Лекция 5: Арифметическое кодирование

А. М. Шур

Кафедра алгебры и фундаментальной информатики УрФУ

11 апреля 2020 г.

Коды Хаффмана, энтропия и оптимальность

- Даны алфавит $\Sigma = \{x_1, \ldots, x_k\}$ и случайная величина $\xi = (x_1\,{}_{|p_1}, \ldots, x_k\,{}_{|p_k})$
- Строим по ξ код Хаффмана $\{C_1, \ldots, C_k\} \subset \{0,1\}^*$ и “оптимально” кодируем тексты, сгенерированные ξ, заменяя каждое вхождение x_i на C_i $(i = 1, \ldots, k)$
- Насколько это в действительности оптимально?

Коды Хаффмана, энтропия и оптимальность

- Даны алфавит $\Sigma = \{x_1, \ldots, x_k\}$ и случайная величина $\xi = (x_{1\,|p_1}, \ldots, x_{k\,|p_k})$
- Строим по ξ код Хаффмана $\{C_1, \ldots, C_k\} \subset \{0,1\}^*$ и “оптимально” кодируем тексты, сгенерированные ξ, заменяя каждое вхождение x_i на C_i $(i = 1, \ldots, k)$
- Насколько это в действительности оптимально?
- Матожидание длины кода символа $E_{Huff} = \sum_{i=1}^{k} p_i|C_i|$
- Оптимум по теоремам Шеннона равен $E_{Opt} = H(\xi) = \sum_{i=1}^{k} p_i \log \frac{1}{p_i}$
- По неравенству Крафта (см. лекцию 3) $\sum_{i=1}^{k} 2^{-|C_i|} = 1$
  - равенство следует из полноты кодов Хаффмана

$\Rightarrow$ $\eta = (x_{1\,|q_1}, \ldots, x_{k\,|q_k})$, где $q_i = 2^{-|C_i|}$, — случайная величина

$\Rightarrow$ E_{Huff} — сумма произведений вероятностей ξ на логарифмы вероятностей η

$\Rightarrow$ $E_{Huff} > E_{Opt} \iff \eta \neq \xi$

Коды Хаффмана, энтропия и оптимальность

- Даны алфавит $\Sigma = \{x_1, \ldots, x_k\}$ и случайная величина $\xi = (x_{1\,|p_1}, \ldots, x_{k\,|p_k})$
- Строим по ξ код Хаффмана $\{C_1, \ldots, C_k\} \subset \{0,1\}^*$ и “оптимально” кодируем тексты, сгенерированные ξ, заменяя каждое вхождение x_i на C_i $(i = 1, \ldots, k)$
- Насколько это в действительности оптимально?
- Матожидание длины кода символа $E_{Huff} = \sum_{i=1}^{k} p_i |C_i|$
- Оптимум по теоремам Шеннона равен $E_{Opt} = H(\xi) = \sum_{i=1}^{k} p_i \log \frac{1}{p_i}$
- По неравенству Крафта (см. лекцию 3) $\sum_{i=1}^{k} 2^{-|C_i|} = 1$
  - равенство следует из полноты кодов Хаффмана

$\Rightarrow$ $\eta = (x_{1\,|q_1}, \ldots, x_{k\,|q_k})$, где $q_i = 2^{-|C_i|}$, — случайная величина

$\Rightarrow$ E_{Huff} — сумма произведений вероятностей ξ на логарифмы вероятностей η

$\Rightarrow$ $E_{Huff} > E_{Opt} \iff \eta \neq \xi$

★ Код Хаффмана оптимален среди всех возможных способов кодирования, если все вероятности равны отрицательным степеням двойки

★ В остальных случаях алгоритм Хаффмана “округляет” каждую вероятность до одной из соседних степеней двойки

- код становится неоптимальным, поскольку кодируется не “настоящее” распределение ξ, а его “округление” η

Пример: $p_1 = p_2 = 0.4$, $p_3 = 0.2$. Коды символов получат длины $1, 2, 2$, т.е. $q_1 = 0.5$, $q_2 = q_3 = 0.25$. Имеем

$E_{Opt} = 0.8 \cdot \log 2.5 + 0.2 \log 5 \approx 1.52$ бит на символ

$E_{Huff} = 0.4 \cdot \log 2 + 0.6 \cdot \log 4 = 1.6$ бит на символ

Идея арифметического кодирования

Поскольку код Хаффмана — лучший среди префиксных кодов, для улучшения его результатов надо “выйти из плоскости” наличия кода у каждого отдельного символа и кодировать весь текст в целом.

Идея арифметического кодирования

Поскольку код Хаффмана — лучший среди префиксных кодов, для улучшения его результатов надо "выйти из плоскости" наличия кода у каждого отдельного символа и кодировать весь текст в целом.

Построим кодирующую функцию, которая каждому тексту, сгенерированному ξ, ставит в соответствие подполуинтервал $[i, j)$ полуинтервала $[0, 1)$

- Длина интервала равна вероятности текста
- Интервалы, соответствующие текстам одной длины, не пересекаются (и в объединении дают $[0, 1)$)
- Интервал, в который отображается текст, содержится в любом интервале, в который отображается префикс текста
- Кодом текста является самая короткая двоичная запись числа из интервала, в который отображен текст
  - так как все числа имеют вид $0.[\text{двоичная строка}]$, кодом является двоичная строка (мантисса)

Идея арифметического кодирования

Поскольку код Хаффмана — лучший среди префиксных кодов, для улучшения его результатов надо "выйти из плоскости" наличия кода у каждого отдельного символа и кодировать весь текст в целом.

Построим кодирующую функцию, которая каждому тексту, сгенерированному ξ, ставит в соответствие подполуинтервал $[i, j)$ полуинтервала $[0, 1)$

- Длина интервала равна вероятности текста
- Интервалы, соответствующие текстам одной длины, не пересекаются (и в объединении дают $[0, 1)$)
- Интервал, в который отображается текст, содержится в любом интервале, в который отображается префикс текста
- Кодом текста является самая короткая двоичная запись числа из интервала, в который отображен текст
  - так как все числа имеют вид $0.[\text{двоичная строка}]$, кодом является двоичная строка (мантисса)

★ Для единственности декодирования потребуется символ конца текста, либо отдельная передача длины текста, либо дописывание нулей после последней единицы в мантиссе

- первые два способа потребуют порядка $\log n$ дополнительных бит, а третий — чаще всего 0; для теоретических целей нужно рассматривать его (для практических третьему способу понадобится тот же $\log n$ бит, чтобы обозначить, где закончился код; проще всего хранить длину текста или кода)

Схема арифметического кодирования

Пусть $\xi = (x_{\mathbf{1}\,|\,p_{\mathbf{1}}}, \ldots, x_{k\,|\,p_k})$ — дискретная случайная величина, T — текст, сгенерированный ξ. Разобьем интервал $[0, 1)$ на k интервалов с длинами p_i:

