Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Дисциплина «Защита информации и надежность информационных систем»

**Отчёт по лабораторной работе №8**

Сжатие/распаковка данных методом Барроуза-Уилера

Студент: Жук С.С.

ФИТ 3 курс 2 группа

Преподаватель: Савельева М.Г.

**Содержание**

[1 Теоретические сведения 3](#_Toc196009367)

[2 Прямое и обратное преобразование трех символьных и двоичного блоков данных 7](#_Toc196009368)

[3 Сравнительный анализ длительности процессов прямого и обратного преобразований в зависимости от блока данных 12](#_Toc196009369)

[Вывод 13](#_Toc196009370)

# **1 Теоретические сведения**

Сжатие информации является одним из способов ее кодирования. В основе сжатия данных, как одна из первопричин, лежит избыточность.

Основная цель сжатия – обеспечить более компактное представление данных, вырабатываемых источником, т. е. уменьшить физический объем сообщений, генерируемых источником, и сократить время его передачи (стоимость) по каналам связи. Фундаментальная теорема К. Шеннона о кодировании информации утверждает, что «стоимость кодирования всегда не меньше энтропии источника, хотя может быть сколь угодно близка к ней». Поэтому для любого алгоритма сжатия всегда имеется некоторый предел степени (или эффективности) сжатия, определяемый энтропией входного потока (или сжимаемого сообщения).

Все алгоритмы сжатия преобразуют входной поток данных, минимальной единицей которых является бит, а максимальной – байт или несколько байт. Основными техническими характеристиками процессов сжатия и результатов их работы являются:

* степень сжатия, или отношение R объемов исходного (до сжатия, *V*дс) и результирующего (после сжатия, *V*пс) потоков данных (сообщений);
* скорость сжатия − время, затрачиваемое на сжатие некоторого объема информации входного потока до получения из него эквивалентного выходного потока;
* качество сжатия − величина, показывающая, насколько сильно сжат выходной поток при помощи применения к нему повторного сжатия по этому же или иному алгоритму.

Степень сжатия R обычно оценивается следующим образом:

(1.1)

(1.2)

Первое отношение показывает, какую часть объема сообщения (файла) до сжатия занимает сообщение (файл) после сжатия; второе отношение выражает основной физический смысл сжатия и показывает степень сжатия.

Что касается третьей из приведенных технических характеристик (качества сжатия), то она показывает, по существу, совместимость данного метода с другими. Это важно, если принять во внимание то обстоятельство, что практически все современные архиваторы основаны на использовании нескольких разных методов сжатия (кодирования).

По критерию, связанному с характером или форматом данных или степенью соответствия сжимаемых данных распакованным, все методы сжатия разделяют на два класса: обратимое и необратимое сжатие, или иначе: сжатие без потерь и сжатие с частичной потерей информации.

Понятно, что недопустимы никакие потери при упаковке текстовых документов, кодов компьютерных программ, файлов баз данных.

Сжатие с потерей информации реализуется на основе таких известные форматов данных и алгоритмов сжатия, как JPEG и MPEG. Алгоритм JPEG используется при сжатии фотоизображений. Алгоритмы MPEG используют при сжатии видео и музыки.

Методы и алгоритмы сжатия с потерей информации применяют обычно для решения так называемых потребительских задач. Это значит, например, что если фотография передается для просмотра, а музыка для воспроизведения, то подобные алгоритмы применять можно. Если же они передаются для дальнейшей обработки, например для редактирования, то никакая потеря информации в исходном файле недопустима. Считают, что на фотографических иллюстрациях, предназначенных для воспроизведения на экране, потеря 5% информации не критична, а в некоторых случаях можно допустить и 20−25% уровень потерь.

В настоящее время можно встретить достаточно большое число архивирующих и сжимающих утилит, большинство из которых доступны для некоммерческого использования. Поддержка популярных форматов файловых архивов начинает включаться в разные утилиты и программы. Часто используемые форматы становятся стандартными форматами архивов (zip, arj, rar, ha, pak, cab и др.).

BWT-преобразование (Burrows-Wheeler Transform) – техника сжатия информации (в особенности текстов). BWT не сжимает данные в классическом понимании процесса, но преобразует блок данных в формат, исключительно подходящий для сжатия. BWT оперирует сразу целым блоком данных, который выделяется из входного потока (сообщения).

