**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

отчет

**по лабораторной работе №3**

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

Тема: «Хэш-таблицы**»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 7307 | Торопов В.А. |  |
| Преподаватель | Колинько П.Г. |  |

Санкт-Петербург

2019

Оглавление

[Цель работы 3](#__RefHeading___Toc188_4150899979)

[Задание 3](#__RefHeading___Toc190_4150899979)

[Обоснование по выбору размера хэш-таблицы и коэффициентов хэш-функции 3](#__RefHeading___Toc302_2376616454)

[Оценка временной сложности 3](#__RefHeading___Toc967_306870553)

[Результаты работы программы 3](#__RefHeading___Toc969_306870553)

[Код программы 6](#__RefHeading___Toc306_2376616454)

[Вывод 9](#__RefHeading___Toc308_2376616454)

# Цель работы

Научиться работать с хэш-таблицами и придумывать хэш-функции.

# Задание

Составить и отладить программу для вычисления шестого множества по

пяти заданным, представленным в форме хэш-таблиц. Элементы множеств

— целые числа из интервала [0, 100].

Формула для вычислений: A \ (B ∩ C) ∪ D ⊕ E;

средняя мощность множества: 10.

# Обоснование по выбору размера хэш-таблицы и коэффициентов хэш-функции

Размер хэш-таблицы равен трехкратной мощности множества. Это обусловлено тем, что в таком случае вероятность коллизии меньше 0,33.

Хэш — функция выбрана такая: h(x) = (a \* x + b) % m,

где m – размер таблицы, а a и b — простые числа.

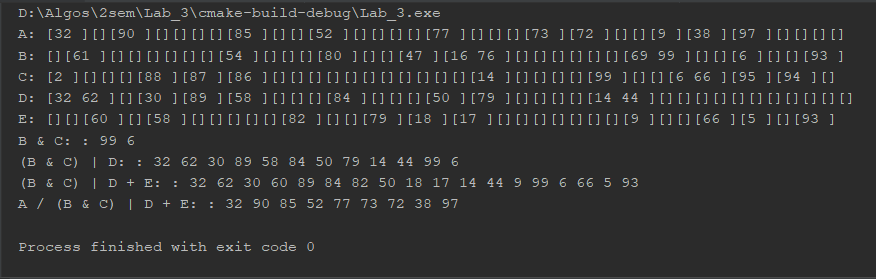
Я взял a = 29, b = 2, m = мощность множества \* 3. Такой выбор обеспечивает равномерное использование всех ячеек таблицы в большинстве практических случаев.

# **Оценка временной сложности**

Если таблица правильно построена и не переполнена, проверка принадлежности элемента множеству, а также вставка и удаление элемента выполняются в ней за постоянное время, примерно такое же, как и в массиве битов. За постоянное время будут выполняться и двуместные операции над множествами: объединение, пересечение и разность.

В худшем случае, когда все данные попадают в одну или несколько ячеек таблицы, образовав неупорядоченные списки, справедливы следующие оценки временной сложности: O(n) — для поиска и удаления элемента; O(n^2) — для двуместной операции над множествами.

# **Результаты работы программы**



# **Код программы**

Hash\_Table.h

1. #ifndef LAB\_3\_HASH\_TABLE\_H  
   #define LAB\_3\_HASH\_TABLE\_H  
   #endif //LAB\_3\_HASH\_TABLE\_H  
   #include <set>  
   #include <random>  
   #include <time.h>  
   #include <string.h>  
     
   **enum**{*N* = 10, *max\_num* = 100};  
   **using namespace** std;  
     
   **struct** list{  
    set<**int**> table;  
   };  
     
   **class** Hash{  
    **char** Name;  
    list H\_table[3\**N*];  
   **public**:  
    **void** create(**char** Name\_set);  
    **bool** proof(**int** a);  
    **void** Out();  
    **void** Out(string \_name);  
    Hash **const operator**& (**const** Hash B);  
    Hash **operator** &= (Hash B);  
    Hash **const operator** | (**const** Hash B);  
    Hash **operator** |= (Hash B);  
    Hash **const operator** + (**const** Hash B);  
    Hash **operator** += (Hash B);  
    Hash **const operator** / (**const** Hash B);  
    Hash **operator** /= (Hash B);  
   };  
     
