Санкт-Петербургский государственный университет

Математико-Механический факультет

Кафедра исследования операций

«Допустить к защите»_____

Заведующий кафедрой

Курсовая работа

Еричев Алексей Олегович

Адаптивное арифметическое кодирование и алгоритм PPM (prediction by partial marching)

Научный руководитель

канд. ф.-м. н., доцент И. В. Агафонова

Санкт-Петербург – 2018

Оглавление

1. Введение

В наши дни обработка и передача цифровой информации используется почти во всех сферах человеческой жизни. Для экономии времени и памяти был разработан ряд алгоритмов, позволяющих сжимать информацию без потерь перед её отправкой. Одним из наиболее известных и широко используемых алгоритмов является алгоритм Хаффмена. Но данный алгоритм, хотя и способен произвести максимально сильное сжатие, даже в лучшем случае не имеет возможности передавать менее одного бита информации на каждый символ сообщения. Поэтому в 70-х годах XX века была разаботана схема арифметического кодирования, позволяющая кодировать некоторые символы менее, чем одним битом. О работе данной схемы, а также о некоторых её улучшениях и приложениях пойдёт речь в моей работе.

Для полноценного раскрытия темы будут рассмотрены (описаны) следующие алгоритмы:

1. Арифметическое кодирование
    а. Пример
    б. Реализация на C++
2. Адаптивное арифметическое кодирование
    а. Пример
    б. Реализация на C++
3. Prediction by Partial Matching
    а. Пример

2. Арифметичексое кодирование

Схема работы арифметического кодирования основана на сопоставлении набору символов (сообщению) отрезка из [0,1]. При поступлении на обработу нового символа длина отрезка уменьшается пропорционально вероятности появления данного символа. После обработки всего сообщения для построенного отрезка выбирается число, принадлежащее его внутренности и равное целому числу, делённому на минимально возможную натуральную степень двойки. Данное число и будет являться кодом данного сообщения. Одной из главных особенностей арифметического кодирования является его непрерывность. Кодируются не отдельные символы или их конечные группы, а всё предыдущее сообщение целиком.

Пусть имеется набор вероятностей для выбранной случайной величины. Составляется таблица, в которой определяется порядок символов, также в неё заносятся значения вероятностей. Для данного распределения, сохраняя порядок, строится исходное покрытие [0,1] отрезками, пересекающимися лишь в граничных точках, длины которых численно равны соответствующим значениям вероятностей, а объединение совпадает с [0,1].

2.1. Алгоритм прямого кодирования

Пусть количество символов равно N.

На вход подаётся первый символ сообщения с порядковым номером k, тогда для него вместо первоначального рассматривается k-ый отрезок и делится на N частей в тех же отношениях, что и исходный. Далее подаётся второй символ с номером m, для которого выбирется m-й отрезок уже нового покрытия и т.д.. Очевидно, что при сжатии получившийся код — число большее 0 и меньшее 1, поэтому при передаче сообщения первый ноль и точку можно отбросить, но для корректного завершения сжатия и для декодирования необходимо использовать специальный выделенный символ конца сообщения ($).

Пример: "Константа$"

Номер	Символ	Вероятность	Отрезок
1	к	0.1	[0,0.1]
2	о	0.1	[0.1,0.2]
3	н	0.2	[0.2,0.4]
4	с	0.1	[0.4,0.5]
5	т	0.2	[0.5,0.7]
6	а	0.2	[0.7,0.9]
7	$	0.1	[0.9,1]

Номер	Вход	Отрезок
-	∅	[0,1]
1	к	[0,0.1]
2	о	[0.01,0.02]
3	н	[0.012,0.014]
4	с	[0.0128,0.013]
5	т	[0.0129,0.01294]
6	а	[0.012928,0.012936]
7	н	[0.0129296,0.0129312]
8	т	[0.0129304,0.01293072]
9	а	[0.012930624,0.012930688]
10	$	[0.0129306818,0.012930688]

Итоговым отрезком будет являться [0.0129306818,0.012930688].

Число, принадлежащее его внутренности и равное частному целого и минимально возможной натуральной степени двойки — это $\frac{1735527}{2^{27}} = 0.11010011111011011001111_2$. Тогда кодом сообщения будет 11010011111011011001111.

L(Константа$)=21, тогда L_1=2.1 (бит/символ)

2.2. Обратное кодирование

Для обратного кодирования необходимо помимо самого кода также знать таблицу с номерами символов и их вероятностями.