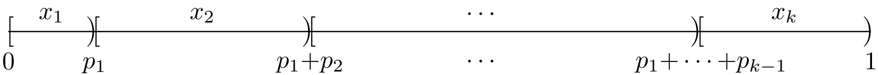
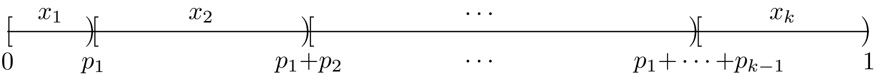


Схема арифметического кодирования

Пусть $\xi = (x_{1\,|\,p_1}, \ldots, x_{k\,|\,p_k})$ — дискретная случайная величина, T — текст, сгенерированный ξ. Разобьем интервал $[0, 1)$ на k интервалов с длинами p_i:



Текст $T = x_{i_1}x_{i_2}\cdots x_{i_n}$ кодируется посимвольно по следующим правилам:

- для x_{i_1} берется интервал, помеченный этим символом
- для $x_{i_1}\cdots x_{i_r}x_{i_{r+1}}$ берется интервал для $x_{i_1}\cdots x_{i_r}$, на него проектируется “разметка” интервала $[0, 1)$ и берется подинтервал, помеченный $x_{i_{r+1}}$

- Пример для $T = x_2x_k$:

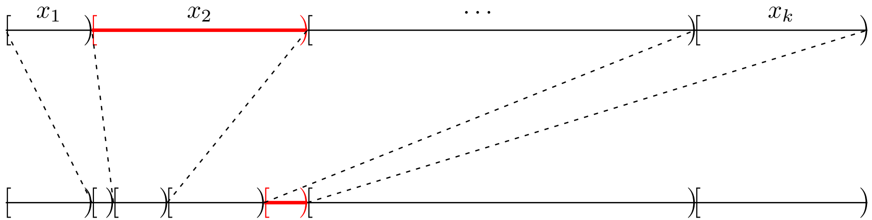
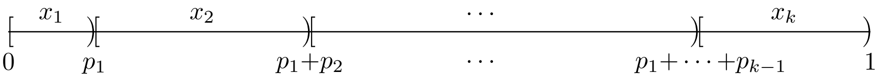


Схема арифметического кодирования

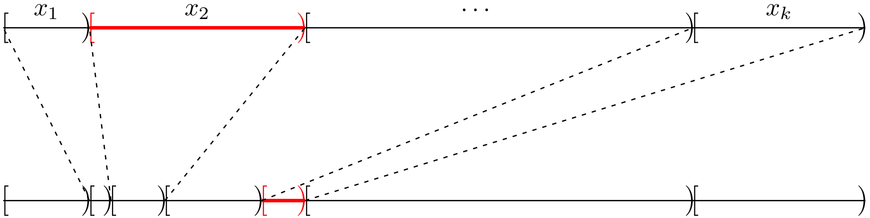
Пусть $\xi = (x_{1\,|p_1}, \ldots, x_{k\,|p_k})$ — дискретная случайная величина, T — текст, сгенерированный ξ. Разобьем интервал $[0,1)$ на k интервалов с длинами p_i:



Текст $T = x_{i_1}x_{i_2}\cdots x_{i_n}$ кодируется посимвольно по следующим правилам:

- для x_{i_1} берется интервал, помеченный этим символом
- для $x_{i_1}\cdots x_{i_r}x_{i_{r+1}}$ берется интервал для $x_{i_1}\cdots x_{i_r}$, на него проектируется "разметка" интервала $[0,1)$ и берется подинтервал, помеченный $x_{i_{r+1}}$

- Пример для $T = x_2x_k$:



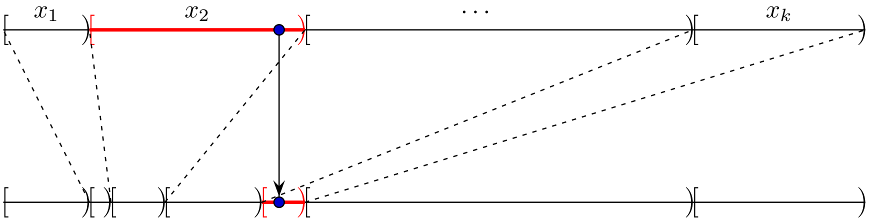
- По окончании процесса в полученном интервале берется рациональная точка со знаменателем вида 2^m с наименьшим m (она единственна - проверьте!)
- Код текста — мантисса двоичной записи этой точки
  - про нули в конце — потом

Схема декодирования

Число $z \in [0,1)$ декодируется в текст $T = x_{i_1} x_{i_2} \cdots x_{i_n}$ по следующим правилам:

- x_{i_1} — метка интервала, в котором лежит z; объявляем интервал текущим
- на каждом последующем шаге проектируем $[0,1)$ на текущий интервал; символ, в новый интервал которого попало z — очередной символ текста, а сам интервал становится текущим

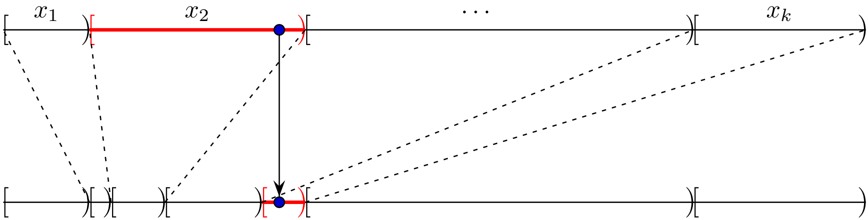
- Пример (два шага):



Число $z \in [0,1)$ декодируется в текст $T = x_{i_1} x_{i_2} \cdots x_{i_n}$ по следующим правилам:

- x_{i_1} — метка интервала, в котором лежит z; объявляем интервал текущим
- на каждом последующем шаге проектируем $[0,1)$ на текущий интервал; символ, в новый интервал которого попало z — очередной символ текста, а сам интервал становится текущим

- Пример (два шага):



Когда заканчивать?

