# Лекция 4: Статическое и динамическое сжатие на примере метода Хаффмана

А. М. Шур

Кафедра алгебры и фундаментальной информатики УрФУ

20 марта 2017 г.

# Кодирование и сжатие

- ★ Каждый метод кодирования может “сам по себе” служить методом сжатия
  - не очень хорошим; лучше использовать его совместно с другими методами моделирования

# Кодирование и сжатие

★ Каждый метод кодирования может “сам по себе” служить методом сжатия
- не очень хорошим; лучше использовать его совместно с другими методами моделирования

Напомним, что

- Даны алфавит $\Sigma = \{x_1, \ldots, x_k\}$ и случайная величина $\xi = (x_{1\,|p_1}, \ldots, x_{k\,|p_k})$
- Мы нашли оптимальное кодирование текстов, сгенерированных $\xi$, на основе префиксных кодов
  - по “кубику” $\xi$ вычисляется код Хаффмана $\{C_1, \ldots, C_k\} \subset \{0,1\}^*$
  - код текста $T = x_{i_1} \cdots x_{i_n}$ есть $\mathtt{enc}(T) = C_{i_1} \cdots C_{i_n}$, причем $\mathtt{enc}$ — инъекция
  - матожидание длины закодированного символа (а значит, и закодированного текста) минимально по всем префиксным кодам
  - $\mathtt{enc}$ и $\mathtt{enc}^{-1}$ вычислимы онлайн за время, пропорциональное $|T| + |\mathtt{enc}(T)|$

# Кодирование и сжатие

★ Каждый метод кодирования может “сам по себе” служить методом сжатия
- не очень хорошим; лучше использовать его совместно с другими методами моделирования

Напомним, что
- Даны алфавит $\Sigma = \{x_1, \ldots, x_k\}$ и случайная величина $\xi = (x_{1\,|p_1}, \ldots, x_{k\,|p_k})$
- Мы нашли оптимальное кодирование текстов, сгенерированных $\xi$, на основе префиксных кодов
  - по “кубику” $\xi$ вычисляется код Хаффмана $\{C_1, \ldots, C_k\} \subset \{0,1\}^*$
  - код текста $T = x_{i_1} \cdots x_{i_n}$ есть $\mathtt{enc}(T) = C_{i_1} \cdots C_{i_n}$, причем $\mathtt{enc}$ — инъекция
  - матожидание длины закодированного символа (а значит, и закодированного текста) минимально по всем префиксным кодам
  - $\mathtt{enc}$ и $\mathtt{enc}^{-1}$ вычислимы онлайн за время, пропорциональное $|T| + |\mathtt{enc}(T)|$

★ В задаче сжатия данных у нас есть только текст $T$ как последовательность символов $k$-символьного алфавита
- например, $T$ — блок данных, рассматриваемый как последовательность байт, т.е. $k = 256$

и нет никакой $\xi$...

# Статическое сжатие по Хаффману

Возьмем очень простую модель: $T$ сгенерирован посимвольно бросками некоторого кубика $\xi$

- По Шеннону, $T$ — "типичный" результат последовательности испытаний над $\xi$

$\Rightarrow$ Частоты символов в $T$ пропорциональны их вероятностям в $\xi$

$\Rightarrow$ Посчитаем частоты, поделим их на $n$ и получим вероятности для $\xi$

- теперь можно кодировать выход $\xi$, используя материал предыдущей лекции
- ★ можно на $n$ не делить, построив (то же самое) дерево Хаффмана по частотам

Это называется "Статический метод сжатия по Хаффману"

# Статическое сжатие по Хаффману

Возьмем очень простую модель: $T$ сгенерирован посимвольно бросками некоторого кубика $\xi$

- По Шеннону, $T$ — "типичный" результат последовательности испытаний над $\xi$

$\Rightarrow$ Частоты символов в $T$ пропорциональны их вероятностям в $\xi$

$\Rightarrow$ Посчитаем частоты, поделим их на $n$ и получим вероятности для $\xi$

- теперь можно кодировать выход $\xi$, используя материал предыдущей лекции
- ★ можно на $n$ не делить, построив (то же самое) дерево Хаффмана по частотам

Это называется "Статический метод сжатия по Хаффману"

Договоримся в дальнейшем называть алгоритм, вычисляющий `enc`, кодером, а алгоритм, вычисляющий $\texttt{enc}^{-1}$ — декодером. Мы иногда будем "одушевлять" этих участников процесса.

# Статическое сжатие по Хаффману

Возьмем очень простую модель: $T$ сгенерирован посимвольно бросками некоторого кубика $\xi$

- По Шеннону, $T$ — "типичный" результат последовательности испытаний над $\xi$

$\Rightarrow$ Частоты символов в $T$ пропорциональны их вероятностям в $\xi$

$\Rightarrow$ Посчитаем частоты, поделим их на $n$ и получим вероятности для $\xi$

- теперь можно кодировать выход $\xi$, используя материал предыдущей лекции
- ★ можно на $n$ не делить, построив (то же самое) дерево Хаффмана по частотам

Это называется "Статический метод сжатия по Хаффману"

Договоримся в дальнейшем называть алгоритм, вычисляющий $\mathtt{enc}$, кодером, а алгоритм, вычисляющий $\mathtt{enc}^{-1}$ — декодером. Мы иногда будем "одушевлять" этих участников процесса.

