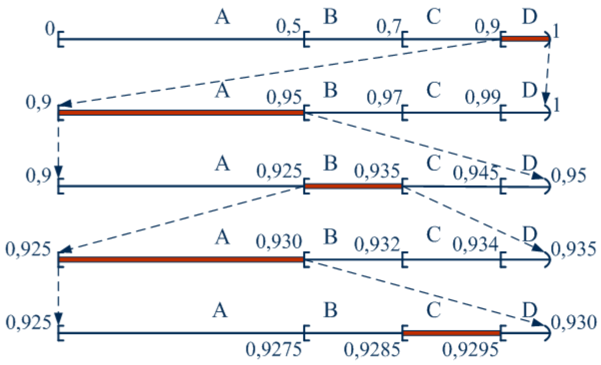
Арифметическое кодирование – Отчётное ДЗ

Идея кодирования

Принцип кодирования: сообщение представляется в виде вещественного числа, однозначно его определяющего.

Для источника с 4-мя состояниями, необходимо закодировать сообщение «DABAC»

A	B	C	D
0,5	0,2	0,2	0,1



В качестве кода можно выбрать любое число из интервала $[0{,}9285; 0{,}9295)$, например, $0{,}929$.

Последовательности символов $\{x(1),\dots,x(M)\}$, созданных дискретным источником сообщений X с известным алфавитом и вероятностями появления символов $\{(x_k, p_k)\}_{k=1}^{N}$, ставится в соответствие некоторое число, однозначно задающее данную последовательность.

Арифметическое кодирование

Задача – закодировать сообщение $\{x(1),\dots,x(M)\}$ источника с алфавитом $\{x_k\}_{k=1}^{N}$, для которого задано разбиение на вероятностные интервалы $\Delta_k = [L_{x_k}; U_{x_k})$.

Шаг 1. Установить границы текущего интервала $L=0, U=1$ и счетчик символов $j=1$.

Шаг 2. Определить новые границы текущего интервала:

$$W = U - L,$$
$$U = L + W \times U_{x(j)},$$
$$L = L + W \times L_{x(j)}.$$

Шаг 3. Увеличить счетчик символов $j = j+1$. Если $j > M$, то перейти на шаг 4, иначе – на шаг 2.

Шаг 4. Выбрать число $B \in [L; U)$, содержащее наименьшее число цифр.

Задание. Для бинарного источника с распределением вероятностей

z_1	z_2
0,9	0,1

построить выходной битовый код для всех возможных пар символов, сравнить средние битовые затраты с энтропией.

Проверить правильность кодирования, использую программу task1.m:

```
function task1
% арифметическое кодирование
P = [0.9 0.1]; % вероятности
S = [1 1]; % сообщение
L_cur = 0;
U_cur = 1;
L = cumsum([0 P(1:end-1)]);
U = cumsum(P);
for i = 1:length(S)
   W = U_cur - L_cur;
   U_cur = L_cur + W * U(S(i));
   L_cur = L_cur + W * L(S(i));
end
L_cur_bin = dec2bin(L_cur * 2^10)
U_cur_bin = dec2bin(U_cur * 2^10)
```

Энтропия источника $H = -0{,}1 \cdot \log_2 0{,}1 - 0{,}9 \cdot \log_2 0{,}9 \approx 0{,}47$ бита.

Проведем кодирование пар символов. Получим следующие интервалы:

AA: $[0;\, 0{,}81) = [0{,}000\ldots;\, 0{,}1100111101\ \ldots)$, ***Число*** = 0,1 (1 бит);

AB: $[0{,}81;\, 0{,}9) = [0{,}1100111101\ \ldots;\, 0{,}11100110\ 01\ldots)$, ***Число*** = 0,111 (3 бита);

BA: $[0{,}9;\, 0{,}99) = [0{,}11100110\ 01\ldots;\, 0{,}11111101\ 01\ldots)$, ***Число*** = 0,1111 (4 бита);

BB: $[0{,}99;\, 1) = [0{,}11111101\ 01\ldots;\, 0{,}11111111\ 11\ldots)$, ***Число*** = 0,1111111 (7 битов).

