МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ ФГБОУ ВО

"СЕВЕРО-КАВКАЗСКИЙ ГОРНО-МЕТАЛЛУРГИЧЕСКИЙ ИНСТИТУТ"

(ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ)

Факультет: *Информационных технологий и электронной техники*

Кафедра: *Информатики и вычислительной техники*

Специальность: *Информатика и вычислительная техника*

Профиль: *Автоматизированные системы обработки информации и управления*

Пояснительная записка

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

на тему: «Создать программу, автоматизирующую оптимизацию производительности участков поиска элементов данных в контейнерах std::list и std::vector с помощью индексированных контейнеров std::map в программах, написанных на C++»

Студент: Меликян Р.А.
Руководитель доц., к.т.н. Томаев М.Х.
Проект рассмотрен кафедрой и допущен к защите в ГАК
Заведующий кафедрой проф., д.т.н. Гроппен В.О.

Владикавказ 2021

Реферат

Пояснительная записка: 74 стр, 4 табл., 10 рис., 13 ист., 1 прил.

Оптимизация, C++, STL, Контейнеры, std:vector, std::list, std::map, std::find, C#, Итераторы.

Объект разработки: «Создать программу, автоматизирующую оптимизацию производительности участков поиска элементов данных в контейнерах std::list и std::vector с помощью индексированных контейнеров std::map в программах написанных на C++».

Цель выпускной работы: заключается в поиске оптимальной стратегии замен контейнеров «list» и «vector» контейнером «map» на участках непосредственного поиска элементов в контейнерах при условии, что требуемые для её реализации дополнительные ресурсы ОП не превысят верхней границы.

Использованное прикладное и системное программное обеспечение:

- Операционная система Windows 7 Ultimate 64-bit;
- Среда разработки Microsoft Visual Studio 2015 Community

Типы используемых вычислительных средств:

При разработке своей работы использовал персональный компьютер со следующими характеристиками: AMD FX(tm) – 8120 Eight-Core Processor 3.10 GHz, NVIDIA GeForce GTX 650, 8 Gb ОЗУ.

Полученные результаты:

Во время выполнения выпускной работы был создан программный продукт, оптимизирующий код на языке C++ заменой контейнеров «list» и «vector» контейнером «map» в тех местах, где производится поиск элементов в контейнерах. Для проверки данного утверждения были проведены серии экспериментов.

Оглавление

Введение

C++ - компилируемый, статически типизированный язык программирования общего назначения.

Поддерживает такие парадигмы программирования, как процедурное, ОО - программирование, обобщённое программирование. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. C++ сочетает свойства, как низкоуровневого, так и высокоуровневого языка. В отличие от языка C, огромное внимание уделено поддержке ООП и обобщённого программирования.

Библиотека стандартных шаблонов (STL) (англ. *Standard Template Library*) — набор согласованных обобщённых алгоритмов, контейнеров, средств доступа к их содержимому и различных вспомогательных функций в C++.

Библиотека стандартных шаблонов до включения в стандарт C++ была сторонней разработкой, вначале — фирмы HP, а затем SGI. Стандарт языка не называет её «STL», так как эта библиотека стала неотъемлемой частью языка. Однако многие люди до сих пор используют это название, чтобы отличать её от остальной части стандартной библиотеки (потоки ввода-вывода (iostream), подраздел Си и др.).

Проект под названием STLPort, основанный на SGI STL, осуществляет постоянное обновление STL, iostream и строковых классов. Некоторые другие проекты также занимаются разработкой частных применений стандартной библиотеки для различных конструкторских задач. Каждый производитель компиляторов C++ обязательно поставляет какую-либо реализацию этой библиотеки, так как она является очень важной частью стандарта и широко используется.

Контейнер управляет выделяемой для его элементов памятью и предоставляет функции-члены для доступа к ним, либо непосредственного, либо через итераторы (объекты, обладающие схожими с указателями свойствами).

В библиотеке достаточно различных контейнеров, о трёх из них пойдёт речь в моей работе. Важно знать, чем они отличаются, их сильные и слабые стороны, чтобы грамотно использовать их.

Глава 1. Аналитический обзор

1.1 Оптимизация программ

Под оптимизацией программы подразумевают такие преобразования, в результате которых она становится более эффективной, т.е. становится более экономной по памяти или/и более быстрой по выполнению тех же функций, что и до оптимизации.

Оптимизация проводится по двум частным критериям: время выполнения программы и объём памяти, которую она использует. Но эти два критерия противоречат друг другу, т.к. чтобы уменьшить время работы, необходимо увеличить размер потребляемой памяти и наоборот. В этом случае программист из личных побуждений отдаёт предпочтение одному из критериев.

Частично оптимизацию программы может компилятор. Но в основном этот процесс зависит от квалификации программиста и невозможно дать алгоритм, оптимизирующий любую программу. Можно лишь обратить внимание на те аспекты, где скрыты резервы оптимизации и проиллюстрировать их на примерах.

Существует два подхода к оптимизации программ: «чистка» и перепрограммирование. Оба подхода имеют как достоинства, так и недостатки.

Первый подход заключается в исправлении очевидных небрежностей в исходной программе. Его достоинство - данный метод требует мало времени. Однако повышение эффективности при этом обычно незначительно.

Второй подход состоит в переделке исходной программы. Можно переделать часть программы, которая, например, расходует наибольшую часть времени. Этот подход обеспечивает обычно наилучший результат, но и

самый дорогой. Он приемлем, если оптимизируемая программа подвергалась значительным изменениям.

Оптимизация кода - различные методы преобразования кода ради улучшения его характеристик и повышения эффективности. Среди целей оптимизации можно указать уменьшения объема кода, объема используемой программой оперативной памяти, ускорение работы программы, уменьшение количества операций ввода вывода.

Главное из требований, которые обычно предъявляются к методу оптимизации - оптимизированная программа должна иметь тот же результат и побочные эффекты на том же наборе входных данных, что и неоптимизированная программа. Впрочем, это требование может и не играть особой роли, если выигрыш за счет использования оптимизации может быть сочтен более важным, чем последствия от изменения поведения программы.

1.2 Виды оптимизации

Оптимизация кода может проводиться, как и вручную, программистом, так и автоматизировано. В последнем случае оптимизатор может быть как отдельным программным средством, так и быть встроенным в компилятор (т.е. оптимизирующий компилятор). Кроме того, следует отметить, что современные процессоры могут оптимизировать порядок выполнения инструкций кода.

Существуют такие понятия как высокоуровневая и низкоуровневая оптимизация. Высокоуровневые оптимизации в большинстве проводятся программистом, который, оперируя абстрактными сущностями (функциями, процедурами, классами и т.д.) и представляя себе общую модель решения задачи, может оптимизировать дизайн системы. Оптимизации на уровне элементарных структурных блоков исходного кода (циклов, ветвлений и т.д.) тоже обычно относят к высокому уровню; некоторые выделяют их в

отдельный ("средний") уровень. Низкоуровневая оптимизация производится на этапе превращения исходного кода в набор машинных команд, и зачастую именно этот этап подвергается автоматизации. Впрочем, программисты на ассемблере считают, что никакая машина не превзойдет в этом хорошего программиста (при этом все согласны, что плохой программист сделает еще хуже и машины).

1.3 Методы оптимизация кода C++

1. *Оптимизация работы с массивами*

При работе с массивами использование функций, предназначенных для определённых операций с ними, может существенно увеличить производительность. Так, например, при копировании массивов используется функция memcpy(копирование данных из одного блока памяти в другой) даст заметный прирост производительности по сравнению с поэлементным копированием в цикле for. Пример:

До оптимизации:

```
double arr1[CNT], arr2[CNT];
for (int i = 0; i< CNT; i++)
        arr[i] = 0;
```

Оптимизированный код:

```
double arr1[CNT], arr2[CNT];
memcpy(arr1, arr2, sizeof(double) * CNT);
```

2. *Замена возврата значения ссылочной переменной*

Оператор return создаёт временный экземпляр и копирует в него переданное в return значение, т.е. вызывается конструктор копий. Для возврата значений используется дополнительный аргумент ссылка.

3. *Префиксный и постфиксный оператор*

Префиксный оператор предпочтительнее постфиксного. При работе с примитивными типами, что префиксные, что постфиксные арифметические операции, с большей вероятностью, будут иметь одинаковые результаты производительности. Однако с объектами, операторы постфикса могут заставить объект создавать собственную копию, чтобы сохранить своё начальное значение(которое должно возвращаться в результате операции), а также вызвать возможный побочный эффект операции. Рассмотримпример:

```
class IntegerIncreaser {
int m_Value;
public:
    /* Postfix operator. */
    Integer Increaser operator++ (int) {
    Integer Increaser tmp(*this);
    ++m_Value; return tmp; };
    /* Prefix operator. */
    Integer Increaser operator++ () {
        ++m_Value;
        return *this;
    };

};
```

Поскольку операторы постфикса обязаны возвращать неизменённую версию значения, которое увеличивается (или уменьшается) – независимо от того, используется ли результат на самом деле – скорее всего, он сделает копию. Например, итераторы STL более эффективны при изменении с помощью префиксных операторов.

4. *Оптимизация передачи аргументов в функции*

Данный вид оптимизации делится на 2 метода:

1) Метод макрозамен – даёт выигрыш в производительности, благодаря двум составляющим:

- Время на передачу управления в функцию и возврат из неё – это время складывается из времени выполнения оператора *call*, а также времени на сохранении в стеке основных регистров процессора.
- Время на копирование данных из внешних переменных в локальные переменные.

Метод макрозамен заключается в преобразовании функции в макрос. Макрос – участок кода, который так же, как и функция, имеет имя и аргументы, но в отличие от функции тело макроса вставляется во все места исходного кода, где он вызывается, при этом размер используемого кода программы вырастит пропорционально количеству вызовов макросов.

В языках C/C++ существует поддержка на уровне международного стандарта ключевого слова inline, которое информирует компилятор о необходимости замены функции на макрос. В этом случае, если компилятор обнаружит препятствие для такой замены, то ключевое слово будет проигнорировано.

Для того чтобы принудительно преобразовать функцию в web-макрос, в компилятор фирмы «Microsoft» включена поддержка ключевого слова __forceinline, который игнорирует все условия потенциально препятствующие макрозамене.

Начиная со стандарта C++ 11, для модификации функции в макрос недостаточно простого добавления ключевого слова inlineили __forceinline.

Тело модифицируемой функции необходимо полностью перенести в заголовочный файл.

Пример макрозамены с помощью inline:

До оптимизации:

```
double f(double x, double y) {
    return (sin(x) * cos(y)) / (x * y);
```

```
}
int main(){
        double k = 5, y = 7;
        int z = f(k, y);
        return 0;
}
```

После добавления ключевого слова inline:

```
Inline double f(double x, double y) {
        return (sin(x) * cos(y)) / (x * y);
}
int main(){
double k = 5, y = 7;
int z = f(k, y);
return 0;
}
```

2) Альтернативой использования стандартных средств языка является прямая вставка кода. Данный способ потенциально позволяет получить максимальный выигрыш в производительности, однако на практике является наиболее трудоёмким.

Самой важной проблемой является возможный конфликт имён локальных переменных исходной функции, т.е. при прямой вставке может возникнуть необходимость изменить наименования переменных, модифицируемых функцией.

5. *Замена параметров значений на параметры ссылки*

Один из способов повышения производительности в функции – замена параметров значений на параметры ссылки. В этом случае в функцию передается не копия значений аргументов, а адреса передаваемых значений. Это даст выигрыш в том случае, если размер типа аргумента

больше чем размер указателя на текущей платформе. В следующем примере оптимизация даст выигрыш в производительности для приложения на платформе Win 32, т.к. на этой платформе указатель является 32-х битным, а тип double является 64-х разрядным.

До оптимизации:

```
double f(double x, double y) {
        return (sin(x) * cos(y)) / (x * y);
}

int main(){
        double k = 5, y = 7;
        int z = f(k, y);
return 0;
}
```

После оптимизации:

```
double f(double &x, double &y) {
        return (sin(x) * cos(y)) / (x * y);
}

int main(){
        double k = 5, y = 7;
        int z = f(k, y);
return 0;
}
```

1.4 Выбор участка кода для оптимизации

При оптимизации кода вручную существует ещё одна проблема: нужно знать не только, каким образом проводить оптимизацию, но и в каком месте. На оптимизацию всей программы будет затрачено колоссальное количество ресурсов, поэтому рациональнее всего выбрать участок кода, который называется «критическим». Такой фрагмент называют узким местом или бутылочным горлышком, и для их определения используют специальные программы – профайлер, которые позволяют замерять время работы различных частей программы.

На практике оптимизация зачастую проводится после этапа "хаотического" программирование, поэтому представляет собой смесь из собственно оптимизации, рефакторинга и исправления ошибок: упрощение "причудливых" конструкций – вроде strlen(path.c_str()), логических условий (a.x != 0 && a.x != 0) и т.п. Для такого рода оптимизации профайлеры не пригодны. Однако для обнаружения таких мест используются программы статического анализа – средства поиска семантических ошибок на основе глубоко анализа исходного кода - ведь, как видно из второго примера, неэффективный код может быть следствием ошибок (как, например, опечатки в данном примере - скорее всего, имелось ввиду a.x != 0 && a.y != 0). Хороший статический анализатор обнаружит подобный код, и выведет предупреждающее сообщение.

1.5 Вред и польза оптимизации

Во всём надо относиться рационально, и оптимизация – не исключение. Считается, что новичок в программировании, написавший код на ассемблере, будет работать медленнее, чем код сгенерированный компилятором.

К оптимизации, проводимой оптимизатором, почти нет претензий, причём иногда некоторые оптимизации являются фактическими стандартными и обязательными.

Однако следует понимать, что многочисленные сложные оптимизации на уровне машинного кода могут сильно замедлить процесс компиляции. Причем выигрыш от них может быть чрезвычайно мал по сравнению с оптимизациями общего дизайна системы.

Таким образом, не стоит забывать проводить оптимизацию кода, при этом использовать специальные программные средства, но это стоит делать в меру и с осторожностью.