- Если известна длина n текста — то после n итераций
- Если есть символ конца текста — то после его декодирования

★ Можно использовать дополнительные нули после последней единицы в коде:

- кодер приписывает столько нулей, сколько итераций не менялось число с самой короткой двоичной записью в текущем интервале
- декодер работает, пока z не станет числом с кратчайшей двоичной записью в текущем интервале, плюс столько итераций, столько дописано нулей

Качество кодирования

- Текст T можно рассматривать как результат единичного эксперимента над случайным вектором $(\xi, \ldots, \xi)$ длины n, энтропия которого равна $n \cdot H(\xi)$
  - Это и есть ожидаемая длина кода
- По теоремам Шеннона можно считать, что есть $2^{n \cdot H(\xi)}$ возможных текстов, вероятность каждого из которых равна $2^{-n \cdot H(\xi)}$
- На каждом интервале длины l, $2^{-m} \leqslant l < 2^{-m+1}$, есть число, записываемое дробью со знаменателем 2^m, т.е. двоичная дробь длины m

$\Rightarrow$ арифметический кодер сопоставит тексту последовательность длины $\lceil n \cdot H(\xi) \rceil$
  - Не учтены поправка в минус (может попасться точка со знаменателем 2^{m-c}) и поправка в плюс (нули в конце), но обе являются маленькими константами

★ Можно утверждать, что арифметическое кодирование производит код оптимальной длины

Качество кодирования

- Текст T можно рассматривать как результат единичного эксперимента над случайным вектором $(\xi,\ldots,\xi)$ длины n, энтропия которого равна $n\cdot H(\xi)$
  - Это и есть ожидаемая длина кода
- По теоремам Шеннона можно считать, что есть $2^{n\cdot H(\xi)}$ возможных текстов, вероятность каждого из которых равна $2^{-n\cdot H(\xi)}$
- На каждом интервале длины l, $2^{-m}\leqslant l<2^{-m+1}$, есть число, записываемое дробью со знаменателем 2^m, т.е. двоичная дробь длины m

$\Rightarrow$ арифметический кодер сопоставит тексту последовательность длины $\lceil n\cdot H(\xi)\rceil$
  - Не учтены поправка в минус (может попасться точка со знаменателем 2^{m-c}) и поправка в плюс (нули в конце), но обе являются маленькими константами

★ Можно утверждать, что арифметическое кодирование производит код оптимальной длины

Качество кодирования

- Текст T можно рассматривать как результат единичного эксперимента над случайным вектором $(\xi, \ldots, \xi)$ длины n, энтропия которого равна $n \cdot H(\xi)$
  - Это и есть ожидаемая длина кода
- По теоремам Шеннона можно считать, что есть $2^{n \cdot H(\xi)}$ возможных текстов, вероятность каждого из которых равна $2^{-n \cdot H(\xi)}$
- На каждом интервале длины l, $2^{-m} \leqslant l < 2^{-m+1}$, есть число, записываемое дробью со знаменателем 2^m, т.е. двоичная дробь длины m

$\Rightarrow$ арифметический кодер сопоставит тексту последовательность длины $\lceil n \cdot H(\xi) \rceil$

- Не учтены поправка в минус (может попасться точка со знаменателем 2^{m-c}) и поправка в плюс (нули в конце), но обе являются маленькими константами

★ Можно утверждать, что арифметическое кодирование производит код оптимальной длины

- Текст T можно рассматривать как результат единичного эксперимента над случайным вектором $(\xi, \ldots, \xi)$ длины n, энтропия которого равна $n \cdot H(\xi)$
  - Это и есть ожидаемая длина кода
- По теоремам Шеннона можно считать, что есть $2^{n \cdot H(\xi)}$ возможных текстов, вероятность каждого из которых равна $2^{-n \cdot H(\xi)}$
- На каждом интервале длины l, $2^{-m} \leqslant l < 2^{-m+1}$, есть число, записываемое дробью со знаменателем 2^m, т.е. двоичная дробь длины m

$\Rightarrow$ арифметический кодер сопоставит тексту последовательность длины $\lceil n \cdot H(\xi) \rceil$
  - Не учтены поправка в минус (может попасться точка со знаменателем 2^{m-c}) и поправка в плюс (нули в конце), но обе являются маленькими константами

★ Можно утверждать, что арифметическое кодирование производит код оптимальной длины

Качество кодирования

- Текст T можно рассматривать как результат единичного эксперимента над случайным вектором $(\xi, \ldots, \xi)$ длины n, энтропия которого равна $n \cdot H(\xi)$
  - Это и есть ожидаемая длина кода
- По теоремам Шеннона можно считать, что есть $2^{n \cdot H(\xi)}$ возможных текстов, вероятность каждого из которых равна $2^{-n \cdot H(\xi)}$
- На каждом интервале длины l, $2^{-m} \leqslant l < 2^{-m+1}$, есть число, записываемое дробью со знаменателем 2^m, т.е. двоичная дробь длины m

$\Rightarrow$ арифметический кодер сопоставит тексту последовательность длины $\lceil n \cdot H(\xi) \rceil$

  - Не учтены поправка в минус (может попасться точка со знаменателем 2^{m-c}) и поправка в плюс (нули в конце), но обе являются маленькими константами

★ Можно утверждать, что арифметическое кодирование производит код оптимальной длины

Качество кодирования

- Текст T можно рассматривать как результат единичного эксперимента над случайным вектором $(\xi, \ldots, \xi)$ длины n, энтропия которого равна $n \cdot H(\xi)$
  - Это и есть ожидаемая длина кода
- По теоремам Шеннона можно считать, что есть $2^{n \cdot H(\xi)}$ возможных текстов, вероятность каждого из которых равна $2^{-n \cdot H(\xi)}$
- На каждом интервале длины l, $2^{-m} \leqslant l < 2^{-m+1}$, есть число, записываемое дробью со знаменателем 2^m, т.е. двоичная дробь длины m

$\Rightarrow$ арифметический кодер сопоставит тексту последовательность длины $\lceil n \cdot H(\xi) \rceil$
  - Не учтены поправка в минус (может попасться точка со знаменателем 2^{m-c}) и поправка в плюс (нули в конце), но обе являются маленькими константами

★ Можно утверждать, что арифметическое кодирование производит код оптимальной длины

Проблема: трудоемкость арифметических вычислений, даже с рациональными числами (числители/знаменатели растут)

Качество кодирования

- Текст T можно рассматривать как результат единичного эксперимента над случайным вектором $(\xi, \ldots, \xi)$ длины n, энтропия которого равна $n \cdot H(\xi)$
  - Это и есть ожидаемая длина кода
- По теоремам Шеннона можно считать, что есть $2^{n \cdot H(\xi)}$ возможных текстов, вероятность каждого из которых равна $2^{-n \cdot H(\xi)}$
- На каждом интервале длины l, $2^{-m} \leqslant l < 2^{-m+1}$, есть число, записываемое дробью со знаменателем 2^m, т.е. двоичная дробь длины m

$\Rightarrow$ арифметический кодер сопоставит тексту последовательность длины $\lceil n \cdot H(\xi) \rceil$
  - Не учтены поправка в минус (может попасться точка со знаменателем 2^{m-c}) и поправка в плюс (нули в конце), но обе являются маленькими константами

★ Можно утверждать, что арифметическое кодирование производит код оптимальной длины

