УДК 004

Сжатие данных с применением алгоритмов BWT и RLE на языке программирования C++

Фатеенков Данила Витальевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В статье описывается работа и реализация преобразования Барроуза-Уилера (BWT) и Run-Length Encode алгоритма на языке программирования C++. Также рассматривается эффективность описанных алгоритмов сжатия на примере кодирования Шеннона (сравниваются такие параметры закодированного сообщения как энтропия, средняя длина кодового слова, коэффициент сжатия и избыточность кода).
Ключевые слова: кодирование, сжатие данных, C++, преобразование Барроуза-Уилера, Run-Length Encode

Data compression via BWT and RLE algorithms in C++ programming language

Fateenkov Danila Vitalievich
Sholom-Aleichem Priamursky State University
Student

Abstract
The article describes the work and implementation of Barrows-Wheeler Transform (BWT) and Run-Length Encode algorithm in C++ programming language. The efficiency of the described compression algorithms on the example of Shannon coding is also considered (such parameters of encoded message as entropy, average codeword length, compression ratio and code redundancy are compared).
Keywords: encoding, data compression, C++, Barrows-Wheeler Transform, Run-Length Encode

1. Введение

1.1 Актуальность

Одной из основных задач теории информации является предварительное сжатие данных (с возможностью восстановления) для дальнейшего кодирования и передачи полученного пакета пользователю. В настоящее время существует большое количество архиваторов, которые используют алгоритмы сжатия перед кодированием данных. Примерами таких программ являются WinRAR, 7zip или bzip2. Алгоритмы сжатия данных позволяют преобразовать исходную информацию в более подходящий для кодирования формат, что позволяет ускорить работу с передачей информации

и уменьшить её вес. На сегодняшний день данные могут иметь слишком большой вес для пересылки, поэтому задача эффективного сжатия данных (эффективность достигается при наименьшем размере файла и скорости сжатия) является актуальной.

1.2 Обзор исследований

А.А. Кожемякина и Н.Б. Буторина разработали для сжатия данных программу с использованием языка программирования Python, в которой доступны несколько алгоритмов сжатия информации, в том числе bzip2, использующий BWT и RLE алгоритмы [1]. И.В Ковалёва и А.Н Размахнина реализовали алгоритм преобразования Барроуза-Уилера на C++ [2]. Предложенный способ реализации предполагает использование функций библиотеки “algorithm”, что может создать ограничения для оптимизации кода. В статье журнала “ Information Processing Letters” в 2019-м году было описано применение позиционного преобразования Барроуза-Уилера при работе с парами строк, расстояние Хэмминга между которыми больше заданного порога [3]. В статье журнала “ Procedia Computer Science” в 2021-м году было представлено исследование на тему улучшения алгоритма сжатия данных для диспетчеризации электроэнергии на основе RLE [4].

1.3 Цель исследования

Цель – реализовать алгоритм преобразования Барроуза-Уилера и алгоритм сжатия Run-Length Encode на языке программирования C++.

2. Материалы и методы

Для реализации поставленной цели используется язык программирования C++.

3. Результаты и обсуждения

BWT (Преобразование Барроуза-Уилера) – алгоритм был разработан в 1994 году Майклом Барроузом и Дэвидом Уиллером. Главную цель, которую должен выполнять BWT алгоритм – изменять исходные данные, чтобы достичь максимальной эффективности при сжатии.

Алгоритм изменяет порядок символов во входной строке таким образом, что повторяющиеся подстроки образуют зачастую (это можно увидеть при работе с большими данными) на выходе строку с идущими подряд последовательностями одинаковых символов. Такая последовательность особенно применима в энтропийном кодировании (код Шеннона-Фано, код Хаффмана), так как её можно преобразовать в строку символов и чисел, характеризующих длину подстроки равных символов. Таким образом на выходе получается строка, предназначенная для сжатия с достижением меньшего веса, чем при сжатии без BWT алгоритма.