   **void** Hash::create(**char** Name\_set){  
    Name = Name\_set;  
    **for** (**int** i = 0; i < *N*; i++){  
    **int** a = rand() % *max\_num*;  
    **while**(!proof(a)){  
    a = rand() % *max\_num*;  
    }  
    }  
   };  
     
   **bool** Hash::proof(**int** a){  
    **int** func = (29\*a + 2) % (3\**N*);  
    **if**(H\_table[func].table.find(a) == H\_table[func].table.**end**()){  
    H\_table[func].table.insert(a);  
    **return true**;  
    } **else return false**;  
   }  
     
   **void** Hash::Out(){  
    cout << Name << ": ";  
    **for** (**int** i = 0; i < 3\**N*; i++){  
    cout << "[";  
    **for**(**auto** j:H\_table[i].table){  
    cout << j << " ";  
    }  
    cout << "]";  
    }  
    cout << endl;  
   }  
     
   **void** Hash::Out(string \_name) {  
    cout << \_name << ": ";  
    **for** (**int** i = 0; i < 3\**N*; i++){  
    //cout << "[";  
    **for**(**auto** j:H\_table[i].table){  
    cout << j << " ";  
    }  
    //cout << "]";  
    }  
    cout << endl;  
   }  
     
   Hash **const** Hash::**operator**&(**const** Hash B) {  
    Hash R(\***this**);  
    **return** R &= B;  
   }  
     
   Hash Hash::**operator**&=(Hash B) {  
    **for** (**int** i = 0; i < 3\**N*; i++){  
    **for**(**auto** j: (\***this**).H\_table[i].table){  
    **if**(B.H\_table[i].table.find(j) == B.H\_table[i].table.**end**() || B.H\_table[i].table.size() == 0){  
    (\***this**).H\_table[i].table.erase(j);  
    }  
    }  
    }  
    **return** (\***this**);  
   }  
     
   Hash **const** Hash::**operator**|(**const** Hash B) {  
    Hash R(\***this**);  
    **return** R |= B;  
   }  
     
   Hash Hash::**operator** |= (Hash B) {  
    **for** (**int** i = 0; i < 3\**N*; i++){  
    **for**(**auto** j: B.H\_table[i].table){  
    (\***this**).H\_table[i].table.insert(j);  
    }  
    }  
    **return** (\***this**);  
   }  
     
   Hash **const** Hash::**operator** /(**const** Hash B) {  
    Hash R(\***this**);  
    **return** R /= B;  
   }  
     
   Hash Hash::**operator** /= (Hash B) {  
    **for** (**int** i = 0; i < 3\**N*; i++){  
    **for**(**auto** j: B.H\_table[i].table){  
    (\***this**).H\_table[i].table.erase(j);  
    }  
    }  
    **return** (\***this**);  
   }  
     
   Hash **const** Hash::**operator**+(**const** Hash B) {  
    Hash R(\***this**);  
    **return** R += B;  
   }  
     
   Hash Hash::**operator**+=(Hash B) {  
    Hash A = (\***this**) | B;  
    Hash C = (\***this**) & B;  
    **return** A / C;  
   }

Main.cpp

#include <iostream>  
#include "Hash\_Table.h"  
  
**int** main(){  
 srand(time(0));  
 Hash A, B, C, D, E, R, B\_and\_C;  
 R.create('R');  
 A.create('A');  
 B.create('B');  
 C.create('C');  
 D.create('D');  
 E.create('E');  
 A.Out();  
 B.Out();  
 C.Out();  
 D.Out();  
 E.Out();  
 R = B & C;  
 R.Out("B & C: ");  
 R = R | D;  
 R.Out("(B & C) | D: ");  
 R = R + E;  
 R.Out("(B & C) | D + E: ");  
 R = A / E;  
 R.Out("A / (B & C) | D + E: ");  
 **return** 0;  
}

# **Вывод**

1. В данной работе мы изучили хэш – таблицы, работу с ними, грамотное создание хэш – функций.
2. Сами хэш — таблицы - это хороший способ хранения данных, с возможностью быстрого их добавления, удаления, считывания, поиск и т. д.
3. Благодаря использованию set сильно облегчилась работа с классом.