Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Отчёт**

Лабораторная работа №10 «Сжатие/распаковка данных методом Лемпеля-Зива»

Студент: Водчиц Анастасия

ФИТ 3 курс 1 группа

Преподаватель: Нистюк О.А.

Минск 2025

**Цель**: приобретение практических навыков использования метод Лемпеля − Зива (Lempel-Ziv) для сжатия/распаковки данных.

**Задачи**:

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

# Теоретические сведения

В 1977 г. Авраам Лемпель и Якоб Зив выдвинули идею формирования «словаря» общих последовательностей анализируемых (сжимаемых) данных. При этом сжатие данных осуществляется за счет замены записей соответствующими кодами из словаря. Классический алгоритм Лемпеля − Зива – LZ77, названный так по году представления метода, формулируется следующим образом: «если в проанализированном (сжатом) ранее выходном потоке уже встречалась подобная последовательность байт, причем запись о ее длине и смещении от текущей позиции короче, чем сама эта последовательность, то в выходной файл записывается ссылка (смещение, длина), а не сама последовательность».

Суть метода LZ77 (как и последующих его модификаций) состоит в следующем: упаковщик постоянно хранит некоторое количество последних обработанных символов в буфере. По мере обработки входного потока вновь поступившие символы попадают в конец буфера, сдвигая предшествующие символы и вытесняя самые старые. Размеры этого буфера, называемого также скользящим словарем (англ. sliding dictionary), варьируются в разных реализациях систем сжатия. Скользящее окно имеет длину n, т. е. в него помещается n символов, и состоит из двух частей:

* последовательности длины n1 = n − n2 уже закодированных символов (словарь);
* упреждающего буфера (буфера предварительного просмотра, lookahead) длиной n2 – буфера кодирования.

Нужно найти самое длинное совпадение между строкой буфера кодирования, начинающейся с символа St + 1, и всеми фразами словаря.

Эти фразы могут начинаться с любого символа St − (n1 − 1), St − (n1 − 1) + 1, …, St, выходить за пределы словаря, вторгаясь в область буфера, но должны лежать в окне. Буфер не может сравниваться сам с собой. Длина совпадения не должна превышать размера буфера. Полученная в результате поиска фраза St − (р − 1), St − (р − 1) + 1, St − (р − 1) + (q − 1) кодируется с помощью двух чисел:

1. смещения (англ. offset) от начала буфера p;
2. длины соответствия, или совпадения (англ. match length) q.

Ссылки (p и q − указатели) однозначно определяют фразу. Дополнительно в выходной поток записывается символ s, следующий за совпавшей строкой буфера.

Длина кодовой комбинации (триады – p, q, s) на каждом шаге определяется соотношением l(с ) = logN n1 +logN n2 + 1, где N – мощность алфавита.

После каждого шага окно смещается на q + 1 символов вправо и осуществляется переход к новому циклу кодирования. Величина сдвига объясняется тем, что мы реально закодировали именно q + 1 символов: q – с помощью указателя и 1 − с помощью тривиального копирования.

Передача одного символа в явном виде (s) позволяет разрешить проблему обработки еще ни разу не встречавшихся символов, но существенно увеличивает размер сжатого блока.

# Практические задания

**Задание 1.** Разработать авторское приложение в соответствии с целью лабораторной работы. При этом предусмотреть возможность оперативного изменения размеров окон (n1, n2).