Определим средние битовые затраты:

$$R = \frac{1 \cdot 0{,}81 + 3 \cdot 0{,}09 + 4 \cdot 0{,}09 + 7 \cdot 0{,}01}{2} = 0{,}755 \ (\text{бита}).$$

Битовые затраты заметно превышают значение энтропии.

Получили значение больше энтропии, что не противоречит теории. ►

Из-за сближения в процессе кодирования верхней U и нижней L границ интервала, после окончания процедуры арифметического кодирования M символов первые биты в двоичной записи чисел $L = 0, l_1 l_2 \ldots$ и $U = 0, u_1 u_2 \ldots$ будут совпадать. Пусть количество первых совпавших разрядов $\{b_j\}$ равно K: $u_j = l_j = b_j$, $j = 1, \ldots, K$. Тогда, поскольку $L < U$, первые несовпадающие биты: $l_{K+1} = 0$, $u_{K+1} = 1$, а сами числа L и U можно представить в виде, указанном на слайде выше.

Будем считать, что не все u_k равны 0 при $k \geq K+2$. Действительно, если $U = 0, b_1 b_2 \ldots b_k 100\ldots$, то можно записать $U = 0, b_1 b_2 \ldots b_k 01111\ldots$. Тогда в качестве кодового числа можно всегда выбирать $B = 0, b_1 b_2 \ldots b_K 1$. Поскольку число B всегда заканчивается единичным битом, этот бит можно не хранить и передавать лишь $b_1 b_2 \ldots b_K$ (декодер будет приписывать дополнительный единичный бит автоматически).

Выбор кодового числа

После окончания кодирования границы интервала L и U можно представить в виде

$$U = 0,b_1 b_2 \dots b_K\, 1\, u_{K+2} u_{K+3} \dots _{(2)} = \sum_{k=1}^{K} b_k 2^{-k} + \frac{1}{2^{K+1}} + \overbrace{\sum_{k=K+2}^{\infty} u_k 2^{-k}}^{\le 1/2^{K+1}},$$

$$L = 0,b_1 b_2 \dots b_K\, 0\, l_{K+2}\, l_{K+3} \dots _{(2)} = \sum_{k=1}^{K} b_k 2^{-k} + \underbrace{\sum_{k=K+2}^{\infty} l_k 2^{-k}}_{\le 1/2^{K+1}},$$

где $b_1, b_2, \dots, b_K$ – совпавшие биты в двоичном представлении чисел L и U.

Не все u_k, $k \ge K+2$ равны 0, и не все l_k, $k \ge K+2$ равны 1. Поэтому в качестве кодового числа можно использовать $B = b_1\, b_2 \dots b_K\, 1$. Последний бит – всегда 1, следовательно, его не нужно хранить!

Принимая во внимание вышесказанное, определим средние битовые затраты для кодирования пары символов из предыдущего примера:

AA – кодовое число 0,1 (0 битов);

AB – кодовое число 0,111 (2 бита);

BA – кодовое число 0,1111 (3 бита);

BB – кодовое число 0,1111111 (6 битов).

Следовательно, $R = \dfrac{0 \cdot 0{,}81 + 2 \cdot 0{,}09 + 3 \cdot 0{,}09 + 6 \cdot 0{,}01}{2} = 0{,}255 \; < H = 0{,}47$. Получили, что битовые затраты оказались меньше энтропии! Чтобы понять, как такое стало возможным, рассмотрим процесс декодирования сообщения по числу B.

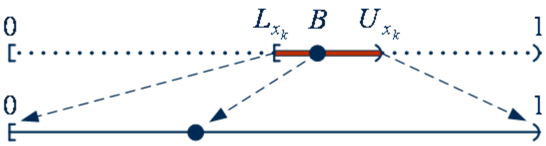
Алгоритм декодирования

Задача – декодировать сообщение из M символов.

Шаг 1. Ввести кодовое число B и установить счетчик символов $j = 0$.

Шаг 2. Найти интервал Δ_k, в который попадает число B. Выдать символ $x(j) = x_k$.

