МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

И ПРОГРАММНОЙ ИНЖЕНЕРИИ (КАФЕДРА №43)

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Старший преподаватель |  |  |  | Е.В. Павлов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №4 |
| «МЕТРИКА ОЦЕНКИ СЛОЖНОСТИ ПОТОКА УПРАВЛЕНИЯ» |
| по дисциплине: «МЕТРОЛОГИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ» |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. | 4631 |  |  |  | С.А. Гришин |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург   
2018

1. **Цель работы**Целью данной работы является изучение способов оценки сложности программного обеспечения на основе метрики Маккейба.

**2. Задание**

Оценить структурную сложность программы по метрике Маккейба:

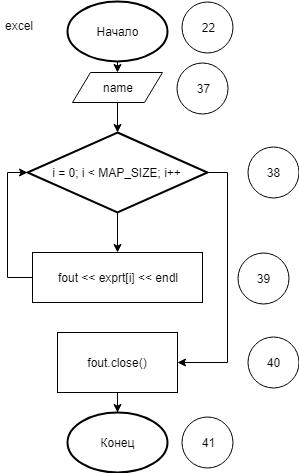
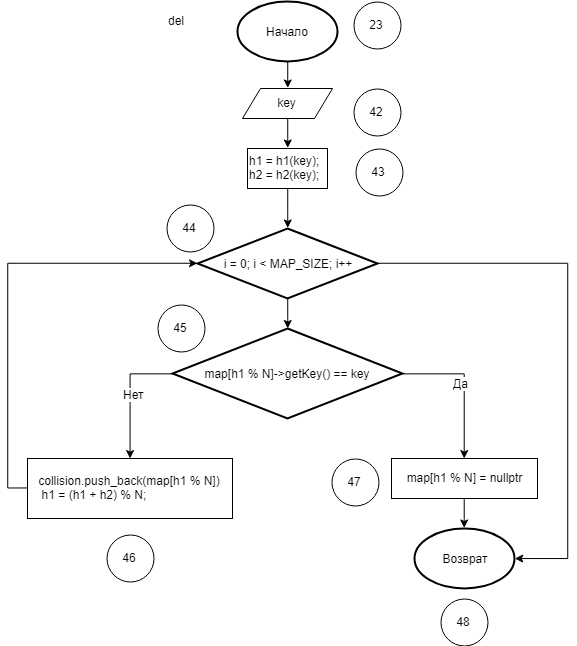
* Начертить блок-схему алгоритма программы;
* По блок-схеме построить граф потока управления;
* Выделить линейно-независимые маршруты и циклы;
* Вычислить цикломатическую сложность;
* Построить матрицы смежности и достижимости.

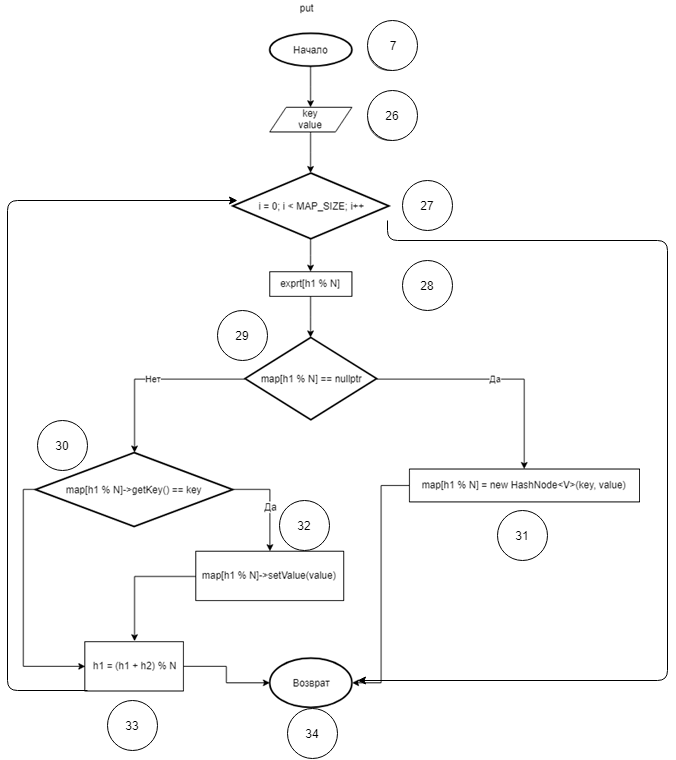
В качестве варианта задания выбрана программа, написанная на языке программирования С++, хеш-таблицу с двойным хешированием.

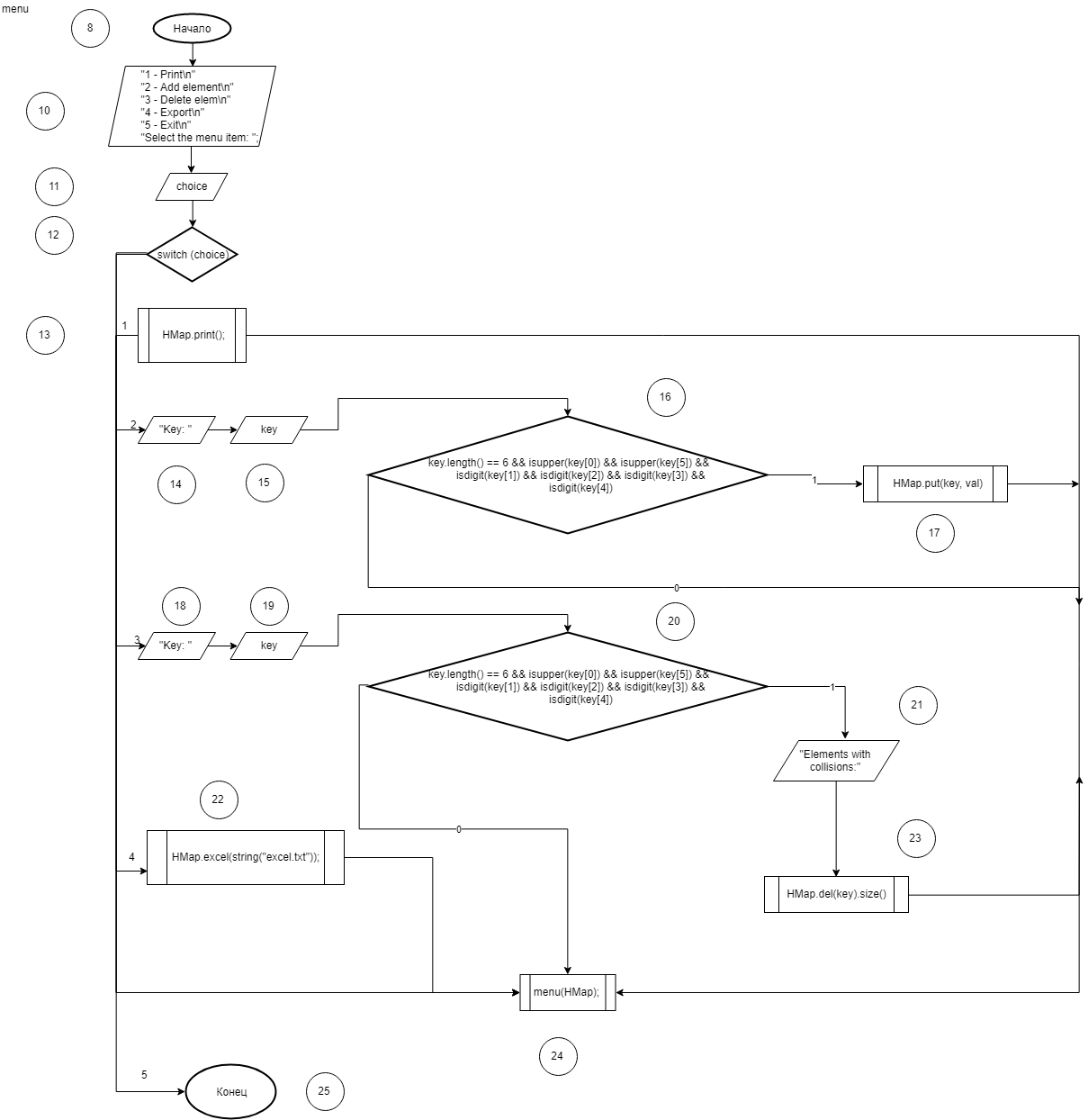
**3. Оценка сложности потока управления**