1.6 STL – стандартная библиотека шаблонов

Под термином библиотека стандартных шаблонов (STL, **S**tandard **T**emplate **L**ibrary) понимают набор интерфейсов и компонентов, первоначально разработанных Александром Степановым, Менг Ли и другими сотрудниками AT&T Bell Laboratories и Hewlett-Packard Research Laboratories в начале 90-х годов (хотя и позже ещё весьма многие приложили руку к тому, что стало на сегодня стандартным компонентом C++). Далее библиотека STL перешла в собственность компании SGI, а также была включена как компонент в набор библиотек Boost. И наконец, библиотека STL вошла в стандарты C++ 1998 и 2003 годов (ISO/IEC 14882:1998 и ISO/IEC 14882:2003) и с тех пор считается одной из составных частей стандартной библиотек C++.

Первоначальной целью STL (это хорошо видно из хронологии комментариев в заголовочных файлах) было создание боле гибкой модели регулярных контейнеров по сравнению с массивами и обобщение на них некоторых широко используемых алгоритмов (таких как поиск, сортировка и некоторых других). Но затея оказалась плодотворнее первоначальных намерений, и была существенно расширена. STL вводит ряд понятий и структур данных, которые почти во всех случаях позволяют сильно упростить программный код.

Вводятся следующие категории понятий:

1. Контейнер – способ хранения набора объектов в памяти.
2. Итератор – средство доступа к содержимому отдельных объектов в контейнере.
3. Алгоритм – определение наиболее стандартных вычислительных процедур на контейнерах.
4. Адаптер – адаптация основных категорий для обеспечения наиболее употребляемых интерфейсов (таких как стек или очередь).
5. Функтор (функциональный объект) – сокрытие функции в объекте для использования её другими категориями.

1.7 STL.Контейнеры

Центральным понятием STL, вокруг которого крутится всё остальное, это контейнер (ещё используют термин коллекция). Контейнер — это набор некоторого количества обязательно однотипных элементов, упакованных в контейнер определённым образом. Простейшим прототипом контейнера в классическом языке C++ является массив. Тот способ, которым элементы упаковываются в контейнер, и определяет тип контейнера и особенности работы с элементами в таком контейнере.

Стандартная библиотека предоставляет различные типобезопасные контейнеры для хранения коллекций связанных объектов. При объявлении переменной контейнера указывается тип элементов, которые будет содержать контейнер. Контейнеры могут создаваться с использованием списков инициализаторов. Они содержат функции-члены для добавления и удаления элементов и выполнения других операций.

Итерация элементов в контейнере и доступ к отдельным элементам осуществляются с помощью итераторов. Вы можете использовать итераторы явно, с помощью их функций-членов и операторов, а также глобальных функций. Вы можете также использовать их неявно, например, с помощью

цикла range-for. Итераторы для всех контейнеров стандартной библиотеки C++ имеют общий интерфейс, но каждый контейнер определяет собственные специализированные итераторы.

Библиотека контейнеров является универсальной коллекцией шаблонов классов и алгоритмов, позволяющих программистам легко реализовывать общие структуры данных, такие как очереди, списки и стеки.

Контейнер управляет выделяемой для его элементов памятью и предоставляет функции-члены для доступа к ним, либо непосредственного, либо через итераторы (объекты, обладающие схожими с указателями свойствами).

Большинство контейнеров обладают, по крайней мере, несколькими общими функциями-членами и общей функциональностью. Выбор оптимального контейнера для конкретного случая зависит не только от предоставляемой функциональности, но и от его эффективности при различных рабочих нагрузках.

Контейнеры можно разделить на три категории: последовательные контейнеры, ассоциативные контейнеры и контейнеры-адаптеры.

Последовательные контейнеры поддерживают указанный пользователем порядок вставляемых элементов. К ним относятся:

Контейнер «*vector»* ведет себя как массив, но может автоматически увеличиваться по мере необходимости. Он поддерживает прямой доступ и связанное хранение и имеет очень гибкую длину. По этим и многим другим причинам контейнер «vector» является наиболее предпочтительным последовательным контейнером для большинства областей применения. Если вы сомневаетесь в выборе вида последовательного контейнера, начните с использования вектора.

Контейнер «*array»* обладает некоторыми преимуществами контейнера «vector», однако его длина не обладает такой гибкостью.

Контейнер «*deque»* (двусторонняя очередь) обеспечивает быструю вставку и удаление в начале и в конце контейнера. Он, как и контейнер

«vector», обладает преимуществами прямого доступа и гибкой длины, но не обеспечивает связанное хранение.

Контейнер «*list*» — это двунаправленный список, который обеспечивает двунаправленный доступ, быструю вставку и удаления в любом месте контейнера, но не поддерживает прямой доступ к элементам контейнера.

В ассоциативных контейнерах элементы вставляются в предварительно определенном порядке — например, с сортировкой по возрастанию. Также доступны неупорядоченные ассоциативные контейнеры. Ассоциативные контейнеры можно объединить в два подмножества: сопоставления («set») и наборы («map»).

Контейнер «map», который иногда называют словарем, состоит из пар "ключ-значение". Ключ используется для упорядочивания последовательности, а значение связано с ключом. Например, «map» может содержать ключи, представляющие каждое уникальное ключевое слово в тексте, и соответствующие значения, которые обозначают количество повторений каждого слова в тексте. «map» — это неупорядоченная версия unordered_map.

«set» — это контейнер уникальных элементов, упорядоченных по возрастанию. Каждое его значение также является и ключом. Set — это неупорядоченная версия unordered_set.

Контейнеры «map» и «set» разрешают вставку только одного экземпляра ключа или элемента. Если необходимо включить несколько экземпляров элемента, следует использовать контейнер «multimap» или «multiset». Неупорядоченные версии этих контейнеров — unordered_multimap и unordered_multiset.

Упорядоченные контейнеры «map» и «set» поддерживают двунаправленные итераторы, а их неупорядоченный аналоги — итераторы с перебором в прямом направлении.

Контейнер-адаптер — это разновидность последовательного или ассоциативного контейнера, который ограничивает интерфейс для простоты и ясности. Контейнеры-адаптеры не поддерживают итераторы.

Контейнер «queue» соответствует семантике FIFO (первым поступил — первым обслужен). Первый элемент, который*отправляется*, то есть вставляется, в очередь, должен быть первым элементом, *извлекаемым* из очереди.

Контейнер priority_queue упорядочен таким образом, что первым в очереди всегда оказывается элемент с наибольшим значением.

Контейнер stack соответствует семантике LIFO (последним поступил — первым обслужен). Последний элемент, отправленный в стек, становится первым извлекаемым элементом.

Поскольку контейнеры-адаптеры не поддерживают итераторы, их невозможно использовать в алгоритмах стандартной библиотеки C++.

1.8 Контейнер Vector

Использование класса vector является альтернативой применению встроенных массивов. Этот класс предоставляет гораздо больше возможностей, поэтому его использование предпочтительней.

Шаблон «vector» расположен в заголовочном файле <vector>. Как и все стандартные компоненты, он расположен в пространстве имён std. Данный интерфейс эмулирует работу стандартного массива C (например, быстрый произвольный доступ к элементам), а также некоторые дополнительные возможности, вроде автоматического изменения размера вектора при вставке или удалении элементов.

Элементы контейнера «vector» хранятся непрерывно, а значит, доступны не только через итераторы, но и через смещения, добавляемые к указателям на элементы (data() или же, для непустых массивов, — &vect[0]).

Это означает, что указатель на элемент вектора может передаваться в любую функцию, ожидающую указатель на элемент массива.

Хранилище вектора обрабатывается автоматически, расширяясь и сужаясь по мере необходимости. Векторы обычно занимают больше места, чем статические массивы, поскольку некоторое количество памяти выделяется про запас на обработку будущего роста. Таким образом, память для вектора требуется выделять не при каждой вставке элемента, а только после исчерпания резервов. Общий объём выделенной памяти можно получить с помощью функции capacity(). Резервная память может быть возвращена системе через вызов shrink_to_fit().

Перераспределения обычно являются дорогостоящими операциями в плане производительности. Функция reserve() может использоваться для предварительного выделения памяти и устранения перераспределений, если заранее известно количество элементов.

В дополнение к функциям прямого доступа к элементам, описанным выше, элементы вектора можно получить посредством итераторов.

Итераторы обычно используются парами, один из которых используется для указания текущей итерации, а второй служит для обозначения конца контейнера. Итераторы создаются при помощи таких стандартных методов как begin() и end(). Функция begin() возвращает указатель на первый элемент, а end() — на воображаемый несуществующий элемент, следующий за последним.

Вектор использует наиболее функционально богатый тип итераторов — RandomAccessIterator (итератор произвольного доступа), который позволяет обходить контейнер в любом порядке, а также изменять содержимое вектора в процессе обхода. Однако, при изменении вектора итератор может стать недействительным.

Пример подсчёта суммы элементов вектора при помощи итераторов:

```
#include"stdafx.h"
#include<vector>
```

```
#include<iterator>
#include<iostream>
using namespace std;

int main()
{
	vector<int>the_vector;
	vector<int>::iteratorthe_iterator;
	for (int i = 0; i< 10; i++) {
		the_vector.push_back(i);
	}
	int total = 0;
	the_iterator=the_vector.begin();
	while (the_iterator!=the_vector.end()) {
		total += *the_iterator;
		++the_iterator;
	}
	cout<<"summa= "<< total <<endl;
	system("pause");
	return 0;
}
```

Вектор сохраняет определённый порядок его элементов, так, что при вставке нового элемента в начале или в середине вектора, последующие элементы перемещаются в обратном направлении с точки зрения их оператора присваивания и конструктора копии. Следовательно, ссылки и итераторы элементов после места вставки становятся недействительным. Пример:

```
#include"stdafx.h"
#include<vector>
#include<iterator>
#include<iostream>
using namespace std;

int main()
{
	std::vector<int>v(2); // Создаём вектор, состоящий из двух элементов типа Int

	// Создаём ссылки на оба элемента
	int &first = v.front();
```

```
    int &last = v.back();

    v.insert(v.begin() + 1, 1, 1); // Добавляем новые элементы в середину вектора

    int i = first; // Неопределённое поведение, если вставка вызвала перераспределение памяти
    int j = last; // Неопределённое поведение, согласно стандарту C++
}
```

Сложность (эффективность) обычных операций над векторами следующая:

- Произвольный доступ — постоянная *O(1)*
- Вставка и удаление элементов в конце — амортизированная постоянная *O(1)*
- Вставка и удаление элементов — линейная по расстоянию до конца вектора *O(n)*

Плюсы и минусы вектора(vector):

- Как и все реализации динамического массива, вектор не использует дополнительных структур данных, данные расположены в памяти рядом, за счёт чего они хорошо кэшируются.
- Вектор может быстро выделять память, необходимую для хранения конкретных данных. Это особенно полезно для хранения данных в списках, длина которых может быть не известна до создания списка, а удаление (за исключением, быть может, в конце) необходимо редко.
- Как и другие контейнеры STL, может содержать примитивные типы данных, сложные или определённые пользователем.
- Вектор разрешает произвольный доступ; то есть на элемент вектора можно ссылаться так же, как на элемент массива (по индексу). Связанные списки и множества, напротив, не поддерживают произвольный доступ и арифметические операции над указателями.

- Удаление элемента из вектора или даже очистка вектора совершенно не обязательно освободит память, связанную с этим элементом. Это потому, что максимальный размер вектора с момента его создания является хорошей оценкой размера для нового вектора.
- Векторы являются неэффективными для вставки элементов в любые места, кроме конца. Такая операция имеет O(n) (см. O-нотация) сложность по сравнению с O(1) для связанных списков. Удаление элемента из произвольного места также имеет сложность O(n) (необходимо сдвинуть к началу все элементы, располагающиеся после удаляемого, что в худшем случае даст n-1 перемещений). Это компенсируется скоростью доступа. Доступ к произвольному элементу вектора имеет сложность O(1) по сравнению с O(n) для связанного списка и O(log n) для сбалансированного двоичного дерева поиска.

1.9 Список List

Контейнер std::list широко применяется в системах управления данными ввиду высокой скорости добавления новых элементов.

Список представляет собой контейнер, который поддерживает быструю вставку и удаление элементов из любой позиции в контейнере. Быстрый произвольный доступ не поддерживается. Он реализован в виде двусвязного списка. В отличие от std::forward_list этот контейнер обеспечивает возможность двунаправленного итерирования, являясь при этом менее эффективным в отношении используемой памяти.

В отличие от других контейнеров для типа «list» не определена операция обращения по индексу или функция at(), которая выполняет похожую задачу.

Тем не менее для контейнера «list» можно использовать функции front() и back(), которые возвращают соответственно первый и последний элементы.

Чтобы обратиться к элементам, которые находятся в середине (после первого и до последнего элементов), придется выполнять перебор элементов с помощью циклов или итераторов:

```
#include"stdafx.h"
#include<list>
#include<iterator>
#include<iostream>
using namespace std;

int main()
{
	std::list<int> numbers = { 1, 2, 3, 4, 5 };

	int first = numbers.front();  // 1
	int last = numbers.back();  // 5

	// переборвцикле
	for (int n : numbers)
		std::cout<< n <<"\t";
	std::cout<<std::endl;

	// переборспомощьюитераторов
	for (auto iter = numbers.begin(); iter!=numbers.end(); iter++)
	{
		std::cout<<*iter<<"\t";
	}
	std::cout<<std::endl;
	return 0;
}
```

1.10 Контейнер map

std::map — отсортированный ассоциативный контейнер, который содержит пары ключ-значение с неповторяющимися ключами. Контейнер map, очень похож на остальные контейнеры, такие как vector, list, deque, но с небольшим отличием. В этот контейнер можно помещать сразу два значения.

