.

Хеширование — способ организации данных, основанный на вычислении по данным — ключу — с помощью легко вычислимой функции — *хеш-функции* — адреса элемента в таблице. Занося данные в такую таблицу, можно легко проверить, входит ли тот или иной элемент в таблицу, поскольку можно быстро вычислить адрес, где должен находиться интересующий пользователя ключ. По этой же причине быстра и операция вставки в хеш-таблицу. К недостаткам хеш-таблиц относится главным образом отсутствие упорядоченности: элементы в ней не поддерживаются в каком-то определённом порядке, вследствие чего хеш-таблица плохо подходит для любых задач, как-то связанных с сортировкой.

Коллизия — ситуация, когда хеш-функция принимает одинаковые значения для разных ключей. Хорошая хеш-функция должна допускать как можно меньшее число коллизий.

Для разрешения коллизий используются разные стратегии: открытая адресация и метод цепочек.

Суть метода открытой адресации состоит в следующем: если ячейка, куда хеш-функция «определила» очередной элемент, уже занята, для элемента ищется новая ячейка каким-либо способом. Самый простой вариант — перебор всех последующих ячеек.

Метод цепочек использует иную идею: элементы с совпадающими значениями хешей объединяются в цепочки — связные списки в том или ином виде. Такой вариант гарантирует, что для нового элемента всегда найдётся

место в таблице, но если большая часть всех хранимых элементов находится в середине или конце получающихся списков, смысл хэш-таблицы в значительной мере теряется.

Выполнение работы.

В рамках работы было решено реализовать метод цепочек, использующий таблицу индексов: элементы хеш-таблицы хранят ссылки на элементы таблицы индексов, которые включают в себя ключ, количество повторений данного ключа и ссылку на следующий элемент цепочки. Таким образом, все элементы хеш-таблицы указывают на головы односвязных списков; каждый список содержит все ключи с одинаковым хешем. Для представления элемента таблицы индексов была реализована простая структура `Symbol`, содержащая описанные выше поля.

Сама хеш-таблица представлена классом `HashTable`, который включает и таблицу индексов, и хеш-таблицу. Хеш-функция реализована в виде метода класса: она возвращает остаток от деления очередного элемента на размер хеш-таблицы.

Алгоритм вставки элемента *e* в таблицу выглядит следующим образом: исследуется цепочка, расположенная в таблице по адресу $H(e)$, где H — хеш-функция. Если элемент обнаружен в цепочке, увеличивается счётчик его вхождений; в противном случае он добавляется в конец цепочки.

Тестирование.

Результаты тестирования программы представлены в таблицах ниже. Для удобства восприятия некоторые строки вывода, не относящиеся к сути, были удалены.

Вывод.

В ходе выполнения лабораторной работы были изучены хеш-таблицы, их преимущества и недостатки. Результатом работы стала программа, реализующая создание и дополнение хеш-таблицы с цепочками.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: elements.h

```
#ifndef ELEMENTS_H
#define ELEMENTS_H

template<typename Elem>
struct Symbol{
    Symbol(): link(nullptr), count(1) {}

    Elem key;
    unsigned count;
    Symbol* link;
};

#endif // ELEMENTS_H
```