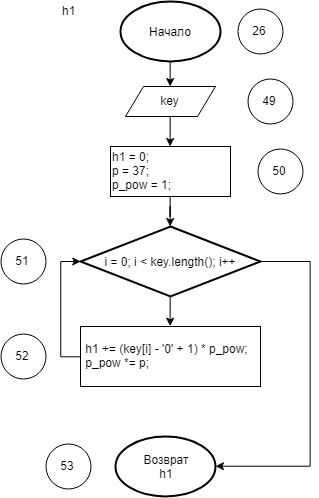
**3.1 Блок-схема**

|  |  |
| --- | --- |
| C:\Users\sun\AppData\Local\Microsoft\Windows\INetCache\Content.Word\main.png | C:\Users\sun\AppData\Local\Microsoft\Windows\INetCache\Content.Word\h2.png |

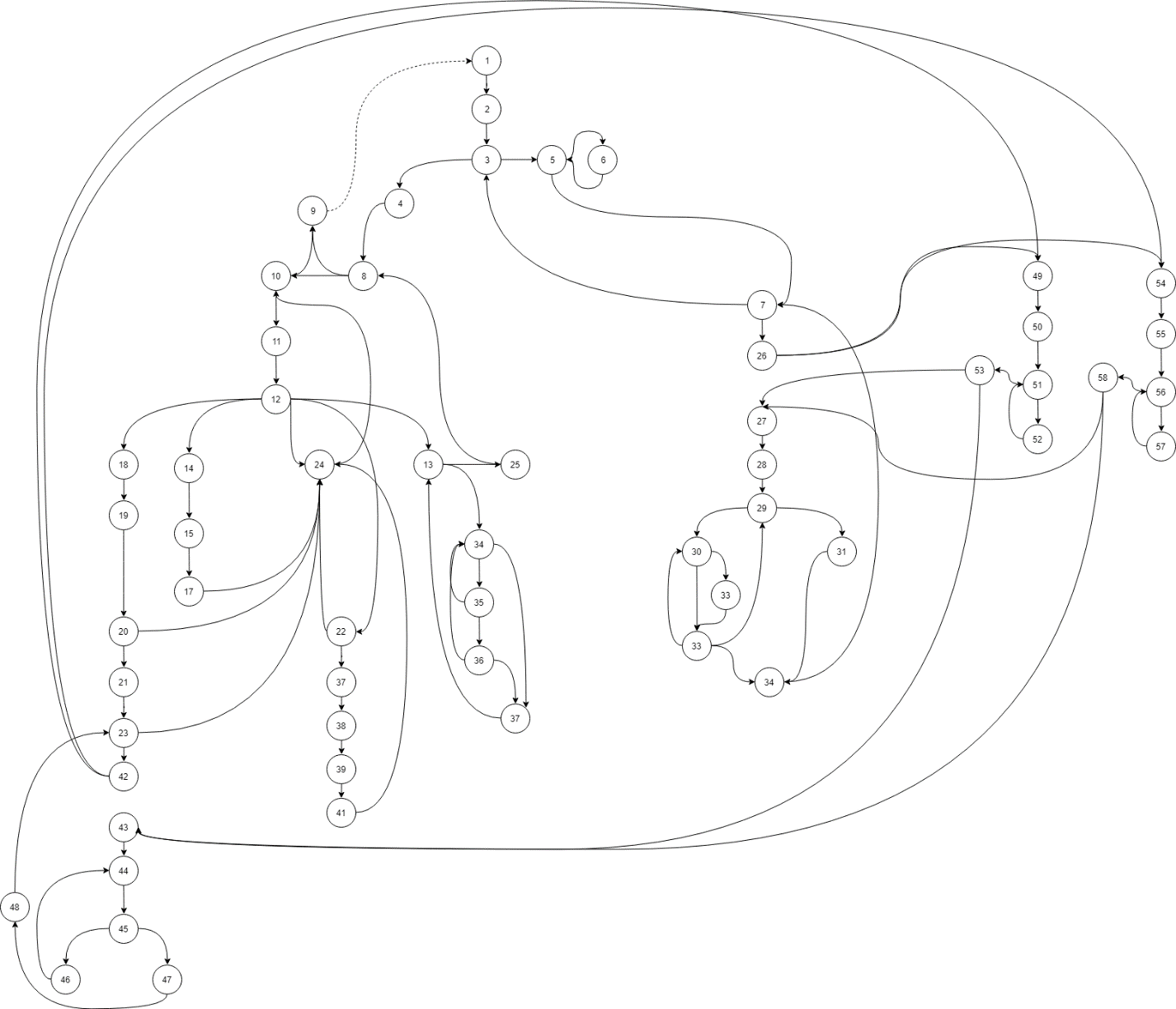




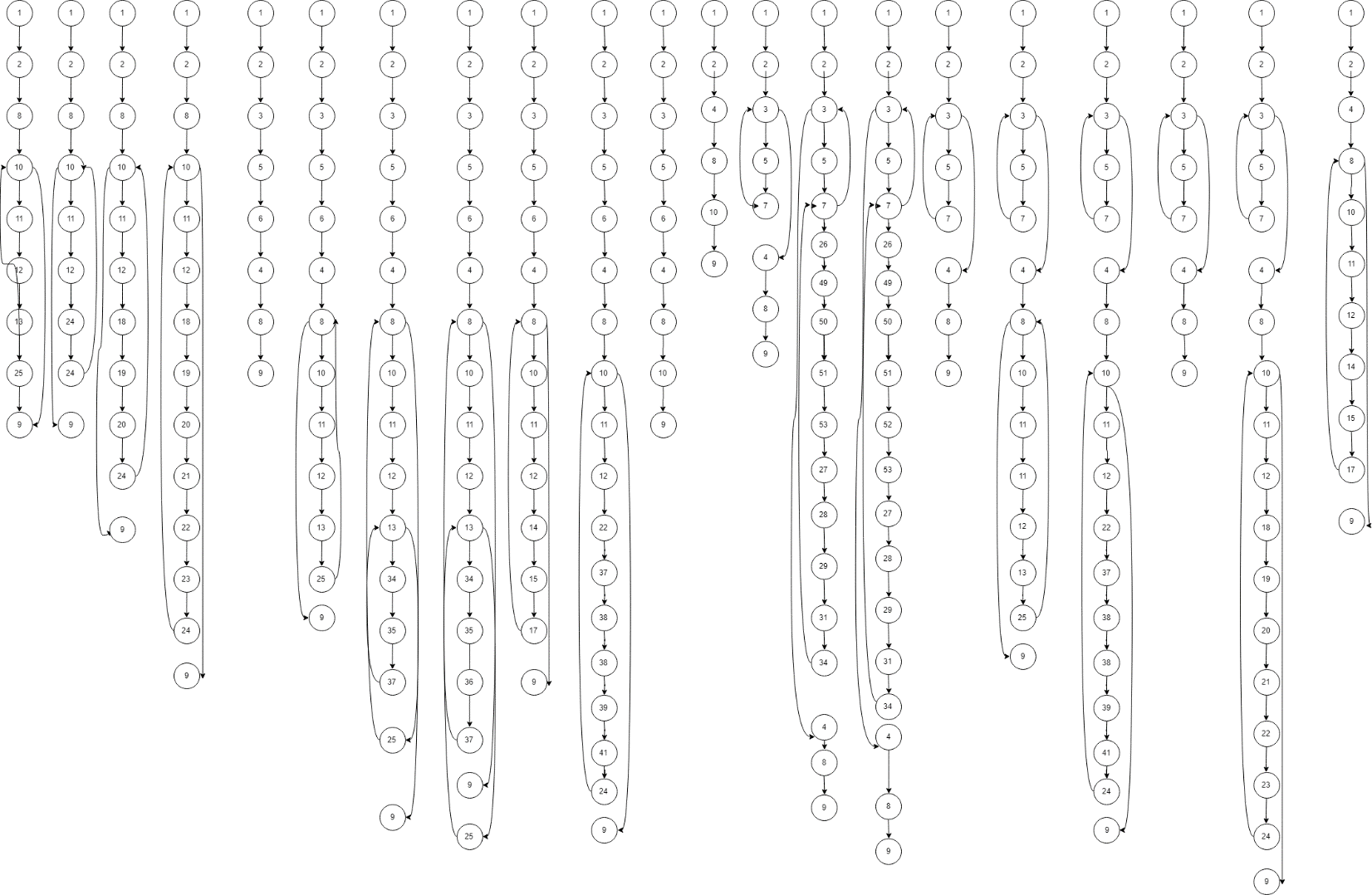




**3.2 Граф потока данных**



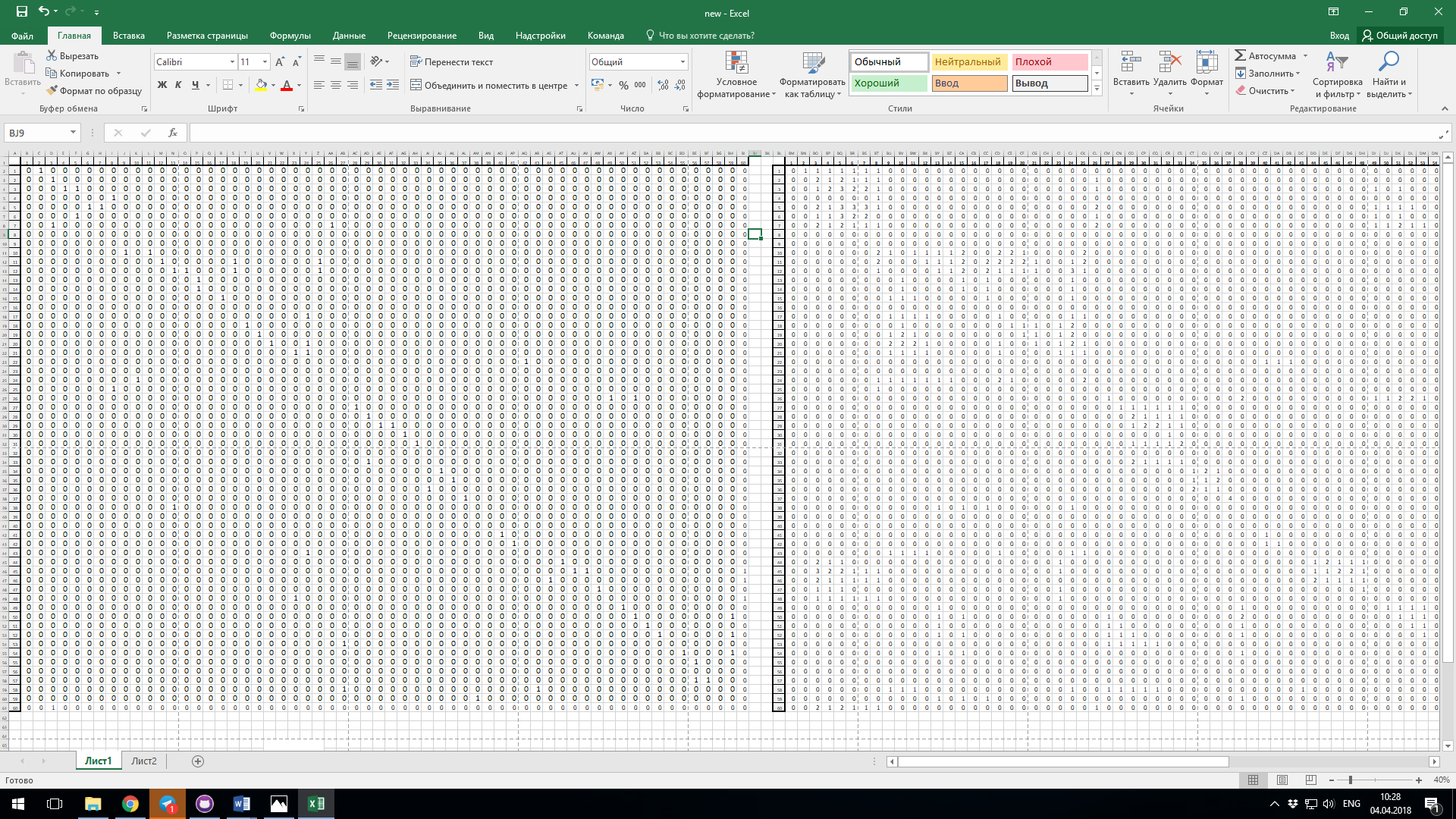
**3.3 Линейно независимые маршруты и циклы**



