Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Защита информации и надёжность информационных систем

Студент: Лопатнюк П.В.

ФИТ 3 курс 1 группа

Преподаватель: Нистюк О.А.

Минск 2025

# Лабораторная работа № 10

# СЖАТИЕ/РАСПАКОВКА ДАННЫХ МЕТОДОМ ЛЕМПЕЛЯ − ЗИВА

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

**Задачи**:

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

# Теоретические сведения

Скользящее окно (Sliding Window): Алгоритм использует концепцию «скользящего окна», которое перемещается по входному потоку данных. Это окно разделено на две части:

1. Окно поиска (Search Buffer, словарь): Содержит N1 уже обработанных символов, предшествующих текущей позиции. В этом буфере производится поиск совпадений. В начале процесса кодирования это окно обычно заполнено символами по умолчанию (например, нулями или специальным символом, не встречающимся в данных).
2. Буфер упреждения (Lookahead Buffer): Содержит N2 следующих символов из входного потока, которые подлежат кодированию.

Процесс кодирования:

Для текущего буфера упреждения ищется самое длинное совпадение его префикса в окне поиска.

Если совпадение найдено, оно кодируется в виде триады (P, Q, S):

* P (смещение, offset): Расстояние (количество символов) назад от текущей позиции (начала буфера упреждения) до начала найденного совпадения в окне поиска. Если совпадения нет, P=0.
* Q (длина, length): Длина найденного совпадения. Если совпадения нет, Q=0. Длина Q ограничена размером буфера упреждения N2 (в некоторых вариантах, как в данной лабораторной, Q может быть ограничено N2-1, чтобы всегда оставался символ для S в буфере упреждения).
* S (следующий символ, symbol): Первый символ в буфере упреждения, следующий после найденного совпадения (длиной Q). Если совпадения не было (Q=0), то S — это просто первый символ буфера упреждения. Если буфер упреждения исчерпан, S может быть специальным символом конца потока (EOF).

Окно поиска сдвигается на Q+1 символов вправо (т.е. Q символов совпадения и символ S считаются обработанными и перемещаются из буфера упреждения в окно поиска).

Процесс повторяется до тех пор, пока весь входной поток не будет закодирован.

Разрешение перекрывающихся копий (Self-referencing/Overlapping copies): Некоторые варианты LZ77, включая тот, что рассматривается в данной работе, позволяют смещению P указывать на позицию в окне поиска таким образом, что длина совпадения Q оказывается больше, чем расстояние от этой позиции до конца окна поиска. Это означает, что часть копируемой последовательности берется из уже скопированной на этом же шаге части. Это позволяет эффективно кодировать повторяющиеся паттерны (например, ‘ababab…’).

Процесс декодирования:

Декодер также поддерживает скользящее окно поиска, инициализированное теми же символами по умолчанию.

При получении триады (P, Q, S):

1. Если Q > 0, из окна поиска копируется Q символов, начиная с позиции, на которую указывает смещение P (на P символов назад от текущего конца окна). При этом также обрабатываются случаи самокопирования, когда P < Q.
2. К скопированной последовательности добавляется символ S.
3. Восстановленные Q+1 символов добавляются в конец окна поиска декодера, и окно сдвигается.

Процесс продолжается до конца закодированного потока (обычно сигнализируемого специальной EOF-триадой, например, (0,0,EOF\_char)).

# Практическое задание

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

Эта структура представляет собой элементарную единицу закодированной информации в LZ77. Она хранит смещение P, длину совпадения Q и следующий за совпадением символ S. Метод ToString() используется для удобного вывода триады в консоль.

|  |
| --- |
| public struct Triplet  {  public int P { get; } // Смещение НАЗАД от текущей позиции (1-based). P=0 если нет совпадения.  public int Q { get; } // Длина совпадения. Q=0 если нет совпадения.  public char S { get; } // Следующий символ.  public Triplet(int p, int q, char s)  {  P = p;  Q = q;  S = s;  }  public override string ToString()  {  // ... (форматирование для вывода) ...  }  } |

Листинг 1.1 – Структура Triplet

Этот метод реализует процесс кодирования входной строки inputText. Инициализирует окно поиска (searchBuffer) символами по умолчанию.