Проблема: трудоемкость арифметических вычислений, даже с рациональными числами (числители/знаменатели растут)
Решение: на каждом шаге “округлять” все вероятности до дробей, имеющих в знаменателе степень двойки, сводя к целочисленной арифметике

Качество кодирования

- Текст T можно рассматривать как результат единичного эксперимента над случайным вектором $(\xi, \ldots, \xi)$ длины n, энтропия которого равна $n \cdot H(\xi)$
  - Это и есть ожидаемая длина кода
- По теоремам Шеннона можно считать, что есть $2^{n \cdot H(\xi)}$ возможных текстов, вероятность каждого из которых равна $2^{-n \cdot H(\xi)}$
- На каждом интервале длины l, $2^{-m} \leqslant l < 2^{-m+1}$, есть число, записываемое дробью со знаменателем 2^m, т.е. двоичная дробь длины m

$\Rightarrow$ арифметический кодер сопоставит тексту последовательность длины $\lceil n \cdot H(\xi) \rceil$

- Не учтены поправка в минус (может попасться точка со знаменателем 2^{m-c}) и поправка в плюс (нули в конце), но обе являются маленькими константами

★ Можно утверждать, что арифметическое кодирование производит код оптимальной длины

Проблема: трудоемкость арифметических вычислений, даже с рациональными числами (числители/знаменатели растут)

Решение: на каждом шаге "округлять" все вероятности до дробей, имеющих в знаменателе степень двойки, сводя к целочисленной арифметике

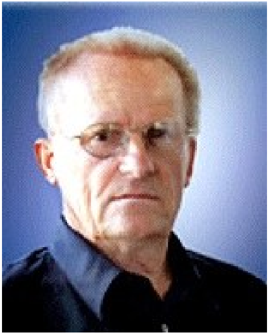
Пример: $p_1 = p_2 = 0.4,\ p_3 = 0.2$. Округлим $0.4 \approx 102/256$, $0.2 \approx 52/256$. Имеем

$E_{Opt} = 0.8 \cdot \log 2.5 + 0.2 \log 5 \approx 1.52193$ бит на символ

$E_{Ari} = 0.8 \cdot \log 256/102 + 0.2 \cdot \log 256/52 \approx 1.52197$ бит на символ

Погрешность энтропии — в 2000 раз меньше, чем у кода Хаффмана

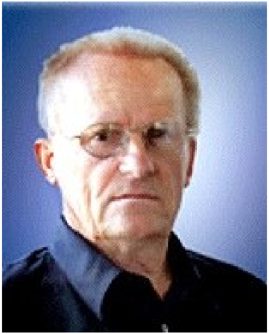
В изобретении и внедрении арифметического кодирования участвовала толпа народу, однако наибольший вклад внес



Йорма Риссанен

Первая работа:
"Generalized Kraft inequality and arithmetic coding"
(IBM Journal of Research and Development, 1976)

В изобретении и внедрении арифметического кодирования участвовала толпа народу, однако наибольший вклад внес



Йорма Риссанен

Первая работа:
“Generalized Kraft inequality and arithmetic coding”
(IBM Journal of Research and Development, 1976)

Метод был защищен кучей патентов, которые истекли не так давно. Из-за этого, в частности, в JPEG используется Хаффман вместо арифметического сжатия, которое давало бы лучшие результаты

Статическая целочисленная реализация

Дан текст T, порожденный $\xi = (x_{1\,|\,p_1}, \ldots, x_{k\,|\,p_k})$, каждому x_j сопоставлен интервал $[\alpha, \beta)$, где $\alpha = \sum_{i=1}^{j-1} p_i$, $\beta = \sum_{i=1}^{j} p_i$

Подготовка:

- Выберем число N (например, 32; в учебном примере у нас будет $N = 8$)
- Заменим вещественный интервал $[0, 1)$ целочисленным $[0..2^N - 1]$
  - можно смотреть на эти целые числа как на значения первых N цифр мантиссы кода $\texttt{enc}(T)$
- Каждый интервал $[\alpha, \beta)$ заменится на $[a..b]$, где $a = \lceil \alpha \cdot 2^N \rceil$, $b = \lceil \beta \cdot 2^N \rceil - 1$

Статическая целочисленная реализация

Дан текст T, порожденный $\xi = (x_{1\,|p_1}, \ldots, x_{k\,|p_k})$, каждому x_j сопоставлен интервал $[\alpha, \beta)$, где $\alpha = \sum_{i=1}^{j-1} p_i$, $\beta = \sum_{i=1}^{j} p_i$

Подготовка:

- Выберем число N (например, 32; в учебном примере у нас будет $N = 8$)
- Заменим вещественный интервал $[0, 1)$ целочисленным $[0..2^N - 1]$
  - можно смотреть на эти целые числа как на значения первых N цифр мантиссы кода $\mathtt{enc}(T)$
- Каждый интервал $[\alpha, \beta)$ заменится на $[a..b]$, где $a = \lceil \alpha \cdot 2^N \rceil$, $b = \lceil \beta \cdot 2^N \rceil - 1$

Шаг кодирования:

- Пусть $[l..h]$ — интервал после предыдущего шага
- Спроектировав $[0..2^N - 1]$ на $[l..h]$, найдем подинтервал $[l'..h']$, соответствующий текущему символу x
  - $l' = l + \left\lceil \frac{a(h-l+1)}{2^N} \right\rceil$, $h' = l + \left\lceil \frac{(b+1)(h-l+1)}{2^N} \right\rceil - 1$, где $[a..b]$ — интервал для x
- Пусть r — число совпадающих старших битов у l' и h'; допишем эти биты к $\mathtt{enc}(T)$, сотрем их у l' и h' и допишем к l' справа r нулей, а к h' — r единиц
  - все числа между l' и h' начинаются с этих битов, поэтому они точно входят в мантиссу итогового кодового числа; мы их записываем и сдвигаем "окно" вправо
- полученные числа l_{new} и h_{new} — границы интервала для следующего шага

Пример кодирования (начало)

Возьмем текст $T = acagaatagaga$, ему соответствует $\xi = (a_{|7/12}, c_{|1/12}, g_{|3/12}, t_{|1/12})$

- Интервалы $[0, 7/12)$; $[7/12, 8/12)$; $[8/12, 11/12)$; $[11/12, 1)$

$\Rightarrow$ при $N = 8$ преобразуются в $[0..149]$; $[150..170]$; $[171..234]$; $[235..255]$

Пример кодирования (начало)

Возьмем текст $T = acagaatagaga$, ему соответствует $\xi = (a_{|7/12}, c_{|1/12}, g_{|3/12}, t_{|1/12})$

- Интервалы $[0, 7/12)$; $[7/12, 8/12)$; $[8/12, 11/12)$; $[11/12, 1)$

$\Rightarrow$ при $N = 8$ преобразуются в $[0..149]$; $[150..170]$; $[171..234]$; $[235..255]$

