Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

Реализация алгоритма сжатия данных

Студентка гр. 953501

Ю. И. Голубович

Руководитель ассистент кафедры информатики

И. А. Удовин

Минск 2020

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой Информатики––––––––––––––––

(подпись)

Волорова Н.А. 2020 г.

ЗАДАНИЕ

по курсовому проекту

Студентке   *Голубович Юлии Игоревне*–––––––– –––––\_\_\_–

1. Тема работы *Реализация алгоритма сжатия данных*\_\_\_\_\_\_\_\_-\_––––\_\_

2. Срок сдачи студентом законченной работы 3*31.05.2020 г*.–––   \_\_\_\_ \_

3. Исходные данные к работе *Операционная система Windows. Язык программирования C++.*

4. Содержание расчётно-пояснительной записки (перечень вопросов, которые подлежат разработке)

*Введение. 1. Анализ предметной области. 2. Разработка. 3. Заключение. Список использованных источников. Приложения../*

5. Консультант по курсовой работе *Удовин И. А.*

6. Дата выдачи задания *01.02.2020 г.*

7. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего объёма работы):

*раздел 1, Введение к 28.03.2020г. – 15 % готовности работы;   
раздел 2 к 15.04.2020г. – 40 % готовности работы;  
раздел 3 к 15.05.2020г. – 70 % готовности работы;  
Заключение, Приложения к 25.05.2020г.– 90 % готовности работы; оформление пояснительной записки и графического материала к 24.05.2020г. – 100 % готовности работы.  
Защита курсового проекта с 31.05.2020 г. по 01.06.2020 г. .*

РУКОВОДИТЕЛЬ *Удовин И.А.*

(подпись)

Задание приняла к исполнению *Голубович Ю. И. 01.02.2020 г.* (дата и подпись студента)

**Оглавление**

[Анализ предметной области 8](#_Toc42171767)

[1.1 Что такое сжатие данных 8](#_Toc42171768)

[1.2 Классификация алгоритмов сжатия 8](#_Toc42171769)

[1.3 Критерии оценки методов сжатия 10](#_Toc42171770)

[1.4 Надежность программ и сложность алгоритмов 11](#_Toc42171771)

[1.5 Алгоритм сжатия методом Хаффмана 11](#_Toc42171772)

[Разработка 15](#_Toc42171773)

[2.1 Выбор инструментария 15](#_Toc42171774)

[2.2 Постановка задачи 15](#_Toc42171775)

[2.3 Реализация алгоритма сжатия методом Хаффмана 16](#_Toc42171776)

[Заключение 19](#_Toc42171777)

[Список источников 20](#_Toc42171778)

[Приложения 21](#_Toc42171779)

[21](#_Toc42171780)

**Введение**

Сжатие информации - проблема, имеющая достаточно давнюю историю, гораздо более давнюю, нежели история развития вычислительной техники, которая (история) обычно шла параллельно с историей развития проблемы кодирования и шифровки информации.

Сжатие сокращает объем пространства, которое требуется для хранения файлов в ЭВМ, и количество времени, необходимое для передачи информации по каналу установ­ленной ширины пропускания. Это есть форма кодирования. Другими целями кодиро­вания являются поиск и исправление ошибок, а также шифрование. Процесс поиска и исправления ошибок противоположен сжатию - он увеличивает избыточность дан­ных, когда их не нужно представлять в удобной для восприятия человеком форме. Удаляя из текста избыточность, сжатие способствует шифрованию, что затрудняет поиск шифра доступным для взломщика статистическим методом.

Существующие алгоритмы сжатия данных можно разделить на два больших класса – с потерями, и без. Алгоритмы с потерями обычно применяются для сжатия изображений и аудио. Эти алгоритмы позволяют достичь больших степеней сжатия благодаря избирательной потере качества. Однако, по определению, восстановить первоначальные данные из сжатого результата невозможно. Алгоритмы сжатия без потерь применяются для уменьшения размера данных, и работают таким образом, что возможно восстановить данные в точности такими, какие они были до сжатия. Они применяются в коммуникациях, архиваторах и некоторых алгоритмах сжатии аудио и графической информации. Основной принцип алгоритмов сжатия базируется на том, что в любом файле, содержащем неслучайные данные, информация частично повторяется. Используя статистические математические модели можно определить вероятность повторения определённой комбинации символов. После этого можно создать коды, обозначающие выбранные фразы, и назначить самым часто повторяющимся фразам самые короткие коды. Для этого используются разные техники, например: энтропийное кодирование, кодирование повторов, и сжатие при помощи словаря. С их помощью 8-битный символ, или целая строка, могут быть заменены всего лишь несколькими битами, устраняя таким образом излишнюю информацию.

Основоположником науки о сжатии информации принято считать Клода Шеннона. Несмотря на то, что результаты исследований Шеннона были по-настоящему востребованы лишь десятилетия спустя, трудно переоценить их значение. Один из самых ранних и хорошо известных методов сжатия - алгоритм Хаффмана, который был и остается предметом многих исследований. Он появился на несколько лет позже, чем [алгоритм Шеннона](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%A8%D0%B5%D0%BD%D0%BD%D0%BE%D0%BD%D0%B0), и является его логическим продолжением.

