**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра САПР**

**ОТЧЁТ**

**по лабораторной работе №3**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: Алгоритмы сжатия без потерь**

Студент гр. 0322 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Шаронин А.Д.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Пестерев Д.О.

Санкт-Петербург

2023

# Задача

Реализовать следующие алгоритмы сжатия символьных данных:

1. Алгоритм Хаффмана (HA)
2. Кодирование длин серий (RKE)
3. Алгоритм Лемпеля-Зива (LZ78)
4. *Преобразование Барроуза-Уиллера (BWT)*
5. *MTF*
6. Арифметическое кодирование (AC)
7. PPM

# Описание реализуемых классов и структур

void Huffman\_alg() — Алгоритм Хаффмана

RLE();//Кодирование длин серий (RLE)

LZ78();//Алгоритм Лемпеля-Зива (LZ78)

BWT();//Преобразование Барроуза-Уиллера (BWT)

MTF();//Преобразование Move-to-Forward

ArithmeticCoding();//Арифметическое кодирование (AC)

**Сравнение временных затрат**

Не все алгоритмы получилось реализовать, так что будем сравнивать только те, которые реально работают, а именно: HA, RLE, LZ78. Преобразования BWT и MTF также работают, но если BWT ещё может помочь при дальнейшем использовании RLE, то как и где может помочь MTF, я не очень понимаю, вероятно я его не до конца сделал или не до конца понял. Сравнивать будем алгоритмы сжатия, а RLE ещё и с использованием BWT. В качестве датасета будем брать enwik8, но для сравнения RLE с использованием BWT будем использовать маленький текст, так как даже при 10% enwik8, BWT потребляет огромное количество памяти (n\*n байт минимум, где n – длина изначального текста). Полученные результаты приведены в таблице ниже.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Изначальный  объём | Итоговый  объём | Процент  сжатия | Время  сжатия |
| HA | 101 128 023 байт | 64 001 110 байт | 63.29% | 400120mils |
| RLE | 110 601 638 байт | 109.38% | 34821.7mils |
| LZ78 | 93 198 674 байт | 92.16% | 490057mils |

Как видно из таблицы алгоритм Хаффмана лучше всех сжимает файлы, но делает это очень долго. Алгоритм LZ78 делает это ещё дольше, а файлы сжимает гораздо хуже. Кодирование длин серий вообще закодировал больше чем было, но быстро, к его чести можно сказать, что чем больше файл, и чем больше там повторяющихся друг за другом символов, тем лучше он сжимает, что мы и продемонстрируем во второй таблице ниже.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Изначальный  объём | Итоговый  объём | Процент  сжатия | Время  сжатия |
| HA | 65 514 байт | 35 848 байт | 54.72% | 257.655mils |
| BWT—>RLE | 6 864 байт | 10.48% | 16.0154mils(RLE) |
| LZ78 | 60 952 байт | 93.04% | 223.694mils |

Как показывают данные, использование преобразования BWT сильно помогает кодированию длин серий при сжатии исходного текста, хотя по времени всё ровно наоборот, так как само преобразование занимает много времени и памяти.

**Пример работы программы**

При запуске программы нас встречает меню, по которому можно передвигаться стрелочками и выбрать нужный алгоритм (Рис. 1).

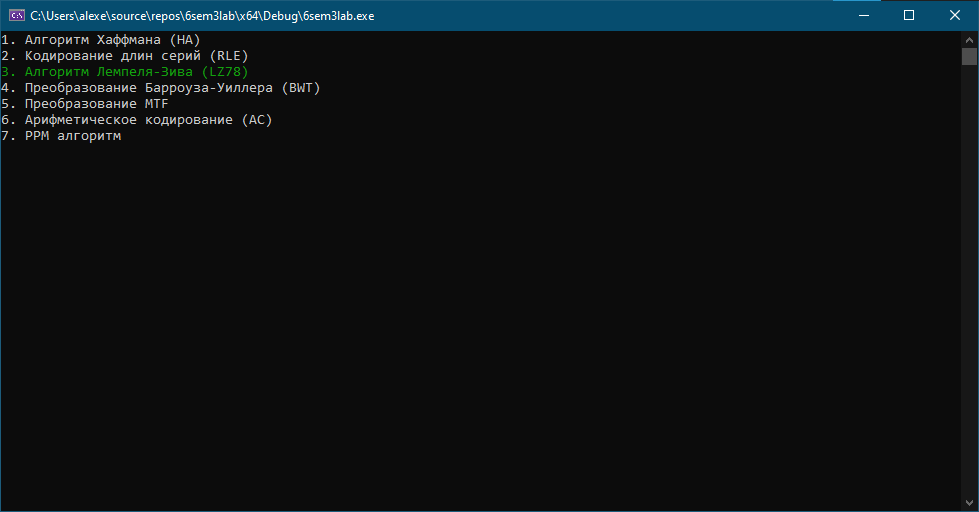


Рисунок 1 — Главное меню

Выбрав интересующий нас алгоритм, нажимаем Enter и попадаем в меню, где можно выбрать способ ввода текста (Рис. 2)

