Курсовая работа по курсу дискретного анализа: алгоритм BWT+MTF+RLE

Выполнила студентка группы 08-308 МАИ *Шевлякова София.*

Условие

Для BWT в конец строки добавляется символ "$"для помощи в дальнейшем декодировании. Это может быть любой символ меньший чем «a». Для MTF используется следующее начальное распределение кодов:
$ -> 0
a -> 1
b -> 2
c -> 3
...
x -> 24
y -> 25
z -> 26
Максимальная длинна текста в тестах 10^5
Формат ввода: вам будут даны тесты двух типов.
Первый тип:
compress
<text>
Текст состоит только из малых латинских букв. В ответ на него вам нужно вывести коды, которыми будет закодирован данный текст.
Второй тип:
decompress
<codes>
Вам даны коды в которые был сжат текст из малых латинских букв, вам нужно его разжать. В RLE коды записываются в порядке: количество, значение
Формат вывода: в ответ на первый тип тестов выведите коды, которыми будет закодирован текст. Каждая пара на отдельной строке. На второй тип тестов выведите разжатый текст.

Метод решения

Для сжатия текста будем использовать комбинацию трех алгоритмов: BWT и MTF для предобработки данных и RLE для кодирования.
Сначала обработаем входной текст с помощью BWT (Burrows-Wheeler transform), алгоритм меняет порядок символов в тексте таким образом, что последовательности многократно чередующихся символов преобразуются в строки одинаковых символов. Для преобразования нам надо записать все циклические сдвиги строки, отсортировать их, и результатом

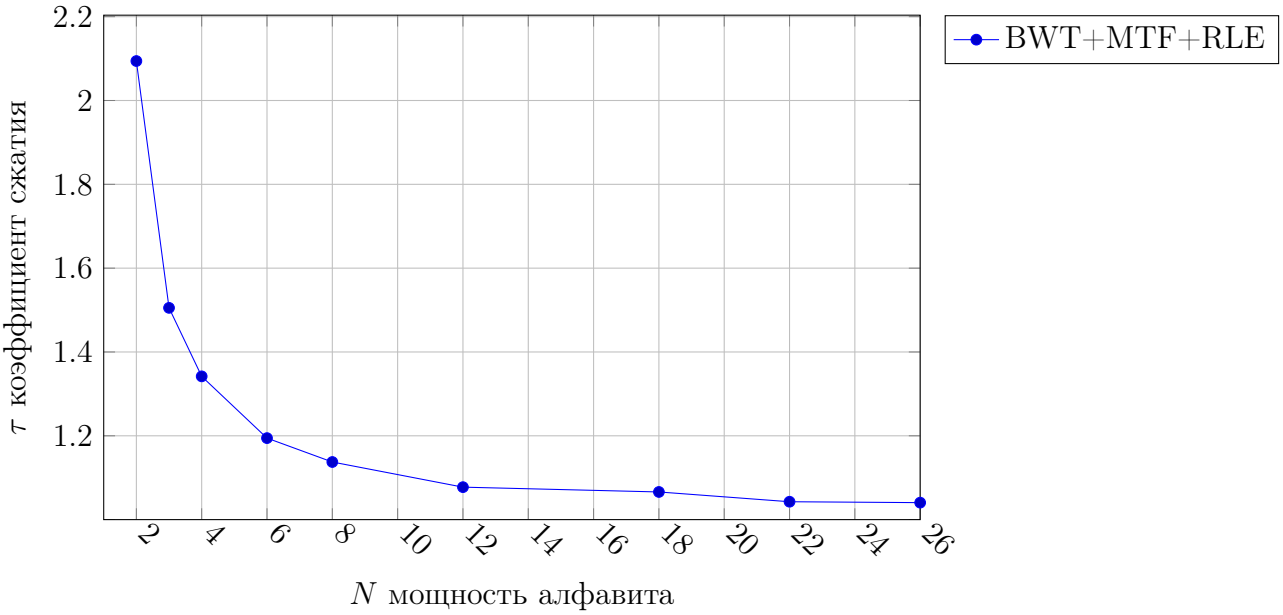
преобразования будет являться строка, полученная из последнего символа строчек отсортированных циклических сдвигов. То есть циклические сдвиги, содержащие одинаковые подстроки, смогут оказаться рядом, а значит на выходе, мы скорее всего получим последовательность повторяющихся символов. Так как хранить все строки накладно, то будем хранить индекса символа, с которого начинается циклический сдвиг.
Теперь мы получили последовательности одинаковых символов, редко перемежающиеся другими символами. Если после этого выполнить шаг по замене каждого символа расстоянием до его предыдущей встречи (алгоритм MTF), то полученный набор чисел будет иметь удачное распределение для применения сжатия. Основной идеей преобразования MTF (move-to-front) является замена каждого входного символа его номером в специальном стеке недавно использованных символов. Последовательности идентичных символов, будут заменены (начиная со второго символа) на последовательность нулей. Если же символ долго не появлялся во входной последовательности, он будет заменён большим числом. Изначально каждое возможное значение записывается в алфавит, в ячейку с номером. В процессе обработки данных этот список изменяется. По мере поступления очередного символа на выход подается номер элемента, содержащего его значение. После чего этот символ перемещается в начало списка, смещая остальные элементы вправо. Этот метод позволяет легко преобразовать данные, насыщенные длинными повторами разных символов в блок данных, самыми частыми символами которого будут нули.
Теперь для преобразованной строки мы можем использовать алгоритм сжатия RLE (run-length encoding), заменяющий повторяющиеся символы на число повторов и символ. Данный алгоритм наиболее эффективен, когда входная строка содержит большое количество рядом расположенных повторяющихся элементов.
Таким образом сочетание всех методов выполняет задачу сжатия исключением повторяющихся подстрок.
Теперь перейдем к декодированию: обратный алгоритм RLE считывает сначала число повторов n, а потом символ, и подает на выход этот символ n раз. Теперь необходимо использовать обратное преобразование для MTF: заполняем алфавит, считываем текущий символ и подаем на выход элемент алфавита по индексу, равному считанному символу, сдвигаем алфавит так, чтобы извлеченный символ оказался первым в алфавите, и так для всей строки.
Большой интерес представляет обратное преобразование BWT. Реализация наивным методом имеет большую асимптотическую сложность: необходимо каждый раз сортировать массив строк при добавлении последнего столбца. Можно заметить, что достаточно знать только то, в какое место перейдет первый символ после сортировки. Поэтому можем хранить два массива: один с отсортированной строчкой, а другой - с исходной. Если хранить для каждого элемента, индекс изначального положения и индекс в отсортированной строчке, то мы будем знать, на какую позицию перейдет символ, а значит, и какой символ на этой позиции к нему приклеится, проитерируемся n раз и получим один из циклических сдвигов строчки. Чтобы восстановить исходную строку, найдем символ $, он гарантированно должен располагаться в конце строки.