Количество нужной человеку информации неуклонно растет с каждым днем. Объемы устройств для хранения данных и пропускная способность линий связи также растут. Однако количество информации растет быстрее. У этой проблемы есть три решения. Первое - ограничение количества информации, но, к сожалению, оно не всегда приемлемо. Например, для изображений это означает уменьшение разрешения, что приведет к потере мелких деталей и может сделать изображения вообще бесполезными (например, для медицинских или космических изображений). Второе — увеличение объема носителей информации и пропускной способности каналов связи. Это решение связано с материальными затратами, причем иногда весьма значительными. Третье решение - использование сжатия информации. Это решение позволяет в несколько раз сократить требования к объему устройств хранения данных и пропускной способности каналов связи без дополнительных издержек (за исключением издержек на реализацию алгоритмов сжатия). Условиями его применимости является избы­точность информации и возможность установки специального програм­много обеспечения либо аппаратуры как вблизи источника, так и вблизи приемника информации. Как правило, оба эти условия удовлетворяются.

Именно благодаря необходимости использования сжатия информации методы сжатия достаточно широко распространены. Однако существуют две серьезные проблемы. Во-первых, широко используемые методы сжатия, как правило, устарели и не обеспечивают достаточной степени сжатия. В то же время они встроены в большое количество программных продуктов и библиотек и поэтому будут использоваться еще достаточно долгое время. Второй проблемой является частое применение методов сжатия, не соответствующих характеру данных. Решение этих проблем позволяет резко повысить эффективность применения алгоритмов сжатия.

****Анализ предметной области****

* 1. ****Что такое сжатие данных****

Сжатие информации – это процесс преобразования информации, хранящейся в файле, к виду, при котором уменьшается избыточность в ее представлении и соответственно требуется меньший объем памяти для хранения.

Сжатие информации в файлах производится за счет устранения избыточности различными способами, например за счет упрощения кодов, исключения из них постоянных битов или представления повторяющихся символов или повторяющейся последовательности символов в виде коэффициента повторения и соответствующих символов. Применяются различные алгоритмы подобного сжатия информации.

Одна из основных характеристик любого алгоритма сжатия – степень сжатия. Степень сжатия зависит от метода сжатия и типа исходного файла. Наиболее хорошо сжимаются файлы графических образов, текстовые файлы и файлы данных, для которых степень сжатия может достигать 5 - 40%, меньше сжимаются файлы исполняемых программ и загрузочных модулей - 60 - 90%. Почти не сжимаются архивные файлы.

* 1. ****Классификация алгоритмов сжатия****

Методы сжатия данных можно разделить на два типа:

1. Неискажающие (loseless) методы сжатия (называемые также методами сжатия без потерь) гарантируют, что декодированные данные будут в точности совпадать с исходными;
2. Искажающие (lossy) методы сжатия (называемые также методами сжатия с потерями) могут искажать исходные данные, например за счет удаления несущественной части данных, после чего полное восстановление невозможно.

Первый тип сжатия применяют, когда данные важно восстановить после сжатия в неискаженном виде, это важно для текстов, числовых данных и т. п. Полностью обратимое сжатие, по определению, ничего не удаляет из исходных данных. Сжатие достигается только за счет иного, более экономичного, представления данных.

Второй тип сжатия применяют, в основном, для видео изображений и звука. За счет потерь может быть достигнута более высокая [степень сжатия](http://mf.grsu.by/UchProc/livak/po/comprsite/theory_classification_01.html#ratio). В этом случае потери при сжатии означают несущественное искажение изображения (звука) которые не препятствуют нормальному восприятию, но при сличении оригинала и восстановленной после сжатия копии могут быть замечены.

Кроме того, можно выделить:

* методы сжатия общего назначения (general-purpose), которые не зависят от физической природы входных данных и, как правило, ориентированы на сжатие текстов, исполняемых программ, объектных модулей и библиотек и т. д., т. е. данных, которые в основном и хранятся в ЭВМ;
* специальные (special) методы сжатия, которые ориентированы на сжатие данных известной природы, например, звука, изображений и т. д. И за счет знания специфических особенностей сжимаемых данных достигают существенно лучшего качества и/или скорости сжатия, чем при использовании методов общего назначения.

По определению, методы сжатия [общего назначения](http://mf.grsu.by/UchProc/livak/po/comprsite/theory_classification_01.html#gener_purp_def) – [неискажающие](http://mf.grsu.by/UchProc/livak/po/comprsite/theory_classification_01.html#loseless_def); [искажающими](http://mf.grsu.by/UchProc/livak/po/comprsite/theory_classification_01.html#lossy_def) могут быть только [специальные](http://mf.grsu.by/UchProc/livak/po/comprsite/theory_classification_01.html#special_def) методы сжатия. Как правило, искажения допустимы только при обработке всевозможных сигналов (звука, изображения, данных с физических датчиков), когда известно, каким образом и до какой степени можно изменить данные без потери их потребительских качеств.

