§ 4 КОДИРОВАНИЕ И ЗАЩИТА ИНФОРМАЦИИ

Горяинов С.И.

ПЕРЕСТРОЙКА БИНАРНЫХ ДЕРЕВЬЕВ В АЛГОРИТМЕ ХАФФМАНА

Аннотация: *Предметом данного исследования является время, затрачиваемое на полную перестройку бинарного дерева, а также степень сжатия текста в алгоритме Хаффмана. В работе определяются зависимости времени работы программы и степени сжатия текста от длины строки при произвольном количестве уникальных символов, при фиксированном количестве уникальных символов, а также при фиксированной длине строки и различном количестве уникальных символов. Показано, что время, требуемое на перестройку бинарного дерева, составляет незначительную часть общего времени работы программы. Алгоритм построения кодов символов включает следующие этапы: 1) считывание файла, содержащего текст;2) подсчёт частот различных символов текста;3) заполнение массивов данных и их сортировка;4) построение бинарного дерева. В ряде источников утверждается, что подход, использующий полную перестройку бинарного дерева неэффективен. Однако это утверждение не подкреплено соответствующими фактами. В данной работе с помощью анализа текстов различной длины и количеством уникальных символов, представленного в табличном и графическом виде, показано, что перестройка бинарного дерева незначительно влияет на время работы программы.*
Ключевые слова: *алгоритм Хаффмана, перестройка бинарного дерева, сжатие текстовых данных, время работы программы, экономное кодирование информации, префиксное кодирование, уникальные символы, степень сжатия текста, длина входной строки, средняя ошибка аппроксимации*

Назначением кодирования информации является экономия свободного места на носителях, а также ускорение процесса передачи данных. Первые работы по данному вопросу появились в 40-х годах прошлого века. В последующие три десятилетия объемы хранимых данных насколько возросли, что возникла проблема более экономного кодирования информации. В наше время объем передаваемых данных значительно увеличился, что делает проблему экономного кодирования информации еще более акту-

DOI: 10.7256/2305-6061.2014.4.13755

альной. Наиболее известными методами сжатия текстовых данных являются алгоритмы арифметического кодирования, Хаффмана, методы Зива–Лемпела, дифференциального кодирования и др.

Алгоритм Хаффмана – классический вариант адаптивного алгоритма оптимального префиксного кодирования информации с минимальной избыточностью. Как метод сжатия данных без потерь, он основан на устранении избыточности представления информации. Экономное кодирование достигается за счет представления маловероятных событий более длинными словами, чем событий с высокой вероятностью наступления. Популярность данного метода в программах сжатия данных обусловлена простотой и скоростью декодирования, а также несложной аппаратной реализацией. Таким образом, алгоритм Хаффмана обладает всеми свойствами полезных кодов источника: свойство единственности декодирования, свойство мгновенной декодируемости, свойство отсутствия префикса.

В данной статье представлена реализация алгоритма Хаффмана на языке C++ с помощью полной перестройки бинарного дерева на некоторых этапах работы. В ряде источников утверждается, что подход, использующий полную перестройку бинарного дерева неэффективен, так как действия, связанные с разбором дерева, его сортировкой и новым построением, занимают большое количество времени, однако это утверждение не подкреплено соответствующими фактами. С помощью анализа текстов различной длины и количеством уникальных символов, представленных в настоящей работе, делается попытка показать, что перестройка бинарного дерева незначительно влияет на время работы программы.

Перейдём к описанию алгоритма. Он включает в себя следующие этапы:

1. Считывание файла, содержащего текст;
2. Подсчёт частот различных символов текста;
3. Заполнение массивов данных и их сортировка;
4. Построение бинарного дерева.

Рассмотрим этот этап более подробно. Элементы массива данных, имеющих большие частоты, располагаются на правых ветвях дерева, а меньшие частоты – на левых ветвях.

При построении кода для каждого символа программа проходит по левым ссылкам узлов дерева до первого встреченного листа и обнаруживает первый элемент для объединения. После этого лист удаляется из дерева, и начинается аналогичный проход для поиска второго элемента, который после обнаружения также удаляется из дерева.

К коду всех символов первого из найденных листов добавляется единица, к коду всех символов второго листа добавляется ноль. Так, например, если символ «а» имел код 01, то, если он окажется в первом из найденных листов, код примет вид 101, если же во втором, то 001.