В цикле обрабатывает входную строку:

* Формирует буфер упреждения (lookaheadBuffer).
* Ищет наиболее длинное совпадение (bestMatchLengthQ) префикса lookaheadBuffer с последовательностью, начинающейся на bestMatchOffsetP позиций назад в searchBuffer (с разрешением «залезания» в lookaheadBuffer для удлинения совпадения). Максимальная длина Q ограничена так, чтобы символ S можно было взять из lookaheadBuffer (если N2>0, то Q <= N2-1, иначе Q=0).
* Определяет следующий символ S (первый символ после найденного совпадения или первый символ буфера упреждения, если совпадения нет).
* Формирует триаду (P, Q, S) и добавляет ее в список результатов.
* Сдвигает окно поиска (searchBuffer) и указатель текущей позиции во входной строке (currentInputPosition) на Q+1 символов.

Процесс завершается, когда вся входная строка обработана, или когда сгенерирована специальная EOF-триада.

|  |
| --- |
| public List<Triplet> Encode(string inputText)  {  // ... (инициализация searchBuffer, encodedTriplets, currentInputPosition, step) ...  while (currentInputPosition < inputText.Length)  {  // ... (формирование lookaheadBuffer) ...  int bestMatchOffsetP = 0;  int bestMatchLengthQ = 0;  // Определение максимальной длины Q для поиска (Q\_max = N2-1 или N2)  int max\_q\_for\_search = 0;  if (lookaheadActualSize > 0 && lookaheadBufferSizeN2 > 0) {  max\_q\_for\_search = lookaheadActualSize -1; // Для правила Q\_max = N2-1  // или max\_q\_for\_search = lookaheadActualSize; // Для правила Q\_max = N2  }  // ... (проверка на <0 ) ...  if (lookaheadBuffer.Length > 0 && max\_q\_for\_search >= 0 ) // Условие для поиска  {  // Цикл по всем возможным смещениям P назад в searchBuffer  for (int p\_offset\_1\_based = 1; p\_offset\_1\_based <= searchBuffer.Length; p\_offset\_1\_based++)  {  int current\_q = 0;  int sb\_match\_start\_index = searchBuffer.Length - p\_offset\_1\_based;  // Цикл наращивания длины совпадения Q, с учетом «залезания в буфер«  for (int k = 0; k < Math.Min(max\_q\_for\_search, lookaheadActualSize) ; k++) // или просто k < lookaheadActualSize если Q\_max=N2  {  char charFromSourcePattern;  if (k < p\_offset\_1\_based) { // Источник в searchBuffer  charFromSourcePattern = searchBuffer[sb\_match\_start\_index + k];  } else { // Источник в lookaheadBuffer (самокопирование)  charFromSourcePattern = lookaheadBuffer[k - p\_offset\_1\_based];  }    if (lookaheadBuffer[k] == charFromSourcePattern) { current\_q++; }  else { break; }  }  // Обновление bestMatchLengthQ и bestMatchOffsetP  // ... (логика выбора лучшего P при равных Q, если требуется) ...  }  }  // ... (определение symbolS и symbolsToShift) ...  // ... (добавление триады в encodedTriplets) ...  // ... (обновление searchBuffer и currentInputPosition) ...  // ... (условия выхода из цикла) ...  }  return encodedTriplets;  } |

Листинг 1.2 – Метод Encode

Метод выполняет обратное преобразование — восстановление исходного текста из списка триад. Инициализирует окно поиска (searchBuffer) символами по умолчанию.

В цикле обрабатывает каждую триаду (P, Q, S):

* Если Q > 0 (есть совпадение), из searchBuffer копируется Q символов, начиная с позиции, удаленной на P символов назад от текущего конца searchBuffer. Логика копирования по одному символу с немедленным добавлением в searchBuffer позволяет корректно обрабатывать самоссылающиеся последовательности.
* Символ S из триады добавляется к восстановленному тексту.
* Все восстановленные символы (скопированная часть и S) добавляются в searchBuffer, который затем сдвигается для поддержания своего размера.
* Процесс завершается после обработки всех триад или при обнаружении специальной EOF-триады.