|  |
| --- |
| **public List<Triplet> Encode(string inputText)**  **{**  **List<Triplet> encodedTriplets = new List<Triplet>();**  **StringBuilder dictionary = new StringBuilder(new string('\0', dictionarySizeN1));**  **int currentPositionInInput = 0; int step = 1;**  **while (currentPositionInInput < inputText.Length)**  **{**  **int lookaheadActualSize = Math.Min(lookaheadBufferSizeN2, inputText.Length - currentPositionInInput);**  **if (lookaheadActualSize == 0) break;**  **string lookaheadBuffer = inputText.Substring(currentPositionInInput, lookaheadActualSize);**  **int bestMatchOffset = 0; int bestMatchLength = 0;**  **string foundMatchString = "";**  **for (int p\_try = 0; p\_try < dictionarySizeN1; p\_try++)**  **{**  **for (int q\_try = 1; q\_try <= lookaheadActualSize; q\_try++)**  **{**  **if (p\_try + q\_try > dictionarySizeN1) break;**  **string dictSub = dictionary.ToString().Substring(p\_try, q\_try); string lookSub = lookaheadBuffer.Substring(0, q\_try);**  **if (dictSub == lookSub)**  **{**  **if (q\_try > bestMatchLength)**  **{**  **bestMatchLength = q\_try;**  **bestMatchOffset = p\_try;**  **foundMatchString = dictSub;**  **}**  **}**  **else break;**  **}**  **}**  **char nextSymbolS;**  **if (bestMatchLength == 0)**  **{**  **nextSymbolS = lookaheadBuffer[0];**  **if (enableVerboseOutput) Console.WriteLine($" Нет совпадения. s = '{nextSymbolS}'");**  **}**  **else**  **{**  **if (currentPositionInInput + bestMatchLength < inputText.Length) nextSymbolS = inputText[currentPositionInInput + bestMatchLength];**  **else nextSymbolS = '\0';**  **if (enableVerboseOutput) Console.WriteLine($" Найдено совпадение: \"{foundMatchString}\" (p={bestMatchOffset}, q={bestMatchLength}). s = '{nextSymbolS.ToString().Replace("\0", "\\0")}'");**  **}**  **encodedTriplets.Add(new Triplet(bestMatchOffset, bestMatchLength, nextSymbolS));**  **int shiftAmount = bestMatchLength + 1;**  **string stringToShiftIntoDictionary;**  **if (currentPositionInInput + shiftAmount <= inputText.Length) stringToShiftIntoDictionary = inputText.Substring(currentPositionInInput, shiftAmount);**  **else stringToShiftIntoDictionary = inputText.Substring(currentPositionInInput, inputText.Length - currentPositionInInput);**  **if (nextSymbolS == '\0' && stringToShiftIntoDictionary.Length > bestMatchLength) stringToShiftIntoDictionary = stringToShiftIntoDictionary.Substring(0, bestMatchLength);**  **if (stringToShiftIntoDictionary.Length > 0)**  **{**  **if (dictionary.Length + stringToShiftIntoDictionary.Length <= dictionarySizeN1) dictionary.Append(stringToShiftIntoDictionary);**  **else**  **{**  **int toRemove = (dictionary.Length + stringToShiftIntoDictionary.Length) - dictionarySizeN1;**  **if (toRemove > 0 && dictionary.Length > 0) dictionary.Remove(0, Math.Min(toRemove, dictionary.Length));**  **dictionary.Append(stringToShiftIntoDictionary);**  **}**  **if (dictionary.Length > dictionarySizeN1) dictionary.Remove(0, dictionary.Length - dictionarySizeN1);**  **}**  **currentPositionInInput += shiftAmount; step++;**  **if (nextSymbolS == '\0' && currentPositionInInput >= inputText.Length) break;**  **}**  **return encodedTriplets;**  **}** |

Листинг 1.1 – Метод для прямого преобразования

|  |
| --- |
| **public string Decode(List<Triplet> encodedTriplets)**  **{**  **StringBuilder decodedText = new StringBuilder();**  **StringBuilder dictionary = new StringBuilder(new string('\0', dictionarySizeN1)); int step = 1;**  **foreach (Triplet triplet in encodedTriplets)**  **{**  **int p = triplet.P; int q = triplet.Q; char s = triplet.S; string partToAppendAndShift = "";**  **if (q > 0)**  **{**  **string matchedString = "";**  **for (int i = 0; i < q; i++)**  **{**  **if (p + i < dictionarySizeN1) matchedString += dictionary[p + i];**  **else break;**  **}**  **decodedText.Append(matchedString);**  **partToAppendAndShift += matchedString;**  **}**  **if (s != '\0')**  **{**  **decodedText.Append(s); partToAppendAndShift += s;**  **}**  **if (partToAppendAndShift.Length > 0)**  **{**  **if (dictionary.Length + partToAppendAndShift.Length <= dictionarySizeN1) dictionary.Append(partToAppendAndShift);**  **else**  **{**  **int toRemove = (dictionary.Length + partToAppendAndShift.Length) - dictionarySizeN1;**  **if (toRemove > 0 && dictionary.Length > 0) dictionary.Remove(0, Math.Min(toRemove, dictionary.Length));**  **dictionary.Append(partToAppendAndShift);**  **}**  **if (dictionary.Length > dictionarySizeN1) dictionary.Remove(0, dictionary.Length - dictionarySizeN1);**  **}**  **step++;**  **if (s == '\0') break;**  **}**  **return decodedText.ToString();**  **}** |

Листинг 1.2 – Метод для обратного преобразование

**Задание 2.** С помощью приложения выполнить прямое и обратное преобразования произвольного текста длиной несколько килобайт. Формат представления параметров p и q выбрать по указанию преподавателя.