Описание программы

В данной программе содержится один файл main.cpp, в котором реализован алгоритм сжатия текста.
Основные этапы работы программы:

- bwt_coder(const string &text, string &result) - алгоритм преобразования Барроуза-Уилера
- mtf_coder(const string &text, vector<int> &result) - алгоритм преобразования Move-To-Front
- rle_coder(const vector<int> &text, vector<pair<int, int» &result) - алгоритм кодирования RLE
- bwt_decoder(const string &text, string &result) - алгоритм обратного преобразования Барроуза-Уилера
- mtf_decoder(const vector<int> &text, string &result) - алгоритм обратного преобразования Move-To-Front
- rle_decoder(const vector<pair<int, int» &text, vector<int> &result) - алгоритм декодирования RLE

Тест производительности



По результатам тестирования видно, что коэффициент сжатия максимален при минимальной мощности алфавита, то есть, когда входная строка содержит большое количество
повторяющихся символов.
Временная сложность работы алгоритма BWT составляет $O(n + n \cdot n \cdot log(n))$, так как за $O(n)$ заполним массив с индексами начала циклического сдвига и за $O(n \cdot n \cdot log(n))$ отсортируем элементы массива циклических сдвигов. Следующий алгоритм MTF выполняется за $O(n)$, так как для каждого элемента входной строки алфавит будем перестраивать за константу. И сложность кодирования RLE тоже линейная, так как нам достаточно один раз пройтись по строке, чтобы закодировать все повторяющиеся символы. Поэтому сложность кодирования $O(n + n \cdot n \cdot log(n) + n + n) = O(n \cdot n \cdot log(n))$. Обратное преобразование BWT выполняется за $O(n \cdot log(n) + n)$, так как нам надо знать индексы элементов в отсортированной строке, а потом за n операций последовательно пройтись по строке. Обратное преобразование MTF имеет линейную сложность, так как мы так же считываем элемент и перестраиваем алфавит за константу. Алгоритм декодирования RLE выполнится за $O(n)$, потому что восстанавливает все символы строки. Получаем, что сложность декодирования $O(n \cdot log(n) + n + n + n) = O(n \cdot log(n))$.

Выводы

В ходе курсового проекта были реализованы три алгоритма сжатия: BWT, MTF и RLE. BWT используется для перестановки символов в строке, что упрощает сжатие последующими алгоритмами. MTF позволяет легко преобразовать данные, насыщенные длинными повторами разных символов в блок данных, самыми частыми символами которого будут нули, такие последовательности эффективно будет сжимать алгоритмом Хаффмана. RLE используется для сжатия последовательностей повторяющихся символов. Используя комбинацию этих трех алгоритмов, удалось достичь значительного сжатия данных. Это позволяет экономить место при хранении и передаче информации, что особенно важно в условиях ограниченной памяти или низкой скорости передачи данных.