# Глава 4. Экспериментальная часть

Для демонстрации работоспособности разработанной программы, проведём эксперименты.

## 4.1 Эффективность оптимизации

Эксперимент заключался в оценке степени прироста производительности, получаемом в результате замены контейнеров «vector» и «list» контейнером «map», на участках поиска элементов в контейнерах.

Описание: в ходе эксперимента, размер контейнеров «vector», «list» и «map» синхронно менялся от 100 000 элементов до 1 000 000 с шагом 100 000. Исходный код тестового примера:

// AppForTest.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include <iostream>

#include <list>

#include <vector>

#include <map>

#include <windows.h>

#define N 800000

#define P 1000

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

std::vector<double> somethingVector(N);

std::list<double> somethingList(N);

std::map<double, int> somethingMap;

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

{

somethingVector[i] = (double)rand();

somethingList.push\_back(somethingVector[i]);

somethingMap.insert(make\_pair(somethingVector[i], i));

}

std::vector<double> ToFind(P);

int ind;

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

{

ind = (int)((double)N \* (double)rand() / (double)RAND\_MAX - 0.5);

ToFind[i] = somethingVector.at(ind);

}

DWORD start1 = GetTickCount();

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

{

auto result1 = std::find(std::begin(somethingVector), std::end(somethingVector), ToFind[i]);

//std::cout << (result1 != somethingVector.end());

}

DWORD time1 = GetTickCount() - start1;

DWORD start2 = GetTickCount();

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

{

auto result2 = std::find(std::begin(somethingList), std::end(somethingList), ToFind[i]);

//std::cout << (result2 != somethingList.end());

}

DWORD time2 = GetTickCount() - start2;

DWORD start3 = GetTickCount();

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

{

auto result3 = somethingMap.find(ToFind[i]);

//std::cout << (result3 != somethingMap.end());

}

DWORD time3 = GetTickCount() - start3;

cout << "\nВремя, затрачиваемое контейнером vector:" << time1 << endl;

cout << "Время, затрачиваемое контейнером list:" << time2 << endl;

cout << "Время, затрачиваемое контейнером map:" << time3 << endl;

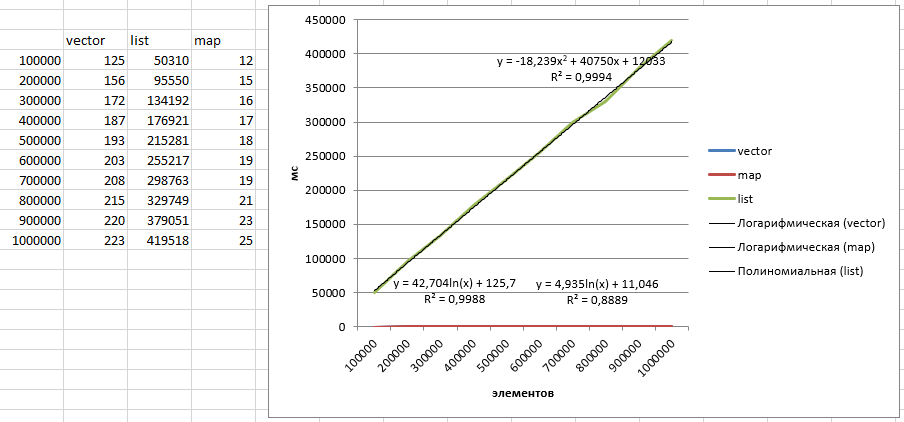
system("pause");

