МИНОБРНАУКИ РФ

ФГБОУ ВПО Тверской государственный технический университет

Кафедра “Программное обеспечение”

Курсовая работа

дисциплина “Базы данных”

Тема: “ Разработка системы автоматизации с использованием СУБД (MySQL/Firebird/PosgreSQL/MSSQL): online – магазин ”

Выполнил: студент группы

ПИН 17.05

Завгороднев Егор

Проверил:

Мальков А.А

Тверь 2018

Оглавление

[Введение 3](#_Toc532936049)

[Аналитическая часть 3](#_Toc532936050)

[Краткая история 3](#_Toc532936051)

[Алгоритмы и их O(N) 5](#_Toc532936052)

[Сравнения алгоритмов 12](#_Toc532936053)

[Проектная часть 13](#_Toc532936054)

[Функционал 13](#_Toc532936055)

[Архитектура приложения 13](#_Toc532936056)

[Классы 14](#_Toc532936057)

[Реализация проекта 16](#_Toc532936058)

[Тестирование 18](#_Toc532936059)

[Заключение 19](#_Toc532936060)

[Литература 20](#_Toc532936061)

# Введение

Цель: Ознакомится с алгоритмами кодирования и декодирования текста, которые позволяют устранить его избыточность и реализовать их.

Задача: Windows Form проект на c#. Реализовать алгоритмы: Хаффмана, арифметическое кодирование, LZ78.

# Аналитическая часть

## Краткая история

В сороковых годах ученые, работающие в области информационных технологий, ясно поняли, что можно разработать такой способ хранения данных, при котором пространство будет расходоваться более экономно. Клод Шеннон, изучая нюансы различий между семантикой (semantics) (что некая сущность значит) и синтаксисом (syntax) (как некая сущность выражается), разработал большинство базовых понятий этой теории. Понимание того, что одно и то же значение (семантика) может быть реализовано различными способами (синтаксис), приводит к закономерному вопросу: "Какой способ выражения чего-либо является наиболее экономичным?" Поиск ответа на этот вопрос привел Шеннона к мысли об энтропии, которая, проще говоря, соотносится с количеством, содержащейся в файле полезной информации. Методы сжатия пытаются увеличивать энтропию файла, то есть уменьшать длину файла, сохраняя при этом всю информацию.

Однако, Шеннон не был первым, кто задумывался о сущности информации и определении ее количества. Первый шаг на этом пути сделал в 1928 г. Хартли. Основной полученный им результат можно сформулировать примерно так: если в заданном множестве, содержащем N элементов, выделен некоторый элемент x, о котором известно лишь, что он принадлежит этому множеству, то, чтобы найти x, необходимо получить количество информации, равное log2 N. Эту формулу обычно называют формулой Хартли.

На текущий момент существует большое количество алгоритмов сжатия данных, которые условно можно разделить на две большие группы:   
1. Поточные и словарные алгоритмы. К этой группе относятся алгоритмы семейств RLE (run-length encoding), LZ\* и др. Особенностью всех алгоритмов этой группы является то, что при кодировании используется не информация о частотах символов в сообщении, а информация о последовательностях, встречавшихся ранее.  
2. Алгоритмы статистического (энтропийного) сжатия. Эта группа алгоритмов сжимает информацию, используя неравномерность частот, с которыми различные символы встречаются в сообщении. К алгоритмам этой группы относятся алгоритмы арифметического и префиксного кодирования (с использованием деревьев Шеннона-Фанно, Хаффмана, секущих).   
В отдельную группу можно выделить алгоритмы преобразования информации. Алгоритмы этой группы не производят непосредственного сжатия информации, но их применение значительно упрощает дальнейшее сжатие с использованием поточных, словарных и энтропийных алгоритмов.

## Алгоритмы и их O(N)

**Код Хаффмана. Сложность:** *O*(*Nlog N*)

Алгоритм Хаффмана— алгоритм оптимального префиксного кодирования алфавита. Был разработан в 1952 году аспирантом Массачусетского технологического института Дэвидом Хаффманом при написании им курсовой работы.

В настоящее время используется во многих программах сжатия данных.

