**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**  
 Дисциплина: «Алгоритмы и структуры данных»

Контрольное домашнее задание   
Исследование алгоритмов сжатия  
Хаффмана и Шеннона-Фано

Выполнил: Мариносян Никита,  
 студент гр. БПИ151.

Преподаватель: Мицюк А. А.

Москва 2016

Оглавление

[Постановка задачи 3](#_Toc468731654)

[Описание алгоритмов и использованных структур данных 5](#_Toc468731655)

[Алгоритм Хаффмана 5](#_Toc468731656)

[Алгоритм Шеннона-Фано 5](#_Toc468731657)

[Структуры данных, использованные при реализации 5](#_Toc468731658)

[При реализации вышеописанных алгоритмов использовались следующие структуры данных: 5](#_Toc468731659)

[Вспомогательные функции и классы, использованные при реализации 6](#_Toc468731660)

[Описание плана эксперимента 7](#_Toc468731661)

[Генерация тестовых файлов и их размер 7](#_Toc468731662)

[Способ подсчета операций 7](#_Toc468731663)

[Результаты эксперимента 8](#_Toc468731664)

[Сравнительный анализ методов (сложность, «+» и «–») 24](#_Toc468731665)

[Заключение (краткие выводы) 25](#_Toc468731666)

[Использованные источники 26](#_Toc468731667)

# Постановка задачи

(1) Реализовать с использованием языка C++ программы для архивирования и разархивирования текстовых файлов. При этом использовать два известных алгоритма кодирования информации:

1. Хаффмана (простой),

2. Шеннона-Фано.  
Обе реализации поместить в одном файле main.cpp, содержащем соответствующие методы:

1. метод архивирования, использующий алгоритм Хаффмана, вход: текстовый файл <name>.txt (кодировка UTF-8)  
   выход: архивированный файл <name>.haff
2. метод разархивирования, использующий алгоритм Хаффмана, вход: архивированный файл <name>.haff  
   выход: разархивированный файл <name>-unz-h.txt (кодировка UTF-8)
3. метод архивирования, использующий алгоритм Шеннона-Фано, вход: текстовый файл <name>.txt (кодировка UTF-8)  
   выход: архивированный файл <name>.shan
4. метод разархивирования, использующий алгоритм Шеннона-Фано. вход: архивированный файл <name>.shan  
   выход: разархивированный файл <name>-unz-s.txt (кодировка UTF-8)

Выбор алгоритма осуществляется с помощью флага командной строки.

Оба алгоритма работают *в два прохода*. Сначала строится *таблица частот встречаемости символов* в конкретном архивируемом файле (кодируем только те символы из набора допустимых, которые реально встречаются в файле). Затем разными способами строится *кодовое дерево*. По нему и архивируется файл. Для разархивирования алгоритмам потребуется знать таблицу, которая использовалась при архивировании. Соответствующая таблица должна сохраняться в архивном файле в самом его начале и использоваться при разархивировании. В начале пишется количество различных символов *n*, имеющихся в кодируемом файле, а затем *n* пар (код символа UTF-8, битовый код в архиве). Порядок — по убыванию частоты встречаемости символа в кодируемом файле.

(2) Провести вычислительный эксперимент с целью оценки реализованных алгоритмов архивации / разархивации. Оценить количество элементарных операций каждого алгоритма.

Для этого  
1. Подготовить тестовый набор из нескольких текстовых файлов разного объема

(20, 40, 60, 80, 100 Кб; 1, 2, 3 Мб — всего 8 файлов) на разных языках (ru, en - кодировка UTF-8) с разным набором символов в каждом файле, а именно:

1. первый набор: символы латинского алфавита и пробел
2. второй набор: символы из первого набора + символы русского алфавита
3. третий набор: символы из второго набора + следующие знаки и

спецсимволы: знаки арифметики „**+ - \* / =**“, знаки препинания „**. , ; : ? !**“,  
символы„**%@#$&~**‘’, скобки разных типов „**()[]{}<>**“, кавычки„**„“**“),

2. Измерить (экспериментально) количество операций (в рамках модели RAM *(см. лекционный материал)*), выполняемых за время работы (архивирования, разархивирования) каждого алгоритма на нескольких различных (не менее трех) файлах для каждого размера входного файла и набора символов (итого получается 8 \* 3 \* 3 = 72 эксперимента по архивированию и 72 по разархивированию для каждого алгоритма, т. е. всего минимум 144 \* 2 = 288, имеет смысл задуматься об автоматизации). Для повышения достоверности результатов каждый эксперимент *можно* повторить несколько (5-10) раз на различных файлах (с одним возможным набором символов) одного размера с последующим усреднением результата.

(3) Подготовить отчет по итогам работы, содержащий постановку задачи, описание алгоритмов и задействованных структур данных, описание реализации, обобщенные результаты измерения эффективности алгоритмов, описание использованных инструментов (например, если использовались скрипты автоматизации), выводы о соответствии результатов экспериментальной проверки с теоретическими оценками эффективности исследуемых алгоритмов.