Инициализация: $l = 0, h = 255$

a: $l_{new} = l' = 0$, $h_{new} = h' = 149$

$\mathtt{enc}(T) = \lambda$

Пример кодирования (начало)

Возьмем текст $T = acagaatagaga$, ему соответствует $\xi = (a_{|7/12}, c_{|1/12}, g_{|3/12}, t_{|1/12})$

- Интервалы $[0, 7/12)$; $[7/12, 8/12)$; $[8/12, 11/12)$; $[11/12, 1)$

$\Rightarrow$ при $N = 8$ преобразуются в $[0..149]$; $[150..170]$; $[171..234]$; $[235..255]$

Инициализация: $l = 0, h = 255$

a: $l_{new} = l' = 0,\ h_{new} = h' = 149$

$\mathtt{enc}(T) = \lambda$

c: $l' = 0 + \left\lceil \frac{150\cdot(149-0+1)}{256} \right\rceil = 88 = [01011000]_2$

$h' = 0 + \left\lceil \frac{171\cdot(149-0+1)}{256} \right\rceil - 1 = 100 = [01100100]_2$

$\mathtt{enc}(T) = 01$

$l_{new} = [01100000]_2 = 96,\ h_{new} = [10010011]_2 = 147$

Пример кодирования (начало)

Возьмем текст $T = acagaatagaga$, ему соответствует $\xi = (a_{|7/12}, c_{|1/12}, g_{|3/12}, t_{|1/12})$

- Интервалы $[0, 7/12)$; $[7/12, 8/12)$; $[8/12, 11/12)$; $[11/12, 1)$

$\Rightarrow$ при $N = 8$ преобразуются в $[0..149]$; $[150..170]$; $[171..234]$; $[235..255]$

Инициализация: $l = 0, h = 255$

a: $l_{new} = l' = 0,\ h_{new} = h' = 149$

$\mathtt{enc}(T) = \lambda$

c: $l' = 0 + \lceil \frac{150\cdot(149-0+1)}{256} \rceil = 88 = [01011000]_2$

$h' = 0 + \lceil \frac{171\cdot(149-0+1)}{256} \rceil - 1 = 100 = [01100100]_2$

$\mathtt{enc}(T) = 01$

$l_{new} = [01100000]_2 = 96,\ h_{new} = [10010011]_2 = 147$

Потенциальная неприятность: маленький интервал $[l..h]$

- Ошибки округления влияют сильнее
- Крайний случай — если число точек на интервале становится меньше $1/p_i$, то символе x_i может соответствовать интервал без единой точки — кодирование невозможно

Расширение интервала

- Интервал — маленький, если l чуть меньше 2^{N-1}, а h — чуть больше
  - точек мало, а старший бит не совпадает

$\Rightarrow$ В этом случае, для некоторого $r \geqslant 1$, $l = [0\,\underbrace{1\cdots1}_{r\ \text{цифр}}\cdots]_2$, $h = [1\,\underbrace{0\cdots0}_{r\ \text{цифр}}\cdots]_2$
  - если $r = 0$, то $h - l > 2^{N-2}$, т.е интервал большой
  - в примере на предыдущем слайде $r = 2$: $l = [01100000]_2$, $h = [10010011]_2$
- Тем самым, нам неизвестен очередной бит $\mathtt{enc}(T)$, но известно, что r следующих битов ему противоположны

Расширение интервала

- Интервал — маленький, если l чуть меньше 2^{N-1}, а h — чуть больше
  - точек мало, а старший бит не совпадает

$\Rightarrow$ В этом случае, для некоторого $r \geqslant 1$, $l = [0\underbrace{1\cdots1}_{r\text{ цифр}}\cdots]_2$, $h = [1\underbrace{0\cdots0}_{r\text{ цифр}}\cdots]_2$

  - если $r = 0$, то $h - l > 2^{N-2}$, т.е интервал большой
  - в примере на предыдущем слайде $r = 2$: $l = [01100000]_2$, $h = [10010011]_2$
- Тем самым, нам неизвестен очередной бит $\mathtt{enc}(T)$, но известно, что r следующих битов ему противоположны

Если к данному шагу на выход отправлена строка S, то $\mathtt{enc}(T)$ — мантисса числа, лежащего между $0.S\,l\,00\cdots0\cdots$ и $0.S\,h\,11\cdots1\cdots$.

Выход: дописать к l бит 0, к h — бит 1; полученный интервал $[2l, 2h{+}1]$ содержит вдвое больше точек, используем его для кодирования

Проблема: для представления l и h есть только N бит

Расширение интервала

- Интервал — маленький, если l чуть меньше 2^{N-1}, а h — чуть больше
  - точек мало, а старший бит не совпадает

$\Rightarrow$ В этом случае, для некоторого $r \geqslant 1$, $l = [0\underbrace{1\cdots1}_{r \text{ цифр}}\cdots]_2$, $h = [1\underbrace{0\cdots0}_{r \text{ цифр}}\cdots]_2$

  - если $r = 0$, то $h - l > 2^{N-2}$, т.е интервал большой
  - в примере на предыдущем слайде $r = 2$: $l = [01100000]_2$, $h = [10010011]_2$
- Тем самым, нам неизвестен очередной бит $\mathtt{enc}(T)$, но известно, что r следующих битов ему противоположны

Если к данному шагу на выход отправлена строка S, то $\mathtt{enc}(T)$ — мантисса числа, лежащего между $0.S\,l\,00\cdots0\cdots$ и $0.S\,h\,11\cdots1\cdots$.

Выход: дописать к l бит 0, к h — бит 1; полученный интервал $[2l, 2h+1]$ содержит вдвое больше точек, используем его для кодирования

Проблема: для представления l и h есть только N бит

Трюк: положим $l_{new} = 2l - 2^{N-1}$, $h_{new} = 2h - 2^{N-1} + 1$,

"раздвинув" интервал в стороны от 2^{N-1};

- в $[l_{new}..h_{new}]$ в 2 раза больше точек, чем в $[l..h]$
- $l = [b_1 b_2 \cdots b_N]_2 \Rightarrow l_{new} = [b_1 b_3 \cdots b_N 0]_2$; то же для h_{new}, но справа единица
- количество расширений (т.е. стираний второго бита) храним в счетчике *bits*
- когда на очередной итерации требуется вывести последовательность битов $b_1 b_2 \cdots b_s$, выводим $b_1 \underbrace{\bar{b}_1 \cdots \bar{b}_1}_{r \text{цифр}} b_2 \cdots b_s$, восстанавливая стертые биты, и обнуляем *bits*

Продолжение примера (кодирование)