- ★ Поскольку декодер не знает, какую $\xi$ придумал себе кодер, глядя на $T$, и не может сам ее восстановить (у него нет $T$), то ему нужно передавать не только $\mathtt{enc}(T)$, но и построенный код Хаффмана
  - можно было бы передать $\xi$, но это не только лишняя работа для декодера, но и лишняя трата памяти — $\xi$, как ни странно, занимает места больше, чем префиксный код!
- насколько экономно можно передать код Хаффмана?

# Кодирование полного дерева

## Предложение

Полное бинарное дерево с $m$ листьями можно однозначно закодировать строкой из $2m - 1$ бит, при этом строка по дереву и дерево по строке строятся за время $O(m)$.

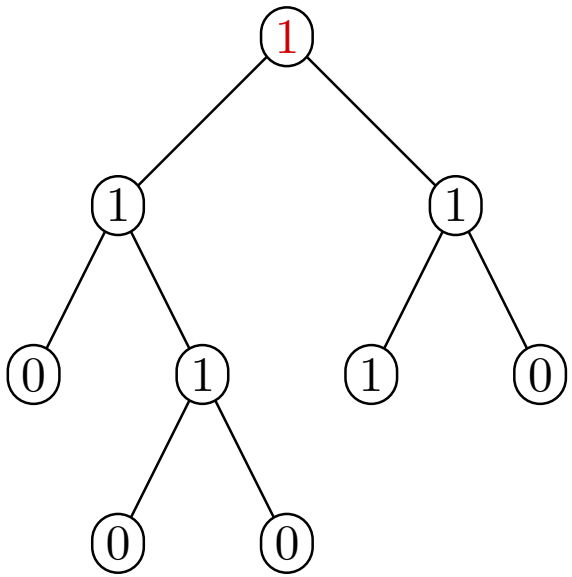
Обойдем дерево в глубину, записывая бит при первом заходе в каждую вершину: 1 для внутренней вершины, 0 — для листа. Этот процесс обратим: по строке можно строить дерево и сразу же обходить его в глубину. При чтении 1 у текущей вершины создается пара детей и текущей вершиной назначается левый ребенок; при чтении 0 текущая вершина помечается как лист, а управление возвращается ее родителю. □

# Кодирование полного дерева

## Предложение

Полное бинарное дерево с $m$ листьями можно однозначно закодировать строкой из $2m - 1$ бит, при этом строка по дереву и дерево по строке строятся за время $O(m)$.

Обойдем дерево в глубину, записывая бит при первом заходе в каждую вершину: 1 для внутренней вершины, 0 — для листа. Этот процесс обратим: по строке можно строить дерево и сразу же обходить его в глубину. При чтении 1 у текущей вершины создается пара детей и текущей вершиной назначается левый ребенок; при чтении 0 текущая вершина помечается как лист, а управление возвращается ее родителю. □



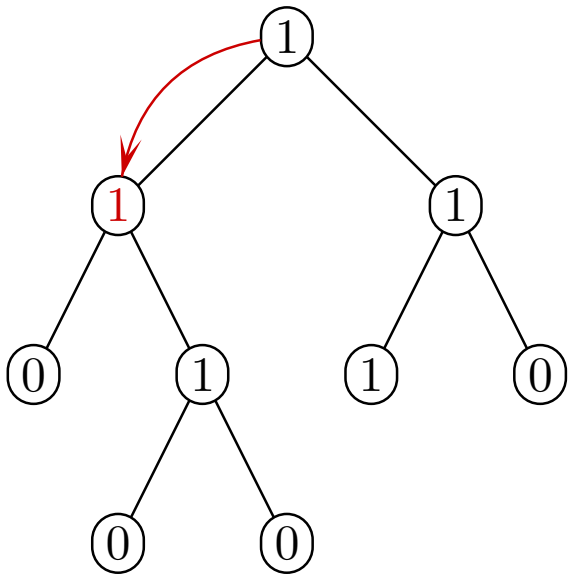

1

# Кодирование полного дерева

## Предложение

Полное бинарное дерево с $m$ листьями можно однозначно закодировать строкой из $2m - 1$ бит, при этом строка по дереву и дерево по строке строятся за время $O(m)$.

