**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра вычислительной техники**

**­­ОТЧЕТ**

**по лабораторной работе №1**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: Хэш-таблицы**

|  |  |  |
| --- | --- | --- |
| Студент гр. 5307 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Правиленко М. А. |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Колинько П. Г. |

Санкт-Петербург

2017

Содержание

[Цель работы 2](#_Toc475905575)

[1. Задание 3](#_Toc475905576)

[2. Оценка временной сложности 3](#_Toc475905577)

[3. Результат работы программы 5](#_Toc475905578)

[4. Вывод 6](#_Toc475905579)

[Приложение 7](#_Toc475905580)

[Исходный код программы 7](#_Toc475905581)

­­­­­­

# Цель работы

Изучить способ представления данных вида хэш-таблица на примере множеств.

# Задание

Составить алгоритм и написать программу на языке C/C++, реализующую создание и обработку множеств целых чисел, представленных в виде хэш-таблицы. Результатом работы программы является множество целых чисел, задаваемое формулой.

Формула индивидуального задания: A U B U C xor D \ E

Интервал значений: [0, 100]

Мощность множества: 26

## **Оценка временной сложности**

Ниже представлена оценка временной сложности для разных операций с множествами в виде хэш-таблиц:

Объединение U: в худшем *O(n2)*, в среднем *O(1)*

Xor: в худшем *O(n2)*, в среднем *O(1)*

Вычитание \: в худшем *O(n2)*, в среднем *O(1)*

В качестве хэш-функции была взята распространённая модель вида:

*h(x) = (a \* x + b) % m*

где a – простое число, близкое к 100 (97)

b – простое число, близкое к 1 (11)

m – размер таблицы

Для полного исключения коллизии на этапе генерации множеств, было решено взять размер хэш-таблицы, равный мощности множества. Генерация множества представлена на рис. 1-2.



Рис. 1. Генерация множества



Рис. 2. Генерация множества (продолжение)

# Результат работы программы

Результат работы программы представлен на рис. 3-4.



Рис. 3. Результат работы программы

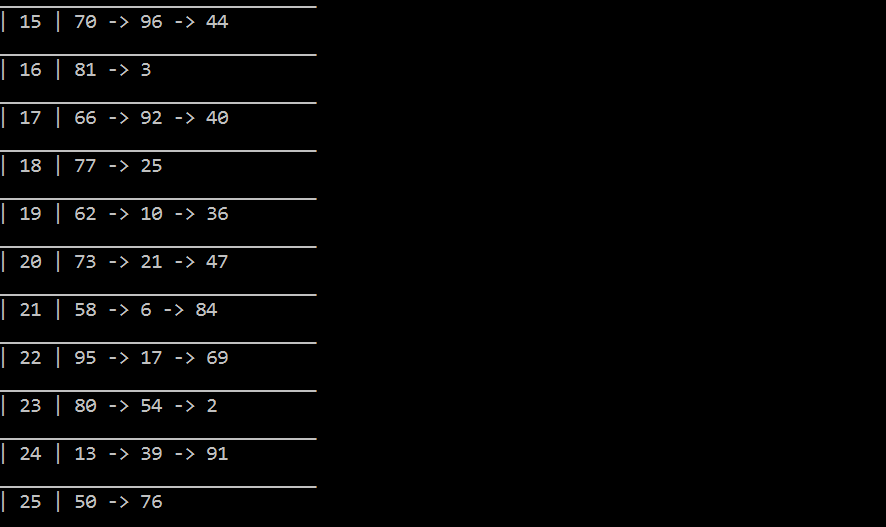


Рис. 4. Результат работы программы (продолжение)

## **Вывод**

В ходе работы были получены практические навыки работы с хэш-таблицами и реализованы различные операции на множествах. Как показывает практика, хэш-таблицы дают наибольшую эффективность при малой коллизии. Уменьшить коллизию можно несколькими способами, одними из которых являются увеличение размера хэш-таблицы и усложнение хэш-функции.