Этот метод кодирования состоит из двух основных этапов:

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

Идея алгоритма состоит в следующем: зная вероятности символов в сообщении, можно описать процедуру построения кодов переменной длины, состоящих из целого количества битов. Символам с большей вероятностью ставятся в соответствие более короткие коды. Коды Хаффмана обладают свойством префиксности (то есть ни одно кодовое слово не является префиксом другого), что позволяет однозначно их декодировать.

Построение кода Хаффмана сводится к построению соответствующего бинарного дерева по следующему алгоритму:

1. Составим список кодируемых символов, при этом будем рассматривать один символ как дерево, состоящее из одного элемента c весом, равным частоте появления символа в строке.
2. Из списка выберем два узла с наименьшим весом.
3. Сформируем новый узел с весом, равным сумме весов выбранных узлов, и присоединим к нему два выбранных узла в качестве детей.
4. Добавим к списку только что сформированный узел вместо двух объединенных узлов.
5. Если в списке больше одного узла, то повторим пункты со второго по пятый.

Теперь для каждого символа выберем кодовое слово (бинарная последовательность, обозначающая путь по дереву к этому символу от корня)

**Пример:**

Закодируем слово **abracadabra**. Тогда алфавит будет ***A*={*a*,*b*,*r*,*c*,*d*},** а набор весов (частота появления символов алфавита в кодируемом слове) ***W*={5,2,2,1,1}**

В дереве Хаффмана будет 5 узлов:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Узел** | **a** | **b** | **r** | **с** | **d** |
| Вес | 5 | 2 | 2 | 1 | 1 |

По алгоритму возьмем два символа с наименьшей частотой — это **c** и **d**. Сформируем из них новый узел **cd** весом 2 и добавим его к списку узлов:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Узел** | **a** | **b** | **r** | **cd** |
| Вес | 5 | 2 | 2 | 2 |

Затем опять объединим в один узел два минимальных по весу узла — **r** и **cd**:

|  |  |  |  |
| --- | --- | --- | --- |
| **Узел** | **a** | **rcd** | **b** |
| Вес | 5 | 4 | 2 |

Еще раз повторим эту же операцию, но для узлов **rcd** и **b**:

|  |  |  |
| --- | --- | --- |
| **Узел** | **brcd** | **a** |
| Вес | 6 | 5 |

На последнем шаге объединим два узла — **brcd** и **a** :

|  |  |
| --- | --- |
| **Узел** | **abrcd** |
| Вес | 11 |

Теперь для каждого символа выберем кодовое слово (бинарная последовательность, обозначающая путь по дереву к этому символу от корня):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Символ** | **a** | **b** | **r** | **с** | **d** |
| Код | 0 | 11 | 101 | 1000 | 1001 |

Таким образом, закодированное слово **abracadabra**

будет выглядеть как **01110101000010010111010**. Длина закодированного слова — **23** бита.

Декодирование:

Проходим по дереву и ищем символ для соответствующего кода

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **11** | **101** | **0** | **1000** | **0** | **1001** | **0** | **11** | **101** | **0** |
| a | b | r | a | c | a | d | a | b | r | a |

**Арифметическое кодирование. Сложность:** *O*(*N*)+O(N^2)

Арифметическое кодирование - алгоритм сжатия информации без потерь, который при кодировании ставит в соответствие тексту вещественное число из отрезка [0;1).

В отличие от алгоритма Хаффмана, не имеет жесткого постоянного соответствия входных символов группам бит выходного потока. Это даёт алгоритму большую гибкость в представлении дробных частот встречаемости символов.

Как правило, превосходит алгоритм Хаффмана по эффективности сжатия, позволяет сжимать данные с энтропией, меньшей 1 бита на кодируемый символ.

Принцип действия:

Пусть имеется некий алфавит, а также данные о частотности использования символов (опционально). Тогда рассмотрим на координатной прямой отрезок от 0 до 1.

Назовём этот отрезок рабочим. Расположим на нём точки таким образом, что длины образованных отрезков будут равны частоте использования символа, и каждый такой отрезок будет соответствовать одному символу.