* 1. ****Критерии оценки методов сжатия****

Основными свойствами какого-либо алгоритма сжатия данных являются:

* качество (коэффициент или степень) сжатия, т. е. отношение длины (в битах) сжатого представления данных к длине исходного представления;
* скорость кодирования и декодирования, определяемые временем, затрачиваемым на кодирование и декодирование данных;
* объем требуемой памяти.

В области сжатия данных, как это часто случается, действует закон рычага: алгоритмы, использующие больше ресурсов (времени и памяти), обычно достигают лучшего [качества сжатия](http://mf.grsu.by/UchProc/livak/po/comprsite/theory_classification_01.html#ratio), и наоборот: менее ресурсоемкие алгоритмы по [качеству сжатия](http://mf.grsu.by/UchProc/livak/po/comprsite/theory_classification_01.html#ratio), как правило, уступают более ресурсоемким.

Понятно, что критерии оценки методов сжатия с практической точки зрения сильно зависят от предполагаемой области применения. Например, при использовании сжатия в системах реального времени необходимо обеспечить высокую скорость кодирования и декодирования; для встроенных систем критический параметр – объем требуемой памяти; для систем долговременного хранения данных – качество сжатия и/или скорость декодирования и т. д.

* 1. Надежность программ и сложность алгоритмов

Надежность программных систем и комплексов очень важна и обеспечивается как безошибочностью программирования и дизайна, так и характеристиками использованных алгоритмов. Если количество ошибок в основном определяется полнотой и качеством тестирования и мало зависит от воли разработчика, то выбор алгоритмов – вполне управляемый и контролируемый процесс.

Для обеспечения конечного и заранее известного времени сжатия (в наихудшем случае), необходимо, чтобы алгоритм обладал хорошо детерминированным временем работы (желательно, мало зависящим от кодируемых данных) и заранее известным объемом требуемой памяти. В частности, выполнение этих требований необходимо при разработке встроенных систем, систем реального времени, файловых систем со сжатием данных и других систем с жесткими ограничениями на разделяемые различными процессами ресурсы.

* 1. Алгоритм сжатия методом Хаффмана

Алгоритм Хаффмана — [жадный алгоритм](https://ru.wikipedia.org/wiki/%D0%96%D0%B0%D0%B4%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) оптимального [префиксного](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D1%84%D0%B8%D0%BA%D1%81%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) [кодирования](https://ru.wikipedia.org/wiki/%D0%AD%D0%BD%D1%82%D1%80%D0%BE%D0%BF%D0%B8%D0%B9%D0%BD%D0%BE%D0%B5_%D0%BA%D0%BE%D0%B4%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) алфавита с минимальной [избыточностью](https://ru.wikipedia.org/wiki/%D0%98%D0%B7%D0%B1%D1%8B%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C). Был разработан в [1952 году](https://ru.wikipedia.org/wiki/1952_%D0%B3%D0%BE%D0%B4) аспирантом [Массачусетского технологического института](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%81%D0%B0%D1%87%D1%83%D1%81%D0%B5%D1%82%D1%81%D0%BA%D0%B8%D0%B9_%D1%82%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%81%D1%82%D0%B8%D1%82%D1%83%D1%82) [Дэвидом Хаффманом](https://ru.wikipedia.org/wiki/%D0%A5%D0%B0%D1%84%D1%84%D0%BC%D0%B0%D0%BD,_%D0%94%D1%8D%D0%B2%D0%B8%D0%B4) при написании им курсовой работы. В настоящее время используется во многих программах сжатия данных.

Этот метод кодирования состоит из двух основных этапов:

1. Построение оптимального кодового дерева.
2. Построение отображения код-символ на основе построенного дерева.

Классический алгоритм Хаффмана на входе получает таблицу частот встречаемости символов в сообщении. Далее на основании этой таблицы строится дерево кодирования Хаффмана (Н-дерево).

1. Символы входного алфавита образуют список свободных узлов. Каждый лист имеет вес, который может быть равен либо вероятности, либо количеству вхождений символа в сжимаемое сообщение.
2. Выбираются два свободных узла дерева с наименьшими весами.
3. Создается их родитель с весом, равным их суммарному весу.
4. Родитель добавляется в список свободных узлов, а два его потомка удаляются из этого списка.
5. Одной дуге, выходящей из родителя, ставится в соответствие бит 1, другой — бит 0. Битовые значения ветвей, исходящих от корня, не зависят от весов потомков.
6. Шаги, начиная со второго, повторяются до тех пор, пока в списке свободных узлов не останется только один свободный узел. Он и будет считаться корнем дерева.

Есть строка длиной в 100 байт, имеющая 6 различных символов в себе. Подсчитав вхождение каждого из символов в строке, получили Таблицу 1.