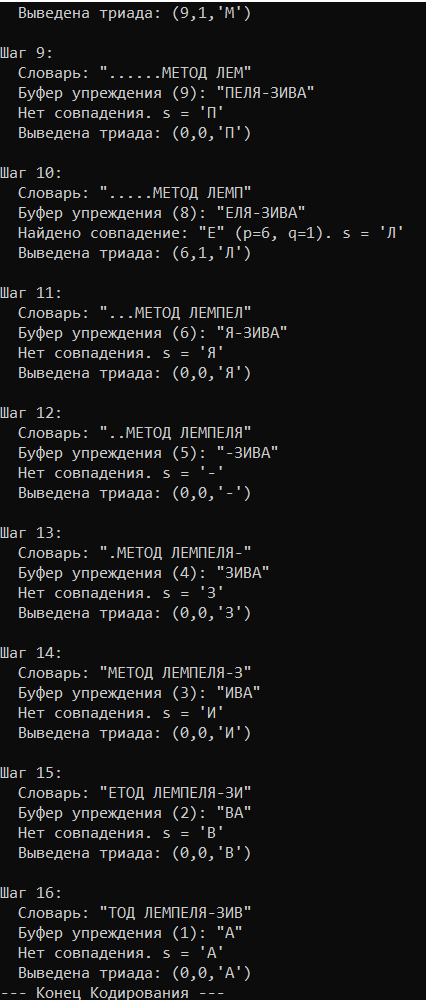
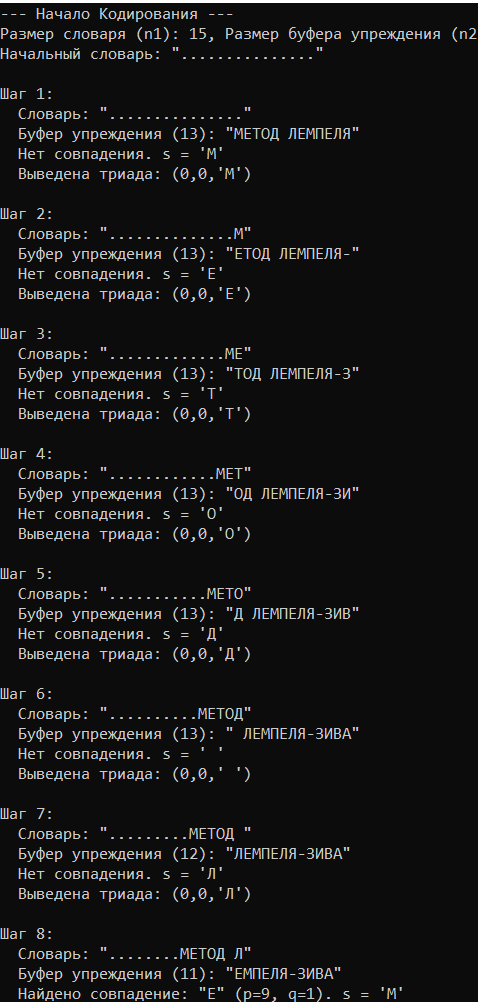