По переданному коду однозначно восстанавливается число, полученное в результате кодировки и принадлежащее итоговому отрезку.

1. По исходной таблице данному числу однозначно сопоставляется отрезок, в котором он содержится. Этому отрезку соответствует символ сообщения. (Если это символ конца сообщения, то процесс декодирования заканчивается)

2. Вычисляется разница числа и левой границы содержащего его отрезка, а также длина данного отрезка, после чего находится частное этих значений. Полученное новое число поступает на обработку на шаг 1.

Пример:

Декодирование предыдущего сообщения: "110100111101101100111"
$0.110100111101101100111_2 = \frac{1735527}{2^{27}}$

Номер	Символ	Вероятность	Отрезок
1	к	0.1	[0,0.1]
2	о	0.1	[0.1,0.2]
3	н	0.2	[0.2,0.4]
4	с	0.1	[0.4,0.5]
5	т	0.2	[0.5,0.7]
6	а	0.2	[0.7,0.9]
7	$	0.1	[0.9,1]

Номер	Число	Отрезок	Символ	Разница	Длина
1	$\frac{1735527}{2^{27}}$	[0,0.1]	к	$\frac{1735527}{2^{27}}$	0.1
2	$\frac{8677635}{2^{26}}$	[0.1,0.2]	о	$\frac{9833743}{5\cdot 2^{26}}$	0.1
3	$\frac{9833743}{2^{25}}$	[0.2,0.4]	н	$\frac{15614283}{5\cdot 2^{25}}$	0.2
4	$\frac{15614283}{2^{25}}$	[0.4,0.5]	с	$\frac{10962551}{5\cdot 2^{25}}$	0.1
5	$\frac{10962551}{2^{24}}$	[0.5,0.7]	т	$\frac{2573943}{2^{24}}$	0.2
6	$\frac{12869715}{2^{24}}$	[0.7,0.9]	а	$\frac{5628319}{5\cdot 2^{24}}$	0.2
7	$\frac{5628319}{2^{24}}$	[0.2,0.4]	н	$\frac{11364379}{5\cdot 2^{24}}$	0.2
8	$\frac{11364379}{2^{24}}$	[0.5,0.7]	т	$\frac{2975771}{2^{24}}$	0.2
9	$\frac{14878855}{2^{24}}$	[0.7,0.9]	а	$\frac{15674019}{5\cdot 2^{24}}$	0.2
10	$\frac{15674019}{2^{24}}$	[0.9,1]	$	-	-

3. Адаптивное арифметическое кодирование

Несмотря на хорошие показатели сжатия (как минимум, не худшие, чем у алгоритма Хаффмена) рассмотренный метод требует двух проходов : первый для сбора частот символов, второй для собственно сжатия. Для уменьшения затрат времени и памяти, а также при отсутствии возможности передачи значений вероятностей символов был разработан адаптивный алгоритм. Он не требует знания таблицы вероятностей, поэтому является однопропроходным. Его принцип заключается в том, что при сопоставлении входящему символу кода меняется ход внутренних вычислений таким образом, что при следующем вхождении на обработу этого же символа ему может сопоставляться уже другой код. То есть алгоритм адаптируется под входящие для сжатия символы.

Суть адаптивного арифметического кодирования отличается от обычного арифметического кодирования лишь подходом к работе с вероятностями. Адаптивный алгоритм использует так называемые "веса".

Изначально вводится понятие заданного множества символов — это множество всех символов, которые могут входить в сообщение. Всем символам сопоставляется вес, от которого зависит вероятность. По мере обработки сообщения вес символа растёт пропорционально количеству его вхождений. Аналогично арифметическому кодированию для корректного окончания сжатия и декодирования необходимо ввести специальный символ конца сообщения.

3.1. Алгоритм адаптивного арифметического кодирования

В начале сжатия вес каждого символа равен 1. Символы выстроены в каком-то определённом порядке, по которому происходит выбор отрезка (например, в лексикографическом). Вероятность появления символа считается равной отношению веса символа к общему весу всех символов. После получения очередного символа и построения соответствующего отрезка его вес увеличивается на 1. Так происходит до тех пор, пока не встречен символ конца сообщения.