Отчет также должен содержать измерения качества архивации (степень сжатия = отношение размеров выходного и входного файлов), оценку связи между степенью сжатия для различных входных файлов (как влияют объем, язык, набор символов, их разнообразие?) и временем работы (количеством операций) для каждого алгоритма.

В отчете необходимо ***явно*** указать, какие части задания были сделаны, а какие нет!

# Описание алгоритмов и использованных структур данных

Оба алгоритма схожи по многим параметрам. Вначале подсчитывается частота встречаемости отдельных символов в тексте (первый обход программы). Затем, основываясь на информации о частоте встречаемости каждого символа, разными способами (алгоритмом Хаффмана или Шеннона-Фано) строится бинарное дерево таким образом, что символу с наибольшей частотой соответствует код наименьшей длины. После этого каждый символ исходного текста кодируется соответствующим ему двоичным кодом и записывается в новый файл (второй обход программы).

## Алгоритм Хаффмана

После составления таблицы частот встречаемости символов, алгоритм состоит из следующих шагов:

1. Выбрать две вершины с наименьшим весом.

2. Создать новую вершину с весом, равным сумме весов выбранных узлов, и присоединить к ней (с помощью указателей) две выбранные вершины в качестве «детей».

3. Добавить к списку новую вершину, созданную из двух других.

4. Если в списке больше одного узла, то повторить пункты с первого по четвертый

Таким образом, дерево строится снизу вверх, т.е. от листьев к корню.

## Алгоритм Шеннона-Фано

Алгоритм отличается способом построения дерева, которое, в отличие от алгоритма Хаффмана, ведется от корня к листьям. После составления таблицы частот встречаемости символов, алгоритм выглядит следующим образом:

1. Символы полученного алфавита разделяются на две части таким образом, чтобы суммарные частоты этих двух частей были максимально близки друг другу.

2. К кодам символов, находящихся в первой части алфавита, приписывается «0», а к кодам символов второй части — «1».

3. Рекурсивно вызывается эта же функция, которая делит полученные части на более маленькие и приписывает соответствующие двоичные цифры к кодам символов, до тех пор, пока в каждой части не останется один элемент.

## Структуры данных, использованные при реализации

## При реализации вышеописанных алгоритмов использовались следующие структуры данных:

* Дерево — набор связных узлов, имитирующих древовидную структуру. Для реализации созданы классы HuffmanVertex и SFVertex, представляющие собой отдельную вершину дерева, в которой хранится информация о символе, его частоте и “детях”.
* Вектор (vector) -  абстрактный тип данных, представляющий собой упорядоченный набор значений, в котором некоторое значение может встречаться более одного раза. Использовался из-за удобства работы (быстрый произвольный доступ к элементам, автоматическое изменение размера при вставке или удалении элементов) и возможности использования стандартной сортировки std::sort.
* Ассоциативный массив (map) - контейнер, который содержит пары ключ-значение с неповторяющимися ключами. Использовался для хранения уникальных для текста символов и соответствующей им частоты.

## Вспомогательные функции и классы, использованные при реализации

Класс HuffmanVertex – вспомогательный класс, имитирующий вершину дерева. Полями являются frequency – частота появления символа в тексте и character – сам символ. Также полями являются два указателя на дочерние вершины типа HuffmanVertex\*: left\_node и right\_node.

Класс SFVertex – вспомогательный класс, имитирующий вершину дерева. Полями являются frequency – частота появления символа в тексте, character – сам символ и code – бинарный код данного символа.

Метод getFrequencyMap – вспомогательный метод построения ассоциативного массива и, хранящем информацию о частоте встречаемости символов в тексте, написанном в файле, путь к которому передается методу в качестве параметра. Также используется для подсчета числа символов в файле.

Метод compressFile – вспомогательный метод создания архивированного файла. Посимвольно записывает в новый файл текст исходного файла в соответствующей кодировке, полученной из таблицы кодов. В начале файла записывает необходимую для разархивации информацию: количество символов в тексте исходного файла, количество уникальных символов, сами уникальные символы, длину кодов каждого символа и коды уникального символа.

Метод decompressFile – универсальный метод разархивации файла для обоих алгоритмов. Однако, чтобы соответствовать спецификации, в функции main созданы отдельные методы разархивации для двух алгоритмов, которые вызывают данный метод. Отмечу, что возможность разархивации файла, имеющего расширение .haff с помощью метода для разархивации файла .shan намеренно ограничена (хотя это возможно). Аналогично нельзя разархивировать файл .haff с помощью метода по извлечению архива .shan. Имеет смысл объединить методы разархивации, однако тогда программа не будет соответствовать требованиям задания.