Рисунок 2.1 – Пример прямого преобразования в консоли

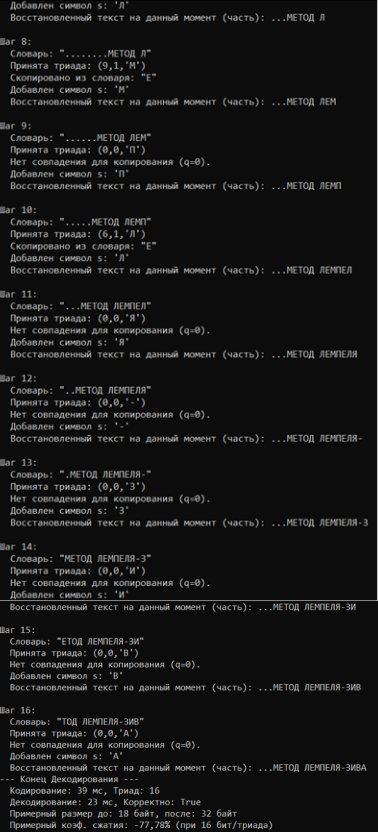
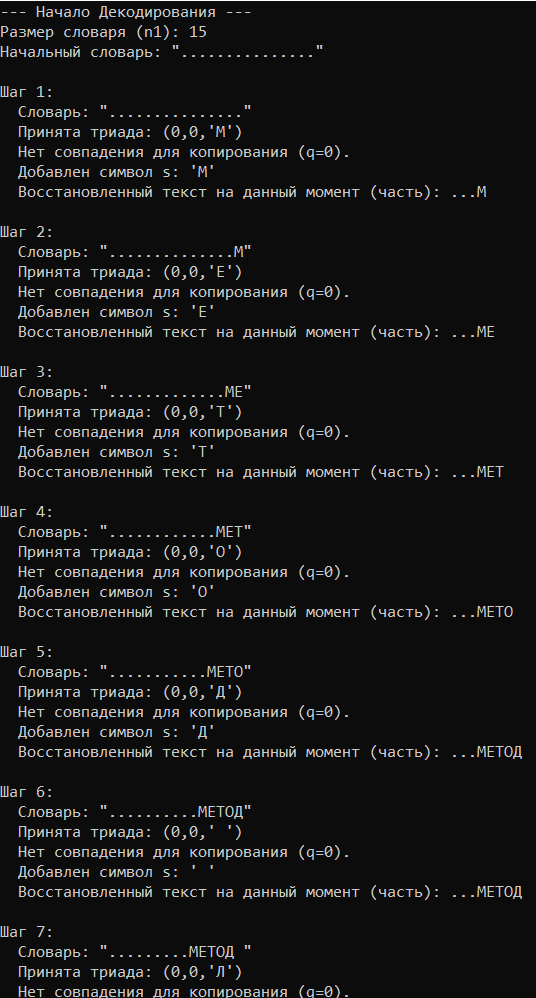
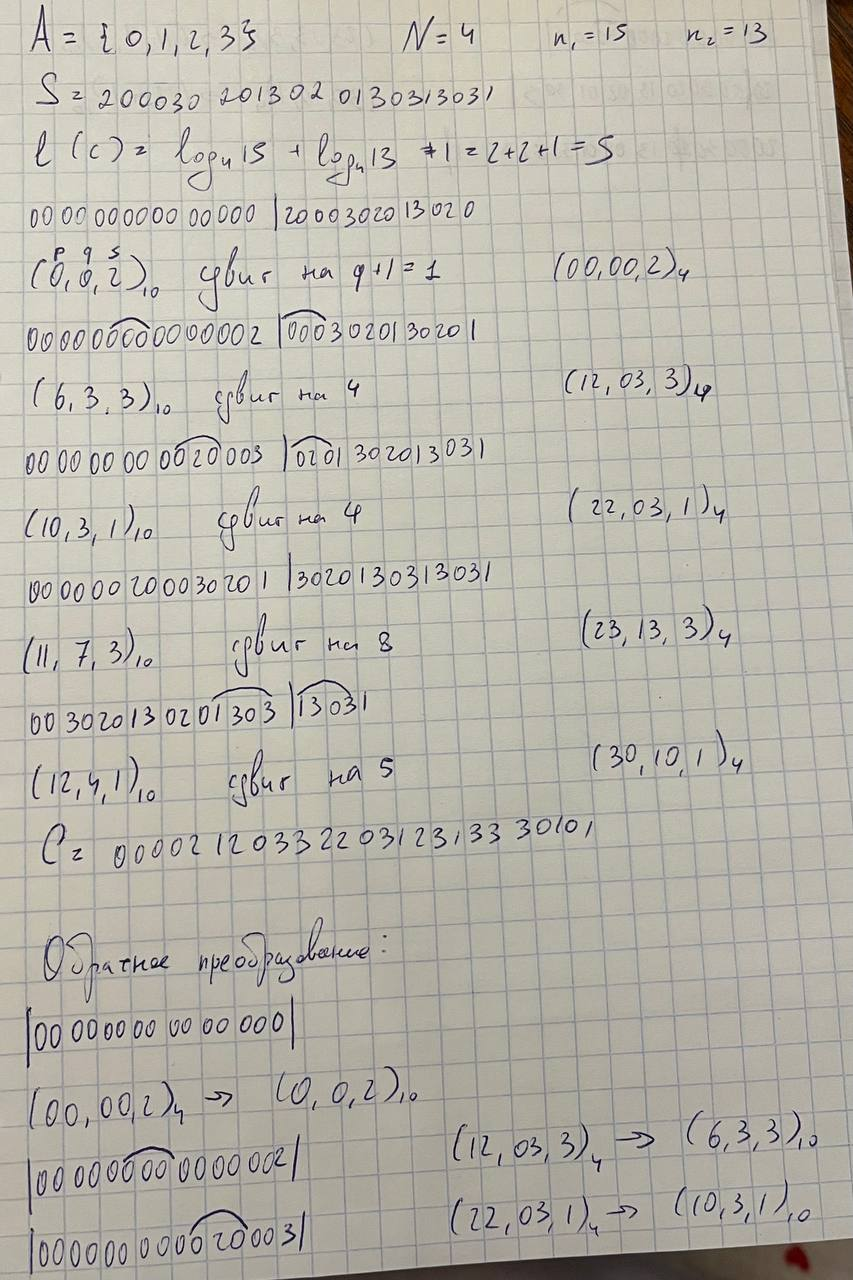