Контейнер map<> (таблица, отображение):

- Содержит упорядоченные пары <ключ, значение>, где ключ и значение могут принадлежать к произвольным типам. Для типа ключа должна быть либо предопределена, либо определена пользователем операция сравнения;
- Элементы с любым значением ключа должны быть уникальны;
- Попытка добавить (метод insert()) к таблице новую пару с уже имеющимся значением ключа завершится неудачей;
- Операция добавления новой пары в таблицу возвращает пару типа <итератор, bool>, у которой второй компонент (логический second) указывает на успешность операции. Если он true, то первый компонент возвращаемого результата (first) даёт итератор добавленного элемента. Если же он false, то возвращается итератор существующего элемента с тем же ключом;
- Операции индексации таблицы ([ ] или at()) требуют в качестве ключа любое значение типа, определённого для ключа;
- Операция индексации at(), при задании ключа-параметра, отсутствующего в составе элементов таблицы, вызывает исключение;
- Напротив, операция индексации [ ], при задании ключа-параметра, отсутствующего в составе элементов таблицы, исключение не вызывает. (наоборот, даже если индексация запрошена только по

чтению, добавляет к контейнеру новый элемент с требуемым значением ключа, но с нулевым полем значения);

Чтобы подключить map, нужно подключить заголовочный файл map и, собственно, простейший код:

```
#include<iostream>
#include<map>

using namespace std;

int main() {
	map<string, int> m;
	m["Вася"] = 0;
	m["Петя"] = 1;
	m["Федор"] = 2;

	cout<< m["Вася"] <<"\n";
	cout<< m["Петя"] <<"\n";
	cout<< m["Федор"] <<"\n";

	return 0;
}
```

Глава 2. Выбор и обоснование алгоритма оптимизации

2.1 Содержательная постановка задачи

Создать программную систему, выполняющую следующие операции:

а) анализ исходного кода и локализация участков, на которых осуществляется последовательный перебор значений контейнеров std::list либо std::vector и сравнение каждого элемента с ключевым значением.

б) генерировать рекомендации по включению в исходный код вспомогательных индексированных контейнеров данных, обеспечивающих быстрый поиск элементов, причем суммарный размер этих контейнеров не должен превышать заданной верхней границы дополнительного объема ОП, выделенного на оптимизацию.

2.1.1Исходные данные к задаче.

а) Исходный код пользователя;

б) Верхняя граница объёма оперативной памяти, выделенной на оптимизацию.

2.2 Формальная постановка задачи

2.2.1 Обозначения.

M –количество контейнер.

n_i – количество элементов i-го контейнера данных;

v_i– размер одного элемента i-го контейнера данных;

P_i – среднее количество операций поиска в i-м контейнере данных;

V– верхняя граница дополнительного объема ОП, выделенной на оптимизацию.

z_i – булева переменная равная единице, если для оптимизации

b_i – элемент массива булевых констант (входные данные). b_i равна единице, если исходным контейнером хранения данных i-го массива являлся std::vector и нулю, в случае, когда исходным контейнером является std::list.

$T_{map}(z_i)$–вспомогательная функция, описывающая среднее время поиска элемента в i-м массиве с использованием вспомогательного контейнера std::map.

s_{map} – скорость формирования контейнера std::map.

$s_{compare}$– средняя скорость сравнения двух произвольных элементов i-го массива.

t_i–среднее время выборки и сравнения двух элементов i-го массива (равна $\frac{v_i}{s_{compare}}$);

$T_{vector}(z_i)$–вспомогательная функция, описывающая среднее время поиска элемента в i-м массиве, реализованного в виде контейнера std::vector.

$T_{list}(z_i)$–вспомогательная функция, описывающая среднее время поиска элемента в i-м массиве, реализованного в виде контейнера std::list.

k–коэффициент пропорциональности > 1, описывающий характер превышения времени выборки данных в контейнере std::list по сравнению с std::vector . Более высокие временные затраты на перемещение между элементами списка объясняются тем что обращение к соседним элементам осуществляется посредством вспомогательных указателей.

2.2.2 Формальная постановка задачи

Сформулируем оптимизационную задачу, минимизирующую суммарное время поиска в контейнерах std::vector и std::list с использованием вспомогательного индексированного контейнера std::map в виде модели (1):

$$\begin{cases} F = \sum_{i=1}^{M} \left(T_{map}(z_i) + b_i T_{vector}(z_i) + (1 - b_i) T_{list}(z_i) \right) \to min; \\ T_{map}(z_i) = z_i \left(P_i t_i \lceil log_2 n_i \rceil + \frac{n_i v_i}{s_{map}} \right); \\ T_{vector}(z_i) = (1 - z_i) P_i t_i \frac{n_i}{2}; \\ T_{list}(z_i) = (1 - z_i) P_i t_i k \frac{n_i}{2}; \\ t_i = \frac{v_i}{s_{compare}}; \\ \sum_{i=1}^{M} z_i n_i v_i \leq V; \\ z_i = 0,1; \end{cases} \quad (1)$$

Следует отметить, что выигрыш в скорости поиска дается не только ценой дополнительной памяти, но также дополнительным процессорным временем, который тратится на создание индексированного массива std::map — соответствует второму слагаемому вспомогательной функции $T_{map}(z_i)$:

$$\frac{n_i v_i}{s_{map}} \quad (2)$$

Пропорциональный характер времени формирования контейнера std::map относительно размера массива подтвержден экспериментально:

Таблица 2.1. *Результаты тестовых замеров времени создания контейнера std::map*

Количество элементов типа double	Время формирования std::map (миллисекунды)
1000000	609
2000000	1250
3000000	1890
4000000	2563
5000000	3219
6000000	3890
7000000	4562
8000000	5234
9000000	5922
10000000	6610

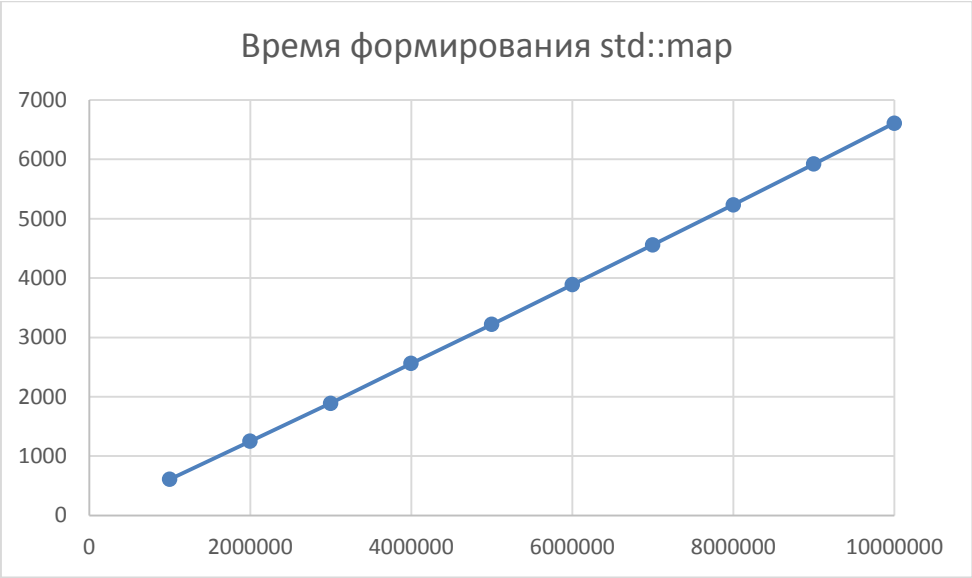

Рис.2.1 Зависимость времени формирования контейнера std::map от размера массива.

2.3 Алгоритм поиска решения

Для того чтобы обеспечить возможность поиска решения задачи (1) при больших размерностях входных данных мной был выбран метод Монте-Карло, приведенный ниже (Алгоритм 2.1):

Алгоритм 2.1.

1. Ввод числа итераций N.
2. i=0.
3. Рекорду **R** присваиваем значение «бесконечность».
4. С помощью генератора случайных чисел генерируются булевы значения (0 либо 1) для каждого элемента $\boldsymbol{z_i}$.
5. Если условие $\sum_{i=1}^{M} z_i n_i v_i \leq V$ не выполняется, то переход к шагу 9, иначе к следующему шагу.
6. Вычисляем значение целевой функции $\boldsymbol{F}$, с помощью выражения, указанного в модели (1).
7. Если F>=R, то переход к шагу 9, иначе к следующему шагу
8. R = F, сохраняем элементы массива {z} в массив $\{z^{optimum}\}$.

9. i = i+1.
10. Если i<=N, то переход к шагу 4, в противном случае к следующему шагу.
11. Решение найдено. R содержит текущее значение целевой функции, а массив $\{z^{optimum}\}$ — значения элементов z_i, соответствующие рекорду.

2.4 Пример решения задачи вручную

Выполним поиск наилучшей стратегии оптимизации для следующего исходного кода пользователя (Листинг 2.1):

Листинг 2.1.

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <iterator>

int main()
{
    int n1 = 3;
    int n2 = 5;

std::vector<int>x1;
std::vector<double>x2;
std::list<int>x3;

//создание и заполнение массивов
    for (int i=0; i<1000; i++) x1.push_back(i);
    for (int i=0; i<700; i++) x2.push_back((double)i);
    for (int i=0; i<500; i++) x3.push_back(i);


 //поиск в массивах
for (int i=0; i<3000; i++)
    {
        auto result1 = std::find(std::begin(x1), std::end(x1),
                        (int)(1000.0*(double)rand()/(double)RAND_MAX) );
std::cout<< (result1!=x1.end()?"Значение найдено!":
                                "Значение не найдено!";
    }

for (int i=0; i<200; i++)
    {
        auto result2 = std::find(std::begin(x2), std::end(x2),
                        700.0*(double)rand()/(double)RAND_MAX );
        std::cout<< (result2=x2.end()?"Значение найдено!":
                                "Значение не найдено!";

auto result3 = std::find(std::begin(x3), std::end(x3),
```

```
                        (int)(500.0*(double)rand()/(double)RAND_MAX) );
std::cout<< (result3=x3.end()?"Значение найдено!":
                              "Значение не найдено!";