$T = acagaatagaga$; $N = 8$; $a : [0..149]$; $c : [150..170]$; $g : [171..234]$; $t : [235..255]$

Продолжение примера (кодирование)

$T = acagaatagaga$; $N = 8$; $a : [0..149]$; $c : [150..170]$; $g : [171..234]$; $t : [235..255]$

c: $l_{new} = [01100000]_2 = 96, \; h_{new} = [10010011]_2 = 147 \Rightarrow bits = 2$

$l_{new} = [00000000]_2 = 0, \; h_{new} = [11001111]_2 = 207$

Продолжение примера (кодирование)

$T = acagaatagaga$; $N = 8$; $a : [0..149]$; $c : [150..170]$; $g : [171..234]$; $t : [235..255]$

c: $l_{new} = [01100000]_2 = 96$, $h_{new} = [10010011]_2 = 147 \Rightarrow bits = 2$

$l_{new} = [00000000]_2 = 0$, $h_{new} = [11001111]_2 = 207$

a: $l' = 0 + \left\lceil \frac{0 \cdot 208}{256} \right\rceil = 0 = [00000000]_2$; $h' = 0 + \left\lceil \frac{150 \cdot 208}{256} \right\rceil - 1 = 121 = [01111001]_2$

$\mathtt{enc}(T) = 01011$; $bits = 0$; $l_{new} = [00000000]_2 = 0$, $h_{new} = [11110111]_2 = 243$

$T = acagaatagaga$; $N = 8$; $a : [0..149]$; $c : [150..170]$; $g : [171..234]$; $t : [235..255]$

c: $l_{new} = [01100000]_2 = 96$, $h_{new} = [10010011]_2 = 147 \Rightarrow bits = 2$

$l_{new} = [00000000]_2 = 0$, $h_{new} = [11001111]_2 = 207$

a: $l' = 0 + \left\lceil \frac{0 \cdot 208}{256} \right\rceil = 0 = [00000000]_2$; $h' = 0 + \left\lceil \frac{150 \cdot 208}{256} \right\rceil - 1 = 121 = [01111001]_2$

$\mathtt{enc}(T) = 01011$; $bits = 0$; $l_{new} = [00000000]_2 = 0$, $h_{new} = [11110111]_2 = 243$

g: $l' = 0 + \left\lceil \frac{171 \cdot 244}{256} \right\rceil = 163 = [10100011]_2$; $h' = 0 + \left\lceil \frac{234 \cdot 244}{256} \right\rceil - 1 = 223 = [11011111]_2$

$\mathtt{enc}(T) = 010111$; $l_{new} = [01000110]_2 = 70$, $h_{new} = [10111111]_2 = 191$;

$bits = 1$; $l_{new} = [00001100]_2 = 12$, $h_{new} = [11111111]_2 = 255$;

Продолжение примера (кодирование)

$T = acagaatagaga$; $N = 8$; $a : [0..149]$; $c : [150..170]$; $g : [171..234]$; $t : [235..255]$

- c: $l_{new} = [01100000]_2 = 96$, $h_{new} = [10010011]_2 = 147 \Rightarrow bits = 2$
  $l_{new} = [00000000]_2 = 0$, $h_{new} = [11001111]_2 = 207$
- a: $l' = 0 + \lceil \frac{0 \cdot 208}{256} \rceil = 0 = [00000000]_2$; $h' = 0 + \lceil \frac{150 \cdot 208}{256} \rceil - 1 = 121 = [01111001]_2$
  $\mathtt{enc}(T) = 01011$; $bits = 0$; $l_{new} = [00000000]_2 = 0$, $h_{new} = [11110111]_2 = 243$
- g: $l' = 0 + \lceil \frac{171 \cdot 244}{256} \rceil = 163 = [10100011]_2$; $h' = 0 + \lceil \frac{234 \cdot 244}{256} \rceil - 1 = 223 = [11011111]_2$
  $\mathtt{enc}(T) = 010111$; $l_{new} = [01000110]_2 = 70$, $h_{new} = [10111111]_2 = 191$;
  $bits = 1$; $l_{new} = [00001100]_2 = 12$, $h_{new} = [11111111]_2 = 255$;
- a: $l' = 12 + \lceil \frac{0 \cdot 244}{256} \rceil = 12 = [00001100]_2$; $h' = 12 + \lceil \frac{150 \cdot 244}{256} \rceil - 1 = 154 = [10011010]_2$
  $\mathtt{enc}(T) = 010111$; $bits = 1$; $l_{new} = [00001100]_2 = 12$, $h_{new} = [10011010]_2 = 154$;

Продолжение примера (кодирование)

$T = acagaatagaga$; $N = 8$; $a : [0..149]$; $c : [150..170]$; $g : [171..234]$; $t : [235..255]$

c: $l_{new} = [01100000]_2 = 96,\ h_{new} = [10010011]_2 = 147 \Rightarrow bits = 2$

$l_{new} = [00000000]_2 = 0,\ h_{new} = [11001111]_2 = 207$

a: $l' = 0 + \lceil \frac{0 \cdot 208}{256} \rceil = 0 = [00000000]_2$; $h' = 0 + \lceil \frac{150 \cdot 208}{256} \rceil - 1 = 121 = [01111001]_2$

$\mathtt{enc}(T) = 01011$; $bits = 0$; $l_{new} = [00000000]_2 = 0,\ h_{new} = [11110111]_2 = 243$

g: $l' = 0 + \lceil \frac{171 \cdot 244}{256} \rceil = 163 = [10100011]_2$; $h' = 0 + \lceil \frac{234 \cdot 244}{256} \rceil - 1 = 223 = [11011111]_2$

$\mathtt{enc}(T) = 010111$; $l_{new} = [01000110]_2 = 70,\ h_{new} = [10111111]_2 = 191$;

$bits = 1$; $l_{new} = [00001100]_2 = 12,\ h_{new} = [11111111]_2 = 255$;

a: $l' = 12 + \lceil \frac{0 \cdot 244}{256} \rceil = 12 = [00001100]_2$; $h' = 12 + \lceil \frac{150 \cdot 244}{256} \rceil - 1 = 154 = [10011010]_2$

$\mathtt{enc}(T) = 010111$; $bits = 1$; $l_{new} = [00001100]_2 = 12,\ h_{new} = [10011010]_2 = 154$;

a: $l' = 12 + \lceil \frac{0 \cdot 143}{256} \rceil = 12 = [00001100]_2$; $h' = 12 + \lceil \frac{150 \cdot 143}{256} \rceil - 1 = 95 = [01011111]_2$

$\mathtt{enc}(T) = 01011101$; $bits = 0$; $l_{new} = [00011000]_2 = 24,\ h_{new} = [10111111]_2 = 191$;