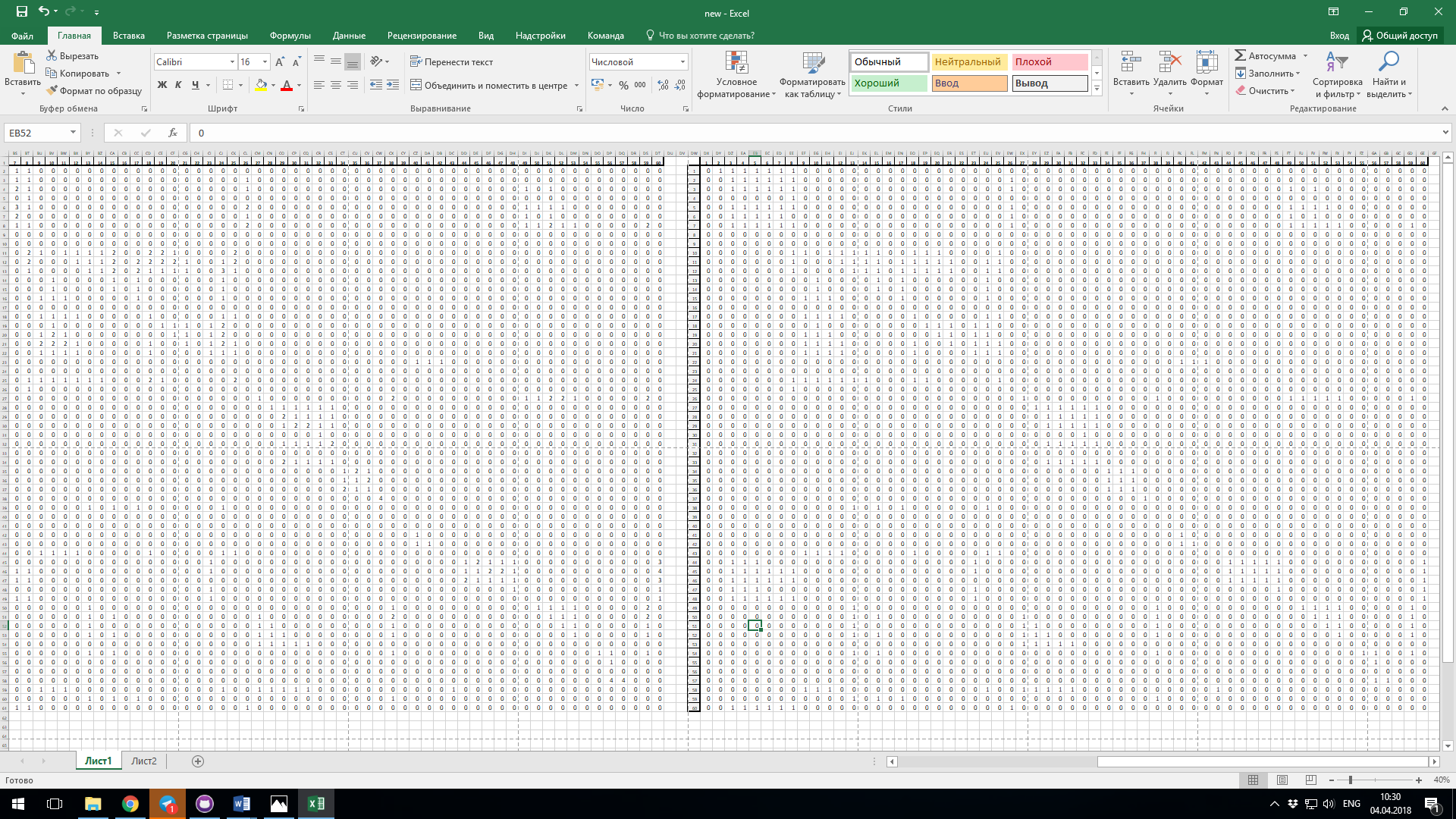
**3.4 Расчет цикломатической сложности (метрика Маккейба)**