После нахождения двух элементов происходит их объединение в один новый элемент, причем частоты обоих суммируются, и получившееся значение присваивается новому элементу. Для нахождения места расположения образованного элемента в дереве сравниваются частоты слева и справа от текущей позиции, начиная с корневого узла. На основании этих сравнений выполняется передвижение по дереву либо влево, либо

DOI: 10.7256/2305-6061.2014.4.13755

вправо. Если находится узел, частота которого равна частоте образованного элемента, происходит его вставка в текущую позицию, а найденный узел прикрепляется к вставленному слева.

Возможен случай, когда частота образованного элемента больше, чем частота узла, стоящего справа относительно текущей позиции. Если при этом выполнить действия, аналогичные действиям выше, и вставить новый элемент в дерево на место текущего, то возможна ошибка обхода дерева, в результате чего образованные коды символов будут некорректны. Для того чтобы это исправить запускается алгоритм, перестраивающий бинарное дерево. После подобной перестройки алгоритм образования новых узлов повторяется.

Программа продолжает свою работу до тех пор, пока в дереве не останется один элемент. Это означает, что все элементы были обработаны и для всех символов, которые располагались в исходном файле, были построены коды.

После этого выводится исходная строка и количество бит, занимаемых этой строкой, закодированная строка и количество бит, которые требуются для ее кодирования.

В качестве входных использовались тексты, содержащие символы русского и английского алфавитов, фрагменты кода программ и другие символы.

В таблице 1 приведены зависимости времени работы программы от длины строки и степень сжатия от числа уникальных элементов. Для анализа выбраны тексты длиной около 5000, 15000, 40000 и 80000 символов.

Таблица1.

Зависимость времени работы программы от длины строки и степени сжатия от числа уникальных символов

№	Длина строки (количество символов)	Время работы программы, мс	Количество уникальных символов	Степень сжатия
	Тексты длиной около 5000 символов			
1	5000	65	97	1.45
2	5012	46	103	1.48
3	5052	62	111	1.46
4	5012	62	101	1.49
5	4982	60	95	1.5
Среднее значение	5011	59	101	1.48
Тексты длиной около 15000 символов				
1	14855	234	92	1.79
2	15115	202	123	1.48
3	15523	202	127	1.46
4	15076	187	135	1.45
5	15083	202	118	1.43
Среднее значение	15130	205	119	1.52

DOI: 10.7256/2305-6061.2014.4.13755

	Тексты длиной около 40000 символов			
1	40613	1170	160	1.4
2	40468	1274	160	1.4
3	40499	1279	161	1.46
4	41044	1379	141	1.54
5	40888	1332	145	1.45
Среднее значение	40702	1287	153	1.45
Тексты длиной около 80000 символов				
1	80555	4243	151	1.47
2	79860	4477	160	1.42
3	80616	4632	160	1.42
4	79570	4508	161	1.42
5	80729	4290	148	1.47
Среднее значение	80266	4430	156	1.44

Средний коэффициент сжатия для таких текстов равен 1,5.

В таблице 2 приведена зависимость времени работы программы от длины входной строки при фиксированном количестве уникальных символов. Для анализа взяты шесть уникальных символов.

Таблица 2.

Зависимость времени работы программы от длины входной строки при фиксированном количестве уникальных символов

Длина строки	**Время работы, мс**	**Длина строки**	**Время работы, мс**	**Длина строки**	**Время работы, мс**
1125	31	18000	249	45000	1360
1500	31	21000	347	48000	1514
1781	31	24000	405	51000	1700
2710	46	27000	499	60000	2324
3410	31	31000	639	70000	3120
5000	46	33000	733	80000	4087
8000	78	36000	858	85000	4586
10000	109	39000	1107	90000	5226
13000	140	42000	1154	95000	5725

Зависимость времени работы от длины входной строки символов представлена графически (рисунок 1), и на указанном участке соответствует функции, заданной уравнением

$$\ln y = -0.04 \ln^3 x + 1.3 \ln^2 x - 13.3 \ln x + 45.4$$

со средней ошибкой аппроксимации 1%.