Таблица 1 – Частоты вхождения символов в строке

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Символ | A | B | C | D | E | F |
| Число вхождений | 10 | 20 | 30 | 5 | 25 | 10 |

Возьмем из таблицы 1 два символа с наименьшей частотой. В данном случае это D (5), и какой-либо символ из F или A (10), можно взять любой из них, например A.

Сформируем из "узлов" D и A новый "узел", частота вхождения для которого будет равна сумме частот D и A (Рисунок 1).

https://studfile.net/html/2706/35/html_PDIOUEcwuQ.zVQ0/img-CvrPyN.png

Рисунок 1 – Первый шаг построения дерева Хаффмана

Номер в рамке - сумма частот символов D и A. Снова ищем два символа с самыми низкими частотами вхождения. Исключаем из просмотра D и A и рассматриваем вместо них новый "узел" с суммарной частотой вхождения. Самая низкая частота теперь у F и нового "узла". Делаем операцию слияния узлов (Рисунок 2).

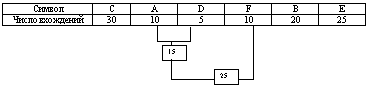


Рисунок 2 – Второй шаг построения дерева Хаффмана

Продолжаем эту процедуру до тех пор, пока все "дерево" не сформировано, т.е. пока все не сведется к одному узлу (Рисунок 3).

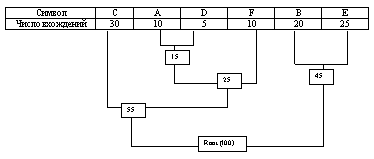


Рисунок 3 – Дерево Хаффмана

Когда дерево создано, можно кодировать строку. Всегда необходимо начинать из корня (Root). Кодируя первый символ (лист дерева С), прослеживаем вверх по дереву все повороты ветвей, и, если поворот левый, то запоминаем 0-й бит, и 1-й бит для правого поворота. Так, для C будем идти влево к 55 (и запомним 0), затем снова влево (0) к самому символу. Код Хаффмана для символа C - 00. Выполнив выше сказанное для всех символов, получим:

C = 00 (2 бита) A = 0100 (4 бита) D = 0101 (4 бита) F = 011 (3 бита) B = 10 (2 бита) E = 11 (2 бита)

При кодировании заменяем символы данными последовательностями.

Разработка

* 1. Выбор инструментария

Для реализации курсовой работы была выбрана интегрированная среда разработки Microsoft Visual Studio 2019, для разработки на языке С++.

Microsoft Visual Studio 2019 позволяет разрабатывать консольные приложения с использованием широких возможностей стандартной библиотеки, а также содержит удобный Just-In-Time отладчик кода, позволяющий легко обнаружить ошибки в коде.

C++ — [компилируемый](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), [статически типизированный](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) общего назначения. Поддерживает такие [парадигмы программирования](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D1%8B_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), как [процедурное программирование](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [объектно-ориентированное программирование](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [обобщённое программирование](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BE%D0%B1%D1%89%D1%91%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). 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 [высокоуровневых](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%81%D0%BE%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), так и [низкоуровневых языков](https://ru.wikipedia.org/wiki/%D0%9D%D0%B8%D0%B7%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F). В сравнении с его предшественником — языком [C](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), — наибольшее внимание уделено поддержке [объектно-ориентированного](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и [обобщённого программирования](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BE%D0%B1%D1%89%D1%91%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5).

## Постановка задачи

В задачу курсовой работы входит реализация одного из алгоритмов сжатия (реализация алгоритма Хаффмана) на языке C++.

* 1. Реализация алгоритма сжатия методом Хаффмана

В программе использованы контейнеры библиотеки STL и итераторы для них:

* Класс priority\_queue - класс-шаблон адаптера контейнера, который предоставляет ограничение функциональности, ограничивая доступ к верхнему элементу некоторого базового типа контейнера, который всегда является самым большим или имеет высший приоритет.
* Шаблон класса unordered\_map описывает объект, который управляет последовательности элементов типа std::pair <const Key, Ty> различной длины. Последовательность слабо упорядочена хэш-функцией, которая разделяет последовательность в упорядоченный набор подпоследовательностей, называемых блоками.
* Итератор - это объект, который может перебирать элементы в контейнере стандартной библиотеки C ++ и предоставлять доступ к отдельным элементам.

Шаги алгоритма:

1. Создание листового узла для каждого символа в строке и добавление его в очередь приоритетов
2. Формирование из узлов дерева Хаффмана

Для создания узла дерева была реализована структура *Node*:

struct Node

{

char ch;

int freq;

Node\* left, \* right;

};

Для построения очереди приоритетов использован класс priority\_queue библиотеки стандартных шаблонов (stl) и создана структура *comparison,* которая реализует предоставление высшего приоритета наиболее редким символам.

struct comparison

{

bool operator()(Node\* l, Node\* r)

{

// элемент с наивысшим приоритетом имеет наименьшую частоту

return l->freq > r->freq;

}

};

Подсчет частоты встречаемости символов хранится в ассоциативном контейнере *unordered\_map<char, int>.*

Дерево Хаффмана формируется следующим образом:

1. Формируется два листа из двух элементов очереди с наивысшими приоритетами. Данные элементы удаляются из очереди.
2. С помощью функции *getNode()* формируется новый узел с двумя полученными листами в качестве дочерних и с частотой, равной сумме их частот.
3. Пункты 1 и 2 повторяются, пока в очереди не останется один элемент. Данный элемент становится корнем дерева.