Рисунок 2.2 – Пример обратного преобразования в консоли



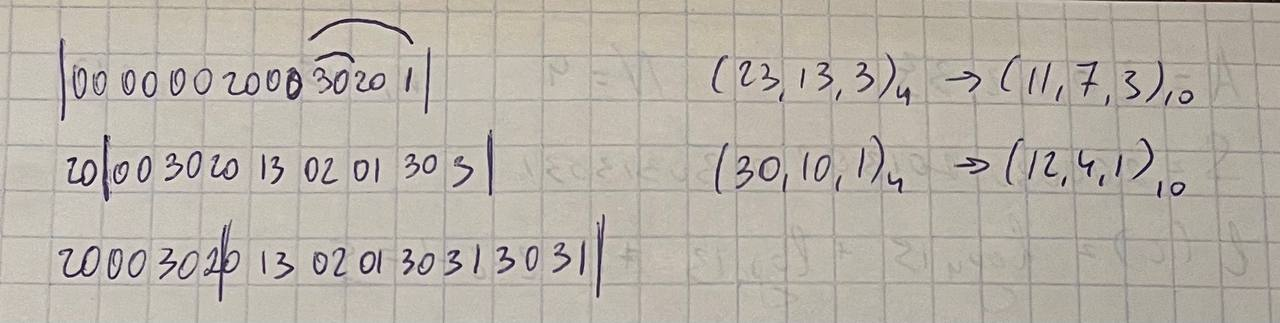


Рисунок 2.3 – Пример прямого и обратного преобразования

**Задание 3.** Изменяя размеры окон, оценить скорость и эффективность выполнения операций сжатия/ распаковки.

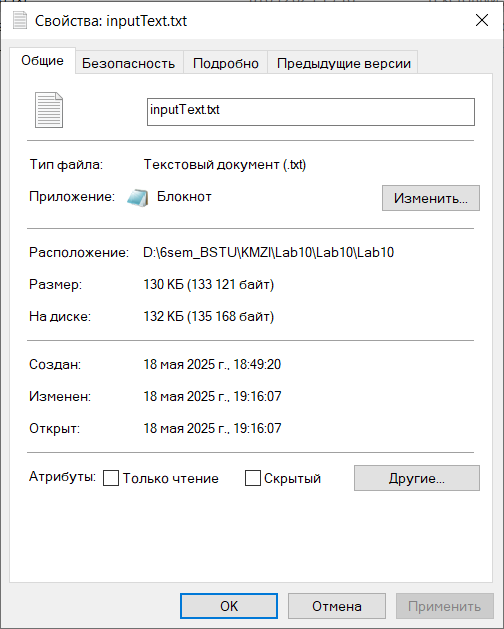


Рисунок 3.1 – Размер файла с входной последовательностью

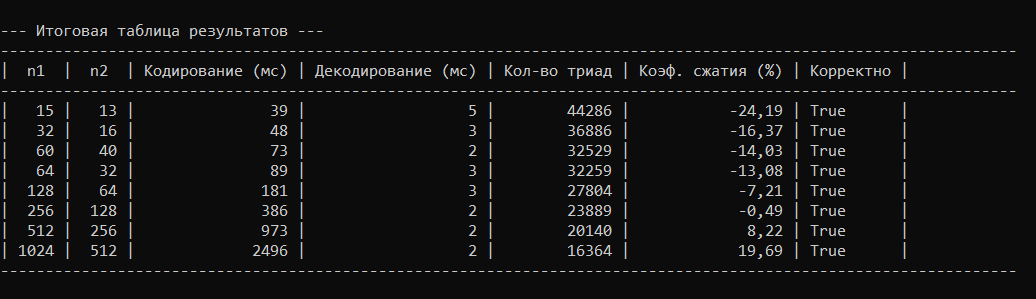


Рисунок 3.2 – Подсчёт эффективности

**Вывод**: в ходе лабораторной работы были успешно реализованы и протестированы основные механизмы алгоритма сжатия данных LZ77. Проведенный анализ влияния размеров скользящего окна позволил выявить зависимости между параметрами алгоритма, скоростью его работы и достигаемой степенью сжатия. Было продемонстрировано, что выбор оптимальных размеров окон зависит от характеристик сжимаемых данных. Полученные практические навыки и теоретические знания могут быть использованы для дальнейшего изучения и применения методов сжатия информации.