    }

}
```

На основании Листинга1 выделим часть исходных данных и поместим их в Таблицу 2.2.

Таблица 2.2. Входные данные, полученные на основании анализа исходного кода.

M	3		
{n}	1000	700	500
{v}	4	8	4
{P}	3000	200	200
{b}	1	1	0

Примем в качестве верхней границы доступного для оптимизации объема ОП значение **7000** байт, а значения скоростей сравнения и создания контейнера 1.00E+13 байт/сек и 1.00E+13 байт/сек соответственно. Рассчитаем вспомогательные переменные t_i по формуле $\frac{v_i}{s_{compare}}$.

Таблица 2.3. Дополнительные входные данные.

s_compare	1.00E+13 байт/сек		
s_map	1.00E+10 байт/сек		
k	15		
{t}	4E-13	8E-13	4E-13
V	**7000 байт**		

Выполним расчет значений целевой функции для всех вариантов стратегий (Таблица 2.4).

z1	z2	z3	Tmap	Tvector	Tlist	Ограничение	F
0	0	0	0	0.000000656	0.0000003	0	0.000000956
0	0	1	2.00717E-07	0.000000656	0	2000	8.56717E-07
0	1	0	5.61512E-07	0.0000006	0.0000003	5600	1.46151E-06
0	1	1	7.62229E-07	0.0000006	0	7600	+Infiniy
1	0	0	4.11959E-07	0.000000056	0.0000003	4000	7.67959E-07
1	0	1	6.12676E-07	0.000000056	0	6000	6.68676E-07

1	1	0	9.73471E-07	0	0.0000003	9600	+Infiniy
1	1	1	1.17419E-06	0	0	11600	+Infiniy
F min							**6.68676E-07**

Наилучшей стратегией является использование вспомогательных контейнеров std::map для 1-го и 3-го массивов, что обеспечит минимальное суммарное время поиска элементов в массивах пользовательского алгоритма, приведенного в Листинге 2.1.

Решение данного примера было получено с помощью программы Excel. Документ, в котором было получено решение, находится в облаке и получить доступ к нему можно по следующей ссылке: https://cloud.mail.ru/public/vBsp/csP9mA4WT.

В следующей главе приведено описание программной реализации описанных подходов.

Глава 3. Программная реализация выбранного алгоритма

3.1 Описание платформы .NET и языка C#

Платформа Microsoft .NET(и связанный с ней язык программирования C#) впервые была представлена примерно в 2002г. и быстро стала одной из основных современных сред разработки программного обеспечения.

До того, как компания Microsoft выпустила язык C# и платформу .NET, разработчики программного обеспечения, создававшие приложения для операционных систем семейства Windows, часто применяли модель программирования COM. Технология COM (Component Object Model – модель компонентных объектов) позволяла строить библиотеки, которые можно было использовать в различных языках программирования.

Когда говорят C#, нередко имеют в виду технологии платформы .NET (WPF, ASP.NET). И, наоборот, когда говорят .NET, нередко имеют в виду C#. Однако, хотя эти понятия связаны, отождествлять их неверно. Язык C# был создан специально для работы с фреймворком .NET, однако само понятие .NET несколько шире.

Синтаксис языка C# выглядит очень похожим на язык Java. На самом деле и Java, и C# являются членами семейства языков программирования, основанного на C (куда также входят C, Objective C, C++) и поэтому они разделяют схожий синтаксис.

Вследствие того, что C# представляет собой гибрид из нескольких языков, он является таким же синтаксически чистым, как и Java, почти столько же простым, как и VB, и практически таким же мощным и гибким, как C++.

C# является объектно-ориентированным и в этом плане много перенял у Java и C++. Например, C# поддерживает полиморфизм, наследование,

перегрузку операторов, статическую типизацию. Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в тоже время гибких, масштабируемых и расширяемых приложений. И C# продолжает активно развиваться, и с каждой новой версией появляется все больше интересных функциональностей, как, например, лямбды, динамическое связывание, асинхронные методы и т.д.

Основные преимущества платформы .NET

Целью создания языка C# и платформы .NET было обеспечение более мощной, гибкой и простой модели программирования по сравнению с COM. .NET Framework – это программная платформа для построения приложений на базе семейства операционных систем Windows, а также многочисленных операционных систем производства не Microsoft, таких как MacOS X и различные дистрибутивы Unix и Linux. Перечень некоторых ключевых средств, поддерживаемых .NET:

- *Возможность взаимодействовать с существующим кодом*. Эта возможность, несомненно, является очень полезной, поскольку позволяет комбинировать существующие двоичные компоненты COM (т.е. обеспечивать взаимодействие с ними) с более новыми программными компонентами .NET и наоборот. С выходом .NET 4.0 и последующих версий возможность взаимодействовать дополнительно упростилась благодаря добавлению ключевого слова dynamic;
- *Поддержка многочисленных языков программирования*. Приложение .NET можно создавать с использованием любого числа языков программирования (C#, VB, F# и т.д.);
- *Общий дополнительный механизм*, разделяемый всеми поддерживающими .NET языками. Одним из аспектов этого механизма является наличие хорошо определённого набора типов, которые способен понимать каждый поддерживающий .NET язык;
- *Языковая интеграция*. В .NET поддерживается межъязыковая наследование, межъязыковая обработка исключений и межъязыковая

отладка кода. Например, базовый класс может быть определён на C#, а затем расширен в VB;

- *Обширная библиотека базовых классов*. Эта библиотека позволяет избегать сложностей, связанных с выполнением низкоуровневых обращений к API– интерфейсам, и предполагает согласованную объектную модель, используемую всеми поддерживающими .NET языками;
- *Упрощённая модель развертывания*. В отличие от COM, библиотеки .NET не регистрируются в системном регистре. Более того, платформа .NET позволяет сосуществовать на одном и том же компьютере несколькими версиями одной и той же сборки *.dll.

JIT-компиляция

Код на C# компилируется в приложения или сборки с расширениями exe или dll на языке CIL. Далее при запуске на выполнение подобного приложения происходит JIT-компиляция (Just-In-Time) в машинный код, который затем выполняется. При этом, поскольку наше приложение может быть большим и содержать кучу инструкций, в текущий момент времени будет компилироваться лишь та часть приложения, к которой непосредственно идет обращение. Если мы обратимся к другой части кода, то она будет скомпилирована из CIL в машинный код. При том уже скомпилированная часть приложения сохраняется до завершения работы программы. В итоге это повышает производительность.

Сравнение управляемого и неуправляемого кода

Возможно, наиболее важный аспект, который следует знать о языке C#, заключается в том, что он порождает код, который может выполняться только в рамках исполняемой среды .NET (использовать C# для построения COM-сервера или неуправляемого приложения C/C++ не допускается). Выражаясь официально, для обозначения кода, ориентированного на исполняющую среду .NET, применяется термин, управляемый код.

Двоичный модуль, который содержит управляемый код, называется сборкой. В противоположность этому, код, который не может обслуживаться непосредственно исполняющей средой .NET, называется неуправляемым кодом.

3.2 Алгоритм работы программы

1. Анализируем текст программы, выбранный пользователем;
2. Разбиваем по шаблону весь текст и находим необходимые блоки кода;
3. Определяем, возможно, неоптимальные участки кода, путём проверки: если в найденных блоках имеется поиск элементов в контейнерах, то, считаем, что это неоптимальный код, и мы будем его рассматривать.
4. Определяем тип рассматриваемых контейнеров;
5. Методом Монте-Карло определяем оптимальное решение, т.е. решение, где выигрыш будет максимальным, и удовлетворяющее введённому ограничению;

3.3 Описание работы программы

Для реализации алгоритма, описанного выше, было разработано приложение на языке C#, платформы .NET 3.5 Windows Forms. Интерфейс программы представляет собой окно, содержащее элементы управления (кнопки) для загрузки пользователем файла с исходным кодом C++(рис. 3.1), а так же его анализом. Помимо них имеются иные элементы для вывода полученных данных анализа кода, а так же результат работы алгоритма.

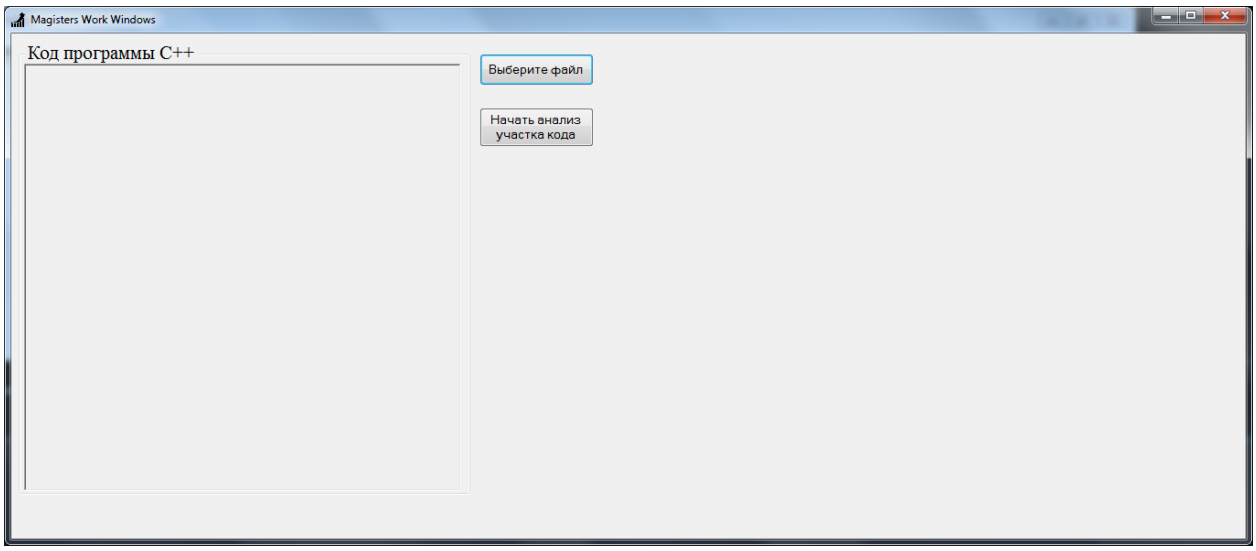

рис.3.1 – Внешний вид программы

Для загрузки файла с исходным кодом программы, написанной на C++, нажимаем на кнопку «Выберите файл». Откроется диалоговое окно, в котором нужно будет выбрать файл (рис. 3.2);

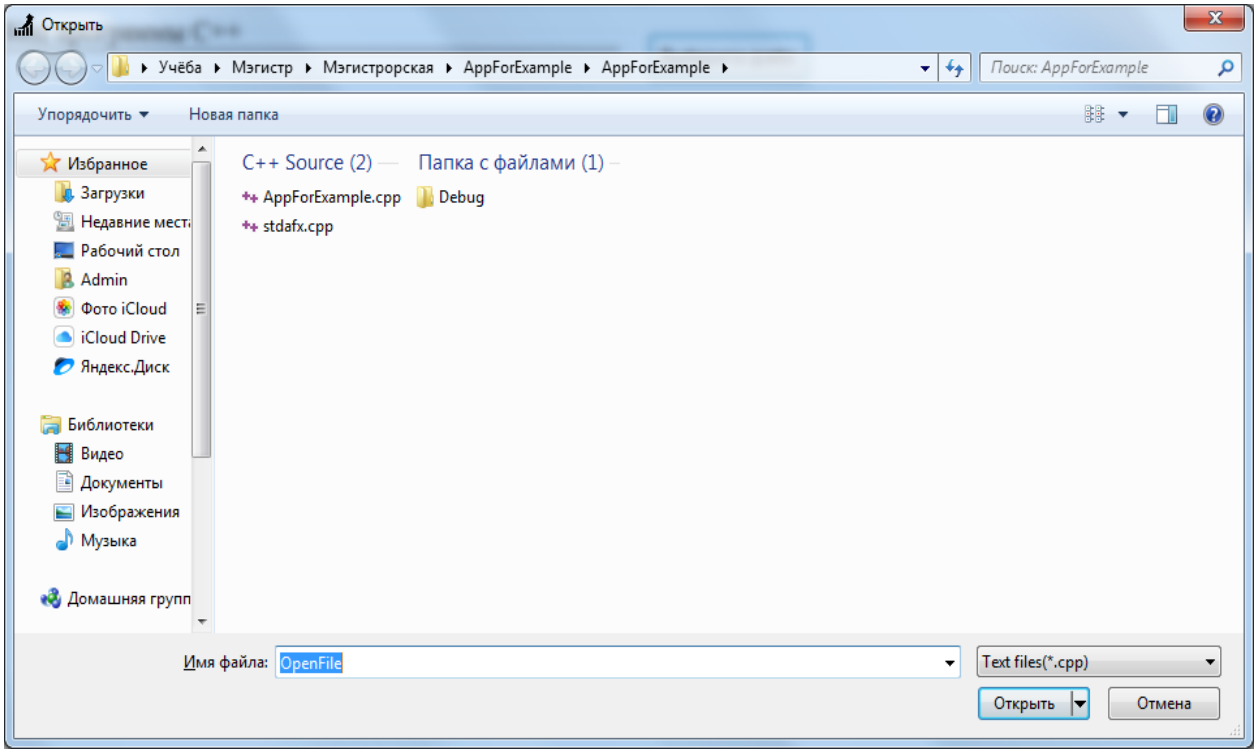

рис. 3.2 – Выбор файла

Когда пользователь выберет файл, то его содержимое, будет выведено на экран (3.3):

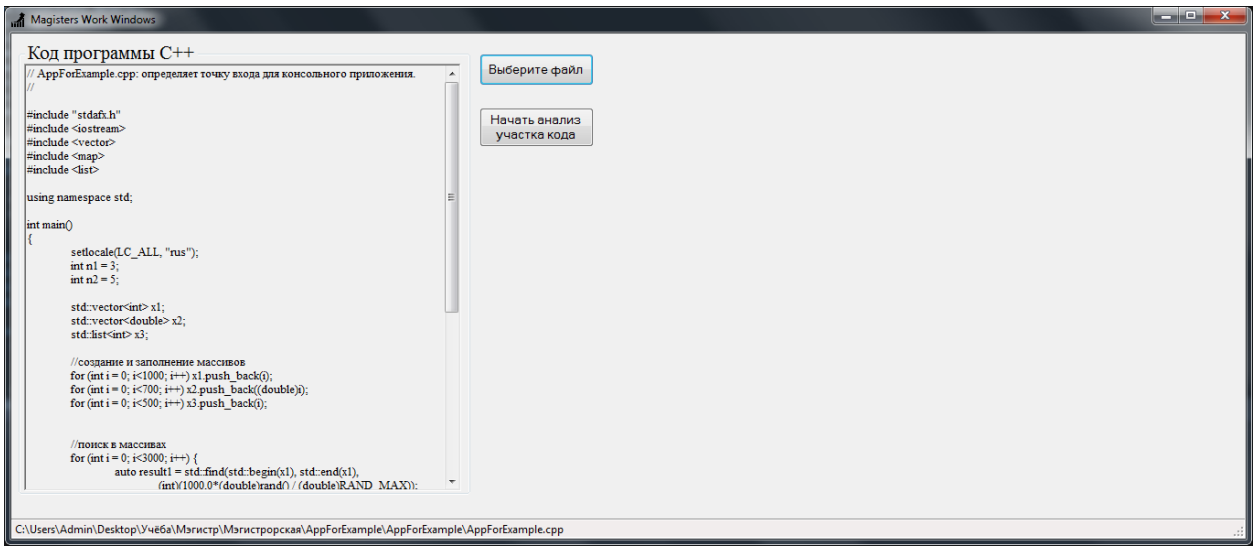

рис. 3.3 – Вывод текста из файла

После выбора файла, пользователь должен нажать на кнопку «Начать анализ участка кода», т.е. программа должна проанализировать код и вывести соответствующие результаты на экран (рис. 3.4);

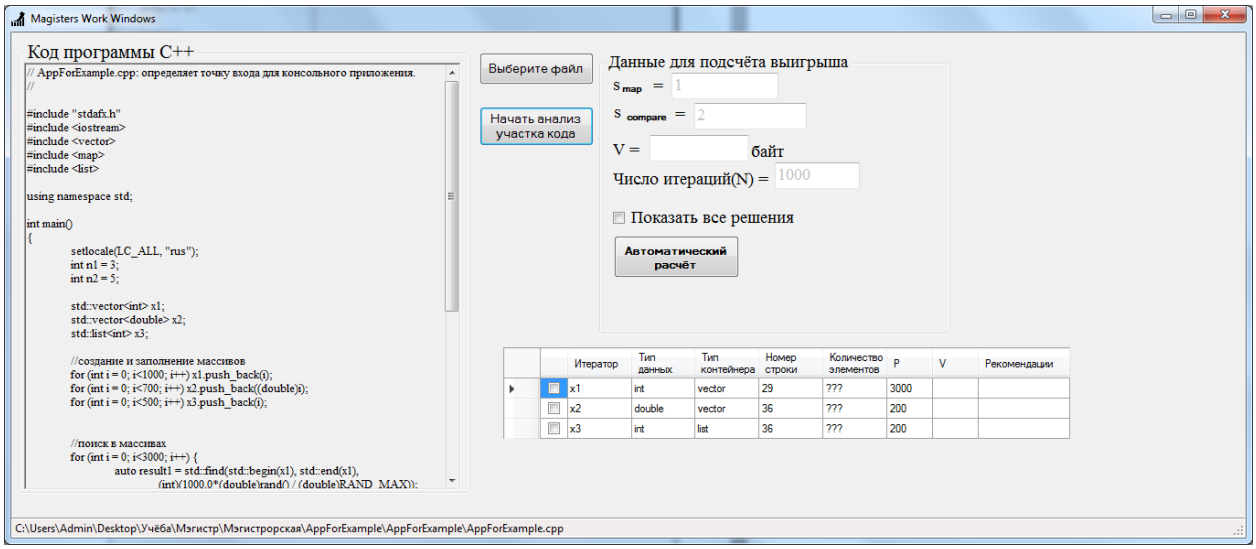

рис. 3.4 – Результат анализа участка кода

После анализа кода, пользователь должен ввести некоторые данные в специальном окне «Данные для подсчёта выигрыша», такие, как: s_{map}(по умолчанию 1), $s_{compare}$(по умолчанию 2), объём доступной ОП(V), число итераций для метода Монте-Карло(N = 10000 по умолчанию) и по желанию пользователя можно сделать активным флаг для вывода подробного

решения. Так же в таблице необходимо заполнить столбец «Количество элементов» у каждого контейнера (рис. 3.5);

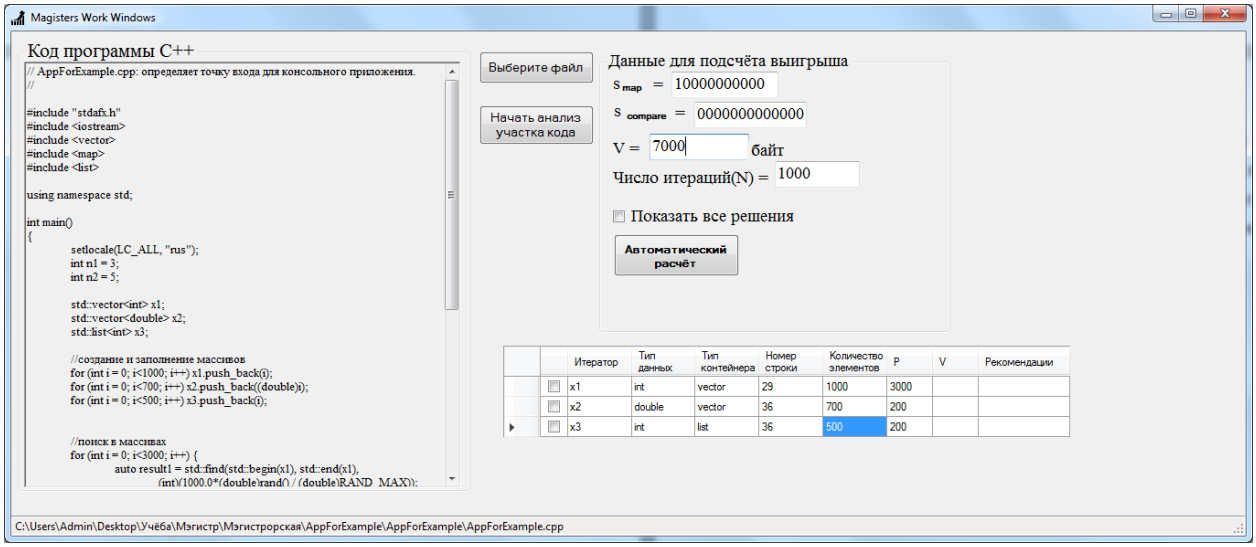

рис. 3.5 – Заполнение необходимых полей для подсчёта

Когда все данные введены, можно нажимать на кнопку «Автоматический расчёт». В результате полным перебором будет подобрана оптимальная последовательность с учётом целевой функции и ограничения (рис. 3.6);

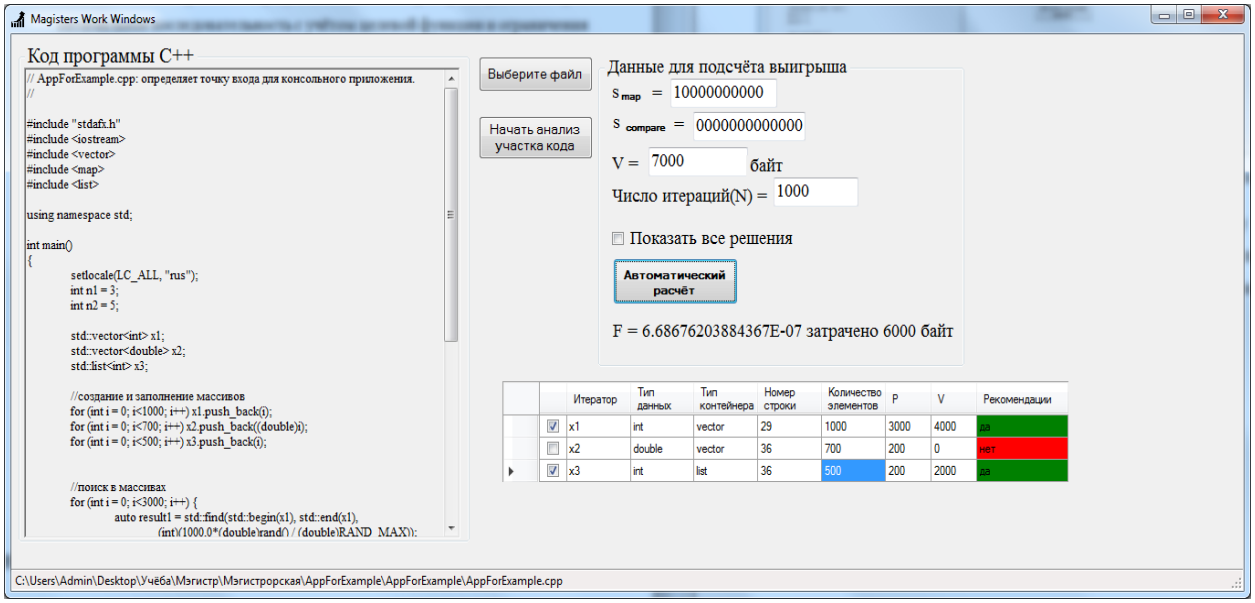

рис. 3.6 – Результат работы программы, при автоматическом расчёте

Так же пользователь сам может выбирать элементы, вопреки тому, что вывела программа, но колонка «Рекомендации» остаётся неизменной, в

случае если пользователь захочет использовать стратегию, предложенную программой.

Листинг программы, которая использовалась в качестве примера (Листинг 3.1):

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