# Описание плана эксперимента

При проведении эксперимента я руководствовался следующим планом:

1. Реализовать построение кодов символов алгоритмом Хаффмана, архивацию и разархивацию на основе данного алгоритма на языке С++.
2. Реализовать построение кодов символов алгоритмом Шеннона-Фано, архивацию и разархивацию на основе данного алгоритма на языке С++.
3. С помощью программы на языке Java подготовить тестовый набор из 72 файлов, как написано в задании.
4. Провести тесты по архивации и разархивации каждого файла с помощью двух алгоритмов (всего 72\*2\*2 = 288 тестов) и получить значения количества элементарных операций для каждого теста.
5. Основываясь на результатах тестов, построить графики и таблицы, требуемые в задании.
6. Составить отчет о проделанной работе, в котором произвести сравнительный анализ методов и сделать основные выводы.

Все пункты данного плана были выполнены. Программа соответствует всем требованиям задания.

## Генерация тестовых файлов и их размер

Исполняемый файл генерации тестовых файлов (Main.jar) и сами тестовые файлы можно найти в архиве с КДЗ в папке TestingFiles. Исполняемый файл генерирует тестовые файлы, основываясь на русскоязычном и англоязычном текстах модернистского романа «Улисс» ирландского писателя Джеймса Джойса (файлы \*set1.txt) и выбирая случайные символа из соответствующих алфавитов (файлы \*set2.txt и \*set3.txt). Ввиду особенностей программной реализации генерации тестовых файлов, их размер может незначительно отличаться от требуемых, однако эта разница очень мала и не влияет на результаты эксперимента.

## Способ подсчета операций

Для подсчета количества элементарных операций в файле «utilities.h» была создана глобальная переменная «operations». В методах, реализующих каждый из алгоритмов (в том числе вспомогательных, записи в файл и чтения из него), было подсчитано количество элементарных операций, и переменная «operations» увеличивалась на на их число с помощью оператора «+=». С целью повышения эффективности работы программы увеличение глобальной переменной производилось не сразу после осуществления элементарных операций, а после небольших участков кода, в которых было посчитано их число. Каждая стандартная операция («+», «-», «=», «&», «\*» и т.д.) считалась за одну элементарную. Методы стандартной библиотеки также считались за одну операцию. Не считались за операцию обращение по индексу и вызов стандартных «свойств» (например .size()). Замечу, что т.к. построение графиков и таблиц ведется относительно конкретной программы, то вне зависимости от способа подсчета операций результат будет один и тот же.

# Результаты эксперимента

С помощью скрипта, вызывающего одну из четырех функций архивации/разархивации для файлов разного размера, всего было проведено 288 тестов. Ниже приведена таблица с полученными в ходе эксперимента результатами (файл «All Tests.xlsx»). В колонке “File” первое число соответствует вызываемому методу (1/2/3/4 см. п. «Постановка задачи»), затем идет размер файла, алфавит и номер набора. В колонке “operations” содержится количество элементарных операций для данного файла.