## **Приложение**

## **Исходный код программы**

Файл hash2.cpp

#include "stdafx.h"

#include "hash2.h"

#include <iomanip>

#include <iostream>

#include <conio.h>

#include <Windows.h>

using namespace std;

int Hash::func(int x)

{

return (97 \* x + 11) % modValue;

}

void Hash::Add(int key)

{

int adress = func(key);

if (cell[adress]->value == -1)

cell[adress]->value = key;

else

{

List \*newObj = new List;

newObj->value = key;

newObj->next = this->cell[adress];

this->cell[adress] = newObj;

}

}

Hash::Hash()

{

int key;

int adress;

bool repeat = 0;

cell = new List\*[size];

modValue = size;

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

{

cell[i] = new List;

cell[i]->next = nullptr;

cell[i]->value = -1;

}

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

{

do

{

repeat = 0;

key = rand() % 101;

adress = func(key);

cout << adress << " ";

if (cell[adress]->value == -1)

cell[adress]->value = key;

else

repeat = 1;

} while (repeat);

}

}

Hash::Hash(int x)

{

int key;

int adress;

bool repeat = 0;

cell = new List\*[size];

modValue = size;

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

{

cell[i] = new List;

cell[i]->next = nullptr;

cell[i]->value = -1;

}

}

Hash::~Hash()

{

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

{

List \*temp = cell[i];

while (temp->next)

{

List \*current = temp;

temp = temp->next;

delete[] current;

}

}

delete [] cell;

}

Hash &Hash::operator | (const Hash & B)const

{

bool canAdd = 1;

Hash \*C = new Hash(0);

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

{

List \*temp = this->cell[i];

while (temp != nullptr)

{

C->Add(temp->value);

temp = temp->next;

}

temp = B.cell[i];

while (temp != nullptr)

{

canAdd = 1;

List \*tempC = C->cell[i];

while (tempC != nullptr)

{

if (temp->value == tempC->value)

canAdd = 0;

tempC = tempC->next;

}

if (canAdd)

C->Add(temp->value);

temp = temp->next;

}

}

return \*C;

}

Hash &Hash::operator / (const Hash & B)const

{

Hash \*C = new Hash(0);

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

{

List\* xB = B.cell[i];

List\* xA = this->cell[i];

List \*temp;

while (xA)

{

temp = xB;

bool flag = 1;

while (temp&&flag)

{

if (xA->value == temp->value)

flag = 0;

temp = temp->next;

}

if (flag)

C->Add(xA->value);

xA = xA->next;

}

}

return \*C;

}

Hash &Hash::operator ^ (const Hash & B)const

{

Hash \*C = new Hash(0);

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

{

List\* xB = B.cell[i];

List\* xA = this->cell[i];

while (xB)

{

bool canAdd = 1;

xA = this->cell[i];

while (xA)

{

if (xB->value == xA->value)

canAdd = 0;

xA = xA->next;

}

if (canAdd)

C->Add(xB->value);

xB = xB->next;

}

xA = this->cell[i];

while (xA)

{

bool canAdd = 1;

List\* xB = B.cell[i];

while (xB)

{

if (xA->value == xB->value)

canAdd = 0;

xB = xB->next;

}

if (canAdd)

C->Add(xA->value);

xA = xA->next;

}

}

return \*C;

}

void Hash::operator = (const Hash & B)

{

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

{

List \*temp = B.cell[i];

while (temp != nullptr)

{

this->Add(temp->value);

temp = temp->next;

}

}

}

void Hash::outHash()

{

system("cls");

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

{

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout << "| " << i << " | ";

List \*temp = cell[i];

while (temp != nullptr)

{

cout << temp->value;

if (temp->next != nullptr)

cout << " -> ";

temp = temp->next;

}

cout << "\n";

}

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(nullptr));

Hash A, B, C, D, E, F(0), G(0), H;

F = A|B|C^D/E;

F.outHash();

\_getch();

return 0;

}

Файл hash2.h

#pragma once

class List

{

public:

int value;

List \*next;

};

class Hash

{

private:

List \*\*cell;

const int size=26;

const int power=26;

int modValue;

public:

Hash();

Hash(int);

~Hash();

Hash &operator | (const Hash&)const;

Hash &operator / (const Hash&)const;

Hash &operator ^ (const Hash&)const;

void operator = (const Hash&);

void outHash();

void Add(int);

int func(int);

};