Санкт-Петербургский политехнический университет Петра Великого

Кафедра прикладной математики и информатики

**Дискретная математика**

Отчет по лабораторной работе № 1

Тема "Кодирование"

**Вариант 3 – «Алгоритм Лемпеля-Зива»**

Выполнил студент гр. 5030102/20202 Тишковец С. Е.

Санкт-Петербург

2024

**Оглавление**

[**Постановка задачи** 3](#_Toc178540353)

[**Язык программирования** 3](#_Toc178540354)

[**Описание алгоритма** 3](#_Toc178540355)

[**Демонстрация алгоритма** 5](#_Toc178540356)

[**Результаты сравнения сжатия с размером исходных данных** 7](#_Toc178540357)

[**Область применимости алгоритма** 8](#_Toc178540358)

[**Формат данных** 8](#_Toc178540359)

[**Источники информации** 8](#_Toc178540360)

# **Постановка задачи**

*Вариант 3 – Алгоритм Лемпеля-Зива*

Поддержать возможность вывода словаря на экран при сжатии и распаковки информации для отслеживания правильности работы алгоритма.

В отчёте сравнить результаты сжатия с размером оригинальных данных на текстах разной длины и сделать выводы об эффективности алгоритма.

# **Язык программирования**

Алгоритм был реализован с помощью **C++**

# **Описание алгоритма**

Алгоритм кодирования:

1. Чтение данных из файла формата .txt
2. Инициализация словаря - создается словарь, который сопоставляет строки (одиночные символы) кодам. Заполняем его символами ASCII (все символы от 0 до 255) и их индексами.
3. Проход по всем символам входной строки:

- Проверка следующего символа – если это не последний символ, добавляем следующий символ к переменной *next*. Это подготавливает пару для проверки в словаре.

- Проверка в словаре – проверяем, существует ли комбинация *current* + *next* в словаре. Если да, обновляем*current*, добавив *next.*

- Обработка, если комбинация не найдена – добавляем код текущей строки и новую комбинацию в словарь с новым кодом.

- Подготовка к следующей итерации – очищаем *next*, чтобы в следующей итерации начать заново.

1. Если в конце осталась какая-то строка в *current*, добавляем её код в *output\_code*.
2. Вывод на экран и запись в файл получившихся кодов.

Алгоритм декодирования:

1. Чтение данных из файла формата .txt
2. Инициализация словаря - создается словарь, который сопоставляет строки (одиночные символы) кодам. Заполняем его символами ASCII (все символы от 0 до 255) и их индексами.
3. Начало декодирования с первого кода. Печать декодированного символа.
4. Проход по оставшимся кодам:

- Начало прохода со второго кода.

- Проверка следующего кода – проверяем, существует ли следующий код в словаре. Если нет, строим строку из предыдущей строки *old* и первого символа *c.* Если следующий код найден, просто получаем соответствующую строку из словаря и выводим её.

- Подготовка к созданию новой комбинации - Обновляем *c*, присваивая ему первый символ текущей строки. Затем создаём новую комбинацию и добавляем её в словарь. Наконец, обновляем переменную*old* для следующей итерации.

1. Вывод на экран и запись в файл декодированного сообщения.

Основная функция:

1. Вывод на экран сообщения о том, что пользователь может выбрать одну из двух операций.
2. Вызов функций для кодирования или декодирования строки (смотря, что выбрал пользователь).
3. Вывод на экран сообщения о том, что пользователь может вывести на экран словарь.
4. Вывод словаря или завершение программы (смотря, что выбрал пользователь).

# **Демонстрация алгоритма**

Пример 1. Кодирование

Допустим, исходный словарь состоит из четырех букв **A, B, C, D** и пробела

Получено сообщение: **ABCD\_ABC**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Шаги** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| **Текст** |  |  |  |  |  | **A** | **B** | **C** | **D** | **\_** | **A** | **B** | **C** |
| **Код** | 0 | 1 | 2 | 3 | 4 |  |  |  |  |  |  |  |  |
| **Словарь** | **\_** | **A** | **B** | **C** | **D** |  |  |  |  |  |  |  |  |

Во второй строке — текст, который надо закодировать, в третьей — код.

Начинаем кодировать текст, одновременно пополняя словарь:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Шаги** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| **Текст** |  |  |  |  |  | **A** | **B** | **C** | **D** | **\_** | **A** |
| **Код** | 0 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 0 |  |
| **Словарь** | **\_** | **A** | **B** | **C** | **D** | **AB** | **BC** | **CD** | **D\_** | **\_A** |  |

Словарь пополняется словами, состоящими из пар соседних букв, и эти слова нумеруются числами верхней строки. До сего момента собственно кодирование идет побуквенно: каждая буква заменяется на соответствующее число из кодировки алфавита. Однако следующий шаг немного отличается.

Если бы мы действовали согласно предшествующему алгоритму, то должны были бы занести в словарь **AB**, но это слово уже есть в словаре под номером **5.** Подставляем вместо него его код:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Шаги** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| **Текст** |  |  |  |  |  | **A** | **B** | **C** | **D** | **\_** | **AB** | **C** |
| **Код** | 0 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 0 | 5 | 3 |
| **Словарь** | **\_** | **A** | **B** | **C** | **D** | **AB** | **BC** | **CD** | **D\_** | **\_A** | **ABC** |  |

В словарь заносим слово, получаемое присоединением к **AB** следующего символа текста, т.е. **C**. По мере увеличения словаря длина кода строки из словаря тоже растет, чтобы учесть новые элементы. 3-х битные группы дают 8 возможных комбинаций. Поэтому, когда в словаре появляется 9-ое слово, алгоритм переходит к 4-х битным группам.