Название файла: hashtable.h

```
#ifndef HASHTABLE_H
#define HASHTABLE_H

#include <fstream>
#include <iostream>
#include <utility>

#include ”elements.h”

using namespace std;

#define SCALE 10

template<typename Elem>
class HashTable{
public:
    HashTable(int length, ofstream& out): free_index(0), out(out){
        size = length*11/10 + 3;
        symb_table = new Symbol<Elem> [SCALE*size];
        data = new Symbol<Elem>* [size];
        if(!data || !symb_table){
            cout << ”Bad allocation!\n”;
            exit(0);
        }
        for(int i = 0; i < size; ++i) data[i] = nullptr;
    }

    ~HashTable(){
        delete [] data;
        delete [] symb_table;
    }

    int hash(Elem elem){
        return elem % size;
    }

    void add(Elem elem){
```

```
    if(!data[hash(elem)]){
        if(free_index >= SCALE*size){
            cout << "Overfull table!\n";
            out << "Overfull table!\n";
            return;
        }
        cout << "Element is not present!\n";
        out << "Element is not present!\n";
        symb_table[free_index].key = elem;
        data[hash(elem)] = &(symb_table[free_index++]);
    }
    else{
        Symbol<Elem>* symb = data[hash(elem)];
        if (symb->key == elem){
            cout << "Element is present (x"<<symb->count<<")!\n";
            out << "Element is present (x"<<symb->count<<")!\n";
            ++symb->count; return;
        }
        else{
            while(symb->link){
                symb = symb->link;
                if (symb->key == elem){
                    cout << "Element is present (x"<<symb->count<<")!\n";
                    out << "Element is present (x"<<symb->count<<")!\n";
                    ++symb->count; return;
                }
            }
            if(free_index >= SCALE*size){
                cout << "Overfull table!\n";
                out << "Overfull table!\n";
                return;
            }
            cout << "Element is not present!\n";
            out << "Element is not present!\n";
            symb_table[free_index].key = elem;
            symb->link = &(symb_table[free_index++]);
        }
    }
}

void printChains(){
    Symbol<Elem>* symb;
    cout << "Chains in hash table of size " << size << ":\n";
    out << "Chains in hash table of size " << size << ":\n";
    for(int i = 0; i < size; ++i){
        cout << "hash = " << i << ": ";
        out << "hash = " << i << ": ";
        if(!data[i]){
            cout << "none\n";
            out << "none\n";
            continue;
        }
        symb = data[i];
        cout << symb->key << " (x" << symb->count << ")";
        out << symb->key << " (x" << symb->count << ")";
        while(symb->link){
            cout << " -> ";
            out << " -> ";
            symb = symb->link;
            cout << symb->key << " (x" << symb->count << ")";
```

```
                out << symb->key << " (x" << symb->count << ")";
            }
            cout << "\n";
            out << "\n";
        }
    }

private:
    Symbol<Elem>** data;
    Symbol<Elem>* symb_table;
    int free_index;
int size;
    ofstream& out;
};

#endif
```

Название файла: main.cpp

```
#include <iostream>
#include <fstream>
#include <string>

#include "hashtable.h"

#define OUTFILE "output.txt"
#define ADD '+'

int main(){
    HashTable<char>* ht = nullptr;
const char* source;

std::ofstream out(OUTFILE);
if(!out.is_open()){
std::cout << "Failed to create output file. Exiting...\n";
return 0;
}
std::string str;
std::cout << "Specify input file name: ";
getline(std::cin, str);
std::ifstream infile(str.c_str());
if(!infile.is_open()){
std::cout << "Failed to open file. Expecting input from console...\n";
std::cout << "Enter EOF (i.e. Ctrl+D) to finish.\n";
}
do{
getline( (infile.is_open() ? infile : std::cin) , str);
if( (infile.is_open() && infile.eof()) || (std::cin.eof()) ) break;
std::cout << "Read line: \"" << str << "\"\n";
out << "Read line: \"" << str << "\"\n";
        if(!str.length()) continue;
source = str.c_str();
        if(*source != ADD || !ht){
            if(*source == ADD){
                std::cout << "Error: an attempt to add to hash table ";
                std::cout << "without creating one.\n";
                out << "Error: an attempt to add to hash table ";
                out << "without creating one.\n";
                continue;
            }
```

```
            else{
                --source;
                delete ht;
            }
            ht = new HashTable<char>(str.length(),out);
            ht->printChains();
        }
        ++source;
while(*source){
            cout << "Adding '" << *source << "'\n";
            out << "Adding '" << *source << "'\n";
            ht->add(*source);
            ht->printChains();
++source;
}
std::cout << "\n";
out << "\n";
}while((infile.is_open() && !infile.eof()) ||
 (!(infile.is_open()) && !std::cin.eof()) );

return 0;
}
```

Таблица 1 – Тестирование на корректных входных данных

Входные данные	Вывод
Mikhail	Read line: ”Mikhail” Chains in hash table of size 10: hash = 0: none hash = 1: none hash = 2: none hash = 3: none hash = 4: h (x1) hash = 5: i (x2) hash = 6: none hash = 7: M (x1) -> k (x1) -> a (x1) hash = 8: l (x1) hash = 9: none
+g	Read line: ”+g” Adding ’g’ Element is not present! Chains in hash table of size 10: hash = 0: none hash = 1: none hash = 2: none hash = 3: g (x1) hash = 4: h (x1) hash = 5: i (x2) hash = 6: none hash = 7: M (x1) -> k (x1) -> a (x1) hash = 8: l (x1) hash = 9: none
+a	Read line: ”+a” Adding ’a’ Element is present (x1)! Chains in hash table of size 10: hash = 0: none hash = 1: none hash = 2: none hash = 3: g (x1) hash = 4: h (x1) hash = 5: i (x2) hash = 6: none hash = 7: M (x1) -> k (x1) -> a (x2) hash = 8: l (x1) hash = 9: none

Таблица 2 – Тестирование на некорректный ввод

Входные данные	Вывод
+a	Read line: ”+a” Error: an attempt to add to hash table without creating one.