Прямое преобразование (формально – сжатие) выполняется в 4 этапа:

1. выделяется блок данных (строка длиной *k* символов некоторого алфавита мощностью *N*), который обозначим символом *М*;
2. составляется таблица *W*1 размером *k*×*k* всех циклических сдвигов входной строки *M*: *W*1 = (*M*);
3. производится лексикографическая (в алфавитном порядке) сортировка строк таблицы *W*1, в результате чего получается таблица *W*2 того же размера;
4. в качестве выходной строки (обозначим ее *BWT*(*М*), *z*) выбирается последний столбец (*Мk*) таблицы *W*2 преобразования и номер строки *z*, совпадающей с исходной строкой *М*. Как видим, выходная строка (сжатое сообщение) всегда по объему превышает входную.

Обратимость преобразования, т. е. возможность извлечения *М* из *BWT*(*М*), *i* основана на рекурсивности преобразования. Общим для рекурсивных функций, рекурсивных алгоритмов и рекуррентных последовательностей является то, что для вычисления очередного значения функции, получения очередной реализации алгоритма, определения очередного члена последовательности необходимо обращаться к предшествующим их значениям, вычисленным раньше.

Итак, входной для обратного преобразования является информация вида *BWT*(*М*), *i*. Это преобразование заключается в выполнении *k* одинаковых шагов, каждый из которых состоит из 2 операций, с целью воссоздания матрицы *W*2:

1. в крайний справа пустой столбец матрицы записывается последовательность символов *Мk*;
2. производится лексикографическая сортировка столбцов заполненной части воссоздаваемой матрицы.

После *k* шагов матрица *W*2 будет получена. Зная значение числа *z*, получаем входной блок *М* сообщения. Последующий вид матрицы определяется предыдущим ее состоянием.

Пример. Пусть *M* = «столб»; *k* = 5.

Прямое преобразование представлено на рисунке 1.1:

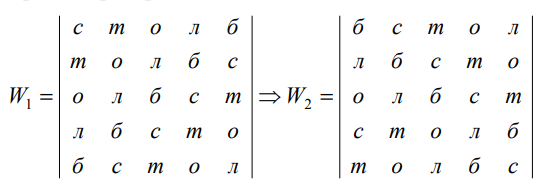


Рисунок 1.1 – Прямое преобразование

Таким образом, имеем: *Мk* = «лотбс» (*k* = 5), *z* = 4 (исходное сообщение – это 4-я строка матрицы *W*2).

Обратное преобразование представлено на рисунке 1.1:

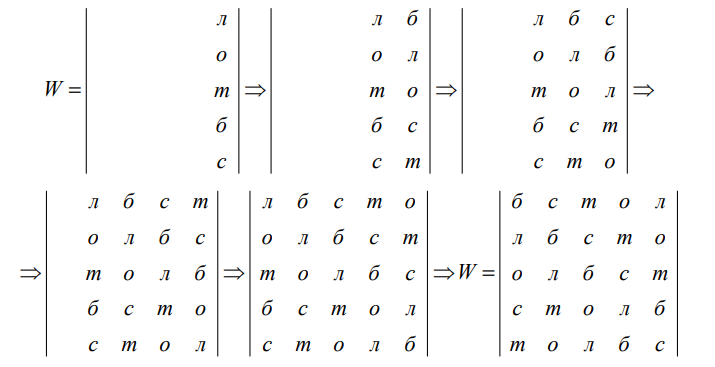


Рисунок 1.2 – Обратное преобразование

На первом шаге в крайний справа незаполненный столбец (пятый) воссоздаваемой матрицы *W*2 (на приведенной условной схеме для математической корректности матрица обозначена символом *W*) записывается значение слова *Мk* = «лотбс», на втором шаге матрица сортируется (реально сортируются строки, содержащие только символы последнего столбца): получим в последнем столбце «блост». Первая операция закончена. Вторая операция: в крайний справа (теперь четвертый) столбец опять записывается входное сообщение «лотбс»; производится сортировка матрицы по значениям 2 последних столбцов. После выполнения указанных операций фактически будет воссоздана (если не возникли ошибки) матрица *W*2. Зная *z* = 4, получим исходное сообщение: *М* = «столб».