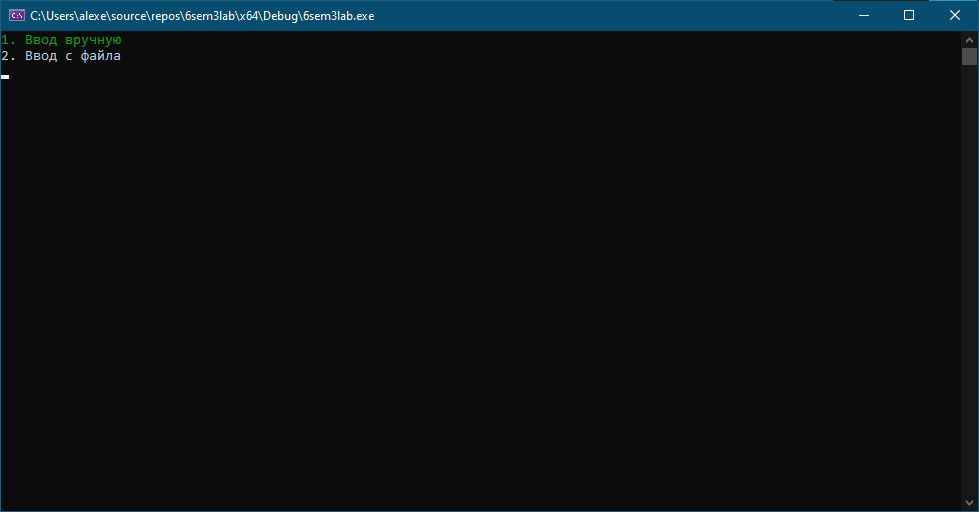


Рисунок 2 — Выбор способа ввода текста

При выборе «Ввод вручную» нам надо просто вписать текст вручную. При выборе «Ввод с файла» нам до указать либо путь до файла, включая имя файла с расширением, либо, если файл в папке с программой, указать только имя файла с расширением (Рис. 3).

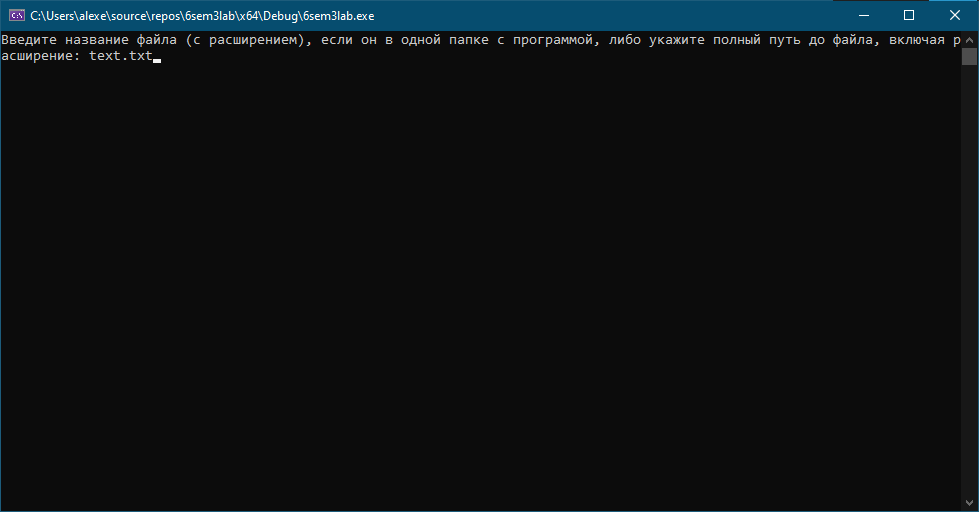


Рисунок 3 — Ввод названия файла

Подтверждаем ввод клавишей Enter и программа кодирует текст в файл output.bin (Рис. 4).