Обойдем дерево в глубину, записывая бит при первом заходе в каждую вершину: 1 для внутренней вершины, 0 — для листа. Этот процесс обратим: по строке можно строить дерево и сразу же обходить его в глубину. При чтении 1 у текущей вершины создается пара детей и текущей вершиной назначается левый ребенок; при чтении 0 текущая вершина помечается как лист, а управление возвращается ее родителю. □



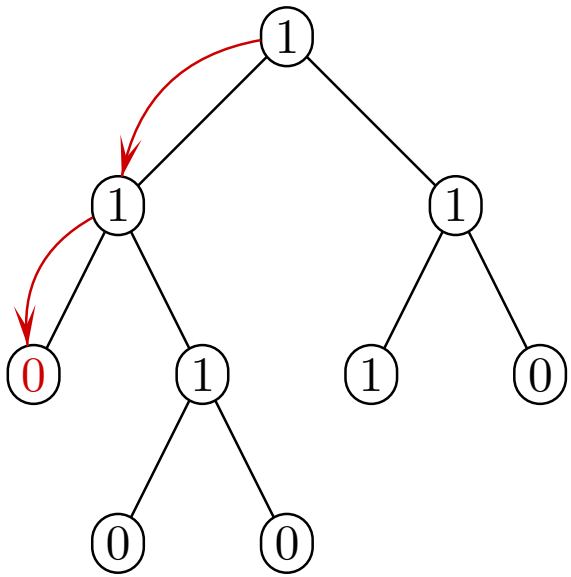

11

# Кодирование полного дерева

## Предложение

Полное бинарное дерево с $m$ листьями можно однозначно закодировать строкой из $2m - 1$ бит, при этом строка по дереву и дерево по строке строятся за время $O(m)$.

Обойдем дерево в глубину, записывая бит при первом заходе в каждую вершину: 1 для внутренней вершины, 0 — для листа. Этот процесс обратим: по строке можно строить дерево и сразу же обходить его в глубину. При чтении 1 у текущей вершины создается пара детей и текущей вершиной назначается левый ребенок; при чтении 0 текущая вершина помечается как лист, а управление возвращается ее родителю. □



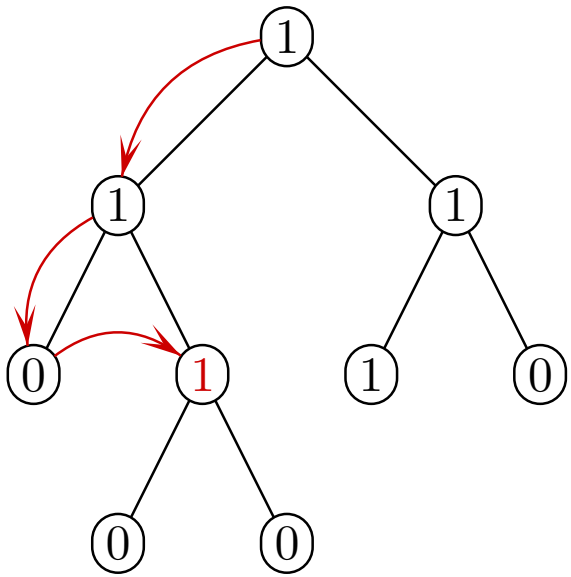

110

# Кодирование полного дерева

## Предложение

Полное бинарное дерево с $m$ листьями можно однозначно закодировать строкой из $2m - 1$ бит, при этом строка по дереву и дерево по строке строятся за время $O(m)$.

Обойдем дерево в глубину, записывая бит при первом заходе в каждую вершину: 1 для внутренней вершины, 0 — для листа. Этот процесс обратим: по строке можно строить дерево и сразу же обходить его в глубину. При чтении 1 у текущей вершины создается пара детей и текущей вершиной назначается левый ребенок; при чтении 0 текущая вершина помечается как лист, а управление возвращается ее родителю. □



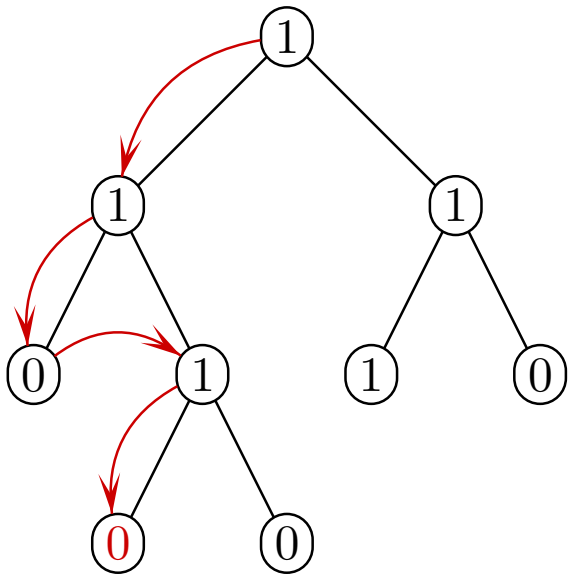

11101

# Кодирование полного дерева

## Предложение

Полное бинарное дерево с $m$ листьями можно однозначно закодировать строкой из $2m - 1$ бит, при этом строка по дереву и дерево по строке строятся за время $O(m)$.