Шаг 3. Привести текущий интервал $\left[L_{x_k}; U_{x_k}\right)$ к $[0;1)$:



$$B = \left(B - L_{x_k}\right) / \left(U_{x_k} - L_{x_k}\right)$$

Шаг 4. Увеличить счетчик символов $j = j + 1$. Если $j \le M$, то перейти на шаг 2, иначе закончить работу.

Задание. Для источника со статистикой

z_1	z_2
0,9	0,1

декодировать сообщение из $M = 6$ символов по кодовому числу 0,111 (оно кодируется 2-мя битами), используя программу task2.m:

```
function task2
% арифметическое декодирование
P = [0.9 0.1]; % вероятности
M = 6; % количество символов в сообщении
B = 0.8750; % B = 0.111
L = cumsum([0 P(1:end-1)]);
U = cumsum(P);
S = zeros(1, M);
for i = 1:M
   S(i) = max(find(L <= B));
   B = (B - L(S(i))) / (U(S(i)) - L(S(i)));
end
S
```

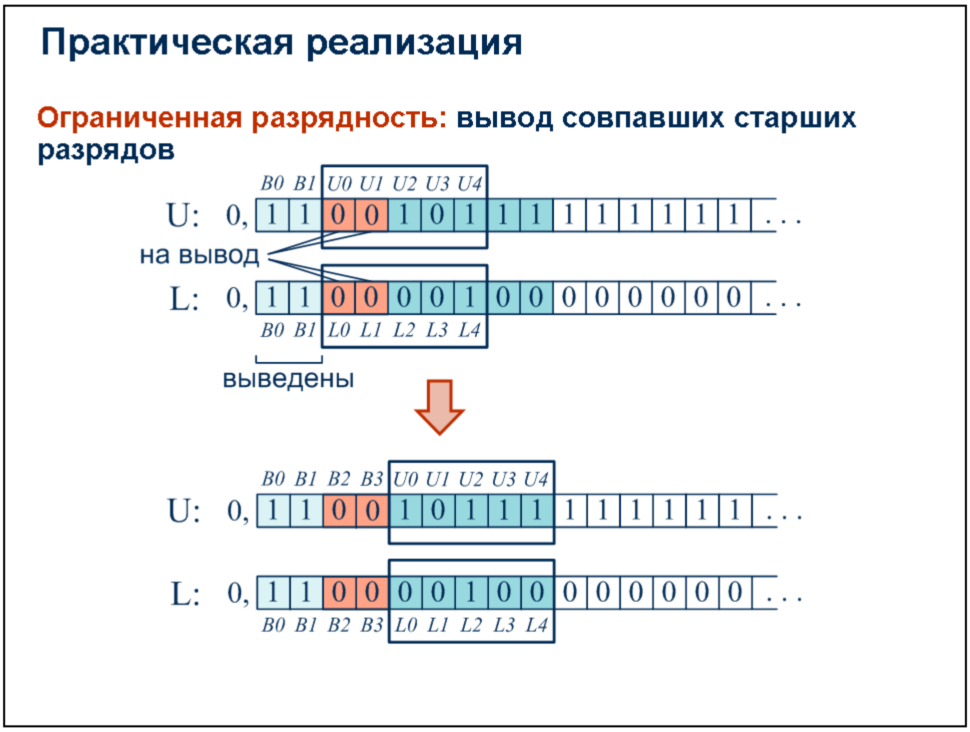
◄ Чтобы не проводить вычисления вручную, воспользуемся программой task2.m. Для указанного кодового числа получим сообщение $ABAAAB$. Но в предыдущем примере кодовое число 0,111 соответствовало последовательности AB! Более того, можно было бы задать $M > 6$, т.е. дальше продолжить декодирование (например, можно запустить программу для $M = 10$).