```
//
#include "stdafx.h"
#include <iostream>
#include <vector>
#include <map>
#include <list>
using namespace std;
int main()
{
	setlocale(LC_ALL, "rus");
	int n1 = 3;
	int n2 = 5;
	std::vector<int> x1;
	std::vector<double> x2;
	std::list<int> x3;
	//создание и заполнение массивов
	for (int i = 0; i<1000; i++) x1.push_back(i);
	for (int i = 0; i<700; i++) x2.push_back((double)i);
	for (int i = 0; i<500; i++) x3.push_back(i);
	//поиск в массивах
	for (int i = 0; i<3000; i++) {
		auto result1 = std::find(std::begin(x1), std::end(x1),
			(int)(1000.0*(double)rand() / (double)RAND_MAX));
```

```
            std::cout << (result1 != x1.end() ? "Значение найдено!" :
                "Значение не найдено!");
        }
        for (int i = 0; i < 200; i++) {
            auto result2 = std::find(std::begin(x2), std::end(x2),
                700.0*(double)rand() / (double)RAND_MAX);
            std::cout << (result2 != x2.end() ? "Значение найдено!" :
                "Значение не найдено!");
            auto result3 = std::find(std::begin(x3), std::end(x3),
                (int)(500.0*(double)rand() / (double)RAND_MAX));
            std::cout << (result3 != x3.end() ? "Значение найдено!" :
                "Значение не найдено!");
        }
        system("pause");
    return 0;
}
```

Листинг 3.1 – Программа на C++ для примера решения

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):

	vector	list	map
100000	125	50310	12
200000	156	95550	15
300000	172	134192	16
400000	187	176921	17
500000	193	215281	18
600000	203	255217	19
700000	208	298763	19
800000	215	329749	21
900000	220	379051	23
1000000	223	419518	25

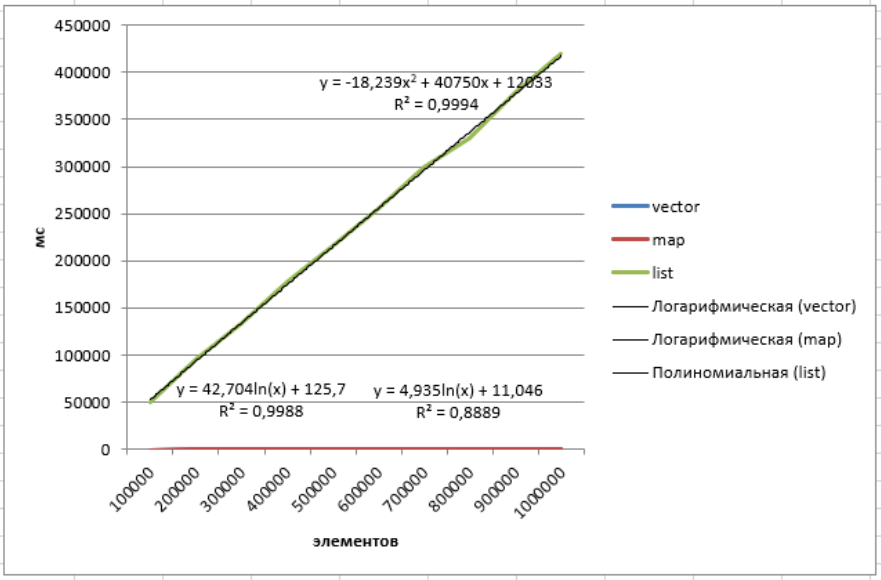

Рисунок 4.1.1 – Зависимость времени поиска элементов от их количества

	vector	list	map
100000	125	50310	12
200000	156	95550	15
300000	172	134192	16
400000	187	176921	17
500000	193	215281	18
600000	203	255217	19
700000	208	298763	19
800000	215	329749	21
900000	220	379051	23
1000000	223	419518	25

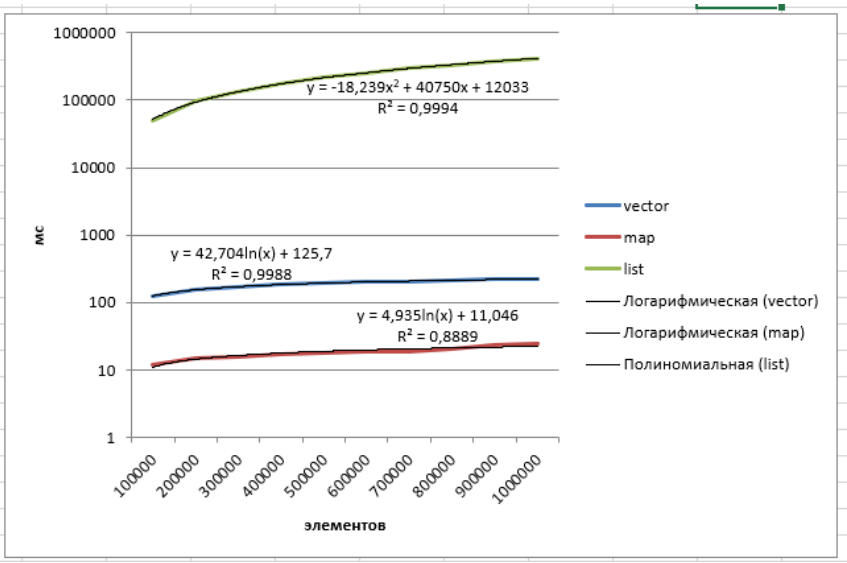

Рисунок 4.1.2 – Зависимость времени поиска элементов от их количества (логарифмический график)

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

N	T
1	6
2	8
3	11
4	14
5	17
6	21
7	24
8	27
9	30
10	32

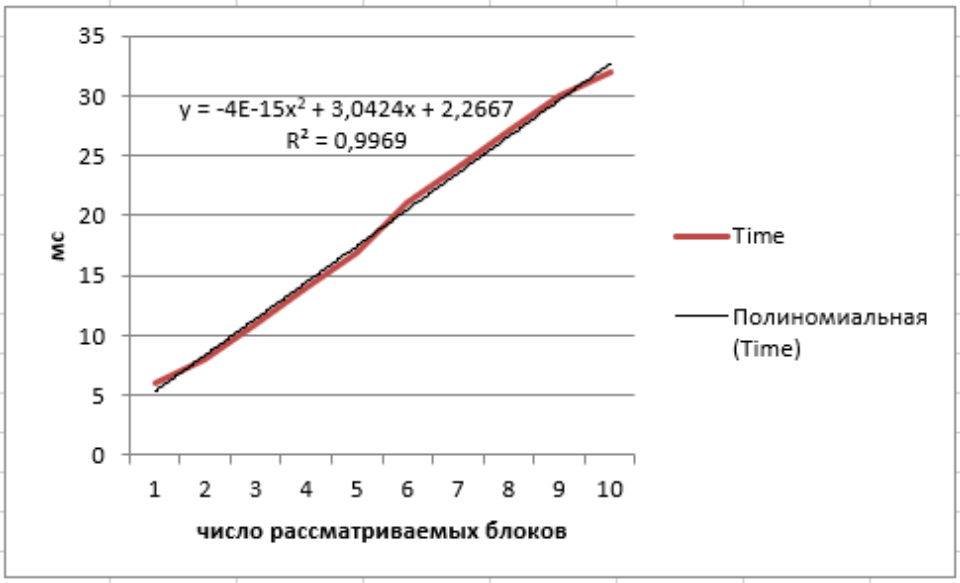

Рисунок 4.2.1 – Зависимость времени анализа программы, от размера кода

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 $s_{map} = 10000000000$, $s_{compare} = 10000000000000$, число итераций N = 10000 и верхняя граница используемой оперативной памяти составляла V = 7000 байт.

Результаты эксперимента представлены на графике 4.4:

N	T
1	37
2	52
3	80
4	107
5	129
6	150
7	177
8	198
9	230
10	255

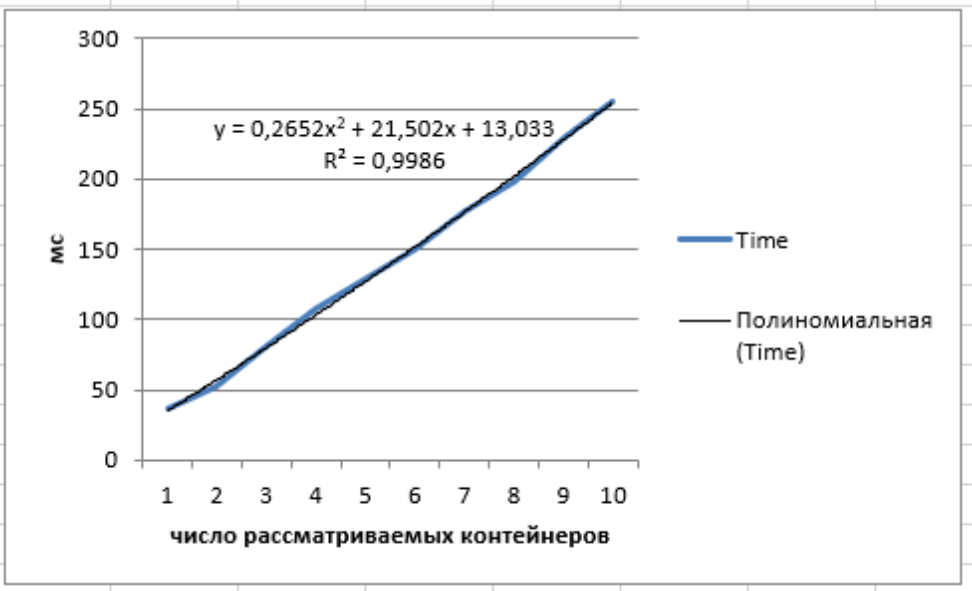

Рисунок 4.3.1 – Зависимости времени нахождения решения от количества блоков

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

Величина ошибки аппроксимации R^2, отображаемая на графике рассчитывается по формуле:

$$R^2 = \left(\frac{\sum_{i=1}^{n} \left(\left(y_i^{\text{эксп}} - \overline{y^{\text{эксп}}}\right)\left(y_i^{f} - \overline{y^{f}}\right) \right)}{\sqrt{\sum_{i=1}^{n} \left(y_i^{\text{эксп}} - \overline{y^{\text{эксп}}}\right) \sum_{i=1}^{n} \left(y_i^{f} - \overline{y^{f}}\right)}} \right)^2 \quad (3)$$

где

$y_i^{эксп}$ — значение, полученное в ходе эксперимента.

y_i^f — значение, вычисленное с использованием аналитической зависимости, полученной аппроксимацией.

$\overline{y^{эксп}}$ и $\overline{y^f}$ —средние значения соответственно экспериментальных и аналитических результатов измерений.

Заключение

В результате исследования и выполнения работы мной были рассмотрены различные методы оптимизации. В том числе и оптимизация программы, написанной на языке C++, где анализировался код приложения и, предлагалась замена участков кода программы.

Для осуществления данной задачи было разработано приложение на языке C#. Также, были проведены исследования и на основании их построены графики, которые показывают эффективность данной работы и оптимизации в целом.

Невозможно писать программу и сразу оптимизировать весь написанный код. Необходимо иметь большой опыт, чтобы сразу подмечать неоптимальные участки кода. Однако, после внедрения оптимизации в проект, эффективность кода повышается в разы. В данном случае, замена контейнеров даёт существенный прирост производительности, не смотря на затраченные ресурсы памяти.

Список литературы

1. Э. Таненбаум. Архитектура компьютера.
2. Вирт Н. Построение компиляторов.
3. Кнут Д. Искусство программирования, том 1. Основные алгоритмы.
4. Статья на habr.com Оптимизация C/C++ кода https://habr.com/ru/post/339406/ - 1 часть
5. Статья на habr.com Оптимизация C/C++ кода https://habr.com/ru/post/339492/ - 2 часть
6. https://www.viva64.com/ru/t/0084/
7. https://docs.microsoft.com/ru-ru/cpp/standard-library/stl-containers?view=vs-2019
8. https://habr.com/ru/company/ua-hosting/blog/278369/
9. https://ru.cppreference.com/w/cpp/container
10. Э.Троелсен. Язык програграммирования C# 5.0 и платформа .NET 4.5
11. Гроппен В.О. Принципы оптимизации программного обеспечения ЭВМ.// Изд. РГУ, Ростов-на-Дону, 1993г.
12. https://metanit.com/sharp/tutorial/1.1.php
13. https://professorweb.ru/my/csharp/charp_theory/level1/index.php

Приложение

Класс Form1:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;

namespace MastersWork {
   public partial class Form1 : Form {
      public Form1() {
         InitializeComponent();
         OpenFile.Filter = "Text files(*.cpp)|*.cpp|All files(*.*)|*.*";
         OpenFile.InitialDirectory =
AppDomain.CurrentDomain.RelativeSearchPath;
      }

      public Dictionary<string, string> lstWithNameAndTypeDataContainers;
      public List<string> lstWithTypeContainers;
      public List<int> lstWithCountOperationFindElementsInContainers;
      public List<int> lstWithIndexStrWithContainers;

      private void ToChooseFile_Click(object sender, EventArgs e) {
         if (OpenFile.ShowDialog() == DialogResult.Cancel)
            return;

         richTextBox1.Text = ReadFile(OpenFile.FileName); ;
         statusStrip1.Visible = true;
         toolStripStatusLabel1.Visible = true;
         toolStripStatusLabel1.Text = OpenFile.FileName;
      }

      private string ReadFile(string path) {
         string text = "";
         try {
            using (StreamReader read = new StreamReader(path,
System.Text.Encoding.Default)) {
               string line = "";
```

```
                while ((line = read.ReadLine()) != null) {
                    text += line + '\n';
                }
            }
        }
        catch (Exception ex) {
            MessageBox.Show("Ошибка чтения из файла: " + ex.Message + "\n" +
ex.Source);
        }
        return text;
    }

    private void CodeAnalysis_Click(object sender, EventArgs e) {
        if (OpenFile.FileName == "") {
            MessageBox.Show("Не выбран файл");
        }
        else {
            ParsingClass parsing = new ParsingClass(richTextBox1.Text);
            parsing.ParsingText();
            lstWithNameAndTypeDataContainers = parsing.infoContainers;
            lstWithTypeContainers = parsing.lstWithTypeIter;
            lstWithCountOperationFindElementsInContainers =
parsing.countOperationFindElement;
            lstWithIndexStrWithContainers = parsing.indexStrContainers;
            MyTable.Visible = true;
            MyTable.RowCount = lstWithNameAndTypeDataContainers.Count;
            ToFillTable();
            CalculationWin.Visible = true;
        }
    }

    /*
    * Заполнение таблицы начальными значениями
    */
    private void ToFillTable() {
        int index = 0;
        MyTable.Height = dataGridViewHeight();
        foreach (KeyValuePair<string, string> tmp in
lstWithNameAndTypeDataContainers) {
            MyTable.Rows[index].Cells[1].Value = tmp.Key;
            MyTable.Rows[index].Cells[2].Value = tmp.Value;
            index++;
        }
        for (int i = 0; i < lstWithNameAndTypeDataContainers.Count; i++) {
```

```
                MyTable.Rows[i].Cells[0].Value = false;
                MyTable.Rows[i].Cells[5].Value = "???";
                MyTable.Rows[i].Cells[3].Value = lstWithTypeContainers[i];
                MyTable.Rows[i].Cells[4].Value = lstWithIndexStrWithContainers[i];
                MyTable.Rows[i].Cells[6].Value =
lstWithCountOperationFindElementsInContainers[i];
            }
        }

        private int dataGridViewHeight() {
            int sum = MyTable.ColumnHeadersHeight;
            foreach (DataGridViewRow row in MyTable.Rows)
                sum += row.Height + 1;
            return sum;
        }

        public double myWin;
        double v, resultV, scomp, smap;
        int n;


        private void AutoCalcWinButton_Click(object sender, EventArgs e) {
            if (LimitMemory.Text == "" || CountIteration.Text == "" ||
IsCheckToPressButton() == false)
                MessageBox.Show("Не все параметры введены!");
            else {
                Stopwatch st = new Stopwatch();
                st.Start();
                byte[] perebWinMonte = new
byte[lstWithNameAndTypeDataContainers.Count];
                myWin = double.MaxValue; // лучшая значение целевой функции