Обойдем дерево в глубину, записывая бит при первом заходе в каждую вершину: 1 для внутренней вершины, 0 — для листа. Этот процесс обратим: по строке можно строить дерево и сразу же обходить его в глубину. При чтении 1 у текущей вершины создается пара детей и текущей вершиной назначается левый ребенок; при чтении 0 текущая вершина помечается как лист, а управление возвращается ее родителю. □



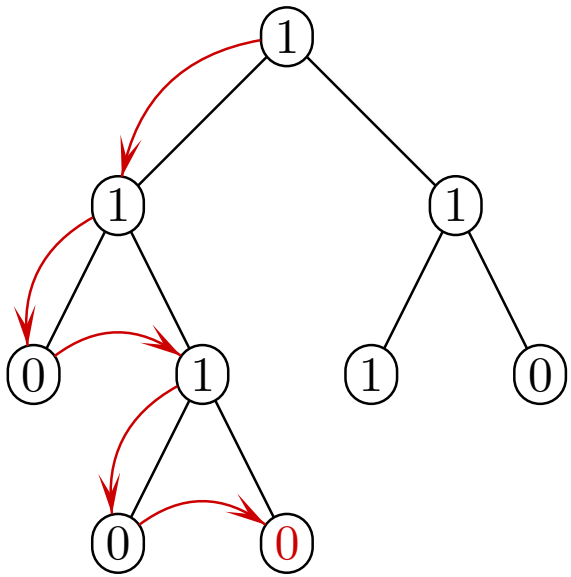

11010

# Кодирование полного дерева

## Предложение

Полное бинарное дерево с $m$ листьями можно однозначно закодировать строкой из $2m - 1$ бит, при этом строка по дереву и дерево по строке строятся за время $O(m)$.

Обойдем дерево в глубину, записывая бит при первом заходе в каждую вершину: 1 для внутренней вершины, 0 — для листа. Этот процесс обратим: по строке можно строить дерево и сразу же обходить его в глубину. При чтении 1 у текущей вершины создается пара детей и текущей вершиной назначается левый ребенок; при чтении 0 текущая вершина помечается как лист, а управление возвращается ее родителю. □



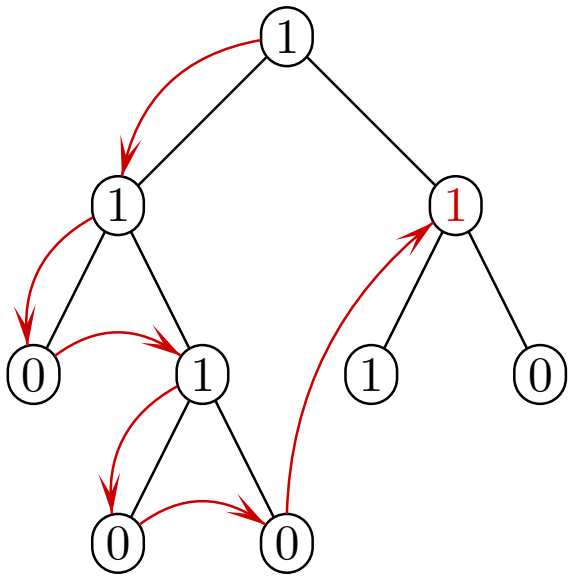

110100

# Кодирование полного дерева

## Предложение

Полное бинарное дерево с $m$ листьями можно однозначно закодировать строкой из $2m - 1$ бит, при этом строка по дереву и дерево по строке строятся за время $O(m)$.

Обойдем дерево в глубину, записывая бит при первом заходе в каждую вершину: 1 для внутренней вершины, 0 — для листа. Этот процесс обратим: по строке можно строить дерево и сразу же обходить его в глубину. При чтении 1 у текущей вершины создается пара детей и текущей вершиной назначается левый ребенок; при чтении 0 текущая вершина помечается как лист, а управление возвращается ее родителю. □



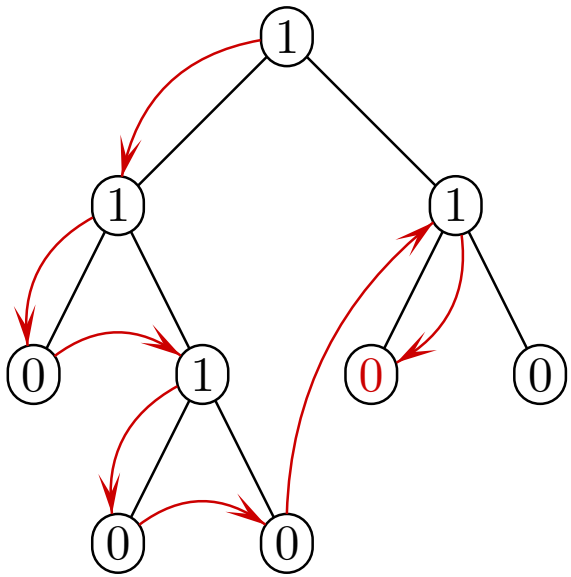

1101001

# Кодирование полного дерева

## Предложение

Полное бинарное дерево с $m$ листьями можно однозначно закодировать строкой из $2m - 1$ бит, при этом строка по дереву и дерево по строке строятся за время $O(m)$.