Теперь возьмём символ из потока и найдём для него отрезок среди только что сформированных, теперь отрезок для этого символа стал рабочим. Разобьём его таким же образом, как разбили отрезок от 0 до 1. Выполним эту операцию для некоторого числа последовательных символов. Затем выберем любое число из рабочего отрезка. Биты этого числа вместе с длиной его битовой записи и есть результат арифметического кодирования использованных символов потока.

Алгоритм:

На вход алгоритму передаются текст для кодирования и список частот встречаемости символов.

1. Рассмотрим отрезок [0;1) на координатной прямой.
2. Поставим каждому символу текста в соответствие отрезок, длина которого равна частоте его появления.
3. Считаем символ из входного потока и рассмотрим отрезок, соответствующий этому символу. Разделим этот отрезок на части, пропорциональные частотам встречаемости символов.
4. Повторим пункт 3 до конца входного потока.

1. Выберем любое число из получившегося отрезка, которое и будет результатом арифметического кодирования.

**Пример:**

Рассмотрим в качестве примера строку ***abracadabra***

|  |  |
| --- | --- |
| Символ | Частота появления |
| **a** | **0,45** |
| **b** | **0,18** |
| **r** | **0,18** |
| **c** | **0,09** |
| **d** | **0,09** |

|  |  |  |
| --- | --- | --- |
| Считанный символ | Левая граница отрезка | Правая граница отрезка |
|  | **0** | **1** |
| **a** | **0** | **0,45** |
| **b** | **0,45** | **0,63** |
| **r** | **0,63** | **0,81** |
| **a** | **0** | **0,45** |
| **c** | **0,81** | **0,90** |
| **a** | **0** | **0,45** |
| **d** | **0,90** | **1** |
| **a** | **0** | **0,45** |
| **b** | **0,45** | **0,63** |
| **r** | **0,63** | **0,81** |
| **a** | **0** | **0,45** |
| Код: **0,26505522 – число из получившегося отрезка** | | |

Декодирование:

Алгоритм по вещественному числу восстанавливает исходный текст.

1. Выберем на отрезке [0;1) , разделенном на части, длины которых равны вероятностям появления символов в тексте, подотрезок, содержащий входное вещественное число. Символ, соответствующий этому подотрезку, дописываем в ответ.
2. Нормируем подотрезок и вещественное число.
3. Повторим пункты 1—2 до тех пор, пока не получим ответ.

|  |  |
| --- | --- |
| **0,26** | a |
| **0,55** | b |
| **0,67** | r |
| **0,26** | a |
| **0,88** | c |
| **0,26** | a |
| **0,94** | d |
| **0,26** | a |
| **0,55** | b |
| **0,67** | r |
| **0,26** | a |

**LZ78. Сложность:** *O*(*N*)

Алгоритм LZ78 алгоритм в явном виде использует словарный подход, генерируя временный словарь во время кодирования и декодирования.

Изначально словарь пуст, а алгоритм пытается закодировать первый символ. На каждой итерации мы пытаемся увеличить кодируемый префикс, пока такой префикс есть в словаре. Кодовые слова такого алгоритма будут состоять из двух частей — номера в словаре самого длинного найденного префикса (pos) и символа, который идет за этим префиксом (next). При этом после кодирования такой пары префикс с приписанным символом добавляется в словарь, а алгоритм продолжает кодирование со следующего символа.

**Пример:**

Закодируем строчку **abracadabra**

|  |  |  |
| --- | --- | --- |
| Словарь | Код |  |
| **a** | **(0,a)** | **abracadabra** |
| **a, b** | **(0,b)** | **bracadabra** |
| **a, b, r** | **(0,r)** | **racadabra** |
| **a, b, r, ac** | **(1,c)** | **cadabra** |
| **a, b, r, ac, ad** | **(1,d)** | **dabra** |
| **a, b, r, ac, ad, ab** | **(1,b)** | **bra** |
| **a, b, r, ac, ad, ab, ra** | **(3,a)** | **a** |

Декодирование происходит аналогично кодированию, на основе декодируемой информации строим словарь и берем из него значения.

|  |  |
| --- | --- |
| Код | Текст |
| **(0,a)** | **a** |
| **(0,b)** | **ab** |
| **(0,r)** | **abr** |
| **(1,c)** | **abrac** |
| **(1,d)** | **abracad** |
| **(1,b)** | **abracadab** |
| **(3,a)** | **abracadabra** |