                resultV = 0; // число задействованной памяти в лучшем переборе
                Random rnd = new Random();
                double[] arrayV = new
double[lstWithNameAndTypeDataContainers.Count]; // массив с занимаемой
памятью каждого контейнера
                byte[] perebor = new byte[lstWithNameAndTypeDataContainers.Count];
// массив выигрышного перебора
                v = Convert.ToDouble(LimitMemory.Text);
                n = Convert.ToInt32(CountIteration.Text);
                scomp = Convert.ToDouble(SCompareText.Text);
                smap = Convert.ToDouble(SMapText.Text);

                for (byte pere = 0; pere < perebor.Length; pere++) {
```

```
                MyTable.Rows[pere].Cells[0].Value = 0;
            }
            if (flagAllSolution.Checked == true)
                AllSolutionsTxtBox.Visible = true;
            else
                AllSolutionsTxtBox.Visible = false;
            for (int i = 0; i < n; i++) {
                for (int p = 0; p < perebor.Length; p++) perebor[p] = (byte)rnd.Next(0,
2);

                double tmpF = 0;
                double tmpV = 0;
                double[] tmpArr = new
double[lstWithNameAndTypeDataContainers.Count];
                for (int j = 0; j < perebor.Length; j++) {
                    if(flagAllSolution.Checked == true) AllSolutionsTxtBox.Text +=
perebor[j] + " ";
                    if (perebor[j] == 1) {
                        tmpV += Convert.ToInt16(MyTable.Rows[j].Cells[5].Value) *
ToDefineType(MyTable.Rows[j].Cells[2].Value.ToString());
                        double t =
ToDefineType(MyTable.Rows[j].Cells[2].Value.ToString()) / scomp;
                        tmpF += Convert.ToInt16(MyTable.Rows[j].Cells[6].Value) * t *
Math.Log(Convert.ToInt16(MyTable.Rows[j].Cells[5].Value), 2) +
((Convert.ToInt16(MyTable.Rows[j].Cells[5].Value) *
ToDefineType(MyTable.Rows[j].Cells[2].Value.ToString())) / smap);
                        tmpArr[j] = Convert.ToInt16(MyTable.Rows[j].Cells[5].Value) *
ToDefineType(MyTable.Rows[j].Cells[2].Value.ToString());
                    } else {
                        if (MyTable.Rows[j].Cells[3].Value.ToString() == "vector") {
                            double t =
ToDefineType(MyTable.Rows[j].Cells[2].Value.ToString()) / scomp;
                            tmpF += Convert.ToInt16(MyTable.Rows[j].Cells[6].Value) *
t * (Convert.ToInt16(MyTable.Rows[j].Cells[5].Value) / 2);
                        }
                        else {
                            double t =
ToDefineType(MyTable.Rows[j].Cells[2].Value.ToString()) / scomp;
                            tmpF += Convert.ToInt16(MyTable.Rows[j].Cells[6].Value) *
t * 15 * (Convert.ToInt16(MyTable.Rows[j].Cells[5].Value) / 2);
                        }
                    }
                }
```

```
                if (flagAllSolution.Checked == true) AllSolutionsTxtBox.Text +=
tmpF + " " + tmpV + "\n";
                if (tmpV <= v && myWin > tmpF) {
                    myWin = tmpF;
                    resultV = tmpV;
                    Array.Copy(tmpArr, arrayV, arrayV.Length);
                    Array.Copy(perebor, perebWinMonte, perebor.Length);
                }
            }

            for (int i = 0; i < perebor.Length; i++) {
                if (perebWinMonte[i] == 0) {
                    MyTable.Rows[i].Cells[8].Style.BackColor = Color.Red;
                    MyTable.Rows[i].Cells[8].Value = "нет";
                }
                else {
                    MyTable.Rows[i].Cells[0].Value = true;
                    MyTable.Rows[i].Cells[8].Style.BackColor = Color.Green;
                    MyTable.Rows[i].Cells[8].Value = "да";
                }
                MyTable.Rows[i].Cells[7].Value = arrayV[i];
            }

            labelAWithWin.Text = "";
            labelAWithWin.Text = "F = " + myWin + " затрачено " + resultV + "
байт";

            st.Stop();
            MessageBox.Show("Time: " + st.ElapsedMilliseconds);
        }
    }

    bool IsCheckToPressButton() {
        for (int i = 0; i < lstWithNameAndTypeDataContainers.Count; i++) {
            if ((MyTable.Rows[i].Cells[5].Value.Equals("???")) ||
(MyTable.Rows[i].Cells[5].Value.Equals("")) ||
(Convert.ToInt16(MyTable.Rows[i].Cells[5].Value) <= 0))
                return false;
        }
        return true;
    }

     double ToDefineType(string type) {
      switch (type) {
```

```
            case "byte":
                return 1;
            case "int":
                return 4;
            case "double":
                return 8;
            case "float":
                return 4;
            case "char":
                return 2;
            default:
                return 4;
        }
    }

    private void SMapText_Enter(object sender, EventArgs e) {
        SMapText.Text = "";
        SMapText.ForeColor = Color.Black;
    }

    private void SMapText_Leave(object sender, EventArgs e) {
        if (SMapText.Text == "") {
            SMapText.Text = "10000000000";
            SMapText.ForeColor = Color.Silver;
        }
    }

    private void SCompareText_Enter(object sender, EventArgs e) {
        SCompareText.Text = "";
        SCompareText.ForeColor = Color.Black;
    }

    int k = 0;
    private void MyTable_CurrentCellDirtyStateChanged(object sender,
EventArgs e) {
        if (Convert.ToInt16(MyTable.SelectedCells[0].ColumnIndex) != 0)
            return;
        k = 0;
        MyTable.EndEdit();
        if (k == 1)
            return;
        else {
            if (LimitMemory.Text == "" || CountIteration.Text == "" ||
IsCheckToPressButton() == false) {
```

```
                MyTable.Rows[MyTable.SelectedCells[0].RowIndex].Cells[0].Value
= 0;
                MessageBox.Show("Не все параметры введены!");
                k = 1;
                return;
            }
            else {
                k++;
                v = Convert.ToDouble(LimitMemory.Text);
                int index = MyTable.SelectedCells[0].RowIndex;
                if (Convert.ToBoolean(MyTable.CurrentCell.Value) == true) {
                    resultV += Convert.ToInt16(MyTable.Rows[index].Cells[5].Value)
* ToDefineType(MyTable.Rows[index].Cells[2].Value.ToString());
                    MyTable.Rows[index].Cells[7].Value = resultV;
                    scomp = Convert.ToDouble(SCompareText.Text);
                    smap = Convert.ToDouble(SMapText.Text);
                    double t =
ToDefineType(MyTable.Rows[index].Cells[2].Value.ToString()) / scomp;
                    myWin += Convert.ToInt16(MyTable.Rows[index].Cells[6].Value)
* t * Math.Log(Convert.ToInt16(MyTable.Rows[index].Cells[5].Value), 2) +
((Convert.ToInt16(MyTable.Rows[index].Cells[5].Value) *
ToDefineType(MyTable.Rows[index].Cells[2].Value.ToString())) / smap);
                    if (resultV <= v) {
                        labelAWithWin.Text = "";
                        labelAWithWin.Text = "F = " + myWin + " затрачено " +
resultV + " байт";
                    }
                    else {
                        labelAWithWin.Text = "";
                        labelAWithWin.Text = "Ограничение не выполняется, " + "
затрачено: " + resultV + " байт";
                    }
                    return;
                }
                else {
                    scomp = Convert.ToDouble(SCompareText.Text);
                    smap = Convert.ToDouble(SMapText.Text);
                    resultV -= Convert.ToInt16(MyTable.Rows[index].Cells[5].Value)
* ToDefineType(MyTable.Rows[index].Cells[2].Value.ToString());
                    double t =
ToDefineType(MyTable.Rows[index].Cells[2].Value.ToString()) / scomp;
                    myWin -= Convert.ToInt16(MyTable.Rows[index].Cells[6].Value)
* t * Math.Log(Convert.ToInt16(MyTable.Rows[index].Cells[5].Value), 2) +
```

```
((Convert.ToInt16(MyTable.Rows[index].Cells[5].Value) *
ToDefineType(MyTable.Rows[index].Cells[2].Value.ToString())) / smap);
                    MyTable.Rows[index].Cells[6].Value = 0;
                    if (resultV <= v) {
                        labelAWithWin.Text = "";
                        labelAWithWin.Text = "F = " + myWin + " затрачено " +
resultV + " байт";
                    }
                    else {
                        labelAWithWin.Text = "";
                        labelAWithWin.Text = "Ограничение не выполняется, " + "
затрачено: " + resultV + " байт";
                    }
                    return;
                }
            }
        }
    }

    private void SCompareText_Leave(object sender, EventArgs e) {
        if (SCompareText.Text == "") {
            SCompareText.Text = "10000000000000";
            SCompareText.ForeColor = Color.Silver;
        }
    }

    private void CountIteration_Enter(object sender, EventArgs e) {
        CountIteration.Text = "";
        CountIteration.ForeColor = Color.Black;
    }

    private void CountIteration_Leave(object sender, EventArgs e) {
        if (CountIteration.Text == "") {
            CountIteration.Text = "10000";
            CountIteration.ForeColor = Color.Silver;
        }
    }

  }
}
```