|  |  |
| --- | --- |
| File | operations |
| 1 20Kb\_EN\_set1.txt | 1010030 |
| 1 20Kb\_EN\_RU\_set1.txt | 1121109 |
| 1 20Kb\_EN\_RU\_S\_set1.txt | 1105603 |
| 1 40Kb\_EN\_set1.txt | 2017461 |
| 1 40Kb\_EN\_RU\_set1.txt | 2246063 |
| 1 40Kb\_EN\_RU\_S\_set1.txt | 2208612 |
| 1 60Kb\_EN\_set1.txt | 3015377 |
| 1 60Kb\_EN\_RU\_set1.txt | 3366131 |
| 1 60Kb\_EN\_RU\_S\_set1.txt | 3298348 |
| 1 80Kb\_EN\_set1.txt | 4036158 |
| 1 80Kb\_EN\_RU\_set1.txt | 4480058 |
| 1 80Kb\_EN\_RU\_S\_set1.txt | 4402424 |
| 1 100Kb\_EN\_set1.txt | 5032195 |
| 1 100Kb\_EN\_RU\_set1.txt | 5591828 |
| 1 100Kb\_EN\_RU\_S\_set1.txt | 5491774 |
| 1 1Mb\_EN\_set1.txt | 53790587 |
| 1 1Mb\_EN\_RU\_set1.txt | 57511405 |
| 1 1Mb\_EN\_RU\_S\_set1.txt | 56436570 |
| 1 2Mb\_EN\_set1.txt | 104060159 |
| 1 2Mb\_EN\_RU\_set1.txt | 116978079 |
| 1 2Mb\_EN\_RU\_S\_set1.txt | 114369882 |
| 1 3Mb\_EN\_set1.txt | 154020820 |
| 1 3Mb\_EN\_RU\_set1.txt | 173543458 |
| 1 3Mb\_EN\_RU\_S\_set1.txt | 170357208 |
| 3 20Kb\_EN\_set1.txt | 1014019 |
| 3 20Kb\_EN\_RU\_set1.txt | 1127494 |
| 3 20Kb\_EN\_RU\_S\_set1.txt | 1109987 |
| 3 40Kb\_EN\_set1.txt | 2025347 |
| 3 40Kb\_EN\_RU\_set1.txt | 2258035 |
| 3 40Kb\_EN\_RU\_S\_set1.txt | 2213471 |
| 3 60Kb\_EN\_set1.txt | 3016602 |
| 3 60Kb\_EN\_RU\_set1.txt | 3380376 |
| 3 60Kb\_EN\_RU\_S\_set1.txt | 3305293 |
| 3 80Kb\_EN\_set1.txt | 4049831 |
| 3 80Kb\_EN\_RU\_set1.txt | 4500244 |
| 3 80Kb\_EN\_RU\_S\_set1.txt | 4408325 |
| 3 100Kb\_EN\_set1.txt | 5033977 |
| 3 100Kb\_EN\_RU\_set1.txt | 5616356 |
| 3 100Kb\_EN\_RU\_S\_set1.txt | 5498095 |
| 3 1Mb\_EN\_set1.txt | 53874275 |
| 3 1Mb\_EN\_RU\_set1.txt | 57735084 |
| 3 1Mb\_EN\_RU\_S\_set1.txt | 56553562 |
| 3 2Mb\_EN\_set1.txt | 104147141 |
| 3 2Mb\_EN\_RU\_set1.txt | 117087897 |
| 3 2Mb\_EN\_RU\_S\_set1.txt | 114465851 |
| 3 3Mb\_EN\_set1.txt | 154428388 |
| 3 3Mb\_EN\_RU\_set1.txt | 174288465 |
| 3 3Mb\_EN\_RU\_S\_set1.txt | 170465973 |
| 2 20Kb\_EN\_set1.haff | 842006 |
| 2 20Kb\_EN\_RU\_set1.haff | 937130 |
| 2 20Kb\_EN\_RU\_S\_set1.haff | 924616 |
| 2 40Kb\_EN\_set1.haff | 1684316 |
| 2 40Kb\_EN\_RU\_set1.haff | 1882660 |
| 2 40Kb\_EN\_RU\_S\_set1.haff | 1852522 |
| 2 60Kb\_EN\_set1.haff | 2518257 |
| 2 60Kb\_EN\_RU\_set1.haff | 2823983 |
| 2 60Kb\_EN\_RU\_S\_set1.haff | 2768945 |
| 2 80Kb\_EN\_set1.haff | 3372307 |
| 2 80Kb\_EN\_RU\_set1.haff | 3759960 |
| 2 80Kb\_EN\_RU\_S\_set1.haff | 3698068 |
| 2 100Kb\_EN\_set1.haff | 4204581 |
| 2 100Kb\_EN\_RU\_set1.haff | 4694131 |
| 2 100Kb\_EN\_RU\_S\_set1.haff | 4614271 |
| 2 1Mb\_EN\_set1.haff | 45068253 |
| 2 1Mb\_EN\_RU\_set1.haff | 48339192 |
| 2 1Mb\_EN\_RU\_S\_set1.haff | 47480821 |
| 2 2Mb\_EN\_set1.haff | 87040525 |
| 2 2Mb\_EN\_RU\_set1.haff | 98404931 |
| 2 2Mb\_EN\_RU\_S\_set1.haff | 96284719 |
| 2 3Mb\_EN\_set1.haff | 128740993 |
| 2 3Mb\_EN\_RU\_set1.haff | 145917541 |
| 2 3Mb\_EN\_RU\_S\_set1.haff | 143382386 |