Таким образом, одного кодового числа недостаточно для того, чтобы однозначно определить сообщение – декодер просто не будет знать, когда ему остановиться. Поэтому на практике используют специальный символ «конец сообщения», или в качестве дополнительной информации передают длину сообщения M. При этом битовые затраты получаются близкими к энтропии, но не меньше её. ►

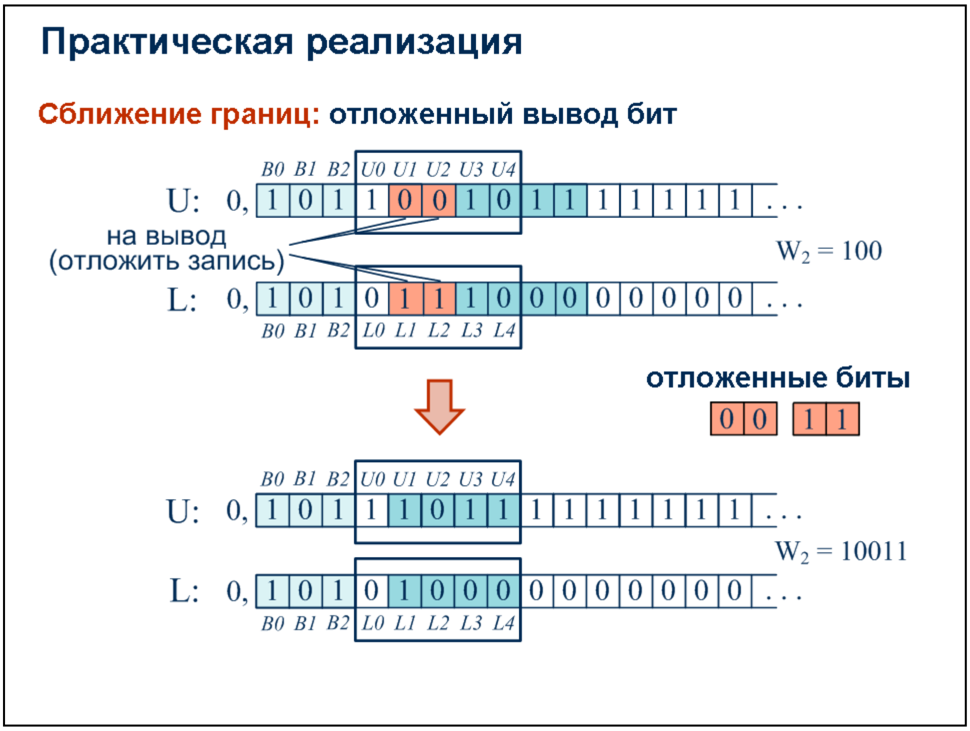
Для реализации арифметического кодера используются числа с фиксированной точкой, при этом предполагается, что их разрядность неограниченная. Однако для представления значений L и U реально могут использоваться регистры конечной разрядности, которой хватает только для кодирования очень коротких сообщений.

В начале работы кодера нижняя граница интервала представляется в виде дроби с бесконечным числом нулей $L_0 = 0 = 0{,}0000\ldots$, а верхняя – в виде дроби с бесконечным числом единиц $U_0 = 1 = 0{,}11111\ldots$ (все вычисления проводятся в двоичной системе).

После того, как в процессе работы кодера несколько старших разрядов регистров U и L совпали, они уже не могут измениться и не участвуют в дальнейшей работе кодера. Следовательно, они могут быть выдвинуты из регистров и записаны в выходной поток, а в освободившиеся младшие разряды для L попадут нули, а для U – единицы.



Проблема возникает, если в процессе кодирования верхняя и нижняя границы интервала отличаются в старших разрядах, но длина интервала недопустимо мала. В этом случае слишком маленькая длина интервала может не позволить закодировать очередной символ, так как при выполнении целочисленной нормализации значения регистров U и L не изменятся и алгоритм зациклится. С этой проблемой можно бороться, отслеживая сближение верхней и нижней границы. Если длина интервала $W = U - L$ на очередной итерации стала недопустимо малой, то выполняется принудительное расширение интервала с помощью выдвигания из регистров одного или нескольких старших разрядов (за исключением самого старшего), ещё не готовых к выводу в выходной поток.



При этом сама запись выдвинутых разрядов в поток откладывается, так как их окончательные значения ещё неизвестны и станут известны только после того, как совпадут старшие разряды регистров, не участвовавшие в выдвигании.