Класс ParsingClass:

```
using System;
using System.Collections.Generic;
using System.Text.RegularExpressions;

namespace MastersWork {
   class ParsingClass {
      private byte[] mas = new byte[2]; // 0 - открывающие скобки, 1 - закрывающие
      private List<string> lstWithBlocks = new List<string>(); // контейнер с блоками циклов
      public List<string> lstWithTypeIter = new List<string>(); // контейнер, где хранятся типы итераторов/контейнеров C++
      public Dictionary<string, string> infoContainers = new Dictionary<string, string>(); // массив, который будет передаваться в основную программу, содержащий имя и тип данных контейнера
      private List<int> indexList = new List<int>(); // контейнер с номерами строк циклов
      public List<int> countOperationFindElement = new List<int>(); // массив, который будет передаваться в основную программу, содержащий число операций поиска
      public List<int> indexStrContainers = new List<int>(); // массив, который будет передаваться в основную программу, содержащий строки, где указан неоптимизированный код
      private string ProgramText { get; }

      public ParsingClass(string programText) {
         ProgramText = programText;
      }

      /*
       * Разбиваем строку на массив строк, где разделитель - символ перехода на новую строчку.
       * Каждую строчку проверяем, используя регулярное выражение.
       * Если строка подходит, то записываем её во временную строку а также запоминаем строчку с началом цикла.
       * Так до тех пор, пока не будет найдена последняя закрывающая скобка.
       * mas[0] == mas[1]
       * Дойдя до конца блока, записываем его в специальный список, но прежде форматируем
       */
      public void ParsingText() {
         string[] splitText = ProgramText.Split('\n');
         Regex regex = new Regex(@"\w*for\w*");
```

```
            Regex pattern = new Regex(@"\s+");

            for (int i = 0; i < splitText.Length; i++) {
                if (regex.IsMatch(splitText[i])) {
                    indexList.Add(i + 1);
                    string strWithFor = "";
                    do {
                        string str = splitText[i];
                        SearchOpenBracket(str);
                        SearchCloseBracket(str);
                        strWithFor += str;
                        i++;

                    } while (mas[0] != mas[1]);
                    i--;
                    mas[0] = 0;
                    mas[1] = 0;
                    lstWithBlocks.Add(strWithFor);
                }
            }
            SearchIterators(splitText);
        }

        private void SearchOpenBracket(string str) {
            int pos = 0;
            while (str.IndexOf('{', pos) != -1) {
                mas[0]++;
                pos = str.IndexOf('{', pos) + 1;
            }
        }

        private void SearchCloseBracket(string str) {
            int pos = 0;
            while (str.IndexOf('}', pos) != -1) {
                mas[1]++;
                pos = str.IndexOf('}', pos) + 1;
            }
        }


        /*Поиск всех контейнеров типа list и vector с помощью регулярных
выражений*/
        private void SearchIterators(string[] splitText) {
            Regex regexList = new Regex(@"std::list\w*");
```

```
            Regex regexVector = new Regex(@"std::vector\w*");

            for (int i = 0; i < splitText.Length; i++) {
                if (regexList.IsMatch(splitText[i]) || regexVector.IsMatch(splitText[i])) {
                    string containerTypeData =
splitText[i].Substring(splitText[i].IndexOf('<') + 1, splitText[i].IndexOf('>') -
splitText[i].IndexOf('<') - 1);
                    string containerName = splitText[i].Split(' ', '=', ';')[1];
                    string containerType =
splitText[i].Substring(splitText[i].LastIndexOf(':') + 1, splitText[i].IndexOf('<') -
splitText[i].LastIndexOf(':') - 1);
                    IdentIterators(containerName, containerTypeData, containerType);
                }
            }
        }

        private void IdentIterators(string name, string typeD, string cType) {
            List<string> tmpMas = new List<string>();
            for (int i = 0; i < lstWithBlocks.Count; i++) {
                Regex regex = new Regex(@"\w*::find");
                if (regex.IsMatch(lstWithBlocks[i])){
                    string[] masString = lstWithBlocks[i].Split('\t');
                    for (int j = 2; j < masString.Length; j++) {
                        if (masString[j].Contains(name) && !tmpMas.Contains(name)) {
                            int countOperationFind =
Convert.ToInt16(masString[1].Substring(masString[1].IndexOf('<') + 1,
masString[1].LastIndexOf(';') - masString[1].IndexOf('<') - 1));
                            indexStrContainers.Add(indexList[i]);
                            countOperationFindElement.Add(countOperationFind);
                            tmpMas.Add(name);
                            infoContainers.Add(name, typeD);
                            lstWithTypeIter.Add(cType);
                            break;
                        }
                    }
                }
            }
        }
    }
}
```

Класс Program.cs:

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace MastersWork {
   static class Program {
      /// <summary>
      /// Главная точка входа для приложения.
      /// </summary>
      [STAThread]
      static void Main() {
         Application.EnableVisualStyles();
         Application.SetCompatibleTextRenderingDefault(false);
         Application.Run(new Form1());
      }
   }
}
```

Form1.Designer.cs

```
namespace MastersWork {
   partial class Form1 {
      /// <summary>
      /// Обязательная переменная конструктора.
      /// </summary>
      private System.ComponentModel.IContainer components = null;

      /// <summary>
      /// Освободить все используемые ресурсы.
      /// </summary>
      /// <param name="disposing">истинно, если управляемый ресурс должен
быть удален; иначе ложно.</param>
      protected override void Dispose(bool disposing) {
         if (disposing && (components != null)) {
            components.Dispose();
         }
         base.Dispose(disposing);
      }

/// <summary>
      /// Требуемый метод для поддержки конструктора — не изменяйте
      /// содержимое этого метода с помощью редактора кода.
      /// </summary>
      private void InitializeComponent() {
```

```
        System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(Form1));
        this.groupBox1 = new System.Windows.Forms.GroupBox();
        this.richTextBox1 = new System.Windows.Forms.RichTextBox();
        this.CodeAnalysis = new System.Windows.Forms.Button();
        this.ToChooseFile = new System.Windows.Forms.Button();
        this.OpenFile = new System.Windows.Forms.OpenFileDialog();
        this.statusStrip1 = new System.Windows.Forms.StatusStrip();
        this.toolStripStatusLabel1 = new
System.Windows.Forms.ToolStripStatusLabel();
        this.CalculationWin = new System.Windows.Forms.GroupBox();
        this.flagAllSolution = new System.Windows.Forms.CheckBox();
        this.SCompareText = new System.Windows.Forms.TextBox();
        this.SMapText = new System.Windows.Forms.TextBox();
        this.label3 = new System.Windows.Forms.Label();
        this.label2 = new System.Windows.Forms.Label();
        this.label4 = new System.Windows.Forms.Label();
        this.label1 = new System.Windows.Forms.Label();
        this.label5 = new System.Windows.Forms.Label();
        this.label6 = new System.Windows.Forms.Label();
        this.CountIteration = new System.Windows.Forms.TextBox();
        this.label9 = new System.Windows.Forms.Label();
        this.AutoCalcWinButton = new System.Windows.Forms.Button();
        this.labelAWithWin = new System.Windows.Forms.Label();
        this.label7 = new System.Windows.Forms.Label();
        this.label8 = new System.Windows.Forms.Label();
        this.LimitMemory = new System.Windows.Forms.TextBox();
        this.MyTable = new System.Windows.Forms.DataGridView();
        this.Column1 = new
System.Windows.Forms.DataGridViewCheckBoxColumn();
        this.Column2 = new
System.Windows.Forms.DataGridViewTextBoxColumn();
        this.Column8 = new
System.Windows.Forms.DataGridViewTextBoxColumn();
        this.Column6 = new
System.Windows.Forms.DataGridViewTextBoxColumn();
        this.Column3 = new
System.Windows.Forms.DataGridViewTextBoxColumn();
        this.Column5 = new
System.Windows.Forms.DataGridViewTextBoxColumn();
        this.Column9 = new
System.Windows.Forms.DataGridViewTextBoxColumn();
        this.Column7 = new
System.Windows.Forms.DataGridViewTextBoxColumn();
```

```
            this.Column4 = new
System.Windows.Forms.DataGridViewTextBoxColumn();
            this.AllSolutionsTxtBox = new System.Windows.Forms.RichTextBox();
            this.groupBox1.SuspendLayout();
            this.statusStrip1.SuspendLayout();
            this.CalculationWin.SuspendLayout();
            ((System.ComponentModel.ISupportInitialize)(this.MyTable)).BeginInit();
            this.SuspendLayout();
            //
            // groupBox1
            //
            this.groupBox1.Controls.Add(this.richTextBox1);
            this.groupBox1.Font = new System.Drawing.Font("Times New Roman",
15.75F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
            this.groupBox1.Location = new System.Drawing.Point(8, 10);
            this.groupBox1.Name = "groupBox1";
            this.groupBox1.Size = new System.Drawing.Size(494, 494);
            this.groupBox1.TabIndex = 11;
            this.groupBox1.TabStop = false;
            this.groupBox1.Text = "Код программы C++";
            //
            // richTextBox1
            //
            this.richTextBox1.Font = new System.Drawing.Font("Times New Roman",
9.75F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
            this.richTextBox1.Location = new System.Drawing.Point(6, 23);
            this.richTextBox1.Name = "richTextBox1";
            this.richTextBox1.ReadOnly = true;
            this.richTextBox1.Size = new System.Drawing.Size(478, 467);
            this.richTextBox1.TabIndex = 6;
            this.richTextBox1.Text = "";
            //
            // CodeAnalysis
            //
            this.CodeAnalysis.Font = new System.Drawing.Font("Microsoft Sans
Serif", 9.75F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(204)));
            this.CodeAnalysis.Location = new System.Drawing.Point(512, 81);
            this.CodeAnalysis.Name = "CodeAnalysis";
            this.CodeAnalysis.Size = new System.Drawing.Size(125, 43);
            this.CodeAnalysis.TabIndex = 10;
            this.CodeAnalysis.Text = "Начать анализ участка кода";
```

```
            this.CodeAnalysis.UseVisualStyleBackColor = true;
            this.CodeAnalysis.Click += new
System.EventHandler(this.CodeAnalysis_Click);
            //
            // ToChooseFile
            //
            this.ToChooseFile.Font = new System.Drawing.Font("Microsoft Sans
Serif", 9.75F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(204)));
            this.ToChooseFile.Location = new System.Drawing.Point(512, 22);
            this.ToChooseFile.Name = "ToChooseFile";
            this.ToChooseFile.Size = new System.Drawing.Size(125, 35);
            this.ToChooseFile.TabIndex = 9;
            this.ToChooseFile.Text = "Выберите файл";
            this.ToChooseFile.UseVisualStyleBackColor = true;
            this.ToChooseFile.Click += new
System.EventHandler(this.ToChooseFile_Click);
            //
            // OpenFile
            //
            this.OpenFile.FileName = "OpenFile";
            //
            // statusStrip1
            //
            this.statusStrip1.Items.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.toolStripStatusLabel1});
            this.statusStrip1.Location = new System.Drawing.Point(0, 530);
            this.statusStrip1.Name = "statusStrip1";
            this.statusStrip1.Size = new System.Drawing.Size(1039, 22);
            this.statusStrip1.TabIndex = 7;
            this.statusStrip1.Text = "statusStrip1";
            this.statusStrip1.Visible = false;
            //
            // toolStripStatusLabel1
            //
            this.toolStripStatusLabel1.Name = "toolStripStatusLabel1";
            this.toolStripStatusLabel1.Size = new System.Drawing.Size(118, 17);
            this.toolStripStatusLabel1.Text = "toolStripStatusLabel1";
            //
            // CalculationWin
            //
            this.CalculationWin.AutoSize = true;
            this.CalculationWin.Controls.Add(this.flagAllSolution);
```