DOI: 10.7256/2305-6061.2014.4.13755

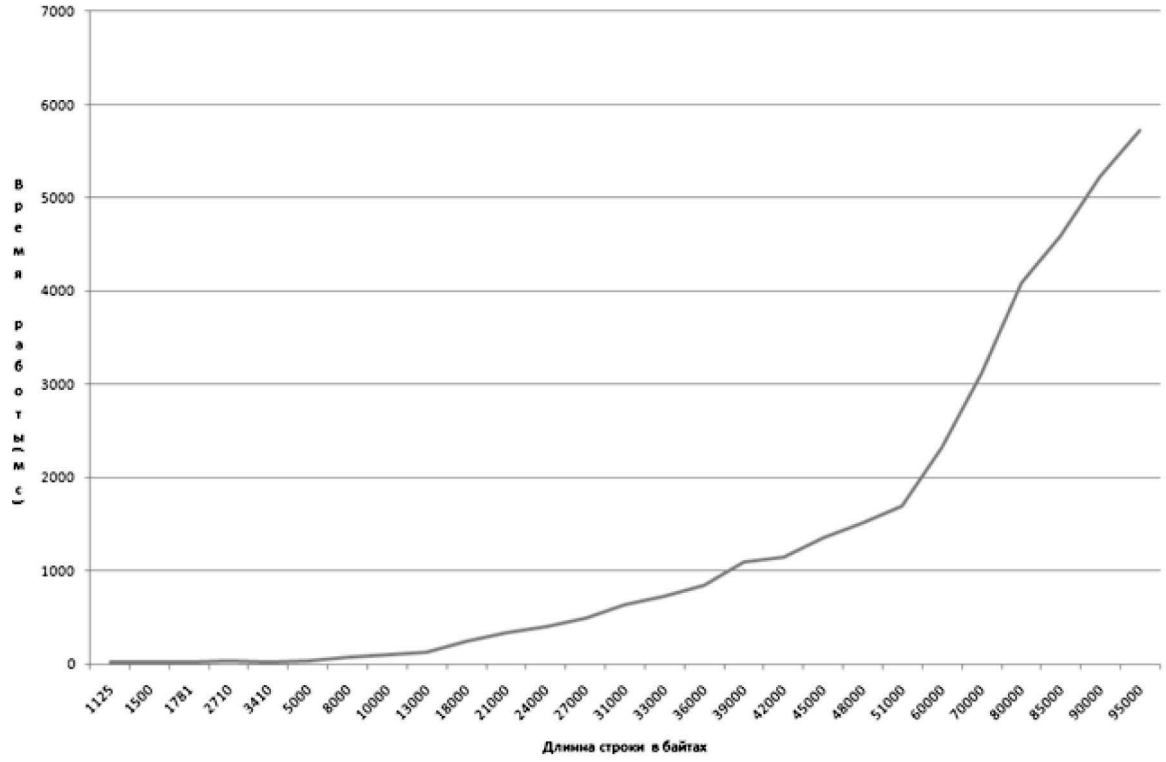

Рисунок 1. Зависимость времени работы программы от длины строки символов

В таблице 3 представлены результаты обработки текстов, в которых частоты символов и их общая длина постоянны, изменяется только количество уникальных символов.

Таблица 3.

Зависимость времени работы программы и степени сжатия от числа уникальных символов

№	Число уникальных символов	Время работы, мс	Степень сжатия
1	10	50	2.28
2	15	58	2.00
3	20	54	1.79
4	25	49	1.68
5	30	50	1.60
6	35	55	1.53
7	40	50	1.47
8	45	59	1.42
9	50	62	1.38
10	55	57	1.36
11	60	49	1,34
12	65	57	1,31
13	70	50	1,28

DOI: 10.7256/2305-6061.2014.4.13755

14	75	55	1,26
15	80	52	1,24
16	85	62	1,22
17	90	55	1,21
18	95	63	1,19
19	100	58	1,18
20	105	58	1,17
21	110	63	1,16
22	115	61	1,15
23	120	54	1,14
24	125	65	1,14
25	130	64	1,13
26	135	57	1,12
27	140	68	1,11
28	145	64	1,10
29	150	74	1,09
30	155	64	1,08

На рисунке 2 представлена зависимость времени работы от числа уникальных символов. Изменения во времени работы программы незначительны. Данные изменения выражаются линейной функцией, которая описывается формулой $y = 0.1x + 49.7$. При этом средняя ошибка аппроксимации составляет 6.3%.