**UTF-8**

UTF-8 (англ. Unicode Transformation Format, 8-bit — «формат преобразования Unicode, 8-битный») — одна из общепринятых и стандартизированных кодировок текста, которая позволяет хранить символы Unicode, используя переменное количество байт (от 1 до 6)

UTF-8 является лишь представлением Unicode в 8-битном виде. Символы с кодами меньше 128 представляются одним байтом (Латинский алфавит, простейшие знаки препинания и арабские цифры), а так как в Unicode они повторяют ASCII, то текст написанный только этими символами будет являться текстом в ASCII.

Символы с кодами от:

* 128 — 2-мя байтами.(Кириллица, расширенная латиница, арабский, армянский, греческий, еврейский и коптский алфавит, сирийское письмо, некоторые знаки препинания).
* 2048 — 3-мя байтами (Все другие современные формы письменности, сложные знаки препинания; математические и другие специальные символы).
* 65536 — 4-мя байтами (Музыкальные символы, смайлы, редкие китайские иероглифы, вымершие формы письменности).

Принцип кодирования:

* Каждый символ превращаем в Unicode.
* Проверяем из какого диапазона символ.
* Если код символа меньше 128, то к результату добавляем его в неизменном виде.
* Если код символа меньше 2048, то берем последние 6 бит и первые 5 бит кода символа. К первым 5 битам добавляем 0xC0 и получаем первый байт последовательности, а к последним 6 битам добавляем 0x80 и получаем второй байт. Конкатенируем и добавляем к результату.
* Похожим образом можем продолжить и для больших кодов, но если символ за пределами U+FFFF придется иметь дело с UTF-16 суррогатами.

## Сравнения алгоритмов

|  |  |  |  |
| --- | --- | --- | --- |
|  | Алгоритм Хаффмана  *O*(*Nlog N*) | Арифметическое кодирование  *O*(*N*)+O(N^2) | LZ\* алгоритмы **:** *O*(*N*) |
| Достоинства | Алгоритм Хаффмана остаётся всегда оптимальным для вторичных алфавитов m2 с более чем двумя символами.  Коды Хаффмана обладают свойством префиксности (то есть ни одно кодовое слово не является префиксом другого), что позволяет однозначно их декодировать. | Не имеет жесткого постоянного соответствия входных символов группам бит выходного потока. Это даёт алгоритму большую гибкость в представлении дробных частот встречаемости символов. Превосходит алгоритм Хаффмана по эффективности сжатия | Эффективность сжатия. |
| Недостатки | Избыточность кодирования обращается в ноль лишь в тех случаях, когда вероятности кодируемых символов являются обратными степенями числа 2.  Для восстановления содержимого сжатого сообщения декодер должен знать таблицу частот | Чрезвычайно высокая сложность процессов кодирования и декодирования. | Невозможность кодирования подстрок, отстоящих друг от друга на расстоянии, большем длины словаря  Малая эффективность при кодировании незначительного объёма данных |

# Проектная часть

## Функционал

Программа кодирует введенный пользователем текст одним из 3 алгоритмов и декодирует полученный код обратно в текст.

## Архитектура приложения

Программа состоит из двух частей: библиотека классов и Windows Form проект.

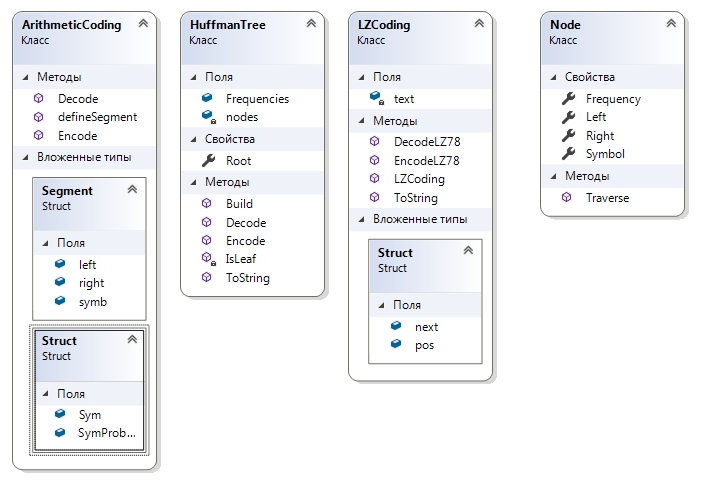
В библиотеке представлены классы и методы для реализации алгоритмов.