Для определения сжатых кодов символов реализована рекурсивная функция *encode()*, которая осуществляет проход по дереву Хаффмана и сохранение двоичных кодов в контейнер *unordered\_map<char, string>.* Сжатый код символа определяется следующим образом: во время прохода от корня к символу при переходе в левый подузел в строку кода записывается 0, при переходе в правый – 1.

Декодирование производится рекурсивной функцией *decode(),* которая переводит двоичную сжатую строку в последовательность читаемых символов.

Для расчёта коэффициента сжатия и наглядного представления масштаба сжатия исходная строка переводится в двоичное представление. Коэффициент сжатия находится по формуле:

 {\displaystyle k={\frac {S\_{o}}{S\_{c}}}},

где *k* — коэффициент сжатия, *S*o — объём исходных данных, а *S*c — объём сжатых. Если *k* = 1, то алгоритм не производит сжатия, если *k* < 1, то алгоритм порождает сообщение большего размера (совершает «вредную» работу).

Для просмотра промежуточных данных реализован вывод дерева Хаффмана (функция *printTree()*), а также пар (символ - частота встречаемости) и (символ – двоичный код).

Реализована функция для выбора языка (английский и русский). Для удобства просмотра кода программы функции вынесены в отдельный файл.

Заключение

В результате выполнения курсовой работы был реализован алгоритм сжатия методом Хаффмана. Алгоритм занимает 4,81 МБ. Из теоретических и экспериментальных данных можно сделать вывод, что степень сжатия строки (за исключением слишком маленькой), зависит лишь от частот появлений тех или иных символов в исходной строке. Если частота появления всех символов примерно одинакова, то сжатие не будет удачным, иначе можно ожидать хорошего сжатия.

Достоинствами метода Хаффмана являются его достаточно высокая скорость и хорошее качество сжатия. Этот алгоритм сравнительно давно известен и широко применяется; примерами могут служить программа compress ОС UNIX (программная реализация) и стандарт кодирования для факсов [Hunter] (аппаратная реализация).

Недостатком кодирования Хаффмана является зависимость степени сжатия от близости вероятностей символов к отрицательным степеням 2; это связано с тем, что каждый символ кодируется целым числом бит. Наиболее ярко это проявляется при кодировании двухсимвольного алфавита: в этом случае сжатие всегда отсутствует, несмотря на различие вероятностей символов; алгоритм фактически "округляет" их до 1/2!

Список источников

1. Методические указания к лабораторным работам и практическим занятиям по курсу "Теория информации и кодирование" "Сжатие данных (кодирование текстовой информации алгоритмами Хаффмана и Лемпеля-Зива)" (для студентов специальности 7.080201 - "Информатика"). /Сост.: С.А. Волощук. – Мариуполь: ПГТУ, 2010 – 20с.
2. Wikipedia [Электронный ресурс]: <https://ru.wikipedia.org>
3. АЛГОРИТМЫ СЖАТИЯ [Электронный ресурс]: <http://mf.grsu.by/UchProc/livak/po/comprsite/demo_algos.html>
4. StackOverflow [Электронный ресурс]: <https://stackoverflow.com>
5. METANIT.COM [Электронный ресурс]: <https://metanit.com/>
6. Techie Delight [Электронный ресурс]: <https://www.techiedelight.com/>
7. Microsoft С++ Documentation [Электронный ресурс]: <https://docs.microsoft.com/ru-ru/cpp/standard-library/>