$T = acagaatagaga$; $N = 8$; $a : [0..149]$; $c : [150..170]$; $g : [171..234]$; $t : [235..255]$

c: $l_{new} = [01100000]_2 = 96$, $h_{new} = [10010011]_2 = 147 \Rightarrow bits = 2$

$l_{new} = [00000000]_2 = 0$, $h_{new} = [11001111]_2 = 207$

a: $l' = 0 + \lceil \frac{0 \cdot 208}{256} \rceil = 0 = [00000000]_2$; $h' = 0 + \lceil \frac{150 \cdot 208}{256} \rceil - 1 = 121 = [01111001]_2$

$\mathtt{enc}(T) = 01011$; $bits = 0$; $l_{new} = [00000000]_2 = 0$, $h_{new} = [11110111]_2 = 243$

g: $l' = 0 + \lceil \frac{171 \cdot 244}{256} \rceil = 163 = [10100011]_2$; $h' = 0 + \lceil \frac{234 \cdot 244}{256} \rceil - 1 = 223 = [11011111]_2$

$\mathtt{enc}(T) = 010111$; $l_{new} = [01000110]_2 = 70$, $h_{new} = [10111111]_2 = 191$;

$bits = 1$; $l_{new} = [00001100]_2 = 12$, $h_{new} = [11111111]_2 = 255$;

a: $l' = 12 + \lceil \frac{0 \cdot 244}{256} \rceil = 12 = [00001100]_2$; $h' = 12 + \lceil \frac{150 \cdot 244}{256} \rceil - 1 = 154 = [10011010]_2$

$\mathtt{enc}(T) = 010111$; $bits = 1$; $l_{new} = [00001100]_2 = 12$, $h_{new} = [10011010]_2 = 154$;

a: $l' = 12 + \lceil \frac{0 \cdot 143}{256} \rceil = 12 = [00001100]_2$; $h' = 12 + \lceil \frac{150 \cdot 143}{256} \rceil - 1 = 95 = [01011111]_2$

$\mathtt{enc}(T) = 01011101$; $bits = 0$; $l_{new} = [00011000]_2 = 24$, $h_{new} = [10111111]_2 = 191$;

t: $l' = 24 + \lceil \frac{234 \cdot 176}{256} \rceil = 179 = [10110011]_2$; $h' = 24 + \lceil \frac{256 \cdot 176}{256} \rceil - 1 = 191 = [10111111]_2$

$\mathtt{enc}(T) = 010111011011$; $l_{new} = [00110000]_2 = 48$, $h_{new} = [11111111]_2 = 255$;

$T = acagaatagaga$; $N = 8$; $a : [0..149]$; $c : [150..170]$; $g : [171..234]$; $t : [235..255]$

c: $l_{new} = [01100000]_2 = 96,\ h_{new} = [10010011]_2 = 147 \Rightarrow bits = 2$
$l_{new} = [00000000]_2 = 0,\ h_{new} = [11001111]_2 = 207$

a: $l' = 0 + \lceil \frac{0 \cdot 208}{256} \rceil = 0 = [00000000]_2$; $h' = 0 + \lceil \frac{150 \cdot 208}{256} \rceil - 1 = 121 = [01111001]_2$
$\mathrm{enc}(T) = 01011$; $bits = 0$; $l_{new} = [00000000]_2 = 0,\ h_{new} = [11110111]_2 = 243$

g: $l' = 0 + \lceil \frac{171 \cdot 244}{256} \rceil = 163 = [10100011]_2$; $h' = 0 + \lceil \frac{234 \cdot 244}{256} \rceil - 1 = 223 = [11011111]_2$
$\mathrm{enc}(T) = 010111$; $l_{new} = [01000110]_2 = 70,\ h_{new} = [10111111]_2 = 191$;
$bits = 1$; $l_{new} = [00001100]_2 = 12,\ h_{new} = [11111111]_2 = 255$;

a: $l' = 12 + \lceil \frac{0 \cdot 244}{256} \rceil = 12 = [00001100]_2$; $h' = 12 + \lceil \frac{150 \cdot 244}{256} \rceil - 1 = 154 = [10011010]_2$
$\mathrm{enc}(T) = 010111$; $bits = 1$; $l_{new} = [00001100]_2 = 12,\ h_{new} = [10011010]_2 = 154$;

a: $l' = 12 + \lceil \frac{0 \cdot 143}{256} \rceil = 12 = [00001100]_2$; $h' = 12 + \lceil \frac{150 \cdot 143}{256} \rceil - 1 = 95 = [01011111]_2$
$\mathrm{enc}(T) = 01011101$; $bits = 0$; $l_{new} = [00011000]_2 = 24,\ h_{new} = [10111111]_2 = 191$;

t: $l' = 24 + \lceil \frac{234 \cdot 176}{256} \rceil = 179 = [10110011]_2$; $h' = 24 + \lceil \frac{256 \cdot 176}{256} \rceil - 1 = 191 = [10111111]_2$
$\mathrm{enc}(T) = 010111011011$; $l_{new} = [00110000]_2 = 48,\ h_{new} = [11111111]_2 = 255$;

- ...

g: $\ldots \mathrm{enc}(T) = 010111011011100$; $bits = 2$ $l_{new} = 60,\ h_{new} = 199$;

a: $l' = 60$; $h' = 142$; $\mathrm{enc}(T) = 0101110110111001000$;

  - 1 — завершение числа с кратчайшей записью на отрезке; 000 дописано за три дополнительных шага (число с кратчайшей записью внутри интервала $[\alpha, \beta)$ не менялось три итерации)

Окончание примера (декодирование)

$\mathtt{enc}(T) = 010111011011100110$; $N = 8$;
$a : [0..149)$; $c : [150..170)$; $g : [171..234)$; $t : [235..255)$

- По $\mathtt{enc}(T)$ скользит окно ширины N; w — число, записанное в окне
- На каждом шаге ищем интервал, в который попало w, выдаем его символ, пересчитываем интервал по правилам кодера;
- Окно сдвигаем на общий префикс l' и h' и корректируем при ненулевом $bits$

Окончание примера (декодирование)

$\mathtt{enc}(T) = 010111011011100110$; $N = 8$;
$a : [0..149)$; $c : [150..170)$; $g : [171..234)$; $t : [235..255)$

- По $\mathtt{enc}(T)$ скользит окно ширины N; w — число, записанное в окне
- На каждом шаге ищем интервал, в который попало w, выдаем его символ, пересчитываем интервал по правилам кодера;
- Окно сдвигаем на общий префикс l' и h' и корректируем при ненулевом *bits*

0) $l = 0,\ h = 255$
1) $\color{red}{01011101}1011100110$, $w = 93$; $T = \color{blue}{a}$; $l' = l = 0,\ h' = h = 149,\ \mathit{shift} = 0$