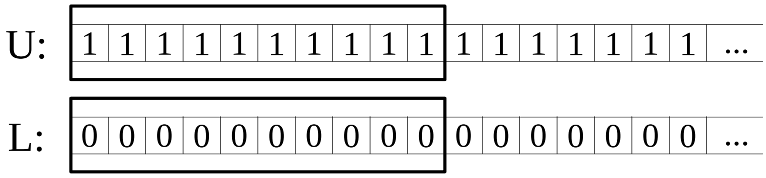
Задание. Для источника со статистикой

A	B	C	D
0,6	0,2	0,1	0,1

закодировать сообщение «DABAC», используя 10-разрядные регистры.

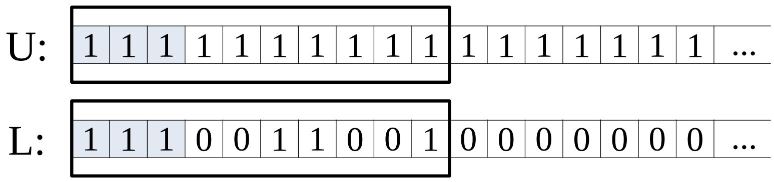
◄ Рассмотрим процесс кодирования.

Начальные значения регистров:

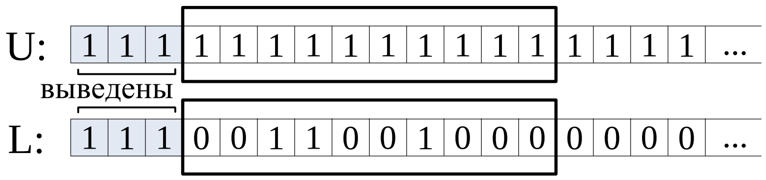


W = 111111111

Кодируем символ «D»:

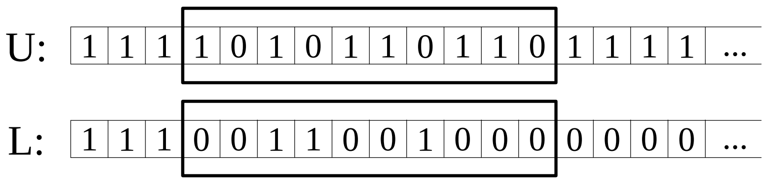


Совпали три старших разряда, их можно записать в выходной поток и выдвинуть из регистров. При этом получим:



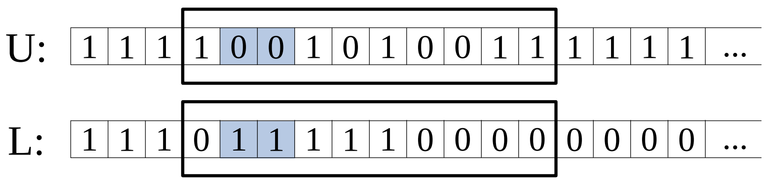
W = 1100110111

Кодируем символ «A»:



W = 0111101110

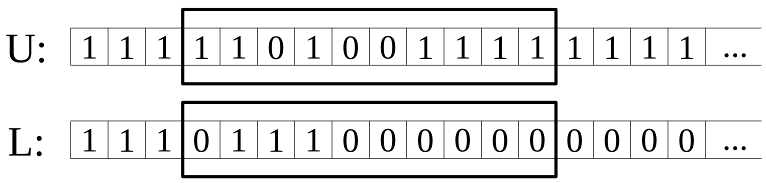
Кодируем символ «B»:



W = 0001100011

Для новых границ получили малую длину интервала, поэтому производим отложенный вывод двух бит, при этом границы изменятся следующим образом:

отложенные биты 0 0 1 1



W = 0110001111

Кодируем символ «A»:

U: 1 1 1 1 0 1 0 1 0 1 1 1 1 1 1 1 1 ...

L: 1 1 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 ...

W = 0011101111

Границы опасно сблизились, поэтому откладываем ещё один бит:

отложенные биты 0 0 0 1 1 1

U: 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 ...

L: 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 ...

W = 0111011111

Кодируем символ «С»:

U: 1 1 1 1 1 0 0 1 0 1 1 1 1 1 1 1 1 ...

L: 1 1 1 1 0 1 1 1 1 1 1 1 1 0 0 0 0 ...

