08 Алгоритм сжатия Барроуза Уилера

Jump to bottom

Alexander Morozov edited this page on 23 May 2019 · 3 revisions

Задание

https://classroom.github.com/a/j7Pxe7D5

Вам предстоит реализовать алгоритм сжатия Барроуза-Уилера. Данный алгоритм прост в реализации, не защищен патентами, а также превосходит по качеству сжатия gzip и PKZIP. Данный алгоритм лежит в основе Unix утилиты по сжатию данных bzip2.

Алгоритм сжатия данных Барроуза-Уилера состоит из трех частей, которые выполняются последовательно:

1. *Преобразование Барроуза-Уилера.* На вход подается текстовый файл на английском языке. Входной файл преобразуется в текстовый файл, в котором есть последовательности с одинаковыми символами, встречающимися несколько раз подряд.
2. *Перемещение к началу.* Текстовый файл с последовательностями одинаковых символов преобразуется в текстовый файл, в котором определенные символы встречаются чаще других.
3. *Алгоритм сжатия Хаффмана.* К текстовому файлу, в котором определенные символы встречаются чаще других, применяется алгоритм сжатия, который заменяет часто встречающиеся символы на короткие кодовые слова, а редкие символы в длинные кодовые слова.

Третий этап, на котором непосредственно происходит сжатие, является эффективным, так как после первого и второго этапа получается текстовый файл, в котором определенные символы встречаются намного чаще других. Для того, чтобы декодировать текстовый файл, необходимо произвести действия алгоритма в обратном порядке: применить декомпрессию Хаффмана, произвести перемещение к началу и произвести обратное преобразование Барроуза-Уилера. Вам необходимо эффективно реализовать преобразование Барроуза-Уилера и алгоритм перемещения к началу.

Алгоритм сжатия/декомпрессии Хаффмана реализовывать не требуется. Вы воспользуетесь уже готовым решением.

Алгоритм перемещения к началу

Основная идея алгоритма перемещения к началу заключается в поддержании упорядоченной последовательности символов алфавита и последовательном символьном чтении входного текста. Как только прочитан символ входного текста, определяется позиция, на которой встречается данный символ. Далее прочитанный символ перемещается в начало символьной цепочки. Позиция следующего входного символа уже определяется с использованием модифицированной цепочки символов.

Рассмотрим пример, где в качестве начальной упорядоченной цепочки символов рассматривается алфавит из 6-ти символов: A, B, C, D, E, F. Требуется закодировать входную строку CAAABCCCACCF. Изначальная упорядоченная цепочка будет обновляться шаг за шагом, когда новый входной символ будет размещен в начале цепочки.

move-to-front	in	out
A B C D E F	C	2
C A B D E F	A	1
A C B D E F	A	0
A C B D E F	A	0
A C B D E F	B	2
B A C D E F	C	2
C B A D E F	C	0
C B A D E F	C	0
C B A D E F	A	2
A C B D E F	C	1
C A B D E F	C	0
C A B D E F	F	5
F C A B D E		

Отметим, что если один и тот же символ встречается во входной строке много раз, то значения в колонке *out* будут небольшими целыми числами. Очень высокая частота некоторых символов будет идеальным сценарием для алгоритма сжатия Хаффмана.

Далее в качестве примера будет рассматриваться текстовый файл *abra.txt*:

```
$ cat abra.txt
ABRACADABRA!
```

Перемещение к началу, этап кодирования.

Вам предстоит работать с расширенной ASCII таблицей, состоящей из 256 символов. Инициализируйте стартовую последовательность символов, поставив на *i-е* место *i-й* символ из расширенной ASCII таблицы. Далее, получив на вход 8-ми битный символ c, напечатайте индекс считанного символа c в текущей последовательности, а потом переместите символ c с его текущего места в начало последовательности.

```
$ moveToFront.out -e abra.txt abra.out
```

Перемещение к началу, этап декодирования.

Инициализируйте стартовую последовательность символов, поставив на *i-е* место *i-й* символ из расширенной ASCII таблицы. Прочитайте 8-ми битный символ *c*, преобразуйте его в число от 0 до 255, напечатайте *c-й* символ из последовательности, а потом переместите в последовательности *c-й* символ в начало. Проверьте, что после декодирования, полученная строка совпадает с исходной строкой.

```
$ moveToFront.out -d abra.out
ABRACADABRA!
```

Вам предлагается реализовать класс *MoveToFront*, где каждый из методов соответствует этапу алгоритма перемещения к началу.

```
struct MoveToFront
{
   // Этап кодирования
   void encode(std::istream& input, std::ostream& output);

   // Этап декодирования
   void decode(std::istream& input, std::ostream& output);
};
```

Сложность. Если *R* мощность алфавита, а *N* количество символов во входном файле, то по времени алгоритм перемещения к началу должен работать за *O(R * N)* и за *O(R + N)* по памяти.