Пример: "Барабан$"

Заданное множество символов: {А,Б,Н,Р}

Расположение символов в лексикографическом порядке: Б,А,Н,Р,$;

Номер	Вход	Вес символов					Суммарный вес	Вероятность	Отрезок
		А	Б	Н	Р	$			
1	Б	1	$\boxed{1}$	1	1	1	5	$\frac{1}{5}$	$[\frac{1}{5},\frac{2}{5}]$
2	А	$\boxed{1}$	2	1	1	1	6	$\frac{1}{6}$	$[\frac{1}{5},\frac{7}{30}]$
3	Р	2	2	1	$\boxed{1}$	1	7	$\frac{1}{7}$	$[\frac{47}{210},\frac{16}{70}]$
4	А	$\boxed{2}$	2	1	2	1	8	$\frac{2}{8}$	$[\frac{47}{210},\frac{9}{40}]$
5	Б	3	$\boxed{2}$	1	2	1	9	$\frac{2}{9}$	$[\frac{113}{504},\frac{1697}{7560}]$
6	А	$\boxed{3}$	3	1	2	1	10	$\frac{3}{10}$	$[\frac{113}{504},\frac{157}{700}]$
7	Н	4	3	$\boxed{1}$	2	1	11	$\frac{1}{11}$	$[\frac{15541}{69300},\frac{10361}{46200}]$
8	$	4	3	2	2	$\boxed{1}$	12	$\frac{1}{12}$	$[\frac{10657}{47520},\frac{10361}{46200}]$

Итоговым отрезком будет $[\frac{10657}{47520},\frac{10361}{46200}]$.

Число, принадлежащее его внутренности и равное частному целого и минимально возможной натуральной степени двойки - это $\frac{470315}{2097152} = \frac{470315}{2^{21}} = 0.1110010110100101011_2$. Тогда кодом сообщения будет 1110010110100101011.

L(Барабан$)=19, тогда L_1=2,375 (бит/символ)

3.2. Обратное кодирование

Декодирование сообщения происходит аналогично обычному арифметическому декодированию, изменения касаются только шага 2. После получения нового числа необходимо перестроить покрытие отрезка [0,1] согласно новому распределению весов.

Пример: Декодирование предыдущего сообщения "11100101101001010 11".

$0.11100101101001010 11_2 = \frac{470315}{2^{21}}$

Номер	Вес символов					Суммарный вес	Число	Отрезок	Символ	Разница	Длина
	А	Б	Н	Р	$						
1	1	1	1	1	1	5	$\frac{470315}{2^{21}}$	$[\frac{1}{5}, \frac{2}{5}]$	Б	$\frac{254423}{5 \cdot 2^{21}}$	$\frac{1}{5}$
2	1	2	1	1	1	6	$\frac{254423}{2^{21}}$	$[0, \frac{1}{6}]$	А	$\frac{254423}{2^{21}}$	$\frac{1}{6}$
3	2	2	1	1	1	7	$\frac{763269}{2^{20}}$	$[\frac{5}{7}, \frac{6}{7}]$	Р	$\frac{100003}{7 \cdot 2^{20}}$	$\frac{1}{7}$
4	2	2	1	2	1	8	$\frac{100003}{2^{20}}$	$[0, \frac{2}{8}]$	А	$\frac{100003}{2^{20}}$	$\frac{1}{4}$
5	3	2	1	2	1	9	$\frac{100003}{2^{18}}$	$[\frac{3}{9}, \frac{5}{9}]$	Б	$\frac{37865}{3 \cdot 2^{18}}$	$\frac{2}{9}$
6	3	3	1	2	1	10	$\frac{113595}{2^{19}}$	$[0, \frac{3}{10}]$	А	$\frac{113595}{2^{19}}$	$\frac{3}{10}$
7	4	3	1	2	1	11	$\frac{567975}{3 \cdot 2^{18}}$	$[\frac{7}{11}, \frac{8}{11}]$	Н	$\frac{742701}{33 \cdot 2^{18}}$	$\frac{1}{11}$
8	4	3	2	2	1	12	$\frac{742701}{3 \cdot 2^{18}}$	$[\frac{11}{12}, 1]$	$	-	-

4. PPM (Prediction by Partial Matching)

Рассмотрим ещё один подход к работе вероятностями при арифметическом кодировании. Пусть вместо безусловных вероятностей появления символа будут оцениваться его условные вероятности при уже известных символах обработанного сообщения. Введём понятие *контекста* — части сообщения, непосредственно предшествующей данному символу и такой, что точно такая же часть уже встречалась в обработанном сообщении. Тогда, рассматривая контексты различной длины, зависящей от уже обработанного сообщения, мы получим алгоритм, имеющий название Prediction by Partial Matching (сокращённо PPM).