Пример. Пусть *M* = «столб»; *k* = 5.

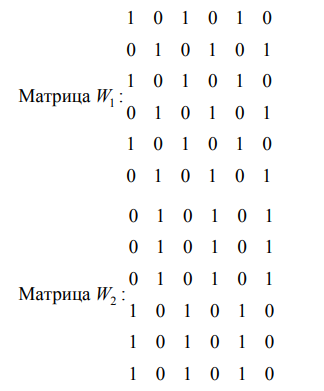


Рисунок 1.3 – Прямое преобразование двоичного блока данных

Результат преобразования: *Мk* = 111000, z = 4 (либо 5, либо 6). Двоичное сообщение может на выходе создавать бинарные последовательности с блоками повторяющихся символов.

# **2 Прямое и обратное преобразование трех символьных и двоичного блоков данных**

Для начала опишем функцию, которая выводит двумерную матрицу в консоль построчно. В конце добавляем пустую строку для визуального разделения. Код представлен в листинге 2.1.

|  |
| --- |
| function showMatrix(matrix) {  matrix.forEach(row => {  *console*.log(row);  });  *console*.log();  } |

Листинг 2.1 – Функция для вывода двумерной матрицы

Следующим шагом будет описана функция, которая выполняет циклические сдвиги строки. Каждый элемент – строка, сдвинутая на одну позицию относительно предыдущей. Сначала добавляем текущую строку, а после выполняем циклический сдвиг: отрезаем первый символ и добавляем его в конец. Функция продемонстрирована в листинге 2.2.

|  |
| --- |
| function createMatrix(input) {  const matrix = [];  for (let i = 0; i < input.length; i++) {  matrix.push(input);  input = input.substring(1) + input[0];  }  return matrix; } |

Листинг 2.2 – Функция для создания циклических сдвигов строки

Далее опишем функцию, которая возвращает копию массива строк, отсортированную в алфавитном порядке. Здесь используем spread-оператор для копирования массива. Программная реализация функции показана в листинге 2.3.

|  |
| --- |
| function copyAndSort(arr) {  return [...arr].sort();} |

Листинг 2.3 – Функция для сортировки строк матрицы в алфавитном порядке

Следующая функция извлекает последний символ из каждой строки матрицы и объединяет их в одну строку. Программная реализация функции представлена в листинге 2.4.

|  |
| --- |
| function extractColumnEnd(grid) {  return grid.map(row => row.at(-1)).join('');} |

Листинг 2.4 – Функция для вывода последнего столбца матрицы

Далее опишем функцию, которая находит индекс строки в массиве. Используется для запоминания позиции оригинального сообщения. Возвращает индекс первого вхождения. Программная реализация функции показана в листинге 2.5.

|  |
| --- |
| function findOriginalIndex(original, array) {  return array.indexOf(original); } |

Листинг 2.5 – Функция для сортировки строк матрицы в алфавитном порядке

Следующая функция приписывает каждому элементу строки к началу строки матриц. Используется при декодировании для построения следующей итерации. Программная реализация функции представлена в листинге 2.6.

|  |
| --- |
| function prependCharacters(chars, matrix) {  return matrix.map((row, i) => chars[i] + row); } |

Листинг 2.6 – Функция для приписывания столбцов в матрицу

Далее опишем функцию, которая восстанавливает *W*2-матрицу на основе закодированной строки. На каждой итерации приписывает символы к текущей матрице и сортирует результат. Программная реализация функции показана в листинге 2.7.

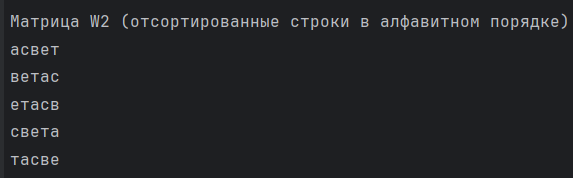
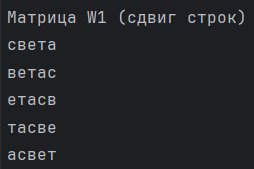
|  |
| --- |
| function restoreMatrixFromEncoded(encodedStr) {  let matrix = *Array*(encodedStr.length).fill('');  for (let i = 0; i < encodedStr.length; i++) {  matrix = prependCharacters(encodedStr, matrix);  showMatrix(matrix);  matrix.sort();  }  return matrix;  } |

Листинг 2.7 – Функция для восстановления *W*2-матрицы

Результат работы программы показан на рисунках 2.1 – 2.8.

