Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Отчёт**

Лабораторная работа №8 «Сжатие/распаковка данных методом Барроуза-Уилера»

Студент: Водчиц Анастасия

ФИТ 3 курс 1 группа

Преподаватель: Нистюк О.А.

Минск 2025

**Цель**: приобретение практических навыков использования метода Барроуза − Уилера для сжатия/распаковки данных.

**Задачи**:

* Закрепить теоретические знания по алгебраическому описанию и использованию методов сжатия/распаковки (архивации/ разархивации) данных на основе метода Барроуза − Уилера (BurrowsWheeler transform, BWT).
* Разработать приложение для реализации метода Барроуза − Уилера.
* Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

# **Теоретические сведения**

Данная лабораторная работа открывает второй раздел практикума, относящийся к методам сжатия данных (англ. data compression), которые играют важную роль в решении проблемы защиты информации, ведь архивация данных необходима не только для экономии места на локальном дисковом носителе, но и для переноса информации, резервирования, резервного копирования и т. п. Исходя из этого, далее дадим краткую характеристику данному классу методов преобразования информации. Сжатие информации является одним из способов ее кодирования.

Развитие методов сжатия данных имеют длинную историю, которая началась задолго до появления компьютеров и компьютерных сетей.

Основная цель сжатия – обеспечить более компактное представление данных, вырабатываемых источником, т. е. уменьшить физический объем сообщений, генерируемых источником, и сократить время его передачи (читай – 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 (англ. compress rating), или отношение R (англ. ratio) объемов исходного (до сжатия, Vдс) и результирующего (после сжатия, Vпс) 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:



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).

Существуют различные подходы к реализации сжатия информации. Они отличаются математической базой, уровнем сложности (простоты) практической реализации, форматом кодируемого потока данных, степенью соответствия сжимаемых и распакованных данных.

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 (англ. lossy compression).

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 (англ. Burrows-Wheeler Transform) – техника сжатия информации (в особенности текстов), основанная на преобразовании, открытом в 1983 г. 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 W1 размером k×k всех циклических сдвигов входной строки M: W1 = (M);
3. производится лексикографическая (в алфавитном порядке) сортировка строк таблицы W1, в результате чего получается таблица W2 того же размера;
4. в качестве выходной строки (обозначим ее BWT(М), z) выбирается последний столбец (Мk) таблицы W2 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. Вспомним, что 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 W2:

1. в крайний справа пустой столбец матрицы записывается последовательность символов Мk;
2. производится лексикографическая сортировка столбцов заполненной части воссоздаваемой матрицы. После k шагов матрица W2 будет получена. Зная значение числа z, получаем входной блок М сообщения. Как видим, последующий вид матрицы определяется предыдущим ее состоянием.

# **Практические задания**

**Задание 1.** Разработать авторское приложение в соответствии с целью лабораторной работы. Входной блок данных может иметь произвольную длину.

|  |
| --- |
| **public static BWTResult Encode(string text)**  **{**  **Console.WriteLine($"--- Запуск прямого преобразования BWT для строки: '{text}' (сдвиги ВПРАВО) ---");**  **if (string.IsNullOrEmpty(text))**  **{**  **Console.WriteLine("Входная строка пуста или null.");**  **return new BWTResult { TransformedText = "", OriginalIndex = -1 };**  **}**  **int n = text.Length;**  **List<string> rotations = new List<string>(n);**  **string currentRotation = text;**  **Console.WriteLine("\n1. Генерация циклических сдвигов (ротаций) ВПРАВО:");**  **for (int i = 0; i < n; i++)**  **{**  **rotations.Add(currentRotation);**  **Console.WriteLine($" Сдвиг {i}: {currentRotation}");**  **if (n > 0)**  **{**  **char lastChar = currentRotation[n - 1];**  **string restOfString = currentRotation.Substring(0, n - 1);**  **currentRotation = lastChar + restOfString;**  **}**  **}**  **Console.WriteLine("\n2. Лексикографическая сортировка сдвигов:");**  **rotations.Sort(StringComparer.Ordinal);**  **Console.WriteLine(" Отсортированная матрица:");**  **for (int i = 0; i < n; i++)**  **{**  **Console.WriteLine($" [{i.ToString().PadLeft(2)}]: {rotations[i]}");**  **}**  **Console.WriteLine("\n3. Извлечение последнего столбца (L):");**  **StringBuilder transformedTextBuilder = new StringBuilder(n);**  **for (int i = 0; i < n; i++)**  **{**  **transformedTextBuilder.Append(rotations[i][n - 1]);**  **}**  **string transformedText = transformedTextBuilder.ToString();**  **Console.WriteLine($" Последний столбец (L): '{transformedText}'");**  **Console.WriteLine("\n4. Поиск индекса исходной строки (I):");**  **int originalIndex = -1;**  **for (int i = 0; i < n; i++)**  **{**  **if (rotations[i].Equals(text, StringComparison.Ordinal))**  **{**  **originalIndex = i;**  **break;**  **}**  **}**  **if (originalIndex != -1)**  **{**  **Console.WriteLine($" Исходная строка '{text}' найдена в отсортированной матрице по индексу: {originalIndex} (это {originalIndex + 1}-я строка)");**  **}**  **else**  **{**  **Console.WriteLine($" Ошибка: Исходная строка '{text}' НЕ найдена в отсортированной матрице!");**  **}**  **Console.WriteLine("--- Прямое преобразование BWT завершено ---");**  **return new BWTResult { TransformedText = transformedText, OriginalIndex = originalIndex };**  **}** |