Суффиксный массив

Для того, чтобы эффективно реализовать преобразование Барроуза-Уилера необходимо воспользоваться суффиксным массивом — лексикографически отсортированным массивом всех суффиксов строки. Например, рассмотрим строку "ABRACADABRA!". В таблице ниже представлены 12 суффиксов рассматриваемой строки. Далее представлены отсортированные суффиксы.

i	Original Suffixes	Sorted Suffixes	index[i]
0	A B R A C A D A B R A !	! A B R A C A D A B R A	11
1	B R A C A D A B R A ! A	A ! A B R A C A D A B R	10
2	R A C A D A B R A ! A B	A B R A ! A B R A C A D	7
3	A C A D A B R A ! A B R	A B R A C A D A B R A !	0
4	C A D A B R A ! A B R A	A C A D A B R A ! A B R	3
5	A D A B R A ! A B R A C	A D A B R A ! A B R A C	5
6	D A B R A ! A B R A C A	B R A ! A B R A C A D A	8
7	A B R A ! A B R A C A D	B R A C A D A B R A ! A	1

i	Original Suffixes	Sorted Suffixes	index[i]
8	B R A ! A B R A C A D A	C A D A B R A ! A B R A	4
9	R A ! A B R A C A D A B	D A B R A ! A B R A C A	6
10	A ! A B R A C A D A B R	R A ! A B R A C A D A B	9
11	! A B R A C A D A B R A	R A C A D A B R A ! A B	2

Значение *i-го* элемента массива *index[i]* соответствует позиции суффикса в исходном массиве, который встречается на *i-м* месте в отсортированном массиве. Это значит, что если *index[11] = 2*, то суффикс, который в отсортированном массиве находится на *11-м* месте, в исходном массиве находится на *2-м* месте.

Вам необходимо реализовать суффиксный массив, реализовав класс *CircularSuffixArray*:

```
struct CircularSuffixArray
{
   // Ctor
   CircularSuffixArray(const std::string& input);

   // Returns size of the input string
   size_t size() const;

   // Returns index of suffix from original array
   size_t operator[](size_t n) const;
};
```

Сложность. Конструктор за время *O(N * log N)*, метод size и оператор [ ] за константное время. По памяти за *O(R + N)*.

Преобразование Барроуза-Уилера.

Основная цель преобразования Барроуза-Уилера заключается в том, чтобы преобразовать входные данные так, чтобы последующий алгоритм сжатия отработал эффективно. Алгоритм переставляет символы входного сообщения так, чтобы получить последовательности повторяющихся символов, но при условии, что преобразованные входные данные можно привести к изначальному виду.

Преобразование Барроуза-Уилера, этап кодирования

Преобразование Барроуза-Уилера строки *s* длины *N* определяется следующим образом: рассмотрим результат сортировки всех суффиксов строки *s*. Преобразование Барроуза-Уилера это последний столбец в отсортированном массиве суффиксов *t[]*, которому предшествует индекс строки, соответствующей изначальной строке в отсортированном массиве. Рассмотрим пример со строкой "ABRACADABRA!":

i	Original Suffixes	Sorted Suffixes	t	index[i]
0	A B R A C A D A B R A !	! A B R A C A D A B R	A	11

i	Original Suffixes	Sorted Suffixes	t	index[i]
1	B R A C A D A B R A ! A	A ! A B R A C A D A B	R	10
2	R A C A D A B R A ! A B	A B R A ! A B R A C A	D	7
*3	A C A D A B R A ! A B R	A B R A C A D A B R A	!	*0
4	C A D A B R A ! A B R A	A C A D A B R A ! A B	R	3
5	A D A B R A ! A B R A C	A D A B R A ! A B R A	C	5
6	D A B R A ! A B R A C A	B R A ! A B R A C A D	A	8
7	A B R A ! A B R A C A D	B R A C A D A B R A !	A	1
8	B R A ! A B R A C A D A	C A D A B R A ! A B R	A	4
9	R A ! A B R A C A D A B	D A B R A ! A B R A C	A	6
10	A ! A B R A C A D A B R	R A ! A B R A C A D A	B	9
11	! A B R A C A D A B R A	R A C A D A B R A ! A	B	2

В столбце *t[]* выделены результаты преобразование Барроуза-Уилера. Так как исходная строка "ABRACADABRA!" оказалась на *3-й* строке, то требуемый индекс равен *3*.

Рассмотрим результат преобразование Барроуза-Уилера:

```
3
ARD!RCAAAABB
```

Заметим, что в получившейся строке 4 раза повторяется символ A и 2 раза повторяется символ B. Эти подпоследовательности делают получившуюся строку удобной для сжатия.

Преобразование Барроуза-Уилера, этап декодирования

Нам необходимо совершить обратное преобразование и трансформировать закодированную строку в исходную. Если *j-й* исходный суффикс является *i-м* суффиксом в отсортированном массиве мы считаем, что значение *next[i]* это строка в отсортированном массиве, где встречается *(j+1)-й* суффикс исходного массива. Например, если *first* это строка, где располагается исходная входная строка, то *next[first]* это индекс сортированного массива суффиксов, где встречается первый суффикс исходной строки (исходная строка, сдвинутая на 1 символ влево); *next[next[first]]* - индекс отсортированного массива суффиксов, где встречается второй суффикс исходной строки, и так далее.