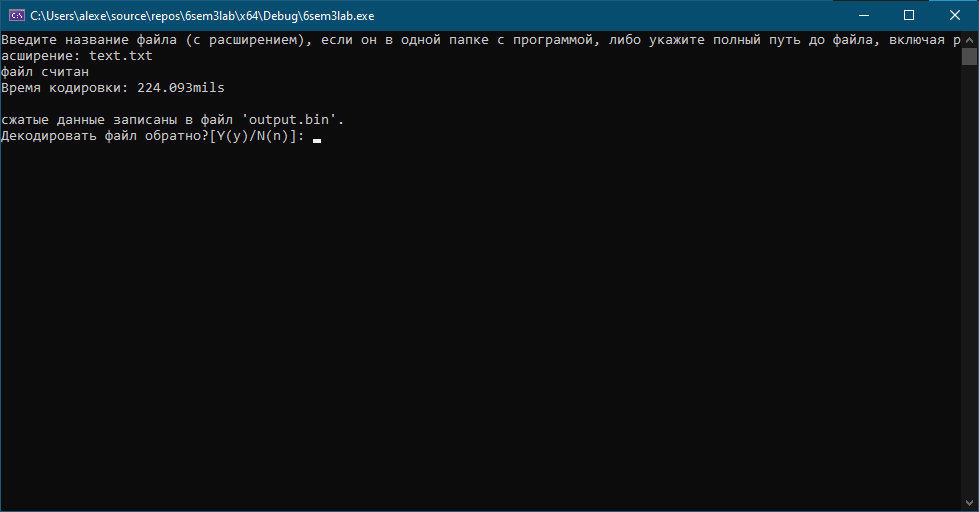


Рисунок 4 — кодирование текста

Дальше мы можем либо обратно раскодировать закодированный текст, либо отказаться от этого, мы раскодируем (Рис. 5).

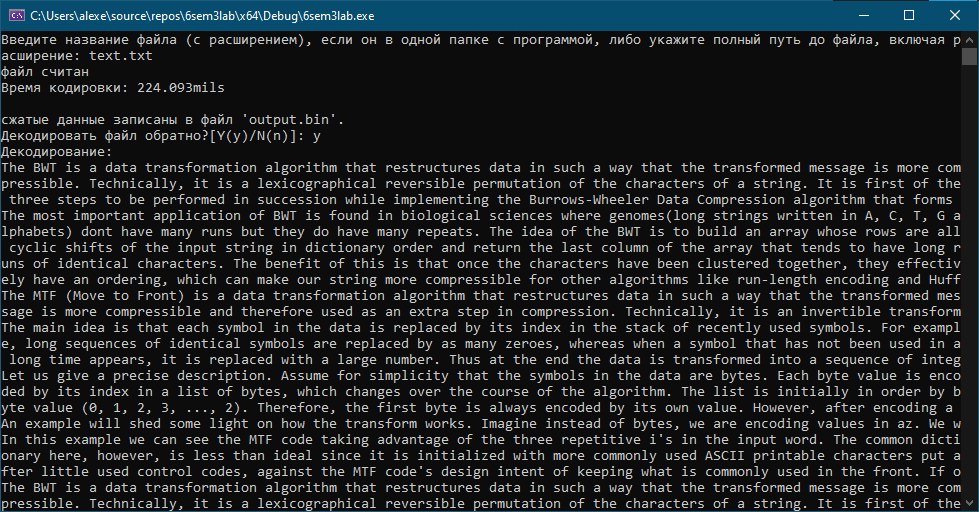


Рисунок 5 — декодирование текста

**Вывод**

В ходе выполнения данной лабораторной работы были реализованы алгоритмы сжатия и преобразования, а именно:

1. Алгоритм Хаффмана (HA)
2. Кодирование длин серий (RKE)

3. Алгоритм Лемпеля-Зива (LZ78)

4. Преобразование Барроуза-Уиллера (BWT)

5*.* MTF

6. Арифметическое кодирование (AC)

7. PPM

Зелёным выделены те, которые были реализованы, а также для низ реализована декодировка. Красным, те которые не реализованы, либо у которых не реализована декодировка. Алгоритм Хаффмана может и не самый быстрый и не лучше всего сжимает, за то самый стабильный, работает с любым текстом, и проще и логичнее в реализации. Остальные алгоритмы мной вероятно были выполнены криво, но это лучшее, что я смог найти в интернете и додумать.

Ссылка   
<https://github.com/FOOZBY/6sem3lab.git>