Получили закодированное сообщение **1 2 3 4 0 5 3**

Пример 2. Декодирование

Получили закодированное сообщение **1 2 3 4 0 5 3**

Стартовое состояние: известна кодировка базового словаря (алфавита). Это позволяет декодировать начальный отрезок закодированного текста:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Шаги** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| **Текст** |  |  |  |  |  | **A** | **B** | **C** | **D** | **\_** |  |
| **Код** | 0 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 0 |  |
| **Словарь** | **\_** | **A** | **B** | **C** | **D** |  |  |  |  |  |  |

Параллельно процессу декодирования запускается процесс заполнения (восстановления) словаря. После того как произошло декодирование первых двух чисел в буквы, из этих букв формируется новое слово и помещается в словарь.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Шаги** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| **Текст** |  |  |  |  |  | **A** | **B** | **C** | **D** | **\_** |  |
| **Код** | 0 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 0 |  |
| **Словарь** | **\_** | **A** | **B** | **C** | **D** | **AB** |  |  |  |  |  |

Номер этого слова **6**. Декодировав следующее число кодировки в букву **C** мы снова образуем словарное слово, подсоединив **C** к **B**:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Шаги** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| **Текст** |  |  |  |  |  | **A** | **B** | **C** | **D** | **\_** |  |
| **Код** | 0 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 0 |  |
| **Словарь** | **\_** | **A** | **B** | **C** | **D** | **AB** | **BC** |  |  |  |  |

Дальнейшая процедура протекает аналогично вплоть до вот этого момента:

Следующее поступившее на вход процесса число **5** не входит в базовый алфавит. Но, однако же, оно уже занесено в разворачиваемый словарь: ему соответствует слово **AB**:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Шаги** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| **Текст** |  |  |  |  |  | **A** | **B** | **C** | **D** | **\_** | **AB** | **C** |
| **Код** | 0 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 0 | 5 | 3 |
| **Словарь** | **\_** | **A** | **B** | **C** | **D** | **AB** | **BC** | **CD** | **D\_** | **\_A** | **ABC** |  |

Процесс продолжается до тех пор, пока не будет раскодирован последний код **3**. Получили декодированное сообщение: **ABCD\_ABC**

# **Результаты сравнения сжатия с размером исходных данных**

Допустим, исходный словарь состоит из четырех букв **A, B, C, D** и пробела

*AABCD AABCD ABCDAB A B C D A B CDA ABCDABCD ABCD A B A A B C D DAA B C D A C B A D C B A A D B C A B A C D ABCDA B C D ABCD A B C D A B C D A B C D A B C D AB A B C D A B C D A B CD A C D A B C D A B C D A A B C D A C D A B D A B C D A B C D A B C AABCD AABCD ABCDAB A B C D A B CDA ABCDABCD ABCD A B A A B C D DAA B C D A C B A D C B A A D B C A B A C D ABCDA B C D ABCD A B C D A B C D A B C D A B C D AB A B C D A B C D A B CD A C D A B C D A B C D A A B C D A C D A B D A B C D A B C D A B C ABCD*

Текст, содержащий 500 символов может быть закодирован

8 битов на один символ × 500 символов = 4000 битами.

Алгоритм **LZW** дал 133 слова в словарь, стоимость кодирования

133 символа × 9 битов = 1197 бит.

1197/4000 ≈ 0.29, т.е. получили 71% экономии.

Пояснения: длина кодов слов из словаря, начиная с 256-го и до 511-го символа, имеет длину 9 бит.

Более длинный кусок, содержащий 1000 символов, дал 213 слов в словарь, стоимость кодирования

213 символов × 9 битов = 1917 бит.

1917/(1000×8) ≈ 0.23, т.е. эффективность сжатия порядка 77%.

С увеличением количества символов эффективность увеличивается, особенно если в тексте много повторяющихся комбинаций символов (слов).

# **Область применимости алгоритма**

1. Область применения:

На момент своего появления алгоритм LZW давал лучший коэффициент сжатия для большинства приложений, чем любой другой хорошо известный метод того времени. Он стал первым широко используемым на компьютерах методом сжатия данных. LZW широко используется и поддерживается в различных программных приложениях и операционных системах, что делает его популярным выбором для сжатия и распаковки.

1. Случаи сбоя:

- Сжатие LZW требует значительного объема памяти для поддержания словаря сжатия. В случае недостатка памяти добавление новых символов в словарь закончится и код выполнится некорректно.

- Если при кодировании или декодировании изначально неправильно инициализировать словарь (например, не все возможные символы алфавита изначально добавлены), это может привести к ошибкам при обработке данных.

- Если входные данные содержат символы, которые не предполагаются в инициализированном словаре, это может привести к сбою в работе алгоритма.

# **Формат данных**

Входные данные: текст для кодирования считывается из файла и должен содержать в себе только те символы, которые проинициализированы в словаре.

Выходные данные: декодированный текст представляет собой набор чисел, которые переводятся в двоичную систему счисления. Последовательность из единиц и нулей разбивается на «восьмерки», тем самым формируя полные байты (8 битов). Один байт представляет собой один символ *char,* а последовательность из этих *char*’ов сохраняется в выходном файле, который создается после декодирования.

# **Источники информации**

* <https://www.geeksforgeeks.org/lzw-lempel-ziv-welch-compression-technique/>
* <https://habr.com/ru/articles/132683/>
* <https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_LZW>