Приложения

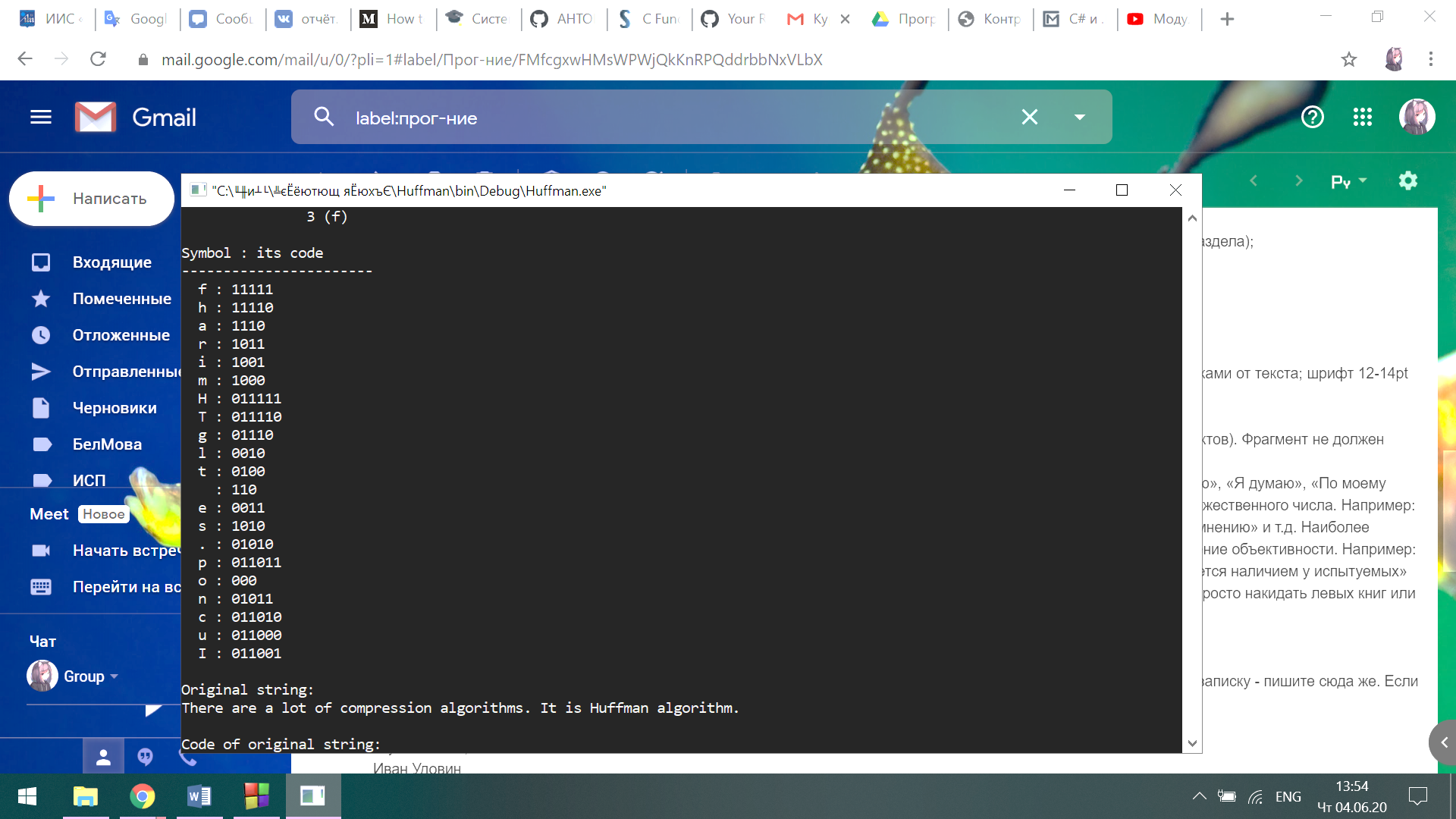
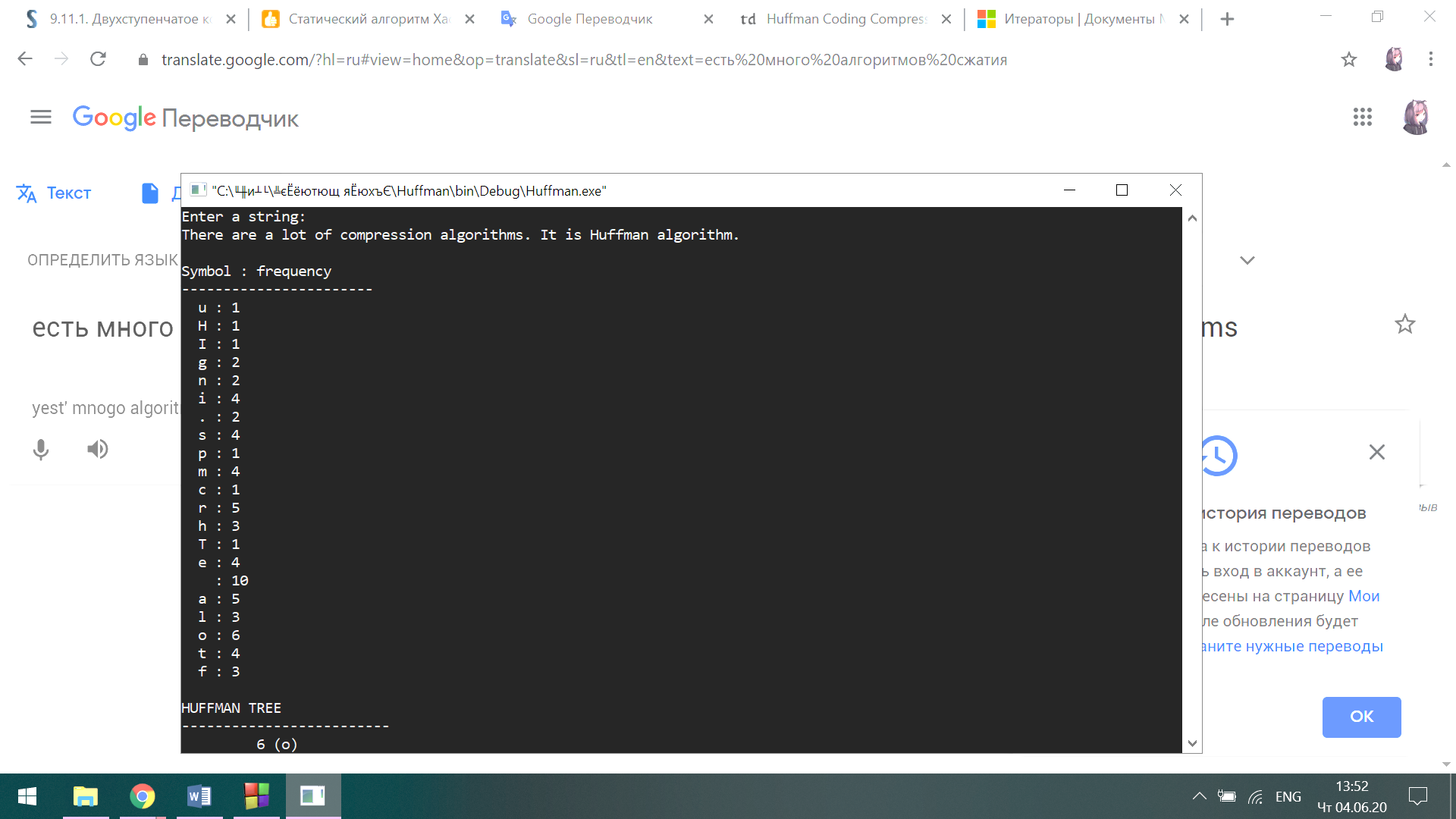