|  |
| --- |
| public string Decode(List<Triplet> encodedTriplets)  {  // ... (инициализация decodedText, searchBuffer, step) ...  for (int tripletIndex = 0; tripletIndex < encodedTriplets.Count; tripletIndex++)  {  Triplet triplet = encodedTriplets[tripletIndex];  int p = triplet.P;  int q = triplet.Q;  char s = triplet.S;  // Обработка специальной EOF-триады (0,0,defaultFillChar) как последней  // ... (условие выхода) ...  if (q > 0 && p > 0) // Копируем, если есть длина и валидное смещение  {  // Цикл для копирования Q символов, обрабатывающий самокопирование  for (int i = 0; i < q; i++)  {  char charToCopy = searchBuffer[searchBuffer.Length - p];  decodedText.Append(charToCopy);  searchBuffer.Append(charToCopy);  if (searchBuffer.Length > dictionarySizeN1)  {  searchBuffer.Remove(0, 1);  }  }  }    // Добавляем символ S, если это не специальный маркер конца на последней триаде  // ... (логика добавления S и обновления searchBuffer) ...  }  return decodedText.ToString();  } |

Листинг 1.3 – Метод Decode

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

В данном разделе демонстрируется пошаговая работа алгоритма LZ77 на примере кодирования и декодирования тестовой строки.

**Параметры алгоритма для демонстрации:**

Входная строка (S): 2000302013020130313031303130313333333 (длина 37 символов)

Размер окна поиска (N1): 15 символов

Размер буфера упреждения (N2): 13 символов

Символ по умолчанию для окна поиска: '0' (ноль)

Правило для максимальной длины совпадения Q: Q не может превышать N2-1, чтобы всегда оставался как минимум один символ в буфере упреждения для S. Таким образом, max\_Q = 13 - 1 = 12.

Начальное состояние:

Окно поиска (Search Buffer): Заполнено 15 символами '0'.  
 [000000000000000]

Входная строка: 2000302013020130313031303130313333333

**Шаг 1:** Окно поиска: [000000000000000]

Буфер упреждения (Lookahead): [2000302013020] (Остаток: 130313031303130313333333)

Поиск совпадения: Символ '2' (первый из буфера упреждения) не найден в окне поиска, состоящем из нулей.

Результат: Совпадения нет (P=0, Q=0). Следующий символ S='2'.

Выведена триада: (0,0,'2')

Обновление: Окно поиска сдвигается на Q+1 = 1 символ. Символ '2' добавляется в окно.

**Шаг 2:** Окно поиска: [000000000000002]

Буфер упреждения: [0003020130201] (Остаток: 30313031303130313333333)

Поиск совпадения: Ищем префикс буфера упреждения («000...») в окне поиска. Найдена последовательность «000».

Эта последовательность «000» имеется в начальных нулях окна поиска. Выбирается самое «свежее» (ближайшее к буферу упреждения) или по другому правилу лекции. В данном случае, «000» соответствует части searchBuffer[15-4 ... 15-4+3-1] = searchBuffer[11..13].

**Результат**: Найдено совпадение «000» (Q=3). Смещение P=4 (на 4 символа назад от конца текущего окна поиска, т.е. 0000000000[000]02). Следующий символ S='3' (четвертый символ буфера упреждения).

Выведена триада: (4,3,'3')

Обновление: Окно поиска сдвигается на Q+1 = 4 символа. Символы «0003« добавляются в окно.

**Шаг 3:** Окно поиска: [000000000020003]

Буфер упреждения: [0201302013031] (Остаток: 3031303130313333333)

Поиск совпадения: Ищем «020...». Найдено «020».

searchBuffer[15-6 ... 15-6+3-1] = searchBuffer[9..11]. Окно 000000000[020]003.

**Результат**: Совпадение «020» (Q=3). Смещение P=6. Следующий символ S='1'.

Выведена триада: (6,3,'1')

Обновление: Окно сдвигается на 4 символа («0201»).

**Шаг 4:** Окно поиска: [000000200030201]

Буфер упреждения: [3020130313031] (Остаток: 303130313333333)

Поиск совпадения: Ищем «3020130...». Найдено «3020130».

searchBuffer[15-5 ... 15-5+7-1] = searchBuffer[10..16] (но окно 15 символов, значит searchBuffer[10..14]). Окно 0000002000[30201]. И далее самокопирование из буфера упреждения для символов '3' и '0'.

Результат: Совпадение «3020130» (Q=7). Смещение P=5. Следующий символ S='3'.

Выведена триаada: (5,7,'3')

Обновление: Окно сдвигается на 8 символов («30201303»).

**Шаг 5:** Окно поиска: [003020130201303]

Буфер упреждения: [1303130313031] (Остаток: 3333333)

Поиск совпадения: Ищем «130313031303...». Найдено «130313031303».

Длина Q ограничена N2-1 = 13-1 = 12.