Обойдем дерево в глубину, записывая бит при первом заходе в каждую вершину: 1 для внутренней вершины, 0 — для листа. Этот процесс обратим: по строке можно строить дерево и сразу же обходить его в глубину. При чтении 1 у текущей вершины создается пара детей и текущей вершиной назначается левый ребенок; при чтении 0 текущая вершина помечается как лист, а управление возвращается ее родителю. □



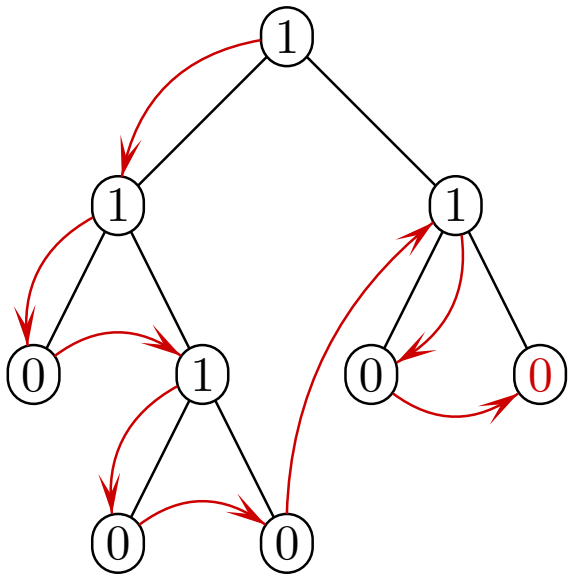

11010010

# Кодирование полного дерева

## Предложение

Полное бинарное дерево с $m$ листьями можно однозначно закодировать строкой из $2m - 1$ бит, при этом строка по дереву и дерево по строке строятся за время $O(m)$.

Обойдем дерево в глубину, записывая бит при первом заходе в каждую вершину: 1 для внутренней вершины, 0 — для листа. Этот процесс обратим: по строке можно строить дерево и сразу же обходить его в глубину. При чтении 1 у текущей вершины создается пара детей и текущей вершиной назначается левый ребенок; при чтении 0 текущая вершина помечается как лист, а управление возвращается ее родителю. □




110100100

# Хранение дерева Хаффмана

★ Чтобы закодировать дерево Хаффмана, используем приведенный выше способ, а после каждого нуля вставим $\log k$ бит “изначального” кода

- например, 8 бит ASCII-кода при $k = 256$

того символа, который кодируется соответствующим листом дерева

- дерево с предыдущего слайда может превратиться, например, в 110010000111001001000010001011001000001001010100

# Хранение дерева Хаффмана

★ Чтобы закодировать дерево Хаффмана, используем приведенный выше способ, а после каждого нуля вставим $\log k$ бит “изначального” кода

- например, 8 бит ASCII-кода при $k = 256$

того символа, который кодируется соответствующим листом дерева

- дерево с предыдущего слайда может превратиться, например, в 110010000111001001000010001011001000001001010100

- Если в алфавите $k$ символов, из которых в тексте реально встретились $\sigma$, то дерево Хаффмана кодируется строкой из $(2 + \log k)\sigma - 1$ бит
  - в полном дереве с $\sigma$ листьями число вершин равно $2\sigma - 1$

# Хранение дерева Хаффмана

★ Чтобы закодировать дерево Хаффмана, используем приведенный выше способ, а после каждого нуля вставим $\log k$ бит "изначального" кода

- например, 8 бит ASCII-кода при $k = 256$

того символа, который кодируется соответствующим листом дерева

- дерево с предыдущего слайда может превратиться, например, в 110010000111001001000010001011001000001001010100

- Если в алфавите $k$ символов, из которых в тексте реально встретились $\sigma$, то дерево Хаффмана кодируется строкой из $(2 + \log k)\sigma - 1$ бит
  - в полном дереве с $\sigma$ листьями число вершин равно $2\sigma - 1$

! Подумайте, сколько бит потребуется, чтобы передать явно множество всех кодовых слов, и убедитесь, что это невыгодно в большинстве случаев

! Убедитесь, что если явно передавать $\xi$ как массив частот символов (без округления, т.к. округление может изменить код Хаффмана), то потребуется еще больше места, чем на список кодовых слов

# Статическое и динамическое сжатие

Алгоритм статического сжатия текста $T$ по Хаффману

- посчитать частоты всех символов в $T$ (первый проход)
- построить дерево Хаффмана, вычислить его код $K$
- закодировать $T$ на построенном дереве (второй проход), выдать $K$ и $T$