Старшие разряды совпали и равны 1, тогда отложенные биты равны 0, в итоге получаем следующие границы интервала:

U: 1 1 1 1 0 0 0 1 0 0 1 0 1 1 1

отложенные биты

L: 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1

В качестве битового кода выбираем 11110001, что соответствует кодовому числу 0,94141. ►

Замечание. В приведенном выше примере при кодировании последнего символа старшие биты совпали, что позволило определить, чему равны отложенные биты и вывести их в выходной поток. Рассмотрим случай, когда после завершения кодирования сообщения старшие биты в **U** и **L** не совпадают, причем некоторое число N бит уже было отложено и регистры имеют вид:

U: 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 1 1 ...

L: 1 1 1 0 0 0 1 1 1 1 1 1 1 0 0 0 0 ...

С учетом отложенных битов:

U: 1 1 1 1 0 ··· 0 0 1 0 0 1 0 1 1 1 1 1 1 1 ...

N отложенных битов

L: 1 1 1 0 1 ··· 1 0 0 1 1 1 1 1 1 1 0 0 0 0 ...

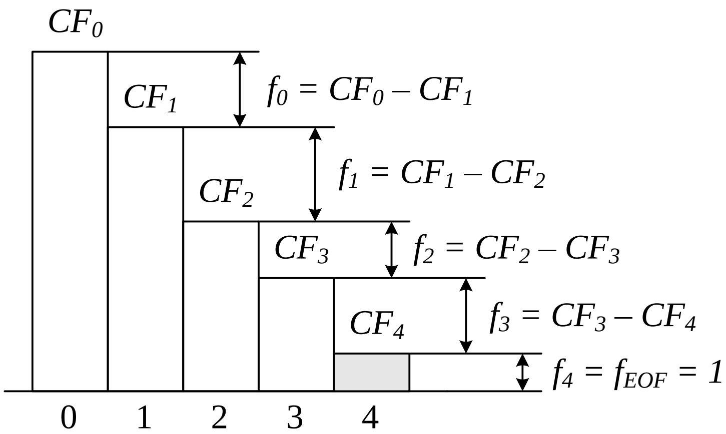
Если значение регистра $\mathbf{L} < 0\ \underbrace{1...1}_{N\ \text{битов}}\ 0\,0...$ (как в приведенном примере), то в качестве кодового можно выбрать следующее число:

1	1	1	0	1	···	1	1

Здесь старший разряд регистра равен 0, за ним следуют $N+1$ единичных битов. В случае, когда $\mathbf{L} \ge 0\,\underbrace{1...1}_{N\ \text{битов}}\,0\,0...$, выбираем кодовое число, в котором старший бит равен 1, и к нему дописываются $N+1$ нулевых битов:

1	1	1	1	0	···	0	0

Накопленная гистограмма и окончание кодирования



Статистику арифметического кодера удобно хранить в виде накопленной (кумулятивной) гистограммы, как показано на рис. выше. Здесь f_k – количество появлений k-го символа, а $CF_k = \sum_{i=k}^{N} f_k$ – «накопленная ненормированная частота» символа, с помощью которой удобно представлять границу интервала. Значение частоты символа (равное длине интервала W) можно получить из разности соседних накопленных частот. Величина CF_0 равна длине всего «единичного» отрезка, на котором производится кодирование. Реальная частота (оценка вероятности) символа вычисляется по формуле $p_k = \dfrac{CF_k - CF_{k-1}}{CF_0}$.

Обычно накопленная гистограмма состоит не из N, а из $N+1$ столбцов. В последнем столбце хранится частота специального символа, который помещается кодером в выходной поток после всех символов, подлежащих кодированию. Этот символ всегда будет встречаться в сообщении только один раз и обозначает конец файла. Он необходим для корректной работы декодера, который, декодировав его, завершает работу. Очевидно, символ конца файла (обычно его называют EOF-символом) не нужен, если декодеру заранее известно, сколько символов находится в закодированном потоке, но обычно декодер такой информации не имеет.

Адаптивное моделирование источника без памяти

В рассмотренных методах кодирования источников сообщений предполагалось, что распределение вероятностей $\{p(x_k)\}_{k=1}^{N}$ появления символов задано. Вместе с тем, статистическая модель источника не всегда известна, а также может изменяться в процессе обработки. Тогда построение модели осуществляется одновременно с кодированием.

