**Федеральное Агентство по Образованию**

Государственное образовательное учреждение высшего профессионального образования

**Санкт-Петербургский Государственный Электротехнический Университет «ЛЭТИ» им. В.И.Ульянова (Ленина)**

**(СПБГЭТУ)**

Кафедра МОЭВМ

Отчет по лабораторной работе №1

по дисциплине «Анализ и проектирование алгоритмов»

Выполнил: Черницын Н. А.

Факультет: КТИ

Группа: 9301

Проверил: Казаков Б.Б.

#### Санкт-Петербург,

# 2013

# ***Формулировка задания:*** Заданы два множества (размеров n и m). Являются ли они пересекающимися?

***Способ решения:*** Решение посредством хеширования. Создается хеш-таблица, содержащая элементы обоих множеств, и проверяется, что коллизии являются результатом хеширования идентичных элементов. На практике это может быть лучшим решением.

***Теоретическая часть:***

Хеш-таблица представляет собой обобщение обычного массива. Однако в то время как ключом массива может быть только число, для хеш-таблицы им может быть любой объект, для которого можно вычислить хеш-код. Интерфейс хеш-таблицы предоставляет нам следующие операции:

* Добавление новой пары ключ-значение
* Поиск значения по ключу
* Удаление пары ключ-значение по ключу

Среднее время поиска значения по ключу в хеш-таблице равно O(1). Это значит, что в среднем, наш поиск не будет зависеть от количества элементов в хеш-таблице, и будет равен некоторому константному значению.

Идея хеширования основана на распределении ключей в обычном массиве H[0..m-1]. Распределение осуществляется вычислением для каждого ключа элемента некоторой хеш-функции h. Эта функция на основе ключа вычисляет целое число n, которое служит индексом для массива H.

Для каждого типа данных можно разработать свою хеш-функцию. Однако есть основные требования к хеш-функции: она должна распределять ключи по ячейкам хеш-таблицы как можно более равномерно, и должна просто вычисляться.

При достаточно маленьком значении m (размера хеш-таблицы) по отношению к n (количеству ключей) или при плохой хеш-функции, может случиться так, что два ключа будут хешированны в одну и ту же ячейку массива H. Такая ситуация называется коллизией. Хорошие хеш-функции стремятся минимизировать вероятность коллизий, однако, учитывая то, что пространство всех возможных ключей может быть больше размера нашей хеш-таблицы H, всё же избежать их вряд ли удастся

***Алгоритм работы программы:***

*Хеширование с цепочками.*

В случае открытого хеширования (другое название хеширования цепочками), мы объедением элементы, хешированные в одну и ту же ячейку, в связный список.

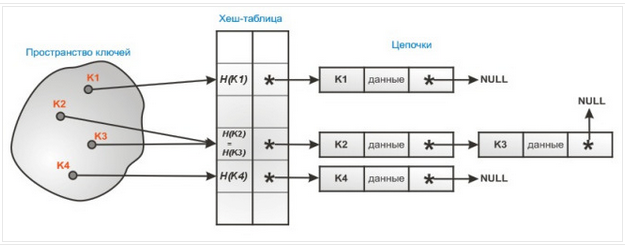


Рисунок 1 – открытое хэширование

Если при добавлении в хеш-таблицу в заданную ячейку мы встречаем ссылку на элемент связного списка, то случается коллизия. Так, мы просто вставляем наш элемент как узел в список.

*Алгоритм:*

1. Сначала идет построение hash – таблицы для множества А.
2. Затем в эту же таблицу добавляется множество B. Из рисунка 1 можно увидеть, что ключи k2 и k3 хэшированы в одну ячейку таблицу. Значит, мы имеем коллизию.
3. Далее идет проверка на пересечение множеств. Если k2 = k3, то наши множества пересекаются, если k2 ≠ k3 – не пересекаются.

В нашем случае hash – функция вычисляется по формуле:

где N – сумма кодов символов в строке.