```
            this.CalculationWin.Controls.Add(this.SCompareText);
            this.CalculationWin.Controls.Add(this.SMapText);
            this.CalculationWin.Controls.Add(this.label3);
            this.CalculationWin.Controls.Add(this.label2);
            this.CalculationWin.Controls.Add(this.label4);
            this.CalculationWin.Controls.Add(this.label1);
            this.CalculationWin.Controls.Add(this.label5);
            this.CalculationWin.Controls.Add(this.label6);
            this.CalculationWin.Controls.Add(this.CountIteration);
            this.CalculationWin.Controls.Add(this.label9);
            this.CalculationWin.Controls.Add(this.AutoCalcWinButton);
            this.CalculationWin.Controls.Add(this.labelAWithWin);
            this.CalculationWin.Controls.Add(this.label7);
            this.CalculationWin.Controls.Add(this.label8);
            this.CalculationWin.Controls.Add(this.LimitMemory);
            this.CalculationWin.Font = new System.Drawing.Font("Times New
Roman", 14.25F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(204)));
            this.CalculationWin.Location = new System.Drawing.Point(643, 22);
            this.CalculationWin.Name = "CalculationWin";
            this.CalculationWin.Size = new System.Drawing.Size(384, 308);
            this.CalculationWin.TabIndex = 12;
            this.CalculationWin.TabStop = false;
            this.CalculationWin.Text = "Данные для подсчёта выигрыша";
            this.CalculationWin.Visible = false;
            //
            // flagAllSolution
            //
            this.flagAllSolution.AutoSize = true;
            this.flagAllSolution.Location = new System.Drawing.Point(15, 169);
            this.flagAllSolution.Name = "flagAllSolution";
            this.flagAllSolution.Size = new System.Drawing.Size(207, 25);
            this.flagAllSolution.TabIndex = 21;
            this.flagAllSolution.Text = "Показать все решения";
            this.flagAllSolution.UseVisualStyleBackColor = true;
            //
            // SCompareText
            //
            this.SCompareText.ForeColor = System.Drawing.Color.Silver;
            this.SCompareText.Location = new System.Drawing.Point(104, 55);
            this.SCompareText.Name = "SCompareText";
            this.SCompareText.Size = new System.Drawing.Size(143, 29);
            this.SCompareText.TabIndex = 20;
            this.SCompareText.Text = "10000000000000";
```

```
            this.SCompareText.Enter += new
System.EventHandler(this.SCompareText_Enter);
            this.SCompareText.Leave += new
System.EventHandler(this.SCompareText_Leave);
            //
            // SMapText
            //
            this.SMapText.ForeColor = System.Drawing.Color.Silver;
            this.SMapText.Location = new System.Drawing.Point(79, 22);
            this.SMapText.Name = "SMapText";
            this.SMapText.Size = new System.Drawing.Size(117, 29);
            this.SMapText.TabIndex = 19;
            this.SMapText.Text = "10000000000";
            this.SMapText.Enter += new System.EventHandler(this.SMapText_Enter);
            this.SMapText.Leave                    +=                    new
System.EventHandler(this.SMapText_Leave);
//
            // label3
            //
            this.label3.AutoSize = true;
            this.label3.Location = new System.Drawing.Point(50, 26);
            this.label3.Name = "label3";
            this.label3.Size = new System.Drawing.Size(31, 21);
            this.label3.TabIndex = 15;
            this.label3.Text = " = ";
            //
            // label2
            //
            this.label2.AutoSize = true;
            this.label2.Font = new System.Drawing.Font("Microsoft Sans Serif",
6.75F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
            this.label2.Location = new System.Drawing.Point(23, 33);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(26, 12);
            this.label2.TabIndex = 14;
            this.label2.Text = "map";
            //
            // label4
            //
            this.label4.AutoSize = true;
            this.label4.Location = new System.Drawing.Point(12, 55);
            this.label4.Name = "label4";
```

```
            this.label4.Size = new System.Drawing.Size(18, 21);
            this.label4.TabIndex = 16;
            this.label4.Text = "s";
            //
            // label1
            //
            this.label1.AutoSize = true;
            this.label1.Location = new System.Drawing.Point(11, 25);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(18, 21);
            this.label1.TabIndex = 13;
            this.label1.Text = "s";
            //
            // label5
            //
            this.label5.AutoSize = true;
            this.label5.Font = new System.Drawing.Font("Microsoft Sans Serif",
6.75F, System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
            this.label5.Location = new System.Drawing.Point(28, 64);
            this.label5.Name = "label5";
            this.label5.Size = new System.Drawing.Size(48, 12);
            this.label5.TabIndex = 17;
            this.label5.Text = "compare";
            //
            // label6
            //
            this.label6.AutoSize = true;
            this.label6.Location = new System.Drawing.Point(74, 57);
            this.label6.Name = "label6";
            this.label6.Size = new System.Drawing.Size(31, 21);
            this.label6.TabIndex = 18;
            this.label6.Text = " = ";
            //
            // CountIteration
            //
            this.CountIteration.ForeColor = System.Drawing.Color.Silver;
            this.CountIteration.Location = new System.Drawing.Point(192, 120);
            this.CountIteration.Name = "CountIteration";
            this.CountIteration.Size = new System.Drawing.Size(93, 29);
            this.CountIteration.TabIndex = 12;
            this.CountIteration.Text = "10000";
            this.CountIteration.Enter += new
System.EventHandler(this.CountIteration_Enter);
```

```
            this.CountIteration.Leave += new
System.EventHandler(this.CountIteration_Leave);
            //
            // label9
            //
            this.label9.AutoSize = true;
            this.label9.Location = new System.Drawing.Point(11, 128);
            this.label9.Name = "label9";
            this.label9.Size = new System.Drawing.Size(185, 21);
            this.label9.TabIndex = 11;
            this.label9.Text = "Число итераций(N) = ";
            //
            // AutoCalcWinButton
            //
            this.AutoCalcWinButton.Font = new System.Drawing.Font("Microsoft
Sans Serif", 9F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(204)));
            this.AutoCalcWinButton.Location = new System.Drawing.Point(16, 200);
            this.AutoCalcWinButton.Name = "AutoCalcWinButton";
            this.AutoCalcWinButton.Size = new System.Drawing.Size(137, 46);
            this.AutoCalcWinButton.TabIndex = 7;
            this.AutoCalcWinButton.Text = "Автоматический расчёт";
            this.AutoCalcWinButton.UseVisualStyleBackColor = true;
            this.AutoCalcWinButton.Click += new
System.EventHandler(this.AutoCalcWinButton_Click);
            //
            // labelAWithWin
            //
            this.labelAWithWin.AutoSize = true;
            this.labelAWithWin.Location = new System.Drawing.Point(12, 262);
            this.labelAWithWin.Name = "labelAWithWin";
            this.labelAWithWin.Size = new System.Drawing.Size(0, 21);
            this.labelAWithWin.TabIndex = 6;
            //
            // label7
            //
            this.label7.AutoSize = true;
            this.label7.Location = new System.Drawing.Point(11, 96);
            this.label7.Name = "label7";
            this.label7.Size = new System.Drawing.Size(44, 21);
            this.label7.TabIndex = 6;
            this.label7.Text = "V = ";
            //
            // label8
```

```
            //
            this.label8.AutoSize = true;
            this.label8.Location = new System.Drawing.Point(162, 98);
            this.label8.Name = "label8";
            this.label8.Size = new System.Drawing.Size(46, 21);
            this.label8.TabIndex = 8;
            this.label8.Text = "байт";
            //
            // LimitMemory
            //
            this.LimitMemory.Location = new System.Drawing.Point(55, 90);
            this.LimitMemory.Name = "LimitMemory";
            this.LimitMemory.Size = new System.Drawing.Size(109, 29);
            this.LimitMemory.TabIndex = 7;

//
            // MyTable
            //
            this.MyTable.AllowUserToAddRows = false;
            this.MyTable.AllowUserToDeleteRows = false;
            this.MyTable.BackgroundColor = System.Drawing.SystemColors.Control;
            this.MyTable.BorderStyle = System.Windows.Forms.BorderStyle.None;
            this.MyTable.ColumnHeadersHeightSizeMode =
System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize
;
            this.MyTable.Columns.AddRange(new
System.Windows.Forms.DataGridViewColumn[] {
            this.Column1,
            this.Column2,
            this.Column8,
            this.Column6,
            this.Column3,
            this.Column5,
            this.Column9,
            this.Column7,
            this.Column4});
            this.MyTable.Location = new System.Drawing.Point(538, 344);
            this.MyTable.Name = "MyTable";
            this.MyTable.ScrollBars = System.Windows.Forms.ScrollBars.None;
            this.MyTable.Size = new System.Drawing.Size(625, 44);
            this.MyTable.TabIndex = 13;
            this.MyTable.Visible = false;
            this.MyTable.CurrentCellDirtyStateChanged += new
System.EventHandler(this.MyTable_CurrentCellDirtyStateChanged);
            //
```

```
            // Column1
            //
            this.Column1.FillWeight = 6.791172F;
            this.Column1.HeaderText = "";
            this.Column1.Name = "Column1";
            this.Column1.Width = 30;
            //
            // Column2
            //
            this.Column2.FillWeight = 119.1196F;
            this.Column2.HeaderText = "Итератор";
            this.Column2.Name = "Column2";
            this.Column2.ReadOnly = true;
            this.Column2.Resizable =
System.Windows.Forms.DataGridViewTriState.False;
            this.Column2.Width = 70;
            //
            // Column8
            //
            this.Column8.FillWeight = 72.88135F;
            this.Column8.HeaderText = "Тип данных";
            this.Column8.Name = "Column8";
            this.Column8.ReadOnly = true;
            this.Column8.Width = 69;
            //
            // Column6
            //
            this.Column6.HeaderText = "Тип контейнера";
            this.Column6.Name = "Column6";
            this.Column6.ReadOnly = true;
            this.Column6.Width = 70;
            //
            // Column3
            //
            this.Column3.FillWeight = 138.9695F;
            this.Column3.HeaderText = "Номер строки";
            this.Column3.Name = "Column3";
            this.Column3.ReadOnly = true;
            this.Column3.Width = 70;
            //
            // Column5
            //
            this.Column5.FillWeight = 184.1371F;
            this.Column5.HeaderText = "Количество элементов";
```

```
this.Column5.Name = "Column5";
this.Column5.Width = 70;
//
// Column9
//
this.Column9.HeaderText = "P";
this.Column9.Name = "Column9";
this.Column9.ReadOnly = true;
this.Column9.Width = 50;
//
// Column7
//
this.Column7.FillWeight = 44.89193F;
this.Column7.HeaderText = "V";
this.Column7.Name = "Column7";
this.Column7.ReadOnly = true;
this.Column7.Width = 50;
//
// Column4
//
this.Column4.FillWeight = 130.235F;
this.Column4.HeaderText = "Рекомендации";
this.Column4.Name = "Column4";
this.Column4.ReadOnly = true;
//
// AllSolutionsTxtBox
//
this.AllSolutionsTxtBox.Location = new System.Drawing.Point(1061, 36);
this.AllSolutionsTxtBox.Name = "AllSolutionsTxtBox";
this.AllSolutionsTxtBox.Size = new System.Drawing.Size(264, 256);
this.AllSolutionsTxtBox.TabIndex = 14;
this.AllSolutionsTxtBox.Text = "";
this.AllSolutionsTxtBox.Visible = false;
//
// Form1
//
this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.ClientSize = new System.Drawing.Size(1350, 552);
this.Controls.Add(this.AllSolutionsTxtBox);
this.Controls.Add(this.MyTable);
this.Controls.Add(this.CalculationWin);
this.Controls.Add(this.groupBox1);
this.Controls.Add(this.CodeAnalysis);
```

```
            this.Controls.Add(this.ToChooseFile);
            this.Controls.Add(this.statusStrip1);
            this.Icon = ((System.Drawing.Icon)(resources.GetObject("$this.Icon")));
            this.Name = "Form1";
            this.Text = "Magisters Work Windows";
            this.groupBox1.ResumeLayout(false);
            this.statusStrip1.ResumeLayout(false);
            this.statusStrip1.PerformLayout();
            this.CalculationWin.ResumeLayout(false);
            this.CalculationWin.PerformLayout();
            ((System.ComponentModel.ISupportInitialize)(this.MyTable)).EndInit();
            this.ResumeLayout(false);
            this.PerformLayout();

        }

    private System.Windows.Forms.GroupBox groupBox1;
        private System.Windows.Forms.RichTextBox richTextBox1;
        private System.Windows.Forms.Button CodeAnalysis;
        private System.Windows.Forms.Button ToChooseFile;
        private System.Windows.Forms.OpenFileDialog OpenFile;
        private System.Windows.Forms.StatusStrip statusStrip1;
        private System.Windows.Forms.ToolStripStatusLabel toolStripStatusLabel1;
        private System.Windows.Forms.GroupBox CalculationWin;
        private System.Windows.Forms.Label label9;
        private System.Windows.Forms.Button AutoCalcWinButton;
        private System.Windows.Forms.Label labelAWithWin;
        private System.Windows.Forms.Label label7;
        private System.Windows.Forms.Label label8;
        private System.Windows.Forms.TextBox LimitMemory;
        private System.Windows.Forms.TextBox CountIteration;
        private System.Windows.Forms.DataGridView MyTable;
        private System.Windows.Forms.DataGridViewCheckBoxColumn Column1;
        private System.Windows.Forms.DataGridViewTextBoxColumn Column2;
        private System.Windows.Forms.DataGridViewTextBoxColumn Column8;
        private System.Windows.Forms.DataGridViewTextBoxColumn Column6;
        private System.Windows.Forms.DataGridViewTextBoxColumn Column3;
        private System.Windows.Forms.DataGridViewTextBoxColumn Column5;
        private System.Windows.Forms.DataGridViewTextBoxColumn Column9;
        private System.Windows.Forms.DataGridViewTextBoxColumn Column7;
        private System.Windows.Forms.DataGridViewTextBoxColumn Column4;
        private System.Windows.Forms.TextBox SCompareText;
        private System.Windows.Forms.TextBox SMapText;
        private System.Windows.Forms.Label label3;
        private System.Windows.Forms.Label label2;
```

```
        private System.Windows.Forms.Label label4;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Label label5;
        private System.Windows.Forms.Label label6;
        private System.Windows.Forms.RichTextBox AllSolutionsTxtBox;
        private System.Windows.Forms.CheckBox flagAllSolution;
    }
}
```