Часто в дополнении с BWT алгоритмом используется RLE (Run-Length Encoding) – алгоритм сжатия, который заменяет последовательность повторяющихся символов на пару, первым элементом которой является сам

символ, а вторым длина последовательности. Данный алгоритм эффективен при работе со строками, в который много цепочек повторяющихся символов. И данному условию может соответствовать результат преобразования Барроуза-Уилера.

BWT алгоритм состоит из нескольких этапов:

1. Составляется массив всех циклических сдвигов строки.

2. Массив сортируется и запоминается индекс входной строки в полученном наборе данных.

3. Составляется строка, в которую входят последние символы каждой строки из массива.

Исходная строка	Массив сдвигов	Сортированный массив	Результат
MOUSE	OUSEM	EMOUS	SEMUO 2
	USEMO	MOUSE	
	SEMOU	OUSEM	
	EMOUS	SEMOU	
	MOUSE	USEMO	

Рисунок 1. Пример работы алгоритма BWT

Первый этап можно реализовать на языке программирования C++ следующим образом:

1. Создаётся динамический массив, в который заносятся все циклические сдвиги.

2. Сдвиг можно получить перестановкой первого символа в самый конец и повторять данное действие, пока строка со сдвигом не будет равна исходной строке.

```
     string shifted_str = main_str;
     string temp_str = "";
     while (temp_str != main_str) {
          temp_str = "";
          char temp_chr = shifted_str[0];

          for (int i=1;i<shifted_str.length();i++) {
               temp_str += shifted_str[i];
          }

          shifted_str = temp_str + temp_chr;
          if (shifted_str != main_str)
all_str.push_back(shifted_str);
          temp_str = shifted_str;
     }
```

Сортировка массива полученных сдвигов возможно с использованием любого существующего на данный момент алгоритма. Также во время

сортировки нужно запоминать позицию исходной строки (это можно сделать также после сортировки поиском в упорядоченном массиве).

Также возможна реализация варианта с использованием особого символа конца строки (задаётся разработчиком). При использовании данного символа нет необходимости запоминать позицию исходной строки в массиве отсортированных сдвигов, так как при сортировке исходная строка помещается на первое место (стандарт ISO 8859). Но данный вариант сложен в реализации, так как язык программирования C++ использует стандарт POSIX при сравнении символов и строк.

Пример работы алгоритма со входной строкой “spatial tree structure” (рис. 2).

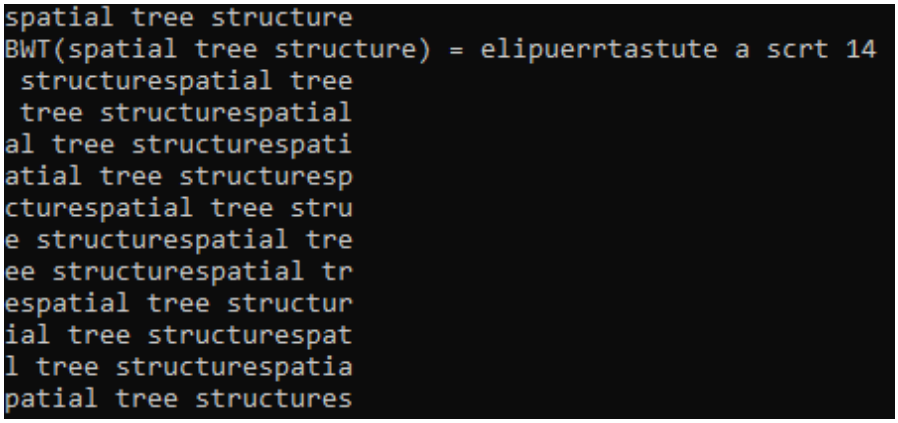

Рисунок 2. Результат работы преобразования Барроуза-Уилера

Декодирование полученного сообщения представляется в виде массива размера n постепенно заполняющихся строк, где n – длина преобразованной строки. Строки заполняются символами преобразованной последовательности и сортируются. Данный алгоритм повторяется до тех пор, пока длина строк в массиве не будет равна n, то есть результат можно представить как матрицу размерности n*n, каждый элемент которой это символ преобразованной BWT алгоритмом последовательности.