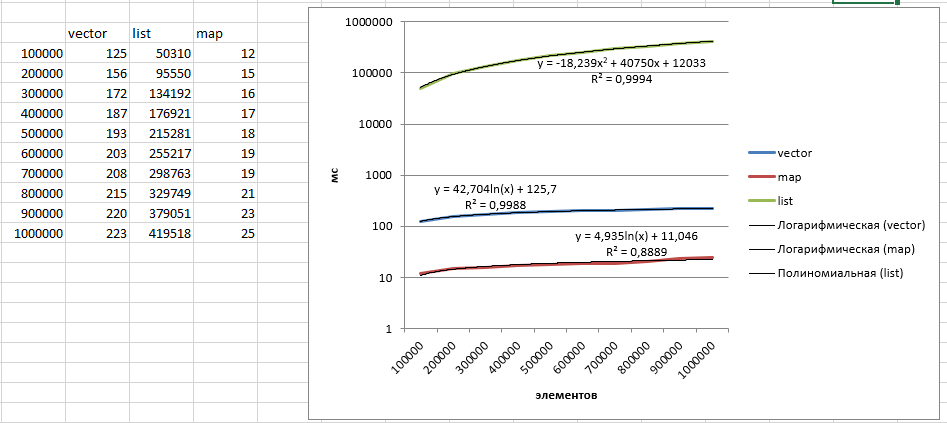
return 0;

}

Листинг 4.1 – Программа на C++ для эксперимента №1

*Результат эксперимента представлен на рисунке 4.1 и 4.2(логарифмический):*

**Рисунок 4.1.1** – **Зависимость времени поиска элементов от их количества**

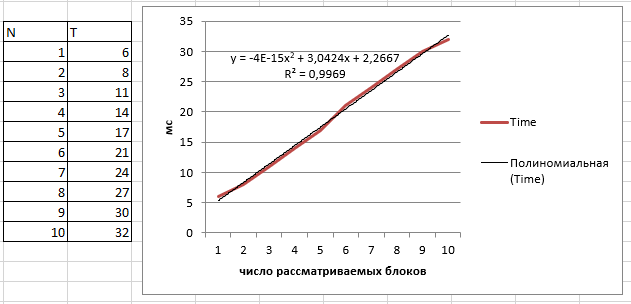
 **Рисунок 4.1.2** – **Зависимость времени поиска элементов от их количества (логарифмический график)**

Эксперимент показал, что замена контейнеров «vector» и «list» контейнером «map», в случае осуществления поиска в контейнере даёт существенный прирост производительности.

## 4.2 Зависимость времени анализа программы, написанной на C++, от размера кода

Эксперимент заключался в увеличение объёма текста программы и оценке производительности анализа кода.

*Результаты эксперимента представлены на графике 4.3*

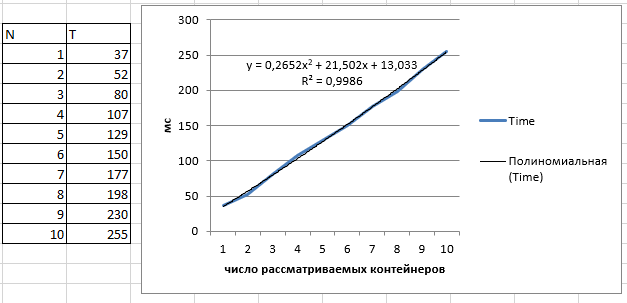
**Рисунок 4.2.1 – Зависимость времени анализа программы, от размера кода**

Из рисунка 4.3 видно, что при увеличении числа рассматриваемых блоков время анализа программы несущественно, но возрастает. Связано это с необходимостью перебора каждого блока для определения не оптимального кода.

## 4.3 Зависимость времени нахождения решения от количества блоков неоптимизированного кода

При проведении эксперимента я брал программу, написанную на C++, где был обнаружен только один контейнер, возможно, подлежащий замене, и в дальнейшем увеличивал их количество от 1 до 10. Во всех программах , , число итераций N = 10000 и верхняя граница используемой оперативной памяти составляла V = 7000 байт.

*Результаты эксперимента представлены на графике 4.4:*

**Рисунок 4.3.1 – Зависимости времени нахождения решения от количества блоков**

Эксперимент показал, что при увеличении числа контейнеров, которые необходимо рассмотреть программе, растёт и время нахождения решения.

Величина ошибки аппроксимации , отображаемая на графике рассчитывается по формуле:

(3)

где

— значение, полученное в ходе эксперимента.

— значение, вычисленное с использованием аналитической зависимости, полученной аппроксимацией.

и —средние значения соответственно экспериментальных и аналитических результатов измерений.