$\mathtt{enc}(T) = 010111011011100110$; $N = 8$;
$a : [0..149)$; $c : [150..170)$; $g : [171..234)$; $t : [235..255)$

- По $\mathtt{enc}(T)$ скользит окно ширины N; w — число, записанное в окне
- На каждом шаге ищем интервал, в который попало w, выдаем его символ, пересчитываем интервал по правилам кодера;
- Окно сдвигаем на общий префикс l' и h' и корректируем при ненулевом *bits*

0) $l = 0$, $h = 255$
1) 010111011011100110, $w = 93$; $T = a$; $l' = l = 0$, $h' = h = 149$, $shift = 0$
2) 010111011011100110, $w = 93$; $T = ac$; $l' = 88$, $h' = 100$, $shift = 2$, $bits = 2$, $l = 0$, $h = 207$

Окончание примера (декодирование)

$\texttt{enc}(T) = 010111011011100110$; $N = 8$;
$a : [0..149)$; $c : [150..170)$; $g : [171..234)$; $t : [235..255)$

- По $\texttt{enc}(T)$ скользит окно ширины N; w — число, записанное в окне
- На каждом шаге ищем интервал, в который попало w, выдаем его символ, пересчитываем интервал по правилам кодера;
- Окно сдвигаем на общий префикс l' и h' и корректируем при ненулевом $bits$

0) $l = 0$, $h = 255$
1) 010111011011100110, $w = 93$; $T = a$; $l' = l = 0$, $h' = h = 149$, $shift = 0$
2) 010111011011100110, $w = 93$; $T = ac$; $l' = 88$, $h' = 100$, $shift = 2$, $bits = 2$, $l = 0$, $h = 207$
3) 010111011011100110, $w = 91$; $T = aca$; $l' = 0$, $h' = 121$, $shift = 1 + 2$, $l = 0$, $h = 243$

Окончание примера (декодирование)

$\mathtt{enc}(T) = 0101110110111001 10$; $N = 8$;
$a : [0..149)$; $c : [150..170)$; $g : [171..234)$; $t : [235..255)$

- По $\mathtt{enc}(T)$ скользит окно ширины N; w — число, записанное в окне
- На каждом шаге ищем интервал, в который попало w, выдаем его символ, пересчитываем интервал по правилам кодера;
- Окно сдвигаем на общий префикс l' и h' и корректируем при ненулевом *bits*

0) $l = 0,\ h = 255$
1) $\color{red}{01011101}1011100110$, $w = 93$; $T = \color{blue}{a}$; $l' = l = 0,\ h' = h = 149,\ shift = 0$
2) $\color{red}{01011101}1011100110$, $w = 93$; $T = a\color{blue}{c}$; $l' = 88,\ h' = 100,\ shift = 2,$
   $bits = 2,\ l = 0,\ h = 207$
3) $01\color{red}{01110110}11100110$, $w = 91$; $T = ac\color{blue}{a}$; $l' = 0,\ h' = 121,\ shift = 1 + 2,$
   $l = 0,\ h = 243$
4) $01011\color{red}{10110111}00110$, $w = 183$; $T = aca\color{blue}{g}$; $l' = 163,\ h' = 223,\ shift = 1,$
   $bits = 1,\ l = 12,\ h = 255$

Окончание примера (декодирование)

$\mathtt{enc}(T) = 0101110110111 00110$; $N = 8$;
$a : [0..149)$; $c : [150..170)$; $g : [171..234)$; $t : [235..255)$

- По $\mathtt{enc}(T)$ скользит окно ширины N; w — число, записанное в окне
- На каждом шаге ищем интервал, в который попало w, выдаем его символ, пересчитываем интервал по правилам кодера;
- Окно сдвигаем на общий префикс l' и h' и корректируем при ненулевом $bits$

0) $l = 0$, $h = 255$
1) $01011101 1011100110$, $w = 93$; $T = a$; $l' = l = 0$, $h' = h = 149$, $shift = 0$
2) $01011101 1011100110$, $w = 93$; $T = ac$; $l' = 88$, $h' = 100$, $shift = 2$, $bits = 2$, $l = 0$, $h = 207$
3) $01011101101 1100110$, $w = 91$; $T = aca$; $l' = 0$, $h' = 121$, $shift = 1 + 2$, $l = 0$, $h = 243$
4) $0101110110111 00110$, $w = 183$; $T = acag$; $l' = 163$, $h' = 223$, $shift = 1$, $bits = 1$, $l = 12$, $h = 255$

... (когда остается $< N$ бит, окно дополняется нулями для вычисления w)

Динамическая реализация

- Общие принципы — такие же, как у динамического кодирования по Хаффману
  - символ $T[i]$ кодируется по частотам символов в $T[1..i-1]$
  - можно делать адаптивное кодирование, периодически масштабируя счетчики
- Можно использовать статический алфавит, присвоив перед началом всем возможным символам частоту 1
- Можно использовать динамический алфавит, используя искусственный символ для добавления новых символов

Динамическая реализация

- Общие принципы — такие же, как у динамического кодирования по Хаффману
  - символ $T[i]$ кодируется по частотам символов в $T[1..i-1]$
  - можно делать адаптивное кодирование, периодически масштабируя счетчики
- Можно использовать статический алфавит, присвоив перед началом всем возможным символам частоту 1
- Можно использовать динамический алфавит, используя искусственный символ для добавления новых символов

★ чтобы найти текущий диапазон для символа, надо знать не только его частоту, но и сумму частот всех символов до него (поделив эту сумму на сумму частот всех символов, получим левый край диапазона). Чтобы делать это быстро, надо хранить частоты в нетривиальной структуре данных, например, в разработанном специально для этой цели дереве Фенвика или в более универсальном дереве отрезков. Использование таких структур позволяет выполнять итерацию за логарифмическое от размера алфавита время

- Общие принципы — такие же, как у динамического кодирования по Хаффману
  - символ $T[i]$ кодируется по частотам символов в $T[1..i-1]$
  - можно делать адаптивное кодирование, периодически масштабируя счетчики
- Можно использовать статический алфавит, присвоив перед началом всем возможным символам частоту 1
- Можно использовать динамический алфавит, используя искусственный символ для добавления новых символов

★ чтобы найти текущий диапазон для символа, надо знать не только его частоту, но и сумму частот всех символов до него (поделив эту сумму на сумму частот всех символов, получим левый край диапазона). Чтобы делать это быстро, надо хранить частоты в нетривиальной структуре данных, например, в разработанном специально для этой цели дереве Фенвика или в более универсальном дереве отрезков. Использование таких структур позволяет выполнять итерацию за логарифмическое от размера алфавита время

★ в 2014 году предложен новый метод кодирования — Arithmetic numeral systems, сохраняющий оптимальность арифметического кодирования и более производительный. Он находится за рамками нашего курса