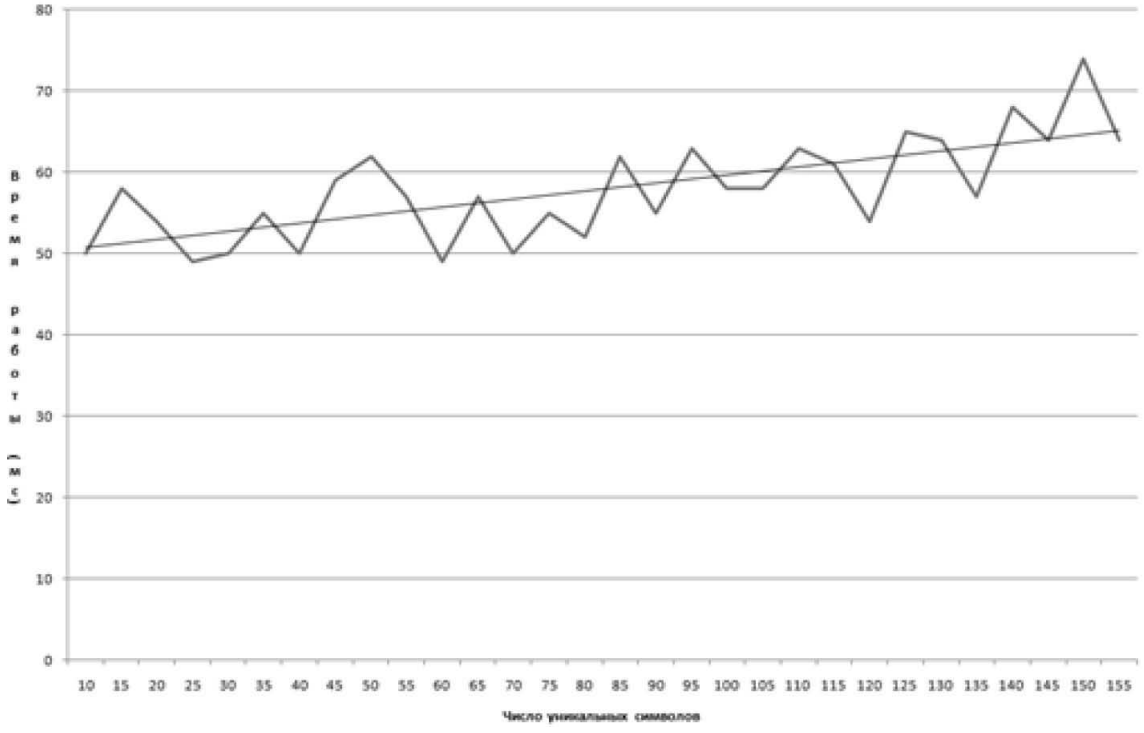

Рисунок 2. Зависимость времени работы программы от числа уникальных символов

DOI: 10.7256/2305-6061.2014.4.13755

График зависимости степени сжатия от числа уникальных символов при одинаковых частотах всех символов представлен на рис. 3. Полученная линия близка к логарифмической функции

$$y = 3 - 0.4 \ln x,$$

средняя ошибка аппроксимации которой составляет 3.4%.

