МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра программной инженерии

**ОТЧЕТ**

по лабораторной работе №2

на тему: «Структурное тестирование программного обеспечения»

по дисциплине: «Качество и тестирование программного обеспечения»

Вариант 7

Выполнила: Карпикова С.П.

Институт приборостроения, автоматизации и информационных технологий

Направление: 09.03.04 «Программная инженерия»

Группа: 71ПГ

Проверили: Ужаринский А.Ю., Конюхова О.В.

Отметка о зачете:

Дата: «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2019 г.

Орел, 2020 г.

**Задание:** Хеш-таблица, не перетирающая элементы при вводе значений с совпадающим ключом, а хранящая список таких элементов и, соответственно, возвращающая их методом get. Метод – двойное хеширование.

Двойным хешированием называется метод разрешения коллизий в хеш-таблице с открытой адресацией, при котором интервал между ячейками фиксирован, как при линейном пробировании, но, в отличие от него, размер интервала вычисляется второй, вспомогательной хеш-функцией, а значит, может быть различным для разных ключей.

**Алгоритм вставки в хэш таблицу**

1. Проверить переполненность таблицы. Если таблица переполнена, то завершить алгоритм, иначе перейти к пункту 2.
2. Вычислить начальный хэш-код h(xk) от основной хэш-функции.
3. Если соответствующая хэш-коду ячейка еще не занята, разместить ключ в эту ячейку и увеличить текущий размер хэш-таблицы, завершить алгоритм, иначе перейти к пункту 4.
4. Вычислить еще один хэш-код g(xk)от дополнительной хэш-функции.
5. Добавить второй хэш-код к первому, взять остаток от деления полученного кода на размер хэш-таблицы N и вернуться к шагу 3.

Вторая хэш-функция должна обладать следующими свойствами:

* Никогда не возвращать значение 0 (иначе алгоритм зациклится).
* Не должна зависеть от основной хэш-функции (иначе не будет эффекта).

Такой подход существенно уменьшает вероятность формирования кластеров и уменьшает количество коллизий.

**Ход работы**

**Программа на С++:**

#include <iostream>

#define TABLE\_SIZE 20

#define PRIME 7

using namespace std;

class DoubleHash

{

int \*hashTable;

int curr\_size;

public:

bool isFull()

{

return (curr\_size == TABLE\_SIZE);

}

int hash1(int key)

{

return (key % TABLE\_SIZE);

}

int hash2(int key)

{

return (PRIME - (key % PRIME));

}

DoubleHash()

{

hashTable = new int[TABLE\_SIZE];

curr\_size = 0;

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

hashTable[i] = -1;

}

void insert(int key)

{

if (isFull())

return;

int index = hash1(key);

if (hashTable[index] != -1)

{

int index2 = hash2(key);

int i = 1;

while (1)

{

int newIndex = (index + i \* index2) %

TABLE\_SIZE;

if (hashTable[newIndex] == -1)

{

hashTable[newIndex] = key;

break;

}

i++;

}

}

else

hashTable[index] = key;

curr\_size++;

}

void get()

{

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

{

if (hashTable[i] != -1)

cout << i << " --> "

<< hashTable[i] << endl;

else

cout << i << endl;

}

}

};

int main()

{

int a[] = { 19, 19, 27, 36, 10, 64, 10, 64, 64, 64, 64, 10, 27, 27, 27 };

int n = sizeof(a) / sizeof(a[0]);

DoubleHash h;

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

h.insert(a[i]);

h.get();

system("pause");

return 0;

}

**Алгоритм вставки в хэш таблицу (insert)**

На основе текста программы метода insert сформирован потоковый граф, представленный на рисунке 1.

**Цикломатическая сложность**

1. V(G) = 4 региона;
2. V(G) = 13 дуг – 11 узлов + 2 = 4;
3. V(G) = 3 предикатных узла + 1 = 4.

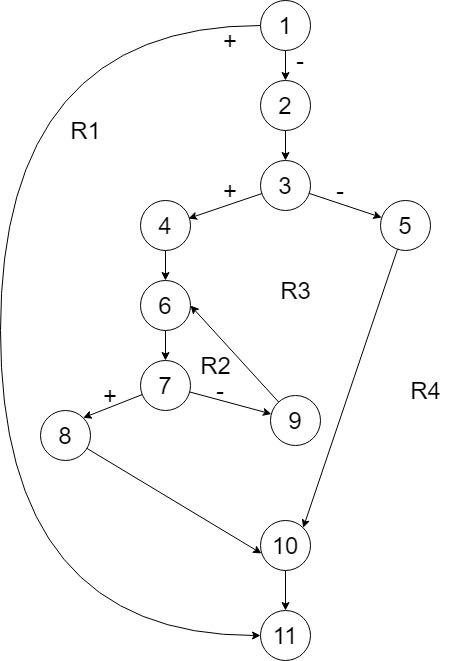


Рисунок 1 – Потоковый граф метода insert.

**Базовое множество**

**Путь 1:** 1-11

**Путь 2:** 1-2-3-5-10-11

**Путь 3:** 1-2-3-4-6-7-…-10-11

**Тестовые варианты**

**ТВ1:**

Исх.данные: TABLE\_SIZE = 5; curr\_size = 5; key = 10; hashTable = {1, 2, 3, 4, 5}

Ожид.рез-т: возврат функции

**ТВ2:**

Исх.данные: TABLE\_SIZE = 5; curr\_size = 0; key = 10; hashTable = {}

Ожид.рез-т: отсутствие коллизии, вставка по первой хеш-функции

**ТВ3:**

Исх.данные: TABLE\_SIZE = 5; curr\_size = 3; key = 10; hashTable = {1, 2, 10}

Ожид.рез-т: коллизия, нахождение индекса и вставка по второй хеш-функции

**Алгоритм возвращения элементов таблицы (get)**

На основе текста программы метода insert сформирован потоковый граф, представленный на рисунке 1.

**Цикломатическая сложность**

1. V(G) = 3 региона;
2. V(G) = 7 дуг – 6 узлов + 2 = 3;
3. V(G) = 2 предикатных узла + 1 = 3.

**Базовое множество**

**Путь 1:** 1-6

**Путь 2:** 1-2-…5-1-6

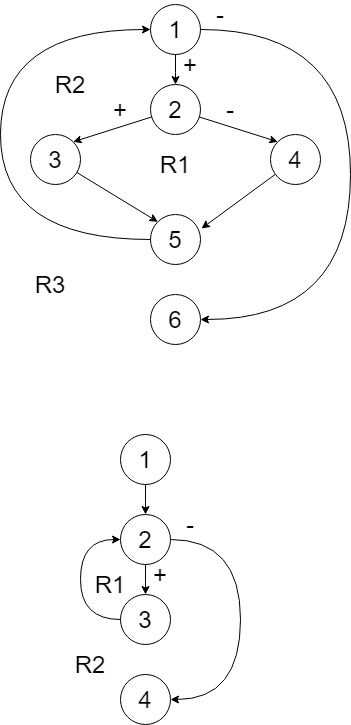


Рисунок 2 – Потоковый граф метода get.

**Тестовые варианты**

**ТВ1:**

Исх.данные: TABLE\_SIZE = 5; hashTable = {}

Ожид.рез-т: вывод пустой таблицы на экран

**ТВ2:**

Исх.данные: TABLE\_SIZE = 5; hashTable = {1, 2, 10}

Ожид.рез-т: вывод таблицы на экран, с заполненными ячейками