M(G) = E – N + P = 78 - 58 + 1 = 21

**4.1 Матрица смежности**

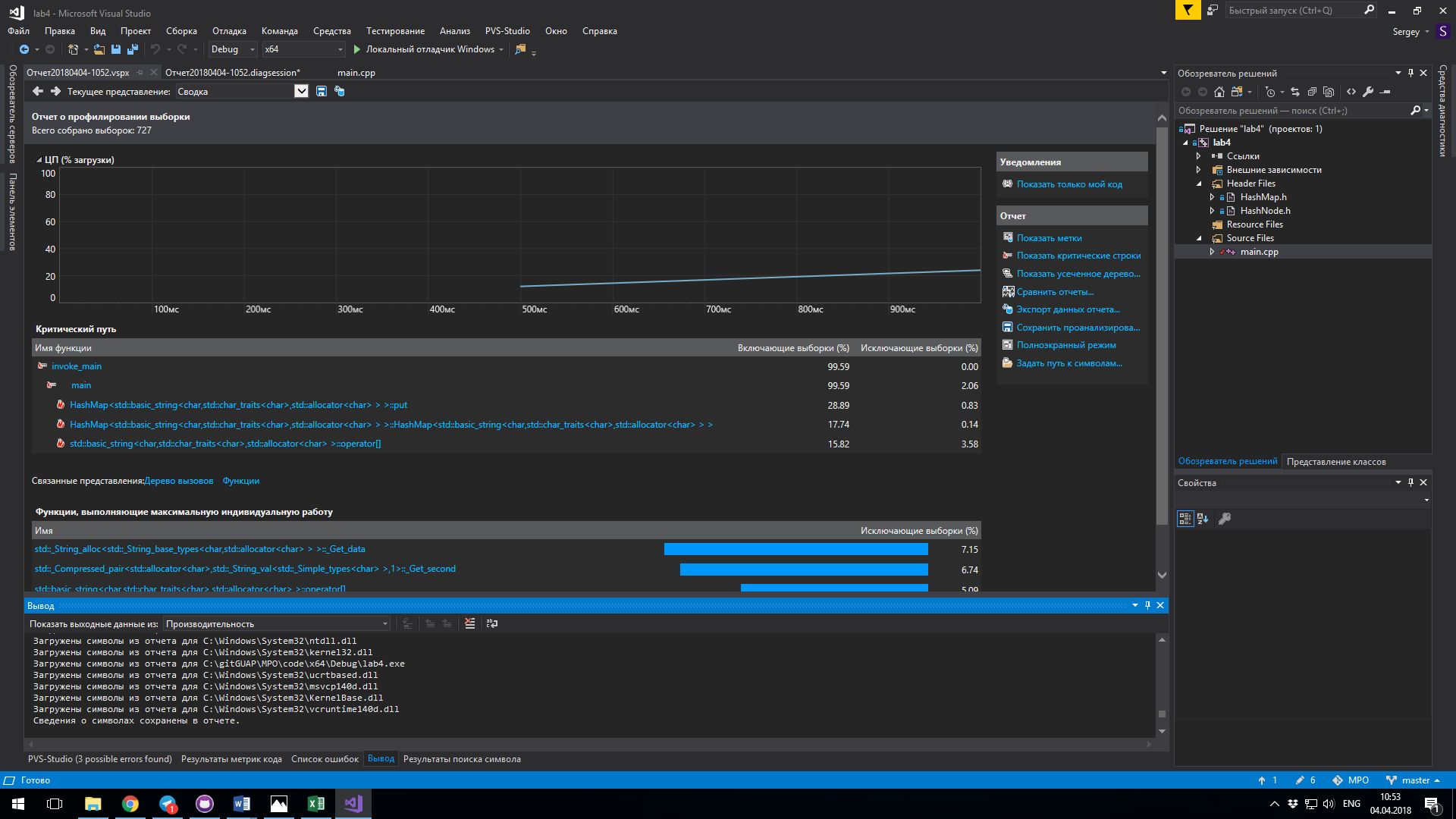


**4.2 Матрица достижимости**

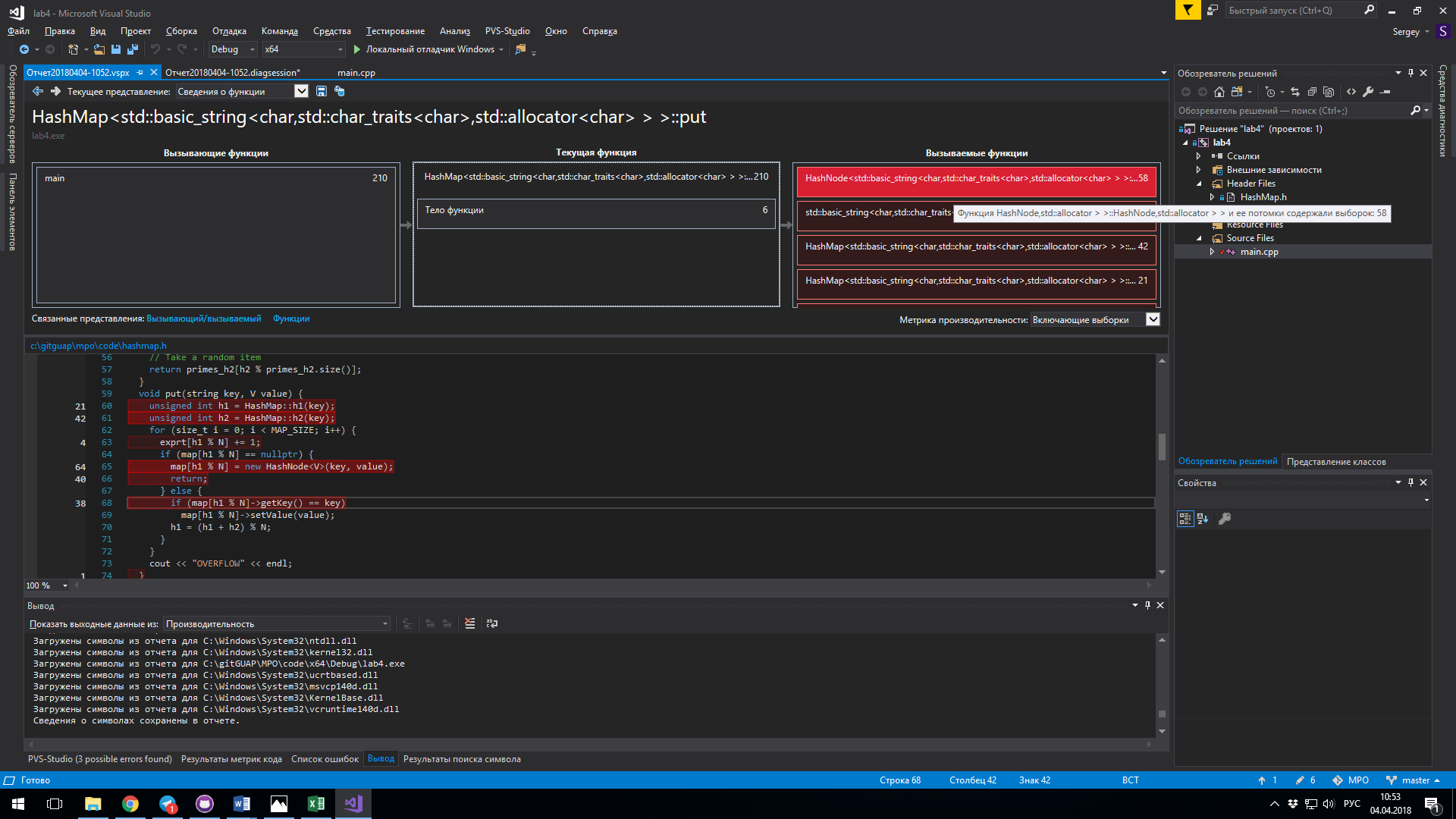


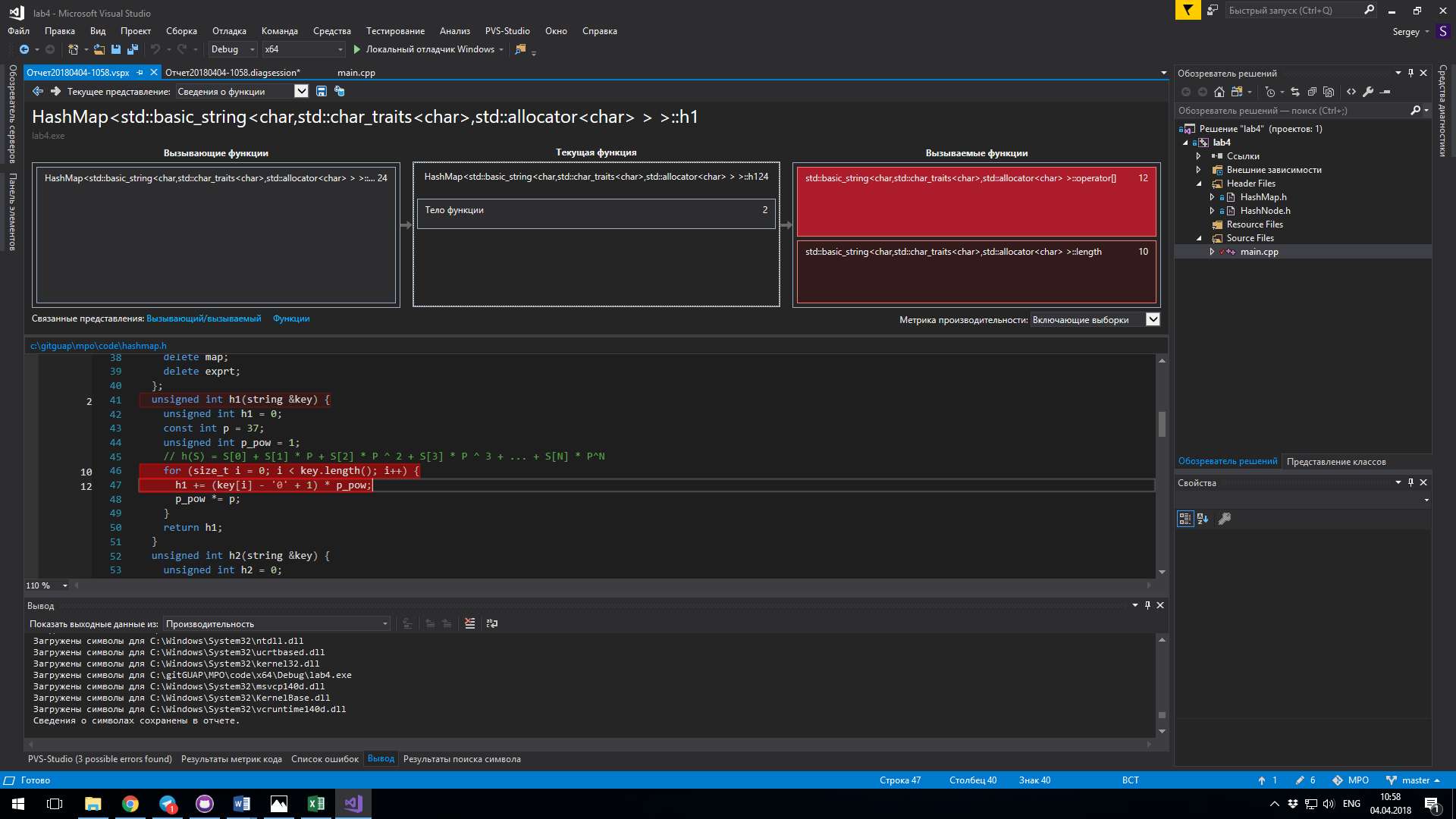
Вычислялась по форуме

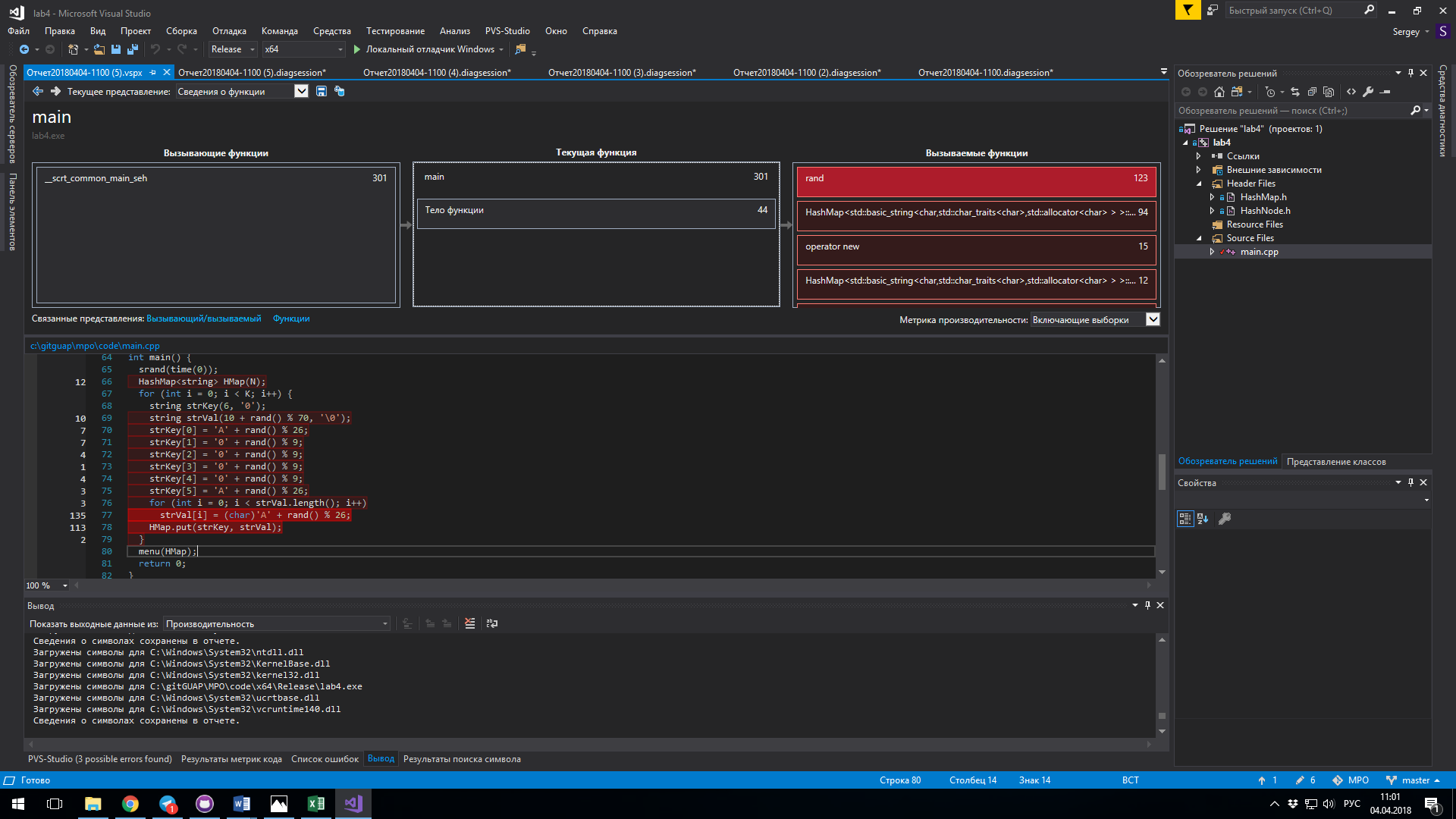
**5. Результат профилирования кода**

Так как у нас происходит тестирование хеш таблицы, а