Смещение P=4 (назад от конца окна ...0201303 указывает на searchBuffer[11]='1'). Копирование продолжается за счет уже скопированных символов из буфера упреждения (самокопирование).

Результат: Совпадение «130313031303» (Q=12). Смещение P=4. Следующий символ S='1' (13-й символ буфера упреждения).

Выведена триада: (4,12,'1')

Обновление: Окно сдвигается на 13 символов («1303130313031«).

**Шаг 6:** Окно поиска: [031303130313031]

Буфер упреждения: [3333333 ] (Остаток: [])

Поиск совпадения: Ищем «3...». Найдено «3».

searchBuffer[15-2]='3'.

Результат: Совпадение «3» (Q=1). Смещение P=2. Следующий символ S='3'.

Выведена триада: (2,1,'3')

Обновление: Окно сдвигается на 2 символа («33«).

**Шаг 7:** Окно поиска: [130313031303133]

Буфер упреждения: [33333 ] (Остаток: [])

Поиск совпадения: Ищем «3333...». Найдено «3333».

searchBuffer[15-1]='3'. Далее самокопирование.

Результат: Совпадение «3333» (Q=4). Смещение P=1. Следующий символ S='3'.

Выведена триада: (1,4,'3')

Обновление: Окно сдвигается на 5 символов («33333»). Вся входная строка обработана. Поскольку следующего символа нет, последняя триада для обозначения конца потока должна быть (0,0, '0' [defaultFillChar]), но так как символ S был взят из строки, то этого не происходит, и алгоритм завершается. (Примечание: в коде есть логика добавления (0,0,EOF), если последний S был defaultFillChar из-за конца строки).

Итоговая последовательность триад:  
(0,0,'2') (4,3,'3') (6,3,'1') (5,7,'3') (4,12,'1') (2,1,'3') (1,4,'3')  
 Всего триад: 7

# Процесс декодирования

Начальное состояние:

Окно поиска (Search Buffer): Заполнено 15 символами '0'.  
[000000000000000]

Закодированные триады: (0,0,'2') (4,3,'3') (6,3,'1') (5,7,'3') (4,12,'1') (2,1,'3') (1,4,'3')

Восстановленный текст: (пусто)

**Шаг 1:**

Принята триада: (0,0,'2')

P=0, Q=0: Копирования из словаря нет.

Символ S='2' добавляется в текст и в окно поиска.

Восстановлено: «2»

Окно поиска: [000000000000002]

**Шаг 2:**

Принята триада: (4,3,'3')

P=4, Q=3: Копируем 3 символа, начиная с 4-й позиции сзади в текущем окне (0000000000[000]02). Копируется «000».

Символ S='3' добавляется.

Восстановлено: «2» + «000» + «3» = «20003»

Окно поиска: [000000000020003] (сдвинулось на «0003»)

**Шаг 3:**

Принята триада: (6,3,'1')

P=6, Q=3: Копируем 3 символа, начиная с 6-й позиции сзади (000000000[020]003). Копируется «020».

Символ S='1' добавляется.

Восстановлено: «20003» + «020» + «1» = «200030201»

Окно поиска: [000000200030201]

**Шаг 4:**

Принята триада: (5,7,'3')

P=5, Q=7: Копируем 7 символов, начиная с 5-й позиции сзади (0000002000[30201], затем самокопирование «30»). Копируется «3020130».

Символ S='3' добавляется.

Восстановлено: «200030201» + «3020130» + «3» = «20003020130201303»

Окно поиска: [003020130201303]

**Шаг 5:**

Принята триада: (4,12,'1')

P=4, Q=12: Копируем 12 символов, начиная с 4-й позиции сзади (00302013020[1303], затем самокопирование «13031303»). Копируется «130313031303».

Символ S='1' добавляется.

Восстановлено: «...1303» + «130313031303» + «1» = «...13031303130313031»

Окно поиска: [031303130313031]

**Шаг 6:**

Принята триада: (2,1,'3')

P=2, Q=1: Копируем 1 символ, с 2-й позиции сзади (0313031303130[3]1). Копируется «3».

Символ S='3' добавляется.

Восстановлено: «...3031» + «3» + «3» = «...303133»

Окно поиска: [130313031303133]

**Шаг 7:**

Принята триада: (1,4,'3')

P=1, Q=4: Копируем 4 символа, с 1-й позиции сзади (13031303130313[3], затем самокопирование «333»). Копируется «3333».

Символ S='3' добавляется.