Листинг 1.1 – Метод прямого преобразования

|  |
| --- |
| **public static string DecodeMatrixMethod(string transformedText, int originalIndex)**  **{**  **Console.WriteLine($"\n--- Запуск ОБРАТНОГО преобразования BWT (метод матрицы) для L='{transformedText}', I={originalIndex} ({originalIndex + 1}-я строка) ---");**  **if (string.IsNullOrEmpty(transformedText) || originalIndex < 0)**  **{**  **Console.WriteLine("Некорректные входные данные для декодирования.");**  **return "";**  **}**  **int n = transformedText.Length;**  **Console.WriteLine($"Длина строки L: {n}");**  **List<string> matrixRows = new List<string>(n);**  **for (int i = 0; i < n; ++i)**  **{**  **matrixRows.Add("");**  **}**  **Console.WriteLine("\nНачинаем построение матрицы справа налево:");**  **for (int j = 0; j < n; j++)**  **{**  **Console.WriteLine($"\nИтерация {j + 1} (построение {n - j}-го столбца):");**  **Console.WriteLine(" 1. Добавление символов L в начало строк:");**  **List<string> tempRows = new List<string>(n);**  **for (int i = 0; i < n; i++)**  **{**  **string newRow = transformedText[i] + matrixRows[i];**  **tempRows.Add(newRow);**  **Console.WriteLine($" '{transformedText[i]}' + \"{matrixRows[i]}\" -> \"{newRow}\"");**  **}**  **matrixRows = tempRows;**  **Console.WriteLine("\n 2. Сортировка строк:");**  **matrixRows.Sort(StringComparer.Ordinal);**  **Console.WriteLine(" Текущее состояние матрицы (строки отсортированы):");**  **for (int i = 0; i < n; i++)**  **{**  **if (matrixRows[i].Length > 0)**  **{**  **Console.Write($" [{i.ToString().PadLeft(2)}]: '{matrixRows[i][0]}' ");**  **if (matrixRows[i].Length > 1)**  **{**  **Console.WriteLine($"| \"{matrixRows[i].Substring(1)}\"");**  **}**  **else**  **{**  **Console.WriteLine();**  **}**  **}**  **else**  **{**  **Console.WriteLine($" [{i.ToString().PadLeft(2)}]: \"\"");**  **}**  **}**  **}**  **Console.WriteLine("\n--- Построение матрицы завершено ---");**  **if (originalIndex >= 0 && originalIndex < n)**  **{**  **string decodedText = matrixRows[originalIndex];**  **Console.WriteLine($"\nИзвлекаем строку по индексу I={originalIndex} ({originalIndex + 1}-я строка): '{decodedText}'");**  **Console.WriteLine("--- Обратное преобразование BWT завершено ---");**  **return decodedText;**  **}**  **else**  **{**  **Console.WriteLine($"\nОшибка: Некорректный индекс {originalIndex}.");**  **Console.WriteLine("--- Обратное преобразование BWT завершено с ошибкой ---");**  **return "";**  **}**  **}** |

Листинг 1.2 – Метод обратного преобразования

**Задание 2.** С помощью приложения выполнить прямое и обратное преобразования 3 отдельных блоков данных, состоящих: а) из собственного имени (можно краткий вариант записи); б) собственной фамилии; в) варианта в соответствии с таблицей ниже. Можно использовать любой из известных методов сортировки символов массива. Выполнить качественный сравнительный анализ длительности процессов прямого и обратного преобразований в зависимости от длины блока данных.

|  |  |
| --- | --- |
| Вариант | Входной блок данных |
| 3 | Мультимиллионер |

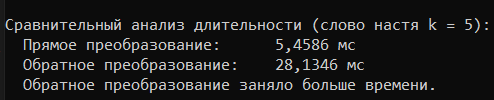


Рисунок 2.1 – Анализ длительности преобразований для задания а

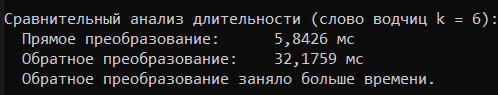


Рисунок 2.2 – Анализ длительности преобразований для задания б

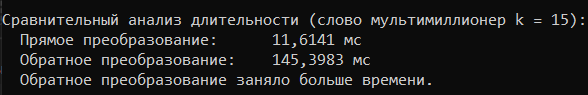


Рисунок 2.3 – Анализ длительности преобразований для задания в

Рассмотрим пример со словом «полка».