не ввода-вывода, то было за решено протестировать полную запись таблицы (25000 элементов) без вывода их на экран.

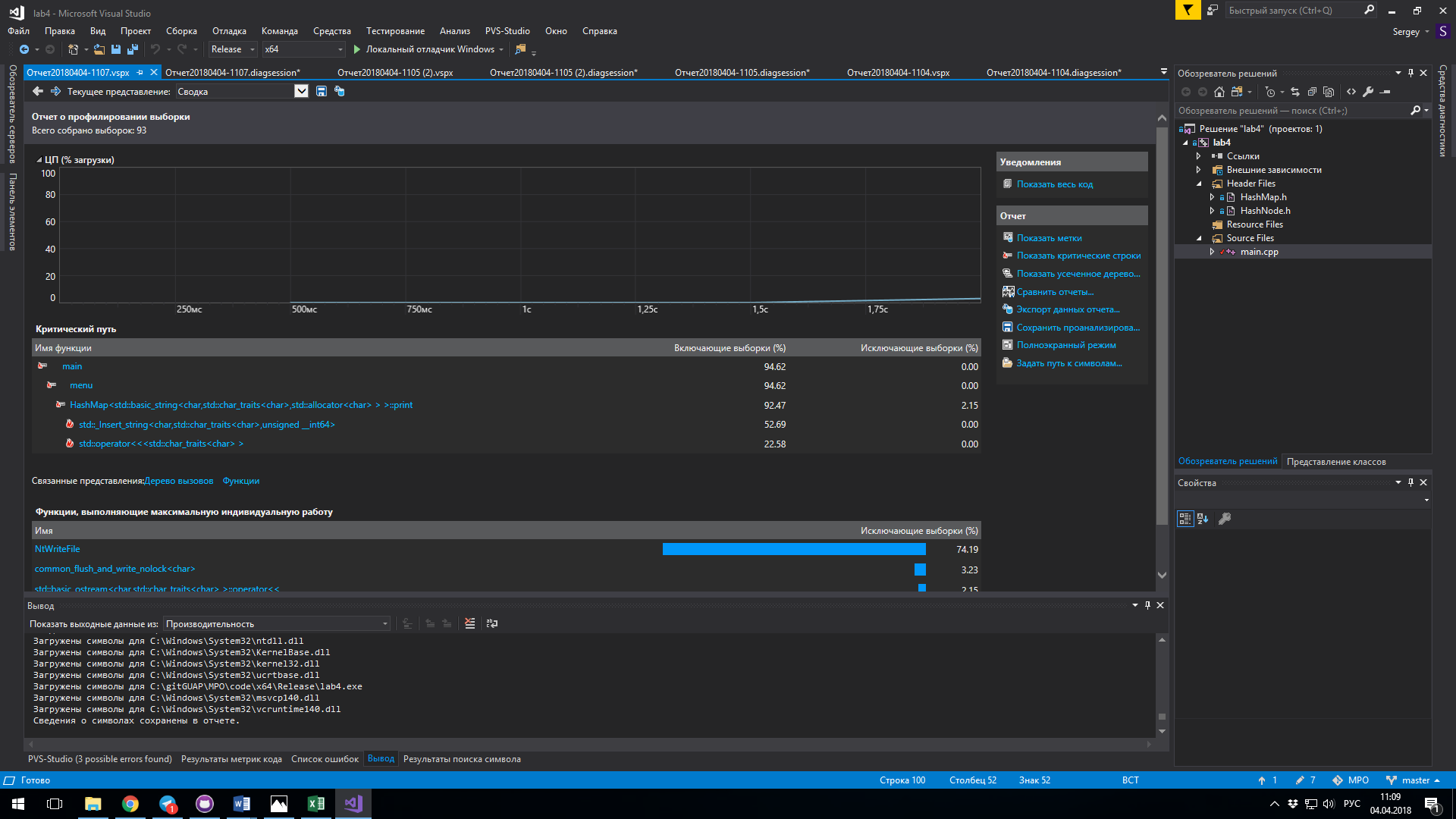
Как мы видим, выделение памяти довольно дорогая операция







Если вывести на экран 500 элементов



**6. Выводы**

В результате выполнения данной лабораторной работы были изучены методы оценки сложности потока управления на основе метрики Маккейба.

Произведен расчет цикломатической сложности программы и построены матрицы смежности и достижимости.

Цикломатическое число Маккейба совпадает с числом линейно-независимых маршрутов и циклов, что свидетельствует о правильно выполненных расчетах.

Произведено профилирование кода и выявлены следующие горячие точки:

- Долгое выделение памяти.