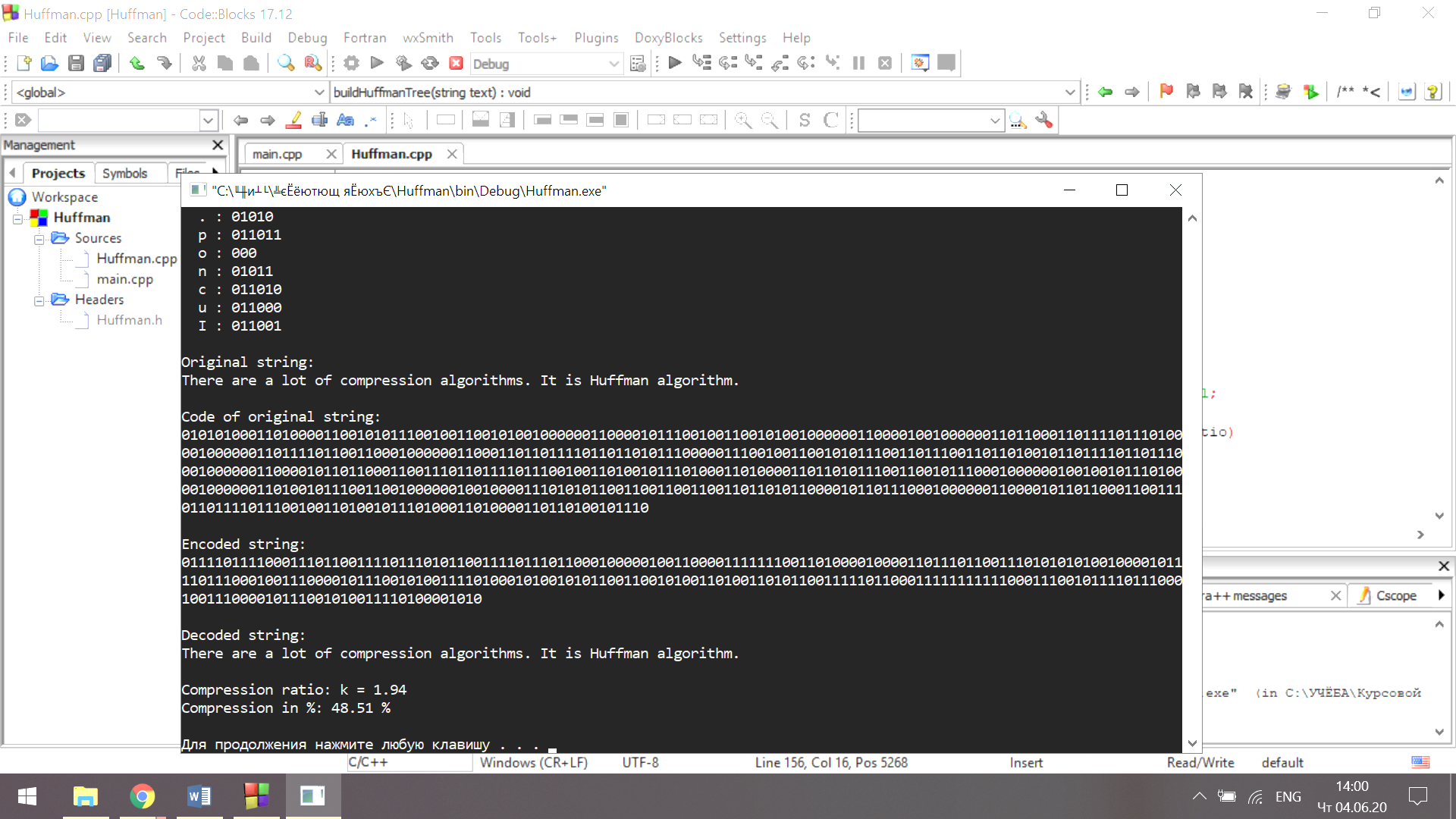
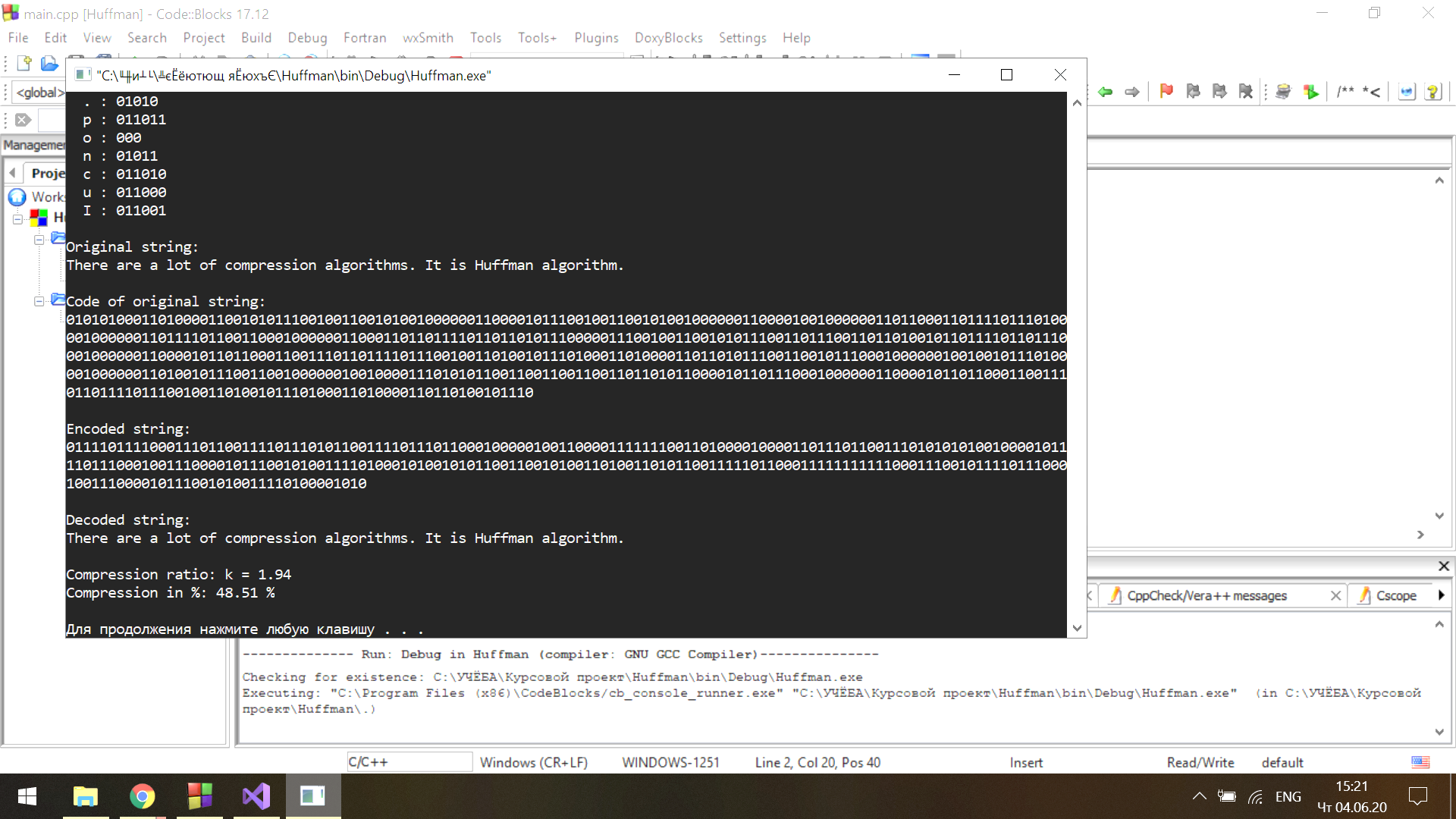


Таблица частот встречаемости символов и таблица соответствия символ – сжатый код.



Коэффициент сжатия строки



Код исходной строки и закодированной