Декодирование строки при помощи массива *t[]*, массива *next[]* и индекса *first*. Использую массив *t[]* мы сможем определить, какие символы располагаются в первом столбце отсортированных суффиксов, так как это те же символы из массива *t[]*, только отсортированные.

i	Sorted Suffixes	t	next

i	Sorted Suffixes	t	next
0	! ? ? ? ? ? ? ? ? ? ?	A	3
1	A ? ? ? ? ? ? ? ? ? ?	R	0
2	A ? ? ? ? ? ? ? ? ? ?	D	6
*3	A ? ? ? ? ? ? ? ? ? ?	!	7
4	A ? ? ? ? ? ? ? ? ? ?	R	8
5	A ? ? ? ? ? ? ? ? ? ?	C	9
6	B ? ? ? ? ? ? ? ? ? ?	A	10
7	B ? ? ? ? ? ? ? ? ? ?	A	11
8	C ? ? ? ? ? ? ? ? ? ?	A	5
9	D ? ? ? ? ? ? ? ? ? ?	A	2
10	R ? ? ? ? ? ? ? ? ? ?	B	1
11	R ? ? ? ? ? ? ? ? ? ?	B	4

Далее, используя массив *next[]* и индекс *first* можно восстановить исходную строку, так как первый символ *i-го* исходного суффикса является *i-м* символов исходной строки. В рассматриваемом примере индекс *first* равен *3*, что значит, что исходная строка в отсортированном массиве суффиксов встречается под *3-м* индексом. Мы можем сделать вывод, что первый символ исходной строки равен *'A'*, а последний *'!'*. Заметим, что *next[first] = 7*, что значит, что следующий по порядку суффикс (исходная строка, сдвинутая на *1* символ влево) встречается под *7-м* индексом. Это значит, что второй символ исходной строки это *'B'*. Далее, *next[next[first]] = 11*, поэтому *3-й* символ исходной строки равен *'R'*.

Формирование массива *next[]* на основе массива *t[]* и индекса *first*. Легко определить элемент *next[]* для символа, который встречается в исходной строке только один раз. Рассмотрим суффикс, который начинается с символа *'C'*. Просматривая первую колонку, мы замечаем его на *8-й* строчке отсортированного массива суффиксов. Следующий суффикс после данного (в исходном массиве) имеет символ *'C'* в конце. Это верно для *5-й* строчки, поэтому *next[8] = 5*. Аналогично можно определить значение *next[]* для символов *'D'* и *'!'*, *next[9] = 2* и *next[0] = 3*.

i	Sorted Suffixes	t	next
0	! ? ? ? ? ? ? ? ? ? ?	A	3
1	A ? ? ? ? ? ? ? ? ? ?	R	
2	A ? ? ? ? ? ? ? ? ? ?	D	
*3	A ? ? ? ? ? ? ? ? ? ?	!	
4	A ? ? ? ? ? ? ? ? ? ?	R	

i	Sorted Suffixes	t	next
5	A ? ? ? ? ? ? ? ? ? ?	C	
6	B ? ? ? ? ? ? ? ? ? ?	A	
7	B ? ? ? ? ? ? ? ? ? ?	A	
8	C ? ? ? ? ? ? ? ? ? ?	A	5
9	D ? ? ? ? ? ? ? ? ? ?	A	2
10	R ? ? ? ? ? ? ? ? ? ?	B	
11	R ? ? ? ? ? ? ? ? ? ?	B	

Однако, если символ встречается с исходной строке несколько раз, то однозначно определить значение *next[]* не получается. Однако, есть правило, которое позволяет справиться с неоднозначностью: если строки *i* и *j* в отсортированном массиве начинаются с одного символа и *i* < *j*, то *next[i]* < *next[j]*.

Например, если рассмотреть символ *'R'*, то можно заметить, что он встречается на первом месте в строках *10* и *11*. На последнем месте данный символ встречается в строках с индексами *1* и *4*. Так как *10* < *11*, то *next[10] = 1* и *next[11] = 4*.

Почему данное правило работает? Так как суффиксы отсортированы, то получается, что лексикографически суффикс под индексом *10* меньше, чем суффикс под индексом *11*. Поэтому неизвестные символы из суффикса под индексом *10* должны быть лексикографически меньше неизвестных символов суффикса под индексом *11*. Кроме этого, мы знаем, что из 2-х суффиксов, заканчивающихся на *'R'* (суффиксы с индексами *1* и *4*), суффикс *1* лексикографически меньше суффикса *4*. Но неизвестные символы из суффиксов *10* и *11* это те самые неизвестные символы из суффиксов *1* и *4*. Поэтому и получается, что *next[10] = 1* и *next[11] = 4*.

Реализуйте преобразование Барроуза-Уилера:

```
struct BWT
{
   // Преобразование Барроуза-Уилера, этап кодирования
   void transform(std::istream& input, std::ostream& output);

   // Обратное преобразование Барроуза-Уилера, этап декодирования
   void inverseTransform(std::istream& input, std::ostream& output);
};
```

Сложность. Преобразование и обратное преобразование за *O(N + R)* по времени и за *O(N + R)* по памяти.

https://github.com/itiviti-cpp/wiki.wiki.git