- Доступ к элементу строки тоже не так быстр

- По сравнению с выводом на экран и генерацией данных для записи, добавление элемента в таблицу не самая медленная вещь.

**7. Использованные источники**1. Метрики кода и их практическая реализация [Электронный ресурс]. – Электронные данные – ООО СМ-Консалт (СМК), 2004-2016. – URL: <http://cmcons.com/articles/CC_CQ/dev_metrics/mertics_part_1/>  
  
2. Значения метрик кода [Электронный ресурс]: документация для Visual Studio 2017. – Электронные данные – Microsoft, 2017. – URL: [https://msdn.microsoft.com/ru- ru/library/bb385914.aspx](https://msdn.microsoft.com/ru-ru/library/bb385914.aspx)

**Приложение**

MAIN.CPP

#include "HashMap.h"

#include <iostream>

#include <string>

#include <time.h>

using namespace std;

const int N = 2500;

const int K = 3;

void menu(HashMap<string> &HMap) {

cout << endl

<< "1 - Print\n"

<< "2 - Add element\n"

<< "3 - Delete elem\n"

<< "4 - Export\n"

<< "5 - Exit\n"

<< "Select the menu item: ";

int choice;

cin >> choice;

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

switch (choice) {

case 1: {

HMap.print();

menu(HMap);

break;

}

case 2: {

string key;

string val;

cout << "Key: ";

getline(cin, key);

if (key.length() == 6 && isupper(key[0]) && isupper(key[5]) &&

isdigit(key[1]) && isdigit(key[2]) && isdigit(key[3]) &&

isdigit(key[4])) {

cout << "Value: ";

getline(cin, val);

HMap.put(key, val);

}

menu(HMap);

break;

}

case 3: {

string key;

cout << "Key: ";

getline(cin, key);

if (key.length() == 6 && isupper(key[0]) && isupper(key[5]) &&

isdigit(key[1]) && isdigit(key[2]) && isdigit(key[3]) &&

isdigit(key[4])) {

cout << "Elements with collisions: " << HMap.del(key).size() << endl;

}

menu(HMap);

break;

}

case 4: {

HMap.excel(string("excel.txt"));

cout << "OK" << endl;

menu(HMap);

break;

}

case 5: {

break;

}

default:

cout << "Incorrect choice." << endl;

menu(HMap);

}

}

int main() {

srand(time(0));

HashMap<string> HMap(N);

for (int i = 0; i < K; i++) {

string strKey(6, '0');

string strVal(10 + rand() % 70, '\0');

strKey[0] = 'A' + rand() % 26;

strKey[1] = '0' + rand() % 9;

strKey[2] = '0' + rand() % 9;

strKey[3] = '0' + rand() % 9;

strKey[4] = '0' + rand() % 9;

strKey[5] = 'A' + rand() % 26;

for (int i = 0; i < strVal.length(); i++)

strVal[i] = (char)'A' + rand() % 26;

HMap.put(strKey, strVal);

}

menu(HMap);

return 0;

}

#pragma once

#include <string>

template <typename V>

class HashNode {

public:

HashNode(const std::string& key, const V& value) : key(key), value(value) {}

std::string getKey() const { return key; }

V getValue() const { return value; }

void setValue(V val) { HashNode::value = val; }

private:

std::string key;

V value;

};

HASHMAP.H

#pragma once

#include "HashNode.h"

#include <fstream>

#include <iostream>

#include <vector>

using namespace std;

template <typename V> class HashMap {

public:

HashMap(int N) {

MAP\_SIZE = N;

map = new HashNode<V> \*[N];

exprt = new int[N];

for (int i = 0; i < N; i++) {

map[i] = nullptr;

exprt[i] = 0;

}

// Generate prime numbers for h2

// 2, 3, 5, 7, 11, 13, 17, 19...N

primes\_h2.push\_back(2);

for (int i = 3; i < N; i++) {

bool prime = true;

for (int j = 0; j < primes\_h2.size() && primes\_h2[j] \* primes\_h2[j] <= i;

j++) {

if (i % primes\_h2[j] == 0) {

prime = false;

break;

}

}

if (prime)

primes\_h2.push\_back(i);

}

};

~HashMap() {

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

delete map[i];

delete map;

delete exprt;

};

unsigned int h1(string &key) {

unsigned int h1 = 0;

const int p = 37;

unsigned int p\_pow = 1;

// h(S) = S[0] + S[1] \* P + S[2] \* P ^ 2 + S[3] \* P ^ 3 + ... + S[N] \* P^N

for (size\_t i = 0; i < key.length(); i++) {

h1 += (key[i] - '0' + 1) \* p\_pow;

p\_pow \*= p;

}

return h1;

}

unsigned int h2(string &key) {

unsigned int h2 = 0;

for (size\_t i = 0; i < key.length(); i++)

h2 += key[i] \* key[i];

// Take a random item

return primes\_h2[h2 % primes\_h2.size()];

}

void put(string key, V value) {

unsigned int h1 = HashMap::h1(key);

unsigned int h2 = HashMap::h2(key);

for (size\_t i = 0; i < MAP\_SIZE; i++) {

exprt[h1 % N] += 1;

if (map[h1 % N] == nullptr) {

map[h1 % N] = new HashNode<V>(key, value);

return;

}

else {

if (map[h1 % N]->getKey() == key)

map[h1 % N]->setValue(value);

h1 = (h1 + h2) % N;

}

}

cout << "OVERFLOW" << endl;

}

vector<HashNode<V> \*> del(string &key) {

unsigned int h1 = HashMap::h1(key);

unsigned int h2 = HashMap::h2(key);

vector<HashNode<V> \*> collision;

for (size\_t i = 0; i < MAP\_SIZE; i++) {

if (map[h1 % N]->getKey() == key) {

delete map[h1 % N];

map[h1 % N] = nullptr;

return collision;

}

else {

collision.push\_back(map[h1 % N]);

h1 = (h1 + h2) % N;

}

}

}

void excel(string &name) {

ofstream fout(name);

for (size\_t i = 0; i < MAP\_SIZE; i++)

fout << exprt[i] << endl;

fout.close();

}

void print() {

cout << endl;

for (size\_t i = 0; i < MAP\_SIZE; i++)

if (map[i] != nullptr)

cout << i << ": " << map[i]->getKey() << " - " << map[i]->getValue()

<< endl;

}

private:

HashNode<V> \*\*map;

int \*exprt;

int MAP\_SIZE;

vector<int> primes\_h2;

};