В WinForm проекте – интерфейс и его связь с классами из библиотеки

## Классы

ClassLibraryTextCoding:

* **SourceProp** - свойстватекста для арифметического кодирования
* **ArithmeticCoding** - методы для арифметического кодирования
* **LZCoding** – методы для алгоритма LZ78
* **Node** – структура узла дерева Хаффмана и метод для нахождения кода символа
* **HuffmanTree** – построение дерева Хаффмана + методы для кодирования и декодирования

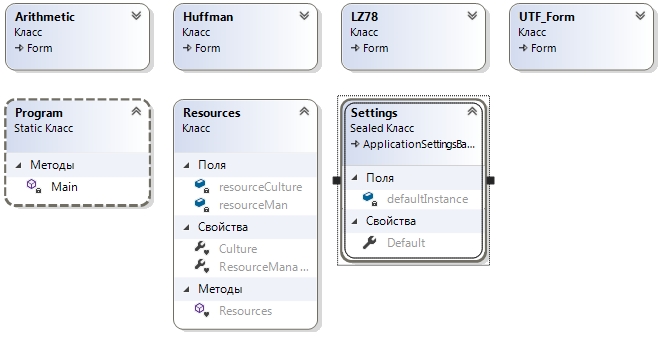


WinForm:

Классы интерфейса для алгоритмов:

* **Arithmetic**

* **Huffman**
* **LZ78**
* **UTF-8**

****

## Реализация проекта

Хронология работы над проектом:

24.11 – Начало разработки , интерфейс программы

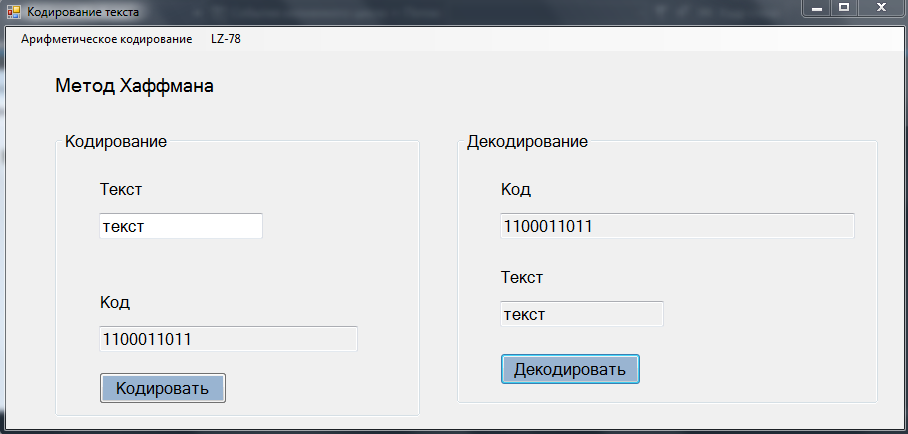
25.11 – Был написан алгоритм арифметического кодирования