Прямое преобразование BWT для входного блока данных (строки) S длины n выполняется в несколько шагов:

Шаг 1. Формируется матрица, состоящая из n строк, где каждая строка является циклическим сдвигом исходной строки S. В данном примере используется циклический сдвиг, где на каждом шаге последний символ строки перемещается в ее начало.

Сдвиг 0: полка

Сдвиг 1: аполк

Сдвиг 2: капол

Сдвиг 3: лкапо

Сдвиг 4: олкап

Шаг 2. Полученная матрица циклических сдвигов сортируется лексикографически (по алфавиту, как в словаре). Сравнение строк обычно происходит по кодам символов. Получаем матрицу M'.

[0]: аполк

[1]: капол

[2]: лкапо

[3]: олкап

[4]: полка

Результатом прямого преобразования BWT является строка L, которая формируется путем конкатенации символов, находящихся в последнем столбце отсортированной матрицы.

Последний столбец матрицы M' содержит символы «клопа». Таким образом, строка L = «клопа».

Помимо строки L, для обратного преобразования необходимо сохранить индекс I строки в отсортированной матрице M', которая совпадает с исходной строкой S.

Исходная строка «полка» находится в отсортированной матрице M на 5-й позиции (индекс 4). Следовательно, I = 5.

Результат прямого преобразования: Пара (L, I). В нашем примере: (клопа, 5).

Обратное преобразование позволяет восстановить исходную строку S, имея только результат прямого преобразования – строку L и индекс I. Метод основан на итеративном восстановлении отсортированной матрицы M'.

Создается пустая матрица размером n x 0 (или, что эквивалентно, n пустых строк).

(клопа, n=5): Изначально имеем 5 пустых строк.

Итеративное построение и сортировка (n итераций): Процесс повторяется n раз. На каждой итерации j (от 1 до n):

* Добавление столбца L: Текущие строки матрицы расширяются путем добавления в начало каждой строки i соответствующего символа L[i].
* Сортировка строк: Полученные после добавления столбца строки снова сортируются лексикографически.

Итерация 1 (Построение 5-го столбца M'). Добавляем L = «клопа» к пустым строкам: к, л, о, п, а. Сортируем:

[0]: а

[1]: к

[2]: л

[3]: о

[4]: п

Итерация 2 (Построение 4-го столбца M'). Добавляем L = «клопа» к строкам а, к, л, о, п: ка, лк, ол, по, ап. Сортируем:

[0]: ап

[1]: ка

[2]: лк

[3]: ол

[4]: по

Итерация 3 (Построение 3-го столбца M'). Добавляем L = «клопа» к строкам ап, ка, лк, ол, по: кап, лка, олк, пол, апо. Сортируем:

[0]: апо

[1]: кап

[2]: лка

[3]: олк

[4]: пол

Итерация 4 (Построение 2-го столбца M'). Добавляем L = «клопа» к строкам апо, кап, лка, олк, пол: капо, лкап, олка, полк, апол. Сортируем:

[0]: апол

[1]: капо

[2]: лкап

[3]: олка

[4]: полк

Итерация 5 (Построение 1-го столбца M'). Добавляем L = «клопа» к строкам апол, капо, лкап, олка, полк: капол, лкапо, олкап, полка, аполк. Сортируем:

[0]: аполк

[1]: капол

[2]: лкапо

[3]: олкап

[4]: полка

После n итераций получена полная отсортированная матрица циклических сдвигов M'. Исходная строка S находится в этой матрице по индексу I, сохраненному во время прямого преобразования.

Извлекаем строку с индексом 4 (отправлено 5, но индексы в массиве считаются от нуля) из финальной матрицы: полка.

Результат обратного преобразования: Исходная строка S: полка.

**Задание 3.** Перевести первые 3 символа из блока данных, указанного в варианте таблицы, в бинарную последовательность в соответствии с кодами ASCII. Выполнить прямое и обратное преобразование. Оценить время прямого и обратного преобразований.

1. м:

* Unicode код: U+043C
* Представление в UTF-8 (Hex): D0 BC
* Представление в UTF-8 (Двоичный): 11010000 10111100

1. у:

* Unicode код: U+0443
* Представление в UTF-8 (Hex): D1 83
* Представление в UTF-8 (Двоичный): 11010001 10000011

1. л:

* Unicode код: U+043B
* Представление в UTF-8 (Hex): D0 BB
* Представление в UTF-8 (Двоичный): 11010000 10111011

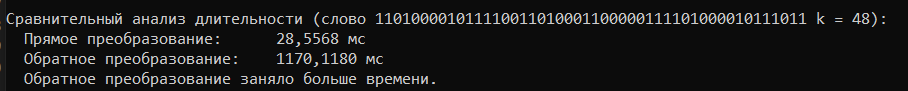


Рисунок 3.1 – Анализ длительности преобразований для бинарной последовательности

**Задание 4.** Результаты оформить в виде отчета по установленным правилам.

**Вывод**: в ходе выполнения лабораторной работы было изучено и реализовано Преобразование Барроуза-Уилера (BWT) – блочный алгоритм сортировки символов, предназначенный для предварительной обработки данных перед сжатием.