***Листинг:***

* *Main.cpp*

#include <iostream>

#include <string>

#include <fstream>

//#include "list.h"

#include "hashtable.h"

Hash::HashTable\* load(Hash::HashTable\* table, std::string filename)

{

std::cout<<"Чтение файла "<<filename<<":"<<std::endl;

std::ifstream file(filename.c\_str());

if (!file || !file.is\_open())

{

std::cerr<<"ERROR! Invalid FILE!"<<std::endl;

return NULL;

}

std::string data;

if (!table)

table=new Hash::HashTable;

while (file>>data)

{

table->add(data);

std::cout<<data<<"\t";

//mass->add(data);

}

std::cout<<"[OK]"<<std::endl;

file.close();

return table;

}

int main()

{

Hash::HashTable\* table;

std::cout<<"Построение хэш-таблицы для множеств..."<<std::endl;

table=load(table,"mnogestvo\_A.txt");

if (!table)

goto err;

//table->printTable();

table=load(table,"mnogestvo\_B.txt");

if (!table)

goto err;

std::cout<<"[OK]"<<std::endl;

table->printTable();

std::cout<<"Проверка коллизий:"<<std::endl;

if (table->check())

{

std::cout<<"Множества не пересекаются!"<<std::endl;

return 1;

}

std::cout<<"Множества пересекаются!"<<std::endl;

return 0;

err:

std::cerr<<"ERROR! Ошибка множества!"<<std::endl;

return -1;

}

* *list.cpp*

#include "list.h"

list::list()

{

}

* *list.h*

#ifndef LIST\_H

#define LIST\_H

#include <iostream>

namespace List

{

template <class TYPE>

class \_node

{

public:

\_node();

//private:

\_node<TYPE>\* \_next;

TYPE \_data;

};

template <class TYPE>

class list

{

public:

list();

~list();

bool reset();

bool next();

void add(TYPE data);

void print(int TAB);

bool empty();

bool check();

private:

\_node<TYPE>\* \_head;

\_node<TYPE>\* \_tail;

protected:

\_node<TYPE>\* \_curr;

};

}

#endif // LIST\_H

/\*\*

\*Реализация функций класса \_node

\*/

template <class TYPE>

List::\_node<TYPE>::\_node()

{

\_next=NULL;

}

/\*\*

\*Реализация функций класса list

\*/

template <class TYPE>

List::list<TYPE>::list()

{

\_head=NULL;

\_curr=NULL;

\_tail=NULL;

}

template <class TYPE>

List::list<TYPE>::~list()

{

while(\_head)

{

\_curr=\_head;

\_head=\_head->\_next;

delete \_curr;

}

}

template <class TYPE>

bool List::list<TYPE>::reset()

{

if (!\_head)

return false;

\_curr=\_head;

return true;

}

template <class TYPE>

bool List::list<TYPE>::next()

{

if (!\_curr || !\_curr->\_next)

return false;

\_curr=\_curr->\_next;

return true;

}

template <class TYPE>

void List::list<TYPE>::add(TYPE data)

{

if (!\_head)

{

\_head=new List::\_node<TYPE>;

\_head->\_data=data;

\_curr=\_tail=\_head;

return;

}

\_tail->\_next=new List::\_node<TYPE>;

\_tail=\_tail->\_next;

\_tail->\_data=data;

}

template <class TYPE>

void List::list<TYPE>::print(int TAB)

{

List::\_node<TYPE>\* temp=\_head;

for(; temp; temp=temp->\_next)

{

for (int i=0; i<TAB; i++)

std::cout<<" ";

std::cout<<temp->\_data<<std::endl;

}

}

template <class TYPE>

bool List::list<TYPE>::empty()

{

return !\_head;

}

template <class TYPE>

bool List::list<TYPE>::check()

{

bool res=true;

if (empty())

return res;

//TYPE temp;

List::\_node<TYPE>\* temp1=\_head;

List::\_node<TYPE>\* temp2=temp1->\_next;

for(; temp1 && res; temp1=temp1->\_next)

for(; temp2 && res; temp2=temp2->\_next)

if (temp1->\_data==temp2->\_data)

{

std::cout<<"find="<<temp1->\_data<<std::endl;

res=false;

return res;

}

return res;

}

* *hashtable.cpp*

#include "hashtable.h"

Hash::HashTable::HashTable()

{

\_table=new elementTable[Hash::M];

for (unsigned int i=0; i<Hash::M; i++)

{

\_table[i].data=new List::list<std::string>;

\_table[i].hashCode=i;

}

}

Hash::HashTable::~HashTable()

{

delete \_table;

}

int Hash::HashTable::hashCode(std::string data)

{

int res=0;

for(unsigned int i=0; i<data.size(); i++)

res+=data[i];

res=res % Hash::M;

return res;