4.1. Алгоритм PPM

Введём сокращённое обозначение для части сообщения :

$$x_i^k = (x_i, x_{i+1}, \ldots, x_{k-1}, x_k), \text{ где } x_j \text{ — j-ый символ сообщения.}$$

Пусть последовательность первых t символов x_1^t уже обработана. Рассмотрим алгоритм обработки символа x_{t+1}:

1. Находится констекст x_{t-d+1}^t наибольшей длины d, не превосходящей заданной величины D.

2. Для всех возможных значений x_{t+1} вычисляются оценки условных вероятностей символа с использованием полученного контекста.

3. Значение символа x_{t+1} кодируется с учётом вероятности, полученнонной на шаге 2.

Пусть s — любая обработанная часть сообщения, полностью совпадающая с контекстом x_{t-d+1}^t (кроме него самого).

Для вычисления вероятности символа $x_{t+1} = a$ воспользуемся формулой:

$$\hat{\text{p}}(a|\mathbf{s}) = \frac{\tau_{\mathbf{t}}(\mathbf{s}, \mathbf{a})}{\tau_{\mathbf{t}}(\mathbf{s}) + \mathbf{1}}, \text{ при } \tau_{\mathbf{t}}(\mathbf{s}, \mathbf{a}) > \mathbf{0}$$

где $\tau_t(\mathbf{s}, \mathbf{a})$ — количество появлений символа a вслед за последовательностью s,
$\tau_t(\mathbf{s})$ — количество всех таких s.

Для корректной работы с вероятностями введём специальный *esc*-символ, сигнализирующий о том, что символ a ранее не встречался вслед за последовательностью s (т.е. $\tau_t(\mathbf{s}, a)$=0). Тогда шаг 2 будет иметь вид:

1. Если $\tau_t(\mathbf{s}, a) \neq 0$, то вероятность $\hat{\text{p}}(\text{a}|\mathbf{s})$ передаётся на шаг 3.

2. Если $\tau_t(\mathbf{s}, a) = 0$, то передаётся *esc*-символ, контекст уменьшается на один символ и вновь поступает на обработку на шаг 2.1 .

Таким образом, символ будет передан с учётом какого-то контекста (возможно, нулевой длины) или же как новый, не встречавшийся ранее символ.

Итоговая вероятность умножается на вероятность *esc*-символа при каждом его появлении:

$$\hat{\text{p}}(esc|\mathbf{s}) = \frac{1}{\tau_t(\mathbf{s}) + 1}$$

Для улучшения оценки вероятности был разработан метод исключений, суть которого заключается в уменьшении выборки последовательностей s для данного контекста, зная контекст длины на 1 большей.

Пусть изветен контекст x_{t-d+1}^{t} и пусть последоваельноть x_{t-d+1}^{t+1} раньше не встречалась, но обрабатывались последовательности $(x_{t-d+1}, \ldots, x_t, y)$ и $(x_{t-d+1}, \ldots, x_t, z)$. Тогда по алгоритму будет возвращён *esc*-символ и контекст уменьшится на единицу x_{t-d+2}^{t}. Для соответствующих ему последовательностей s будет производиться оценка условной вероятности символа x_{t+1}. Метод говорит, что не все s стоит рассмотривать, так как из-за передачи *esc*-символа уже известно, что символы y и z не могут следовать за x_{t-d+2}^{t}. Следовательно, можно уточнить формулу $\tau_t(\mathbf{s})$:

$$\tau'_t(\mathbf{s}) = \tau_t(\mathbf{s}) - \tau_t(x_{t-d+2}, \ldots, x_t, y) - \tau_t(x_{t-d+2}, \ldots, x_t, z)$$

Пример:

Рассмотрим работу данного алгоритма на примере, взятом из книги [1].

"IF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CAN"

Пусть D=5.

Тогда в графе "Контекст s" знаки # обозначают контексты нулевой длины. В графе "$\tau_t(\mathbf{s})$" записано число появлений данного контекста и, если он укорачивался, то также записаны через запятую число появлений укороченных контекстов. В графе "PPM" вероятности, посчитанные с помощью исключений, помечены штрихом.