В качестве начального распределения вероятностей источника можно выбрать равномерное: $p_k = 1/N$, $k = 1,..., N$, а затем, закодировав первый символ сообщения $x(1) = x_j$, нужно внести изменения в модель, повысив вероятность символа x_j: $p_j = 2/(N+1)$, $\forall k \neq j \quad p_k = 1/(N+1)$. Декодер, начиная работать с тем же распределением вероятностей, что и кодер, после декодирования первого символа сообщения затем повышает в модели источника соответствующую вероятность по тому же правилу, которое использовал кодер. Такая адаптация статистических моделей производится после каждого кодирования/декодирования очередного символа. Фактически, в качестве статистической модели при этом используется гистограмма частот, полученная по выборке из обработанных символов. По мере обработки (кодирования или декодирования) сообщения объем выборки растет, и накопленная гистограмма все более точно описывает истинное распределение вероятностей стационарного источника. Очевидно, что для адекватного описания распределения вероятностей объем выборки должен намного превышать количество ячеек в гистограмме, т.е. количество символов N в алфавите источника. Поэтому чем меньше символов в алфавите, тем быстрее «настраивается» модель источника.

С точки зрения практической реализации адаптивное арифметическое кодирование отличается от неадаптивного только наличием функции обновления накопленной гистограммы частот.

Отметим, что увеличение частоты одного символа в накопленной гистограмме влечёт за собой увеличение накопленной частоты всех следующих за ним символов, поэтому адаптивное моделирование является вычислительно сложной задачей и замедляет работу арифметического кодера.

Кроме того, при достижении некоторой максимальной накопленной частотой некоторого наибольшего значения необходимо проводить нормализацию всей накопленной гистограммы. Самый простой способ нормализации – деление частоты каждого символа на 2.

Домашнее задание (отчётное)

1. *MATLAB*. Для источника с 4-мя состояниями:

   $p(A)=0{,}2;\;\; p(B)=0{,}5;\;\; p(C)=0{,}2;\;\; p(D)=0{,}1$

   закодировать и декодировать по полученному кодовому числу сообщение «CBAABD», используя регистры «бесконечной» разрядности. (Для этого модифицировать task1.m, task2.m). Привести полученный скрипт и код сообщения. В качестве сообщения вместо CBAABD использовать по вариантам:
   1. CBABD
   2. DABBC
   3. BBADC
   4. ADCBB
   5. BABDC
   6. CABBD
   7. BACDB
   8. ABCDB
   9. BDCAB
   10. AAAAD
   11. DAAAC
2. Изучить исходные коды неадаптивного арифметического кодера arcode.c. При кодировании в данной программе используется входной двоичный файл, в каждом байте которого содержится символ, подлежащий кодированию. Таким образом, максимальная длина алфавита кодера равна 256 символам.
3. Изменить arcode.c, реализовав адаптивную модель для арифметического кодера, которая обновляет гистограмму частот после кодирования/декодирования каждого обработанного символа сообщения. В отчет включить модифицированный программный код соответствующей процедуры update_model(int symbol).
4. Сравнить эффективность сжатия данных неадаптивного (с фиксированной гистограммой, найденной в результате предварительного статистического анализа обрабатываемых данных) и адаптивного кодера (полученного в п.3) на различных типах данных, полученных как естественным (текстовые, исполняемые файлы, исходные коды, зашумленные и незашумленные проквантованные изображения из прошлого ДЗ), так и искусственным путем. В последнем случае необходимо сгенерировать входные данные с различной статистикой символов, а также входные данные с резко изменяющейся на протяжении файла (нестационарной) статистикой: Например, создав наборы данных $x_1, x_2, \ldots, x_N$ и $y_1, y_2, \ldots, y_N$ с <u>существенно</u> различающимися статистиками, изучить характеристики адаптивного кодера при обработке последовательностей $x_1, x_2, \ldots, x_N, y_1, y_2, \ldots, y_N$ и $x_1, y_1, x_2, y_2, \ldots, x_N, y_N$. Исследование должно быть выполнено в общей сложности не менее чем на 5 различных файлах размера ~ 100 кБ.