# Статическое и динамическое сжатие

Алгоритм статического сжатия текста $T$ по Хаффману

- посчитать частоты всех символов в $T$ (первый проход)
- построить дерево Хаффмана, вычислить его код $K$
- закодировать $T$ на построенном дереве (второй проход), выдать $K$ и $T$

- Нужны два прохода, алгоритм не онлайновый (-), оба прохода быстрые (+)
- Нужно передавать дерево (-), декодеру не нужно его строить (+)
- ★ Плохо, если текст неоднородный — вначале одно распределение частот, потом другое; тогда при подсчете частот все усреднится, сильно увеличив энтропию, и на это повлиять нет возможности

# Статическое и динамическое сжатие

Алгоритм статического сжатия текста $T$ по Хаффману

- посчитать частоты всех символов в $T$ (первый проход)
- построить дерево Хаффмана, вычислить его код $K$
- закодировать $T$ на построенном дереве (второй проход), выдать $K$ и $T$

- Нужны два прохода, алгоритм не онлайновый (-), оба прохода быстрые (+)
- Нужно передавать дерево (-), декодеру не нужно его строить (+)
- ★ Плохо, если текст неоднородный — вначале одно распределение частот, потом другое; тогда при подсчете частот все усреднится, сильно увеличив энтропию, и на это повлиять нет возможности

Альтернатива статическому сжатию — динамическое:

- Алгоритм работает онлайн
- Модель источника перестраивается по ходу чтения текста, опираясь все время только на уже прочитанный текст
- Декодер точно так же, как и кодер, перестраивает модель источника, и поэтому всегда знает, на какой модели был закодирован очередной символ

# Статическое и динамическое сжатие

Алгоритм статического сжатия текста $T$ по Хаффману

- посчитать частоты всех символов в $T$ (первый проход)
- построить дерево Хаффмана, вычислить его код $K$
- закодировать $T$ на построенном дереве (второй проход), выдать $K$ и $T$

- Нужны два прохода, алгоритм не онлайновый (-), оба прохода быстрые (+)
- Нужно передавать дерево (-), декодеру не нужно его строить (+)
- ★ Плохо, если текст неоднородный — вначале одно распределение частот, потом другое; тогда при подсчете частот все усреднится, сильно увеличив энтропию, и на это повлиять нет возможности

Альтернатива статическому сжатию — динамическое:

- Алгоритм работает онлайн
- Модель источника перестраивается по ходу чтения текста, опираясь все время только на уже прочитанный текст
- Декодер точно так же, как и кодер, перестраивает модель источника, и поэтому всегда знает, на какой модели был закодирован очередной символ

- Онлайн (+), не надо передавать ничего, кроме сжатого текста (+)
- Алгоритм сложнее (-), больше работы декодеру (-)
- ★ Можно сделать адаптивным, подстраиваясь под неоднородный текст, масштабируя или отбрасывая старую статистику

# Динамический метод Хаффмана

Впервые предложены Фаллером (Faller) в 1973. Далее излагается версия алгоритма Фаллера–Галлагера–Кнута.

- Текст $T$ кодируется за один проход слева направо
- На шаге, на котором кодируется $T[i]$, текущее дерево Хаффмана построено по частотам в тексте $T[1..i-1]$
  - листья дерева помечены теми символами, которые уже встречались в тексте
  - символы, которых нет в $T[1..i-1]$, в дереве не представлены
  - для кодирования символов, встречаемых впервые, предусмотрена специальная вершина с меткой `new`
- При кодировании старого символа на выход пишется его код в текущем дереве Хаффмана
- При кодировании нового символа на выход пишется код символа `new` в текущем дереве Хаффмана + изначальный $(\log k)$-символьный код нового символа
- В начале дерево состоит из единственного листа с символом `new`
- ★ После прочтения $T[i]$ дерево перестраивается, чтобы стать деревом Хаффмана, построенным по частотам в тексте $T[1..i]$
  - Основная проблема — как сделать это быстро

# Динамический метод Хаффмана

Впервые предложены Фаллером (Faller) в 1973. Далее излагается версия алгоритма Фаллера–Галлагера–Кнута.

- Текст $T$ кодируется за один проход слева направо
- На шаге, на котором кодируется $T[i]$, текущее дерево Хаффмана построено по частотам в тексте $T[1..i-1]$
  - листья дерева помечены теми символами, которые уже встречались в тексте
  - символы, которых нет в $T[1..i-1]$, в дереве не представлены
  - для кодирования символов, встречаемых впервые, предусмотрена специальная вершина с меткой `new`
- При кодировании старого символа на выход пишется его код в текущем дереве Хаффмана
- При кодировании нового символа на выход пишется код символа `new` в текущем дереве Хаффмана + изначальный $(\log k)$-символьный код нового символа
- В начале дерево состоит из единственного листа с символом `new`
- ★ После прочтения $T[i]$ дерево перестраивается, чтобы стать деревом Хаффмана, построенным по частотам в тексте $T[1..i]$
  - Основная проблема — как сделать это быстро

# Динамический метод Хаффмана

Впервые предложены Фаллером (Faller) в 1973. Далее излагается версия алгоритма Фаллера–Галлагера–Кнута.