Восстановлено: «...3133» + «3333» + «3» = «...313333333»

Окно поиска: [313031303133333]

Конец декодирования.  
 Итоговый восстановленный текст: 2000302013020130313031303130313333333  
 Корректность: Восстановленный текст полностью совпадает с исходным.

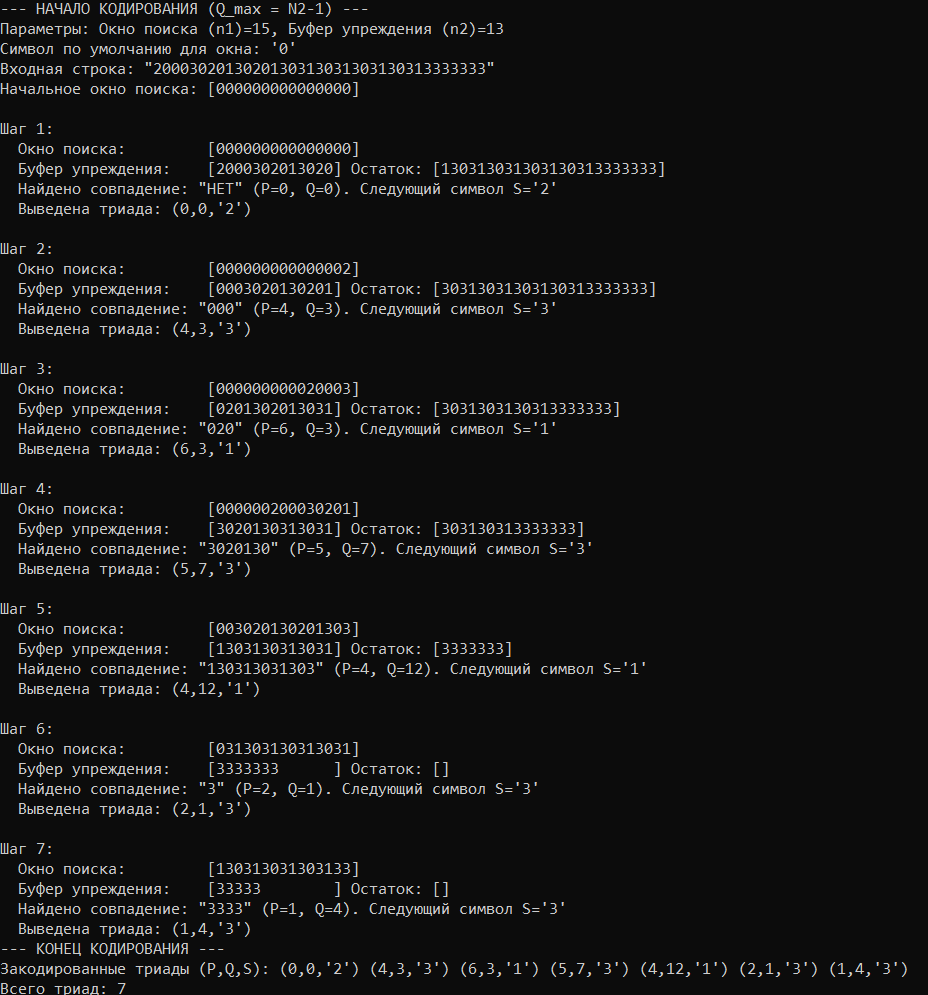


Рисунок 2.1 – Пример прямого преобразования в консоли

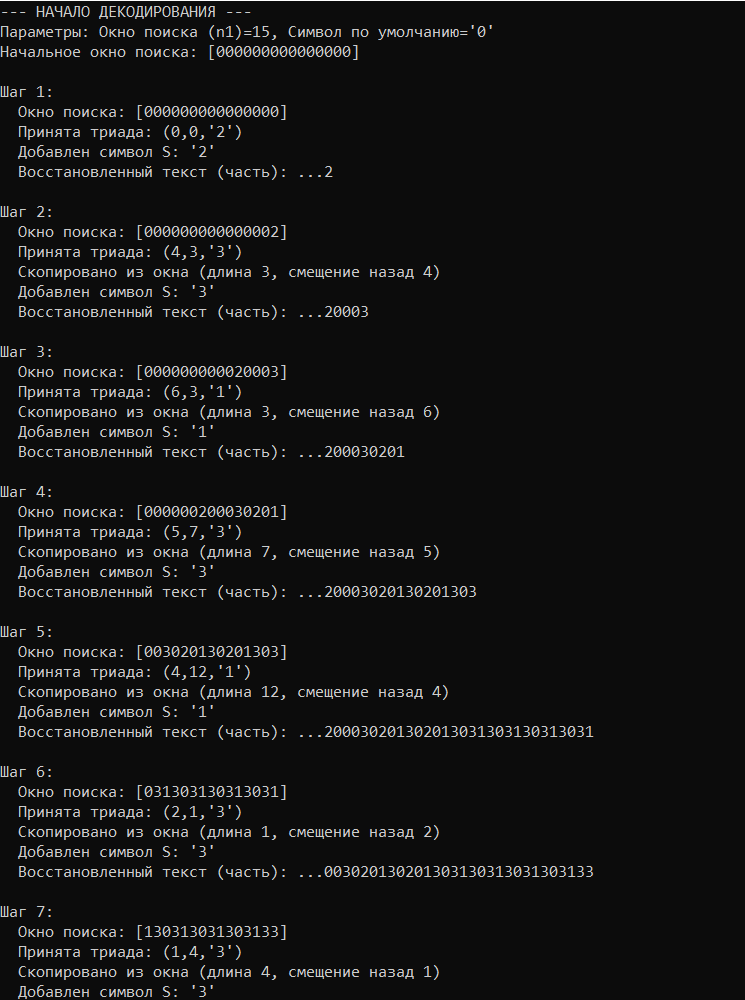


Рисунок 2.2 – Пример обратного преобразования в консоли

1. **Изменяя размеры окон, оценить скорость и эффективность (используя соотношения на с. 76) выполнения операций сжатия/распаковки.**

При маленьких размерах окон (15,13; 32,16 и т.д.) вы получаете отрицательный коэффициент сжатия. Это означает, что сжатый файл больше исходного. Это ожидаемо, так как накладные расходы на хранение триад для коротких или редких совпадений превышают экономию.

По мере увеличения размеров окон (N1 и N2) коэффициент сжатия улучшается. При N1=256, N2=128 он все еще отрицательный, но уже ближе к нулю.

При N1=512, N2=256 вы впервые получаете положительное сжатие (4,69%). Это значит, что окна стали достаточно большими, чтобы находить достаточно длинные и/или частые повторения, которые оправдывают представление в виде триад.