}

void Hash::HashTable::add(std::string data)

{

//std::cout<<hashCode(data)<<"="<<data<<std::endl;

//std::cout<<\_table[hashCode(data)].data<<std::endl;

\_table[hashCode(data)].data->add(data);

}

void Hash::HashTable::printTable()

{

std::cout<<"Хэш-таблица:"<<std::endl;

for (unsigned int i=0; i<Hash::M; i++)

if (!\_table[i].data->empty())

{

std::cout<<\_table[i].hashCode<<":"<<std::endl;

\_table[i].data->print(4);

}

}

bool Hash::HashTable::check()

{

bool res=true;

std::cout<<"Поиск одинаковых элементов в коллизиях:"<<std::endl;

for (unsigned int i=0; i<Hash::M && res; i++)

res=\_table[i].data->check();

std::cout<<"[OK]"<<std::endl;

return res;

}

* *hashtable.h*

#ifndef HASHTABLE\_H

#define HASHTABLE\_H

#include <string>

#include "list.h"

namespace Hash

{

const unsigned int M=256;

struct elementTable

{

int hashCode;

List::list<std::string>\* data;

};

class HashTable

{

public:

HashTable();

~HashTable();

int hashCode(std::string data);

void add(std::string data);

void printTable();

bool check();

private:

elementTable\* \_table;

};

}

#endif // HASHTABLE\_H

***Результат работы программы:***

1. ***Пересекающиеся множества***

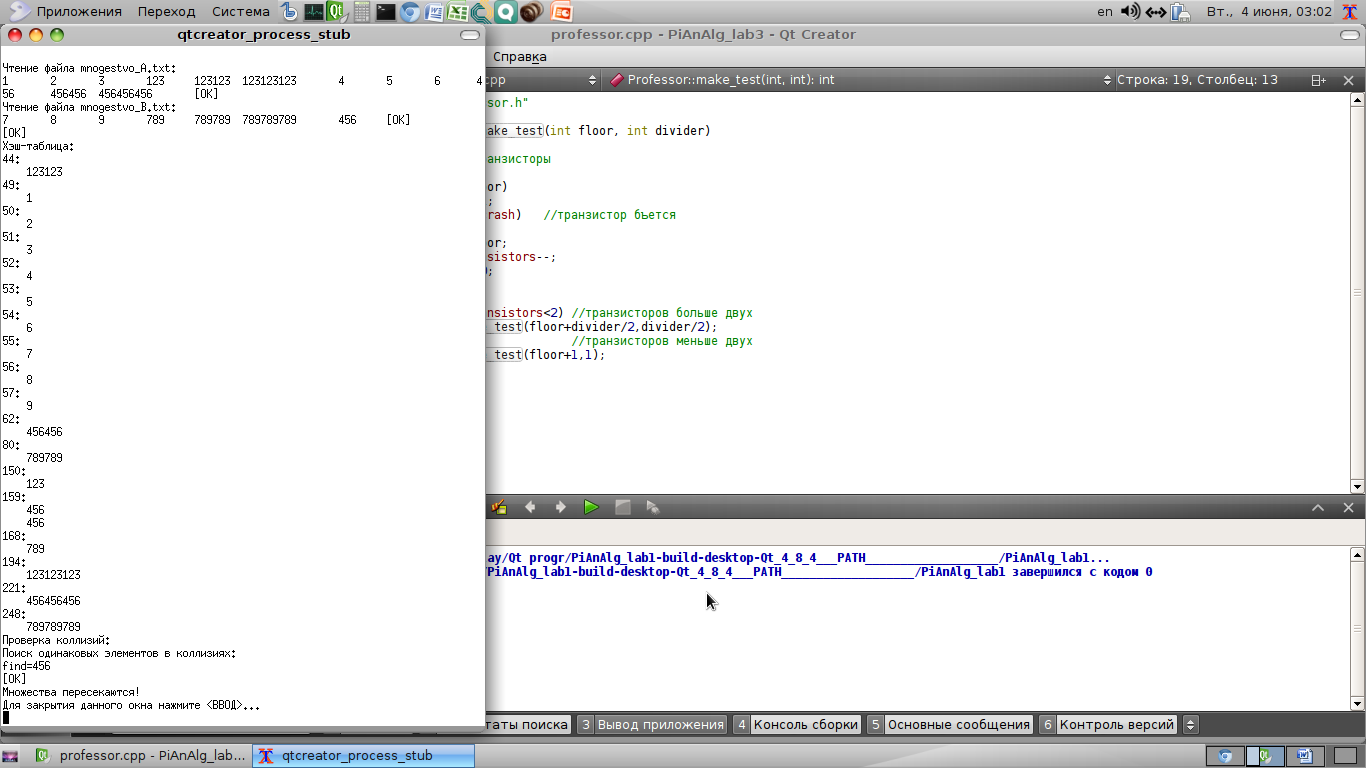
******

Рисунок 2 – результат работы программы (пересекающиеся множества)