- Текст $T$ кодируется за один проход слева направо
- На шаге, на котором кодируется $T[i]$, текущее дерево Хаффмана построено по частотам в тексте $T[1..i-1]$
  - листья дерева помечены теми символами, которые уже встречались в тексте
  - символы, которых нет в $T[1..i-1]$, в дереве не представлены
  - для кодирования символов, встречаемых впервые, предусмотрена специальная вершина с меткой `new`
- При кодировании старого символа на выход пишется его код в текущем дереве Хаффмана
- При кодировании нового символа на выход пишется код символа `new` в текущем дереве Хаффмана + изначальный $(\log k)$-символьный код нового символа
- В начале дерево состоит из единственного листа с символом `new`
- ★ После прочтения $T[i]$ дерево перестраивается, чтобы стать деревом Хаффмана, построенным по частотам в тексте $T[1..i]$
  - Основная проблема — как сделать это быстро

# Динамический метод Хаффмана

Впервые предложены Фаллером (Faller) в 1973. Далее излагается версия алгоритма Фаллера–Галлагера–Кнута.

- Текст $T$ кодируется за один проход слева направо
- На шаге, на котором кодируется $T[i]$, текущее дерево Хаффмана построено по частотам в тексте $T[1..i-1]$
  - листья дерева помечены теми символами, которые уже встречались в тексте
  - символы, которых нет в $T[1..i-1]$, в дереве не представлены
  - для кодирования символов, встречаемых впервые, предусмотрена специальная вершина с меткой `new`
- При кодировании старого символа на выход пишется его код в текущем дереве Хаффмана
- При кодировании нового символа на выход пишется код символа `new` в текущем дереве Хаффмана + изначальный $(\log k)$-символьный код нового символа
- В начале дерево состоит из единственного листа с символом `new`
- ★ После прочтения $T[i]$ дерево перестраивается, чтобы стать деревом Хаффмана, построенным по частотам в тексте $T[1..i]$
  - Основная проблема — как сделать это быстро

# Свойство братьев

Допустим, что надо построить дерево Хаффмана с $m$ листьями

- Построение состоит из $m-1$ шага; на каждом два дерева сливаются в одно
- В процессе построения занумеруем вершины числами от 0 до $2m-2$:
  - на каждом шаге корни двух слитых деревьев получают пару самых больших доступных номеров, причем корень с бóльшим весом получает меньший (т.е. нечетный) номер (листья, слитые на первом шаге, получают номера $2m-3$ и $2m-2$, . . . , поддеревья, слитые последними — номера 1 и 2)
  - после этого корень получает номер 0

# Свойство братьев

Допустим, что надо построить дерево Хаффмана с $m$ листьями

- Построение состоит из $m-1$ шага; на каждом два дерева сливаются в одно
- В процессе построения занумеруем вершины числами от 0 до $2m-2$:
  - на каждом шаге корни двух слитых деревьев получают пару самых больших доступных номеров, причем корень с бóльшим весом получает меньший (т.е. нечетный) номер (листья, слитые на первом шаге, получают номера $2m-3$ и $2m-2$, ..., поддеревья, слитые последними — номера 1 и 2)
  - после этого корень получает номер 0
- Итак, получаем полное бинарное дерево с $m$ листьями, в котором
  1. вершины имеют положительные (можно и неотрицательные) веса
  2. вес отца равен сумме весов сыновей
  3. вершины занумерованы от 0 до $2m-2$ так, что $w(0) \geqslant w(1) \geqslant \cdots \geqslant w(2m-2)$ и для любого $i$ вершины $2i-1$ и $2i$ — братья

Выделенное свойство называется свойством братьев (sibling property

# Свойство братьев

Допустим, что надо построить дерево Хаффмана с $m$ листьями

- Построение состоит из $m-1$ шага; на каждом два дерева сливаются в одно
- В процессе построения занумеруем вершины числами от 0 до $2m-2$:
  - на каждом шаге корни двух слитых деревьев получают пару самых больших доступных номеров, причем корень с бóльшим весом получает меньший (т.е. нечетный) номер (листья, слитые на первом шаге, получают номера $2m-3$ и $2m-2$, ..., поддеревья, слитые последними — номера 1 и 2)
  - после этого корень получает номер 0
- Итак, получаем полное бинарное дерево с $m$ листьями, в котором
  1. вершины имеют положительные (можно и неотрицательные) веса
  2. вес отца равен сумме весов сыновей
  3. вершины занумерованы от 0 до $2m-2$ так, что $w(0) \geqslant w(1) \geqslant \cdots \geqslant w(2m-2)$ и для любого $i$ вершины $2i-1$ и $2i$ — братья

Выделенное свойство называется свойством братьев (sibling property

## Предложение

Полное бинарное дерево $D$ с $m$ листьями и свойствами 1–3 является деревом Хаффмана, построенным по набору $w_1, \ldots, w_m$ весов листьев $D$.