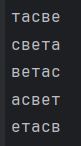


Рисунок 2.1 – Вывод результатов для 1 блока данных









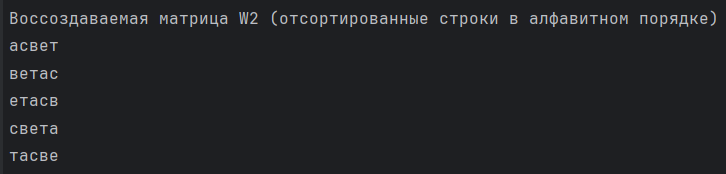
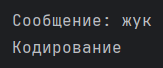




Рисунок 2.2 – Вывод результатов для 1 блока данных



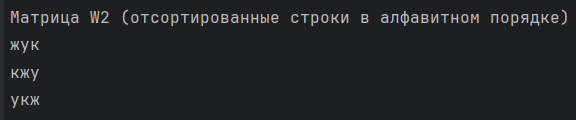
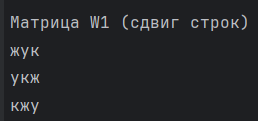
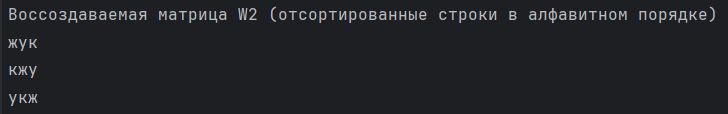








Рисунок 2.4 – Вывод результатов для 2 блока данных



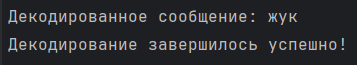
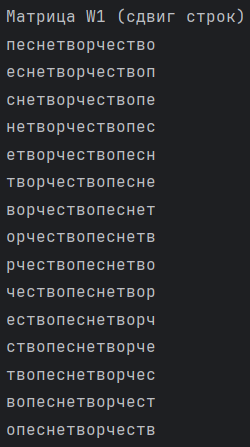
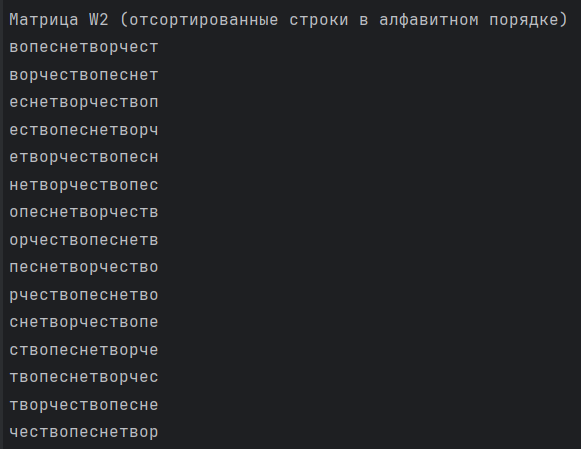
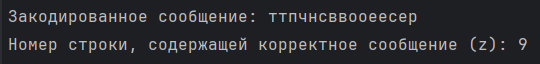


Рисунок 2.5 – Вывод результатов для 2 блока данных







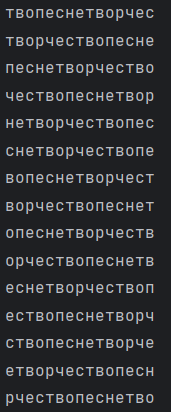
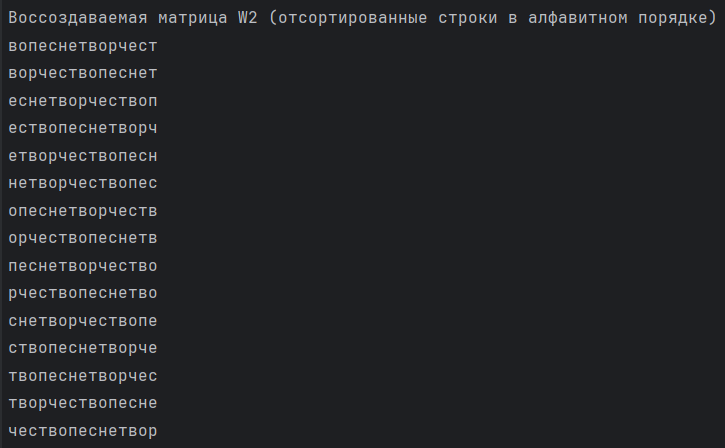
 