После завершения формирования массива выбирается элемент с ранее сохранённым индексом.

```
     vector<string> decoding_str;
     for (int i=0;i<encoded_str.length();i++) {
          decoding_str.push_back({encoded_str[i]});
     }
     string char_sort_temp = "";
     array_sort(decoding_str);
     for (int i=0;i<encoded_str.length()-1;i++) {
          for (int j=0;j<encoded_str.length();j++)
decoding_str[j] = encoded_str[j] + decoding_str[j];

          array_sort(decoding_str);
     }
     string decoded_str = decoding_str[index];
```

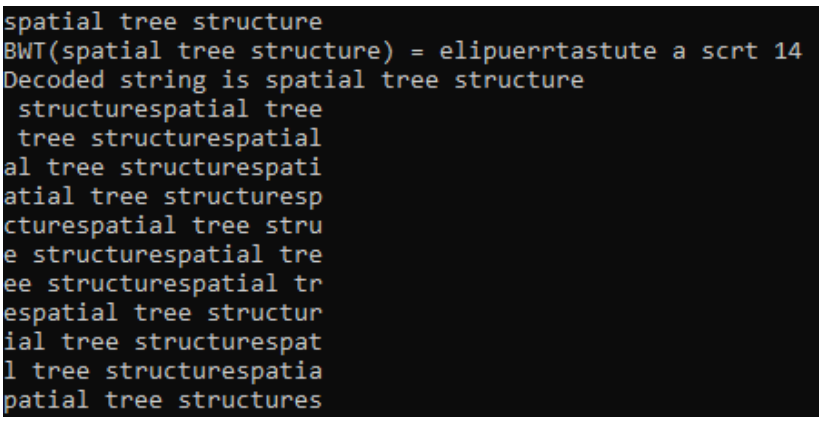

Рисунок 3. Результат работы алгоритма декодирования

Алгоритм декодирования можно оптимизировать. Несложно заметить, что сортировка изменённых строк ставит элементы в одну и ту же позицию на каждом шаге. Это видно, если наблюдать сортировку строк с позиции добавленного символа. Таким образом, можно запоминать индекс, на который перемещается строка при сортировке после добавления нового символа, что позволит не повторять сортировку на каждом шаге цикла и ускорит работу декодирования.

Вторым этапом сжатия строки является RLE-кодирование. Главными его особенностями можно считать простоту реализации и эффективность сжатия. Так как нам необходимо только изменить все подстроки с повторяющимися элементами на пары (сам элемент и длина подстроки), то алгоритм способен сжать строку в один проход, что будет эффективно по времени (особенно при работе с большими данными). Алгоритм состоит из нескольких шагов:

1. Выбирается первый символ и сравнивается с впереди стоящими до тех пор, пока не встретится отличный от исходного символ. При каждом сравнении счётчик, характеризующий длину подстроки, увеличивается на 1.

2. В случае не совпадении сравниваемых символов, значение счётчика добавляется в массив длин последовательностей и обнуляется, а символом, который будет использоваться для сравнения, выбирается последний символ, с которым сравнивался предыдущий.

3. Формируется новая строка – результат выполнения алгоритма.

Например, необходимо закодировать строку “aaaabbddde”. Если убрать все повторяющиеся символы, то в ней можно выделить 4 последовательности. Заменяя последовательности на пары (первый элемент – символ, второй – количество его повторений в последовательности), то результатом такого преобразования будет новая строка “a4b2d3e1”. Если оставить только символы в результате, то получится строка, подходящая для дальнейшего преобразования алгоритмами энтропийного кодирования. В результате показатели зашифрованной строки, предварительно преобразованной через алгоритм RLE, будут иметь лучшие значения (увеличится коэффициент сжатия и уменьшится избыточность кода, например).

```
	string RLE_str = "";
	vector<int> quantity_vec; string nextStep_str = "";
	char temp_RLE_char = encoded_str[0]; int quantity_RLE_char =
0;
	for (int i=0;i<encoded_str.length();i++) {
		if (encoded_str[i] == encoded_str[i+1]) {
			quantity_RLE_char++;
		}
		else {
			RLE_str += temp_RLE_char +
to_string(quantity_RLE_char+1);
			nextStep_str += temp_RLE_char;
			quantity_vec.push_back(quantity_RLE_char+1);
			temp_RLE_char = encoded_str[i+1];
			quantity_RLE_char = 0;
		}
	}
```

```
spatial tree structure
BWT(spatial tree structure) = elipuerrtastute a scrt 14
Run-Length Encoded string: e1l1i1p1u1e1r2t1a1s1t1u1t1e1 1a1 1s1c1r1t1
Decoded string is spatial tree structure

--------------------------------
```