Шаг	Буква	Контекст **s**	$\tau_t(\mathbf{s})$	PPM	
				$\hat{p}(esc\|\mathbf{s})$	$\hat{p}(a\|\mathbf{s})$
1	I	#	0	1	1/256
2	F	#	1	1/2	1/255
3	_	#	2	1/3	1/254
4	W	#	3	1/4	1/253
5	E	#	4	1/5	1/252
6	_	#	5		1/6
7	C	_	1,6	1/2×1/6′	1/251
8	A	#	7	1/8	1/250
9	N	#	8	1/9	1/249
10	N	#	9		1/10
11	O	N	1,10	1/2×1/9′	1/248
12	T	#	11	1/12	1/247
13	_	#	12		2/13
14	D	_	2,13	1/3×1/12′	1/246
15	O	#	14		1/15
16	_	O	1,15	1/2	3/15′
17	A	_	3,16	1/4	1/14′
18	S	A	1,17	1/2×1/16′	1/245
19	_	#	18		4/19
20	W	_	4		1/5
21	E	_ W	1		1/2
22	_	_ WE	1		1/2
23	W	_ WE _	1,1,1,5	1/2×1′ × 1′	2/5′
24	O	_ W	2,2,23	1/3 ×1′	2/22′
25	U	O	2,24	1/3 ×1/18′	1/244
26	L	#	25	1/26	1/243
27	D	#	26		1/27
28	_	D	1,27	1/2	6/25′
29	W	_	6		3/7′
30	E	_ W	3		2/4
31	_	_ WE	2		2/3
32	S	_ WE _	2,2,2,7,31	1/3×1′ × 1 × 1/3′	1/23′
33	H	S	1,32	1/2×1/25′	1/242
34	O	#	33		3/34
35	U	O	3		1/4
36	L	OU	1		1/2
37	D	OUL	1		1/2
38	_	OULD	1		1/2
39	D	OULD_	1,1,1,8	1/2×1′ × 1′ × 1′	1/4′
40	O	_ D	1		1/2
41	_	_ DO	1		1/2
42	A	_ DO _	1		1/2
43	S	_ DO _ AS	1		1/2
44	_	DO_ AS	1		1/2
45	W	O_ AS_	1		1/2
46	E	_ AS _ W	1		1/2
47	_	AS_ WE	1		1/2
48	C	S _ WE _	1,3	1/2	1/3′
49	A	_ WE _ C	1		1/2
50	N	WE _ CA	1		1/2

5. Заключение

В ходе проведённой работы были детально разобраны алгоритмы арифметического кодирования, адаптивного арифметического кодирования и Prediction by Partial Matching, а также приведены примеры, демонстрирующие работу каждого из них. Для обычной и адаптивной версии арифметического кодирования на языке C++ были отдельно написаны реализации сжатия и распаковки переданного сообщения, содержащиеся в приложении.

Стоит отметить, что все описанные алгоритмы имеют широкое практическое применение в настоящее время.

Так, различные версии статистического алгоритма PPM используются во многих известных архиваторах, таких как RAR, 7-Zip и WinZip. Однако, (алгоритм) PPM служит лишь для предсказывания частот появления символов, остновную часть обработки входящего потока данных производит алгоритм арифметического кодирования, поэтому для работы с этим популярным средством сжатия информации необходимо понимать принципы работы самого арифметического кодирования.

Список литературы

1. **Кудряшов Б.Д.** *Теория информации* — Санкт-Петербург: СПбГУ ИТМО, 2010. - 188с.
2. **Лидовский В.В.** *Теория информации: Учебное пособие* — М.: Компания Спутник+, 2004. - 111 с.
3. **Семенюк В.В.** *Экономное кодирование дискретной информации* — СПб.: СПбГИТМО (ТУ), 2001. — 115 с.

6. Приложение

Реализация сжатия приведённых примеров на C++:

Пример 1: "Константа$"

```
#include <iostream>

using namespace std;

int main(){
        int n=7; // количество символов
        int b[]={1,2,3,4,5,6,3,5,6,7}; // порядок вхождения символов в сообщение
        int k=sizeof(b)/sizeof(b[0]); // длина сообщения
        long double c[]={0,1}; // изначальный отрезок
        double d[]={0.1,0.1,0.2,0.1,0.2,0.2,0.1}; // вероятности появления символов
        long double a[n]; // массив для хранения границ отрезков
        a[0]=0;
        for (int j=1;j<=n;j++){
            a[j]=d[j-1]+a[j-1]; // заполнение массива с помощью вероятностей
            };
        int i=-1;
        do  {  //обработка сообщения до тех пор пока не встетился символ конца строки
                i++;
                double x=(c[1]-c[0])/k; // деление отрезка на длинну сообщения
                c[1]=c[0]+x*(a[b[i]]*k); // новая правая граница
                c[0]+=x*a[b[i]-1]*k; // новая левая граница
            } while (b[i]!=7); // условие встречи символа конца строки

        cout<<"["<<c[0]<<";"<<c[1]<<"]"; // итоговый отрезок

    return 0;
}
```