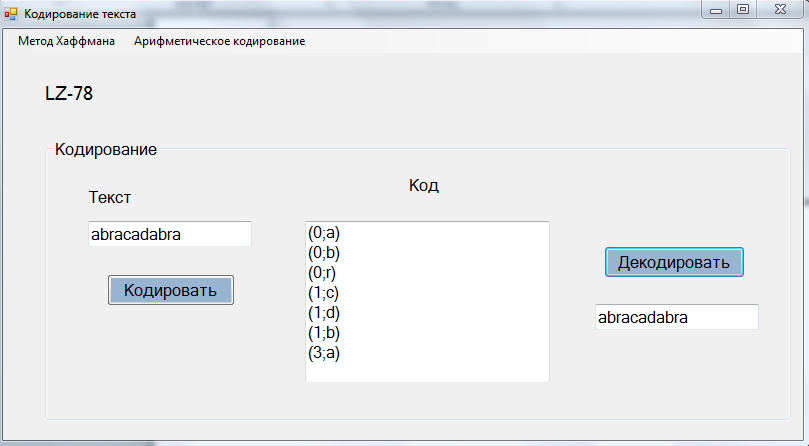
26.11 – Полностью рабочий алгоритм LZ78

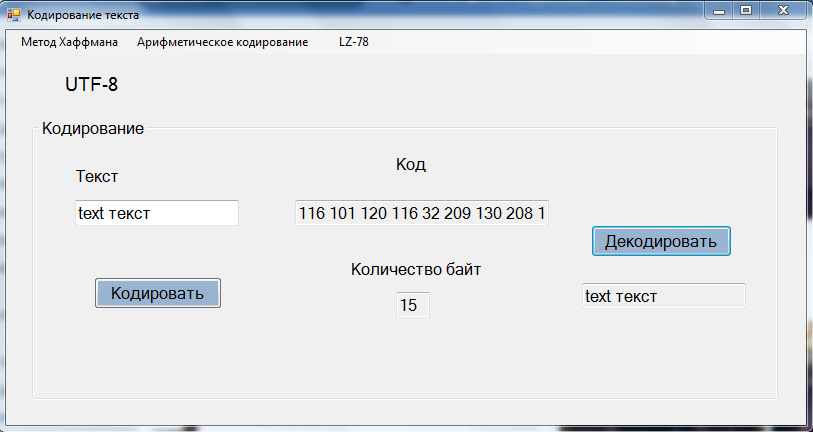
27.11 – Добавлен код Хаффмана, интерфейс привязан к библиотеке классов

8.12 - Кодирование текста в UTF-8 , исправление ошибок программы , Unit-тестирование

Интерфейс программы:







## 

Среда разработки:

Microsoft Visual Studio

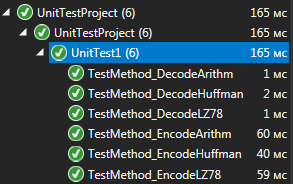
Инструменты:

C# + .NET

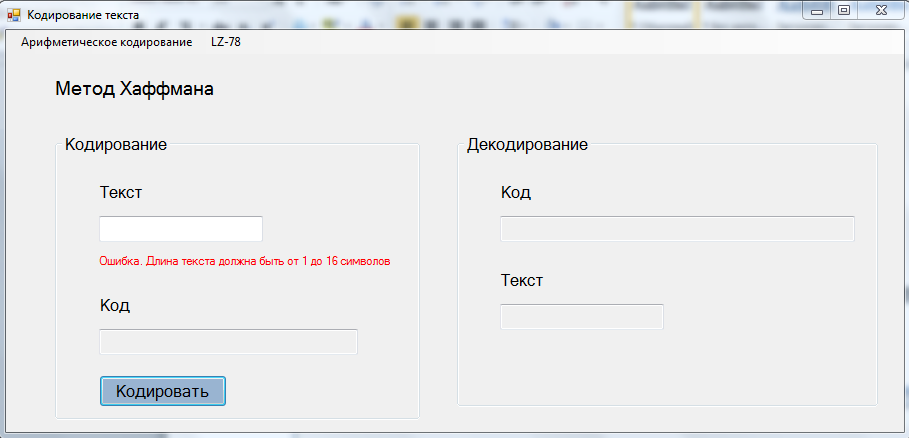
## Тестирование

Для автоматического тестирования я добавил в проект UnitTestProject.

Результаты тестов:



Также программа обрабатывает неправильные действия пользователя, которые приводят к исключениям.



# Заключение

В данной курсовой работе я реализовал 3 алгоритма кодирования и декодирования текста.

Проект был успешно протестирован.

# Литература

<https://ru.wikipedia.org> - Информация о алгоритмах

<http://www.compression.ru> - Всё о сжатии данных, изображений и видео

<https://habr.com/post/251295/> - Методы сжатия данных

<https://habr.com/post/104219/> - Оценка сложности алгоритмов

<http://neerc.ifmo.ru> – Информация о алгоритмах на сайте университета ИТМО