| 4 20Kb\_EN\_set1.shan | 844683 |
| 4 20Kb\_EN\_RU\_set1.shan | 940178 |
| 4 20Kb\_EN\_RU\_S\_set1.shan | 925509 |
| 4 40Kb\_EN\_set1.shan | 1690452 |
| 4 40Kb\_EN\_RU\_set1.shan | 1890400 |
| 4 40Kb\_EN\_RU\_S\_set1.shan | 1853722 |
| 4 60Kb\_EN\_set1.shan | 2518488 |
| 4 60Kb\_EN\_RU\_set1.shan | 2833741 |
| 4 60Kb\_EN\_RU\_S\_set1.shan | 2771696 |
| 4 80Kb\_EN\_set1.shan | 3383537 |
| 4 80Kb\_EN\_RU\_set1.shan | 3775000 |
| 4 80Kb\_EN\_RU\_S\_set1.shan | 3700011 |
| 4 100Kb\_EN\_set1.shan | 4205311 |
| 4 100Kb\_EN\_RU\_set1.shan | 4712915 |
| 4 100Kb\_EN\_RU\_S\_set1.shan | 4616661 |
| 4 1Mb\_EN\_set1.shan | 45141276 |
| 4 1Mb\_EN\_RU\_set1.shan | 48532801 |
| 4 1Mb\_EN\_RU\_S\_set1.shan | 47579804 |
| 4 2Mb\_EN\_set1.shan | 87116373 |
| 4 2Mb\_EN\_RU\_set1.shan | 98498498 |
| 4 2Mb\_EN\_RU\_S\_set1.shan | 96365226 |
| 4 3Mb\_EN\_set1.shan | 129098930 |
| 4 3Mb\_EN\_RU\_set1.shan | 146569882 |
| 4 3Mb\_EN\_RU\_S\_set1.shan | 143473966 |
| 1 20Kb\_EN\_set2.txt | 1258949 |
| 1 20Kb\_EN\_RU\_set2.txt | 1253480 |
| 1 20Kb\_EN\_RU\_S\_set2.txt | 1325861 |
| 1 40Kb\_EN\_set2.txt | 2513065 |
| 1 40Kb\_EN\_RU\_set2.txt | 2486288 |
| 1 40Kb\_EN\_RU\_S\_set2.txt | 2632013 |
| 1 60Kb\_EN\_set2.txt | 3767629 |
| 1 60Kb\_EN\_RU\_set2.txt | 3726446 |
| 1 60Kb\_EN\_RU\_S\_set2.txt | 3941278 |
| 1 80Kb\_EN\_set2.txt | 5022489 |
| 1 80Kb\_EN\_RU\_set2.txt | 4964886 |
| 1 80Kb\_EN\_RU\_S\_set2.txt | 5252394 |
| 1 100Kb\_EN\_set2.txt | 6277047 |
| 1 100Kb\_EN\_RU\_set2.txt | 6202187 |
| 1 100Kb\_EN\_RU\_S\_set2.txt | 6553136 |
| 1 1Mb\_EN\_set2.txt | 64268237 |
| 1 1Mb\_EN\_RU\_set2.txt | 63433631 |
| 1 1Mb\_EN\_RU\_S\_set2.txt | 67041624 |
| 1 2Mb\_EN\_set2.txt | 128542405 |
| 1 2Mb\_EN\_RU\_set2.txt | 126908528 |
| 1 2Mb\_EN\_RU\_S\_set2.txt | 134006510 |
| 1 3Mb\_EN\_set2.txt | 192814864 |
| 1 3Mb\_EN\_RU\_set2.txt | 190342712 |
| 1 3Mb\_EN\_RU\_S\_set2.txt | 201092068 |
| 3 20Kb\_EN\_set2.txt | 1260803 |
| 3 20Kb\_EN\_RU\_set2.txt | 1261148 |
| 3 20Kb\_EN\_RU\_S\_set2.txt | 1330262 |
| 3 40Kb\_EN\_set2.txt | 2516040 |
| 3 40Kb\_EN\_RU\_set2.txt | 2499506 |
| 3 40Kb\_EN\_RU\_S\_set2.txt | 2637513 |
| 3 60Kb\_EN\_set2.txt | 3771342 |
| 3 60Kb\_EN\_RU\_set2.txt | 3746055 |
| 3 60Kb\_EN\_RU\_S\_set2.txt | 3947988 |
| 3 80Kb\_EN\_set2.txt | 5026735 |
| 3 80Kb\_EN\_RU\_set2.txt | 4992402 |
| 3 80Kb\_EN\_RU\_S\_set2.txt | 5260600 |
| 3 100Kb\_EN\_set2.txt | 6282047 |
| 3 100Kb\_EN\_RU\_set2.txt | 6236532 |
| 3 100Kb\_EN\_RU\_S\_set2.txt | 6562104 |
| 3 1Mb\_EN\_set2.txt | 64282223 |
| 3 1Mb\_EN\_RU\_set2.txt | 63670531 |
| 3 1Mb\_EN\_RU\_S\_set2.txt | 67099794 |
| 3 2Mb\_EN\_set2.txt | 128561736 |
| 3 2Mb\_EN\_RU\_set2.txt | 127361807 |
| 3 2Mb\_EN\_RU\_S\_set2.txt | 134113872 |
| 3 3Mb\_EN\_set2.txt | 192841164 |
| 3 3Mb\_EN\_RU\_set2.txt | 191010420 |
| 3 3Mb\_EN\_RU\_S\_set2.txt | 201251932 |