При N1=1024, N2=512 сжатие еще лучше (16,82%).

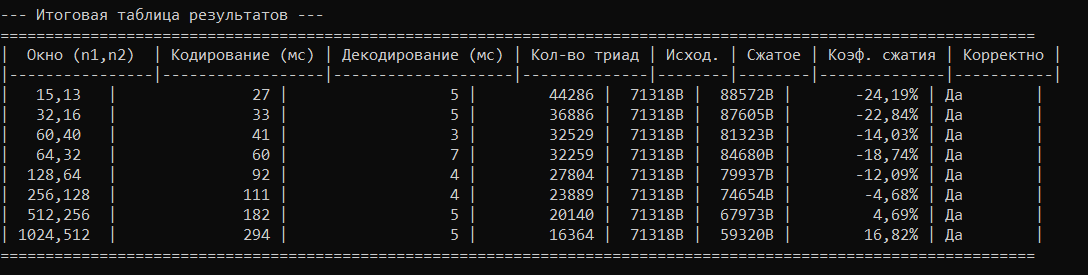


Рисунок 3.2 – Подсчёт эффективности

# Вывод

Алгоритм LZ77 демонстрирует эффективность сжатия, зависящую от выбранных размеров окна поиска (N1) и буфера упреждения (N2), а также от структуры входных данных. Для тестовых данных (размером 71318 байт) было показано, что при малых размерах окон (например, N1=15, N2=13) происходит расширение данных (коэффициент сжатия отрицательный, до -24,19%), так как накладные расходы на кодирование коротких или редких совпадений в виде триад превышают экономию от их замены. С увеличением размеров N1 и N2 алгоритм получает возможность находить более длинные и/или более удаленные повторяющиеся последовательности. Это приводит к улучшению коэффициента сжатия, который становится положительным при размерах окон N1=512, N2=256 (4,69%) и достигает 16,82% при N1=1024, N2=512. Данная тенденция показывает, что для достижения эффективного сжатия методом LZ77 необходимо подбирать достаточно большие размеры буферов, чтобы алгоритм мог использовать присущую данным избыточность. Однако, слишком большие окна могут привести к увеличению времени обработки и потребления памяти.