Декодирование примера 1 : получен код "110100111101101100111"

$0.110100111101101100111_2 = \frac{1735527}{2^{27}}$

```
#include <iostream>

using namespace std;

int main(){
        int n=7; // количество символов в таблице вероятностей
        long double z=(long double)1735527/134217728; // полученный код
        double d[]={0.1,0.1,0.2,0.1,0.2,0.2,0.1}; // таблица вероятностей
        long double a[n]; // массив для хранения границ отрезков
        a[0]=0;
        for (int j=1;j<=n;j++){
            a[j]=d[j-1]+a[j-1]; // заполнение массива с помощью вероятностей
            };
        int i=0;
        while (i!=7) { // условие на символ конца строки
            i=0;
            while (z>a[i]){  // определение отрезка
                i++;
                };
            cout <<i<< endl; // вывод номера соответствующего символа
            long double r=z-a[i-1]; // разница числа-кода и левой границы
            long double b=a[i]-a[i-1]; // длина отрезка
            z=(double)r/b; // новое число-код
        }
    return 0;
}
```

Пример 2: "Барабан$"

```
#include <iostream>

using namespace std;

int main(){
        int n=4+1; // мощность заданного множества + символ конца сообщения
        int b[]={2,1,4,1,2,1,3,5}; //порядок вхождения символов
        int w[]={1,1,1,1,1}; // массив для хранения весов символов
        int v=n; // общий вес всех символов
        long double c[]={0,1}; // отрезок для покрытия
        double d[n-1]; // массив для хранения вероятностей символов
        int k=-1;
        do { // цикл, работающий до встречи символа конца сообщения
          for (int i=0;i<n;i++){
            d[i]=(double)w[i]/v; // заполнение массива вероятностей с помощью весов
            }
          long double a[n]; // массив для хранения границ отрезков
          a[0]=0;
          for (int j=1;j<=n;j++){
            a[j]=d[j-1]+a[j-1]; // заполнение массива границ с помощью вероятностей
            };
          k++;
          double x=(double)(c[1]-c[0])/v; // деление отрезка на количество частей,
                                          равное суммарному весу
          c[1]=c[0]+x*(a[b[k]]*v); // новая правая граница отрезка
          c[0]+=x*a[b[k]-1]*v; // новая левая граница отрезка
          v++; // увеличение общего веса на 1
          w[b[k]-1]++; // увелечение веса соответствующего символа на 1
        } while (b[k]!=5); // условие встречи символа конца строки
        cout<<"["<<c[0]<<";"<<c[1]<<"]"; // вывод итогового отрезка
    return 0;
}
```

Декодирование примера 2 : получен код "11100101101001010 11"

$0.11100101101001010 11_2 = \frac{470315}{2^{21}}$

```
    #include <iostream>

using namespace std;

int main(){
        int n=4+1; // мощность заданного множества + символ конца сообщения
        int w[]={1,1,1,1,1}; // массив для хранения весов символов
        int v=n; // общий вес всех символов
        long double z=(long double)470315/2097152; // переданный код-число
        double d[n-1];  // массив для хранения вероятностей символов
        int i=0;
         while (i!=5) { // условие встречи символа конца сообщения
        for (int k=0;k<n;k++){
        d[k]=(double)w[k]/v; // заполнение массива вероятностей с помощью весов
         };
        long double a[n]; // массив для хранения границ отрезков
        a[0]=0;
        for (int j=1;j<=n;j++){
            a[j]=d[j-1]+a[j-1]; // заполнение массива границ с помощью вероятностей
            };
            i=0;
            while (z>a[i]){ // определение отрезка
                i++;
                };
            long double r=z-a[i-1]; // разница числа-кода и левой границы
            long double b=a[i]-a[i-1]; // длина отрезка
            z=(double)r/b; // новое число-код
             cout <<i<< endl; // вывод номера символа
             v++; // увеличение общего веса символов на 1
            w[i-1]++; // увеличение веса соответствующего символа на 1
        }

    return 0;
}
```