| 2 20Kb\_EN\_set2.haff | 1061135 |
| 2 20Kb\_EN\_RU\_set2.haff | 1053126 |
| 2 20Kb\_EN\_RU\_S\_set2.haff | 1115964 |
| 2 40Kb\_EN\_set2.haff | 2120539 |
| 2 40Kb\_EN\_RU\_set2.haff | 2093746 |
| 2 40Kb\_EN\_RU\_S\_set2.haff | 2221623 |
| 2 60Kb\_EN\_set2.haff | 3180332 |
| 2 60Kb\_EN\_RU\_set2.haff | 3140842 |
| 2 60Kb\_EN\_RU\_S\_set2.haff | 3329888 |
| 2 80Kb\_EN\_set2.haff | 4240372 |
| 2 80Kb\_EN\_RU\_set2.haff | 4186431 |
| 2 80Kb\_EN\_RU\_S\_set2.haff | 4439763 |
| 2 100Kb\_EN\_set2.haff | 5300148 |
| 2 100Kb\_EN\_RU\_set2.haff | 5231007 |
| 2 100Kb\_EN\_RU\_S\_set2.haff | 5540749 |
| 2 1Mb\_EN\_set2.haff | 54288665 |
| 2 1Mb\_EN\_RU\_set2.haff | 53550969 |
| 2 1Mb\_EN\_RU\_S\_set2.haff | 56745737 |
| 2 2Mb\_EN\_set2.haff | 108584971 |
| 2 2Mb\_EN\_RU\_set2.haff | 107143938 |
| 2 2Mb\_EN\_RU\_S\_set2.haff | 113431038 |
| 2 3Mb\_EN\_set2.haff | 162879817 |
| 2 3Mb\_EN\_RU\_set2.haff | 160701092 |
| 2 3Mb\_EN\_RU\_S\_set2.haff | 170221886 |
| 4 20Kb\_EN\_set2.shan | 1062677 |
| 4 20Kb\_EN\_RU\_set2.shan | 1058201 |
| 4 20Kb\_EN\_RU\_S\_set2.shan | 1117679 |
| 4 40Kb\_EN\_set2.shan | 2123055 |
| 4 40Kb\_EN\_RU\_set2.shan | 2103713 |
| 4 40Kb\_EN\_RU\_S\_set2.shan | 2224229 |
| 4 60Kb\_EN\_set2.shan | 3183459 |
| 4 60Kb\_EN\_RU\_set2.shan | 3156427 |
| 4 60Kb\_EN\_RU\_S\_set2.shan | 3333575 |
| 4 80Kb\_EN\_set2.shan | 4243993 |
| 4 80Kb\_EN\_RU\_set2.shan | 4208937 |
| 4 80Kb\_EN\_RU\_S\_set2.shan | 4444779 |
| 4 100Kb\_EN\_set2.shan | 5304438 |
| 4 100Kb\_EN\_RU\_set2.shan | 5259543 |
| 4 100Kb\_EN\_RU\_S\_set2.shan | 5546423 |
| 4 1Mb\_EN\_set2.shan | 54300833 |
| 4 1Mb\_EN\_RU\_set2.shan | 53757756 |
| 4 1Mb\_EN\_RU\_S\_set2.shan | 56794707 |
| 4 2Mb\_EN\_set2.shan | 108601900 |
| 4 2Mb\_EN\_RU\_set2.shan | 107541151 |
| 4 2Mb\_EN\_RU\_S\_set2.shan | 113523279 |
| 4 3Mb\_EN\_set2.shan | 162902859 |
| 4 3Mb\_EN\_RU\_set2.shan | 161287019 |
| 4 3Mb\_EN\_RU\_S\_set2.shan | 170360335 |
| 1 20Kb\_EN\_set3.txt | 1258586 |
| 1 20Kb\_EN\_RU\_set3.txt | 1249209 |
| 1 20Kb\_EN\_RU\_S\_set3.txt | 1325857 |
| 1 40Kb\_EN\_set3.txt | 2513821 |
| 1 40Kb\_EN\_RU\_set3.txt | 2485615 |
| 1 40Kb\_EN\_RU\_S\_set3.txt | 2632069 |
| 1 60Kb\_EN\_set3.txt | 3768006 |
| 1 60Kb\_EN\_RU\_set3.txt | 3732213 |
| 1 60Kb\_EN\_RU\_S\_set3.txt | 3949185 |
| 1 80Kb\_EN\_set3.txt | 5022564 |
| 1 80Kb\_EN\_RU\_set3.txt | 4959625 |
| 1 80Kb\_EN\_RU\_S\_set3.txt | 5250231 |
| 1 100Kb\_EN\_set3.txt | 6277730 |
| 1 100Kb\_EN\_RU\_set3.txt | 6200621 |
| 1 100Kb\_EN\_RU\_S\_set3.txt | 6547398 |
| 1 1Mb\_EN\_set3.txt | 64268971 |
| 1 1Mb\_EN\_RU\_set3.txt | 63427096 |
| 1 1Mb\_EN\_RU\_S\_set3.txt | 67062107 |
| 1 2Mb\_EN\_set3.txt | 128545865 |
| 1 2Mb\_EN\_RU\_set3.txt | 126841407 |
| 1 2Mb\_EN\_RU\_S\_set3.txt | 134037554 |
| 1 3Mb\_EN\_set3.txt | 192819297 |
| 1 3Mb\_EN\_RU\_set3.txt | 190308433 |
| 1 3Mb\_EN\_RU\_S\_set3.txt | 201021474 |