Рисунок 2.6 – Вывод результатов для 3 блока данных



Рисунок 2.7 – Вывод результатов для 3 блока данных

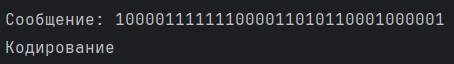




Рисунок 2.8 – Вывод результатов для двоичного блока данных

Прямое преобразование (сжатие) происходит следующим образом: исходное сообщение записывается в виде матрицы *k*×*k*, после чего производятся циклические сдвиги строк. Затем строки матрицы сортируются в алфавитном порядке, в результате чего получается отсортированная матрица *k*×*k*. В качестве выходной строки выбирается последний столбец, и в переменную *z* записывается номер строки исходного сообщения.

Обратное преобразование необходимо для восстановления исходного сообщения. Выходная строка записывается в первый столбец. Следующие операции включают рекурсивное добавление столбцов из матрицы *W*2 по порядку. После того как матрица *k*×*k* воссоздана, следующий этап включает сортировку столбцов. Чтобы получить исходное сообщение, необходимо знать значение *z*. По этому значению извлекаем исходное сообщение из *z* строки.

# **3 Сравнительный анализ длительности процессов прямого и обратного преобразований в зависимости от блока данных**

Для выполнения сравнительного анализа длительности прямого и обратного преобразований будет подключен модуль perf\_hooks, который позволяет измерять время в node.js.





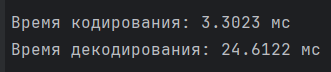




Рисунок 3.1 – Вывод результатов

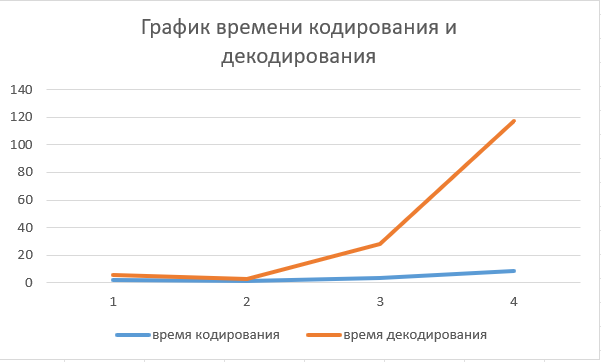


Рисунок 3.2 – График

Прямое сжатие данных методом Барроуза-Уилера происходит быстрее за счет преобразования исходных данных в форму, подходящую для сжатия. Обратное преобразование, или распаковка данных, занимает больше времени из-за необходимости выполнения обратных операций по восстановлению исходной последовательности символов.

Сравнение времени сжатия и распаковки показывает, что сжатие происходит быстрее распаковки более чем в два раза для большого количество символов. Если количество символов мало, то время будет примерно одинаково.

Сжатие и распаковка данных, состоящих только из цифр 0 и 1, происходит аналогичным образом, как и с символами. Основное отличие заключается в том, что сортировка проводится по убыванию. Здесь видно, что с большим количеством цифр в исходном сообщении, время сжатия и распаковки данных слишком различаются.

# **Вывод**

В ходе данной лабораторной работы было создано приложение, реализующее сжатие и распаковку данных с применением трансформации Барроуза–Уилера. Также были изучены теоретические основы, касающиеся принципов работы алгоритма и методов обработки информации.

Можно заключить, что процесс сжатия с использованием метода Барроуза–Уилера обычно проходит значительно быстрее, чем восстановление исходных данных. Это объясняется тем, что при сжатии осуществляется лишь переупорядочивание символов входного текста с целью группировки одинаковых символов, что облегчает последующее сжатие другими алгоритмами. В то же время операция восстановления требует более сложных вычислений, так как необходимо точно воссоздать исходную последовательность символов, что делает распаковку более ресурсоемкой.