Рисунок 5. Результат работы RLE сжатия

Строка “spatial tree structure” при RLE сжатии даёт не выходе не лучший результат, так как её длина слишком маленькая, а повторяющихся символов практически нет. На более длинной строке (для демонстрации взят текст, длина которого 583 символа) можно увидеть эффективность работы алгоритмов BWT и RLE (рисунок 6).

```
BWT(s) = rrreee,,,rrrrrraaarrrddddddddddsssdddsss,,,
Run-Length Encoded string: r3e3,3r6a3r3d10s3d3s3,313
String for the future encoding: re,rardsds,ltsfnrehr
RLE-decoded string: rrreee,,,rrrrrraaarrrddddddddddds
```

Рисунок 6. Пример эффективного сжатия BWT и RLE алгоритмами

Для декодирования можно использовать полученный массив чисел, характеризующих длину последовательности символов. Алгоритм декодирования состоит в восстановлении исходной строки созданием подстрок из каждого символа в закодированной последовательности. Подстрока создаётся с использованием массива чисел повторений символа:

```
	encoded_str = "";
	for (int i=0;i<nextStep_str.length();i++) {
		for (int j=0;j<quantity_vec[i];j++) {
			encoded_str += nextStep_str[i];
		}
	}
```

Закодированная строка энтропийным кодом, предварительно обработанная BWT и RLE алгоритмами будет иметь более хорошие показатели.

Например, необходимо закодировать текст длиной 583 символа (рисунок 7).

Характеристики	Текст, закодированный без сжатия данных	Текст, закодированный со сжатием данных
Длина текста	583	121
Энтропия	4.08429	4.16883
Максимальное значение энтропии	9.18735	6.91886
Средняя длина кодового слова	4.55746	4.55683
Коэффициент сжатия	0.444556	0.602531
Избыточность	0.473172	0.388

Рисунок 7. Сравнение основных характеристик закодированного текста

Исходя из полученных результатов, можно сделать вывод – сжатие текста позволяет привести её к более подходящему виду для шифрования, что выполняет одну из главных задач теории информации. При сжатии понижается одна из главных характеристик текста при кодировании – избыточность. Избыточность - показатель качества кода, оптимальный код обладает минимальной избыточностью и является разностью средней длины кода и энтропии (средняя длина кодового слова всегда больше энтропии). Также повышается коэффициент сжатия – данная величина показывает, какая доля сообщений в закодированном коде является избыточной и чем данная характеристика больше, тем оптимальнее код.

Таким образом, алгоритмы BWT и RLE являются эффективными алгоритмами для сжатия данных. В статье были рассмотрены и реализованы преобразование Барроуза-Уилера и Run-Length Encode алгоритм.

Библиографический список

1. Кожемякина А.А., Буторина Н.Б. Система сжатия информации с выбором оптимального алгоритма. Томск: Издательский Дом Томского государственного университета, 2019. 210 с.
2. Ковалёва И.В, Размахнина А.Н Реализация преобразования Барроуза-Уилера с помощью языка C++ // Постулат. 2016. № 12. С. 56
3. Mäkinen V., Norri T. Applying the Positional Burrows–Wheeler Transform to all-pairs Hamming distance // Information Processing Letters. 2019. Т. 146. С. 17-19
4. Zhanga J., Sun D. Improvement of data compression technology for power dispatching based on run length encoding // Procedia Computer Science. 2021.

Т. 183. С. 526–532
5. Преобразование Барроуза-Уилера URL: https://tftwiki.ru/wiki/Burrows–Wheeler_transform.