| 3 20Kb\_EN\_set3.txt | 1260722 |
| 3 20Kb\_EN\_RU\_set3.txt | 1256281 |
| 3 20Kb\_EN\_RU\_S\_set3.txt | 1330337 |
| 3 40Kb\_EN\_set3.txt | 2516261 |
| 3 40Kb\_EN\_RU\_set3.txt | 2501182 |
| 3 40Kb\_EN\_RU\_S\_set3.txt | 2637585 |
| 3 60Kb\_EN\_set3.txt | 3771500 |
| 3 60Kb\_EN\_RU\_set3.txt | 3750634 |
| 3 60Kb\_EN\_RU\_S\_set3.txt | 3956499 |
| 3 80Kb\_EN\_set3.txt | 5026518 |
| 3 80Kb\_EN\_RU\_set3.txt | 4980963 |
| 3 80Kb\_EN\_RU\_S\_set3.txt | 5258581 |
| 3 100Kb\_EN\_set3.txt | 6282203 |
| 3 100Kb\_EN\_RU\_set3.txt | 6231644 |
| 3 100Kb\_EN\_RU\_S\_set3.txt | 6557187 |
| 3 1Mb\_EN\_set3.txt | 64283346 |
| 3 1Mb\_EN\_RU\_set3.txt | 63660015 |
| 3 1Mb\_EN\_RU\_S\_set3.txt | 67118862 |
| 3 2Mb\_EN\_set3.txt | 128562776 |
| 3 2Mb\_EN\_RU\_set3.txt | 127299549 |
| 3 2Mb\_EN\_RU\_S\_set3.txt | 134150630 |
| 3 3Mb\_EN\_set3.txt | 192844393 |
| 3 3Mb\_EN\_RU\_set3.txt | 190980481 |
| 3 3Mb\_EN\_RU\_S\_set3.txt | 201187583 |
| 2 20Kb\_EN\_set3.haff | 1060855 |
| 2 20Kb\_EN\_RU\_set3.haff | 1049363 |
| 2 20Kb\_EN\_RU\_S\_set3.haff | 1115898 |
| 2 40Kb\_EN\_set3.haff | 2121225 |
| 2 40Kb\_EN\_RU\_set3.haff | 2093159 |
| 2 40Kb\_EN\_RU\_S\_set3.haff | 2221482 |
| 2 60Kb\_EN\_set3.haff | 3180671 |
| 2 60Kb\_EN\_RU\_set3.haff | 3145892 |
| 2 60Kb\_EN\_RU\_S\_set3.haff | 3336797 |
| 2 80Kb\_EN\_set3.haff | 4240438 |
| 2 80Kb\_EN\_RU\_set3.haff | 4181782 |
| 2 80Kb\_EN\_RU\_S\_set3.haff | 4437901 |
| 2 100Kb\_EN\_set3.haff | 5300759 |
| 2 100Kb\_EN\_RU\_set3.haff | 5229634 |
| 2 100Kb\_EN\_RU\_S\_set3.haff | 5535583 |
| 2 1Mb\_EN\_set3.haff | 54289267 |
| 2 1Mb\_EN\_RU\_set3.haff | 53545194 |
| 2 1Mb\_EN\_RU\_S\_set3.haff | 56763697 |
| 2 2Mb\_EN\_set3.haff | 108588049 |
| 2 2Mb\_EN\_RU\_set3.haff | 107084883 |
| 2 2Mb\_EN\_RU\_S\_set3.haff | 113457709 |
| 2 3Mb\_EN\_set3.haff | 162883745 |
| 2 3Mb\_EN\_RU\_set3.haff | 160670914 |
| 2 3Mb\_EN\_RU\_S\_set3.haff | 170160817 |
| 4 20Kb\_EN\_set3.shan | 1062587 |
| 4 20Kb\_EN\_RU\_set3.shan | 1053937 |
| 4 20Kb\_EN\_RU\_S\_set3.shan | 1117637 |
| 4 40Kb\_EN\_set3.shan | 2123245 |
| 4 40Kb\_EN\_RU\_set3.shan | 2105197 |
| 4 40Kb\_EN\_RU\_S\_set3.shan | 2224145 |
| 4 60Kb\_EN\_set3.shan | 3183623 |
| 4 60Kb\_EN\_RU\_set3.shan | 3160469 |
| 4 60Kb\_EN\_RU\_S\_set3.shan | 3340995 |
| 4 80Kb\_EN\_set3.shan | 4243829 |
| 4 80Kb\_EN\_RU\_set3.shan | 4198895 |
| 4 80Kb\_EN\_RU\_S\_set3.shan | 4443037 |
| 4 100Kb\_EN\_set3.shan | 5304587 |
| 4 100Kb\_EN\_RU\_set3.shan | 5255243 |
| 4 100Kb\_EN\_RU\_S\_set3.shan | 5542015 |
| 4 1Mb\_EN\_set3.shan | 54301815 |
| 4 1Mb\_EN\_RU\_set3.shan | 53748459 |
| 4 1Mb\_EN\_RU\_S\_set3.shan | 56811446 |
| 4 2Mb\_EN\_set3.shan | 108602783 |
| 4 2Mb\_EN\_RU\_set3.shan | 107486339 |
| 4 2Mb\_EN\_RU\_S\_set3.shan | 113555015 |
| 4 3Mb\_EN\_set3.shan | 162905714 |
| 4 3Mb\_EN\_RU\_set3.shan | 161260635 |
| 4 3Mb\_EN\_RU\_S\_set3.shan | 170304809 |