1. ***Непересекающиеся множества***

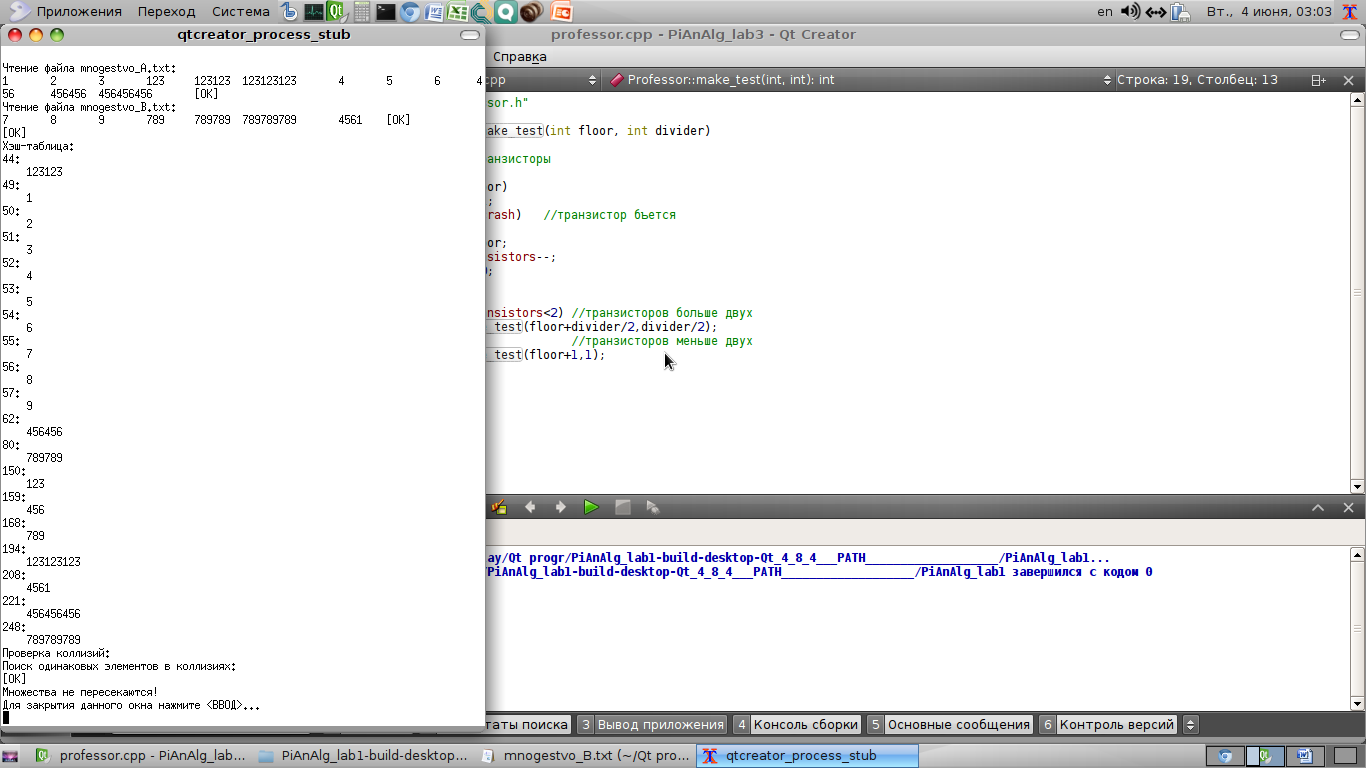
******

Рисунок 3 – результат работы программы (непересекающиеся множества)

***Вывод:*** В данной лабораторной работе я познакомился с hash – таблицами, их свойствами и реализовал на практике алгоритм открытого хэширования. С помощью поиска по хэш-таблице я сравнил два множества на пересечения.