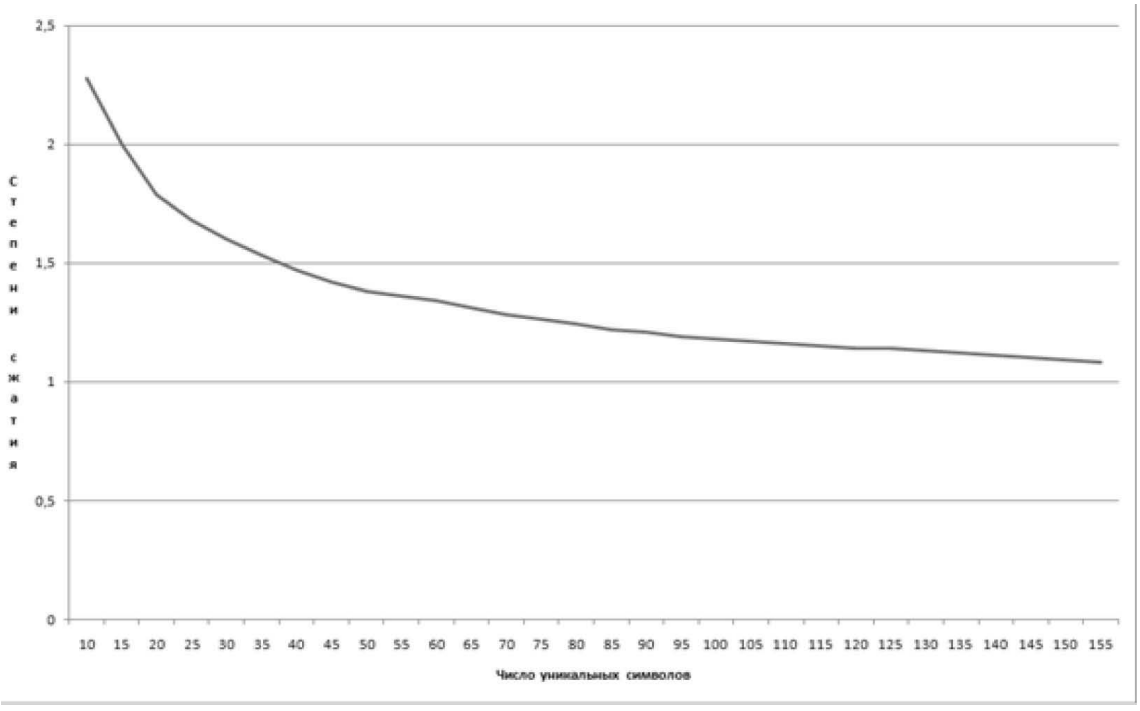

Рисунок 3. Зависимость степени сжатия от числа уникальных символов

Аналогичные значения времени работы и степени сжатия получаются для случая, когда частоты символов взяты случайным образом при фиксированном общем количестве символов файла.

Анализируя зависимости, представленные на рисунках 1 и 2, можно сделать вывод, что время, требуемое на перестройку бинарного дерева, составляет незначительную часть времени работы программы, затрачиваемого на анализ строки, на последующее ее кодирование и на вычисление степени сжатия.

Зависимость времени работы программы от длины входной строки символов (рис. 1) описывается уравнением $\ln y = -0.04 \ln^3 x + 1.3 \ln^2 x - 13.3 \ln x + 45.4$ Количество уникальных символов при построении этой зависимости не менялось. Таким образом, можно сделать вывод, что время, требуемое на построение и перестройку бинарного дерева, колеблется около одного значения, так как на время выполнения программы оказывают влияние и другие процессы, протекающие в операционной системе в данный момент времени. Основная часть времени работы программы тратится на анализ входной строки и её последующее кодирование.

График зависимости времени работы программы от количества уникальных символов

DOI: 10.7256/2305-6061.2014.4.13755

(рис. 2) показывает, что время, затрачиваемое на анализ входной строки и ее кодирование, является константой или меняется незначительно, а основная часть времени работы программы зависит от работы с бинарным деревом. Так как время работы в этом случае подчиняется линейному закону, можно сделать вывод, что наличие полной перестройки бинарного дерева в ходе работы программы оказывает незначительное влияние на продолжительность ее работы.

Библиография :

1. Александров О.Е. Компрессия данных или измерение и избыточность информации. Метод Хаффмана: Методические указания к лабораторной работе / О. Е. Александров – Екатеринбург: УГТУ–УПИ, 2000. – 36 с.
2. Кудряшов Б.Д. Теория информации: Учебник для вузов. – СПб.: Питер, 2009. – 320 с.: ил. – (Серия «Учебник для вузов»)
3. Колмогоров А.Н. Теория информации и теория алгоритмов. – М.: Наука, 1987. – 304 с.
4. Свирид Ю.В. Основы теории информации: Курс лекций. – Мн.: БГУ, 2003. – 139 с.
5. Смирнов М.А. Обзор применения методов безущербного сжатия данных в СУБД: Рукопись. – СПб.: ГУАП, 2004. – 58 с.
6. Шеннон К. Работы по теории информации и кибернетике. – М.: Издательство иностранной литературы, 1963. – 825 с

References:

1. Aleksandrov O.E. Kompressiya dannykh ili izmerenie i izbytochnost' informatsii. Metod Khaffmana: Metodicheskie ukazaniya k laboratornoi rabote / O. E. Aleksandrov – Ekaterinburg: UGTU–UPI, 2000. – 36 s.
2. Kudryashov B.D. Teoriya informatsii: Uchebnik dlya vuzov. – SPb.: Piter, 2009. – 320 s.: il. – (Seriya «Uchebnik dlya vuzov»)
3. Kolmogorov A.N. Teoriya informatsii i teoriya algoritmov. – M.: Nauka, 1987. – 304 s.
4. Svirid Yu.V. Osnovy teorii informatsii: Kurs lektsii. – Mn.: BGU, 2003. – 139 s.
5. Smirnov M.A. Obzor primeneniya metodov bezushcherbnogo szhatiya dannykh v SUBD: Rukopis'. – SPb.: GUAP, 2004. – 58 s.
6. Shennon K. Raboty po teorii informatsii i kibernetike. – M.: Izdatel'stvo inostrannoi literatury, 1963. – 825 s

DOI: 10.7256/2305-6061.2014.4.13755