Построение графиков производилось по таблицам, приведенным в файле «Graphs.xlsx» (там же можно найти и сами графики). EN – набор символов на латинице и пробел. EN\_RU – набор символов на латинице, кириллице и пробел. EN\_RU\_S – набор символов на латинице, кириллице, пробел и другие символы. Huffman compression/decompression – архивация/разархивация алгоритмом Хаффмана. Shannon-Fano compression/decompression – архивация/разархивация алгоритмом Шеннона-Фано.

* Зависимость количества операций (ось OY) от размера файла (ось OX) и размера набора символов (цвет линии) для каждого алгоритма архивирования и разархивирования:
* Зависимость количества операций (ось OY) от размера файла (ось OX), и используемого алгоритма (цвет линии) для каждого набора символов:

* Зависимость количества операций (ось OY) от размера набора символов (ось OX) на файлах максимального размера (3 Мб) для каждого алгоритма архивирования / разархивирования (цвет линии):

.

# Сравнительный анализ методов (сложность, «+» и «–»)

|  |  |  |  |
| --- | --- | --- | --- |
| **Критерий** | **Алгоритм**  **Хаффмана** | **Алгоритм**  **Шеннона-Фано** | **Комментарий** |
| Сложность архивации в теории | O(N+KlogK) | O(N+KlogK) | N – число символов в тексте архивируемого файла  K – число уникальных символов (алфавит) |
| Сложность архивации в программной реализации | O(N+K^2) | O(N+KlogK) | Программная реализация алгоритма Хаффмана отличается от теоретической (программу можно оптимизировать). Однако из-за того, что число символов алфавита K на порядок меньше числа N (количество символов в файле), данная недоработка несильно сказалась на результате эксперимента. |
| Сложность деархивации | O(N+K) | O(N+K) | Сложность деархивации в разработанной программе совпадает с теоретической. |
| Оптимальное кодирование | Да | Не всегда | Из-за неоднозначности разбиения набора символов на группы, алгоритм Шеннона-Фано не всегда дает на выходе оптимальное бинарное дерево |
| Коэффициент сжатия | Выше | Ниже | Следствие возможной не оптимальности построения бинарного дерева. Большая эффективность алгоритма Хаффмана также видна на графиках, приведенных в пункте «Результаты эксперимента», где прямая алгоритма Хаффмана лежит немного ниже прямой алгоритма Шеннона-Фано. |
| Количество элементарных операций | Меньше | Больше | Количество элементарных операций немного меньше у алгоритма Хаффмана. Это также следует из  возможной не оптимальности построения бинарного дерева. |
| Время исполнения | Меньше  (в теории) | Больше  (в теории) | Следствие возможной не оптимальности построения бинарного дерева, аналогично коэффициенту сжатия. |
| Способ построения дерева | Снизу вверх | Сверху вниз | см. п. «Описание алгоритмов» |

# Заключение (краткие выводы)

* Исследование алгоритмов сжатия Хаффмана и Шеннона-Фано было успешно проведено. На языке С++ была разработана программа для архивации и разархивации файлов с помощью данных алгоритмов. Было проведено тестирование работы программы на файлах различного объема (20 КБ – 3 МБ) с измерением количества элементарных операций, коэффициента сжатия и времени исполнения программы.
* Оба алгоритма обладают довольно большим коэффициентом сжатия, который увеличивается с увеличением размера исходного файла. Так файл объемом 3 МБ сжимается до примерно до 2.4 МБ, т.е. k = 3.0/2.4 = 1.25.
* Количество операций при архивации/разархивации алгоритмом Шеннона-Фано больше у алгоритма Хаффмана, т.к. бинарное дерево кодов символов не всегда строится однозначно. Именно поэтому на практике алгоритм Шеннона-Фано используется редко.
* Разница в количестве минимальных операций и времени выполнения между алгоритмами оказалась минимальна при тестировании файлов, соответствующих спецификации задания, т.к. алгоритм Шеннона-Фано на них строит оптимальные деревья или же очень близкие к оптимальным. Однако это будет происходить не всегда, поэтому алгоритм Хаффмана является наиболее эффективным и универсальным.
* Сложность обоих алгоритмов сжатия сравнима с O(N + KlogK), где K – число символов алфавита, а N – общее число символов в исходном файле. Это подтверждают графики, приведенные в разделе «Результаты эксперимента».
* Сложность деархивации обоих алгоритмов – O(N + K)

# Использованные источники

[1] Новиков Ф. А. Дискретная математика для программистов: Учебник для вузов. 3-е изд. — СПб.: Питер, 2009. ISBN 978-5-91180-759-7

[2] Кормен, Томас X., Лейзерсон, Чарльз И., Ривест, Рональд Л., Штайн, Клиффорд. Алгоритмы: построение и анализ, 2-е издание: Пер. с англ. М.: Издательский дом "Вильямc", 2005. ISBN 5-8459-0857-4

[3] [Электронный ресурс]// URL: <https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_Хаффмана> (Дата обращения: 3.12.2016, режим доступа: свободный)

[4] [Электренноый ресурс]// URL: <https://en.wikipedia.org/wiki/Shannon–Fano_coding> (Дата обращения: 3.12.2016, режим доступа: свободный)