Нумерацию вершин дерева $D$ можно получить при построении дерева Хаффмана по набору $w_1, \ldots, w_m$ (алгоритм нумерации описан выше).

# Поддержание текущего дерева Хаффмана

Пусть текущее дерево Хаффмана построено по префиксу $T[1..i-1]$ сжимаемого текста; в префиксе $\sigma$ различных символов

Согласно Предложению, для поддержания текущего дерева Хаффмана достаточно преобразовывать дерево так, чтобы

- веса всех листьев совпадали с текущими частотами символов
- вес отца равнялся сумме весов сыновей
- вершины нумеровались от 0 до $2\sigma$
- $w(0) \geqslant w(1) \geqslant \cdots \geqslant w(2\sigma)$ и для любого $i$ вершины $2i-1$ и $2i$ были братьями

★ Договоримся приписать листу с меткой `new` постоянный вес 0; это гарантирует, что символ будет иметь номер $2\sigma$ (последний)

# Поддержание текущего дерева Хаффмана

Пусть текущее дерево Хаффмана построено по префиксу $T[1..i-1]$ сжимаемого текста; в префиксе $\sigma$ различных символов

Согласно Предложению, для поддержания текущего дерева Хаффмана достаточно преобразовывать дерево так, чтобы

- веса всех листьев совпадали с текущими частотами символов
- вес отца равнялся сумме весов сыновей
- вершины нумеровались от 0 до $2\sigma$
- $w(0) \geqslant w(1) \geqslant \cdots \geqslant w(2\sigma)$ и для любого $i$ вершины $2i-1$ и $2i$ были братьями

★ Договоримся приписать листу с меткой `new` постоянный вес 0; это гарантирует, что символ будет иметь номер $2\sigma$ (последний)

Если символ $x = T[i]$ — новый, добавление листа к дереву происходит так:

- добавляются вершины $2\sigma+1$ (метка $x$, вес 1) и $2\sigma+2$ (метка `new`, вес 0)
- с вершины $2\sigma$ снимается метка `new`, вес меняется на 1 и добавляются сыновья $2\sigma+1$ и $2\sigma+2$

# Поддержание текущего дерева Хаффмана

Пусть текущее дерево Хаффмана построено по префиксу $T[1..i-1]$ сжимаемого текста; в префиксе $\sigma$ различных символов

Согласно Предложению, для поддержания текущего дерева Хаффмана достаточно преобразовывать дерево так, чтобы

- веса всех листьев совпадали с текущими частотами символов
- вес отца равнялся сумме весов сыновей
- вершины нумеровались от 0 до $2\sigma$
- $w(0) \geqslant w(1) \geqslant \cdots \geqslant w(2\sigma)$ и для любого $i$ вершины $2i-1$ и $2i$ были братьями

★ Договоримся приписать листу с меткой `new` постоянный вес 0; это гарантирует, что символ будет иметь номер $2\sigma$ (последний)

Если символ $x = T[i]$ — новый, добавление листа к дереву происходит так:

- добавляются вершины $2\sigma+1$ (метка $x$, вес 1) и $2\sigma+2$ (метка `new`, вес 0)
- с вершины $2\sigma$ снимается метка `new`, вес меняется на 1 и добавляются сыновья $2\sigma+1$ и $2\sigma+2$

Основная процедура: обновление дерева

- Нужно добавить 1 к весу текущей вершины и всех ее предков, включая корень
  - текущая вершина — лист с меткой $x$, если $x$ — старый символ, и отец этого листа, т.е. бывший `new`, если $x$ — новый

★ При этом может нарушиться монотонность последовательности весов — некоторые вершины надо будет менять местами для ее восстановления $\Longrightarrow$

# Поддержание текущего дерева Хаффмана (пример)

$T = acaggaatacac$ (фрагмент последовательности ДНК)

# Поддержание текущего дерева Хаффмана (пример)

$T = acaggaatacac$ (фрагмент последовательности ДНК)


Инициализация
$T[1..0] = \lambda$, $\sigma = 0$

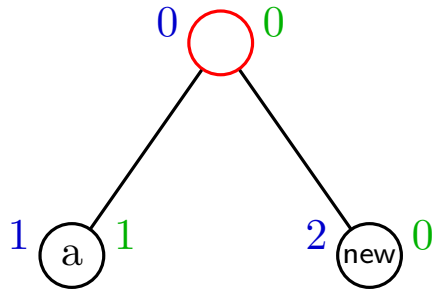
# Поддержание текущего дерева Хаффмана (пример)

$T = acaggaatacac$ (фрагмент последовательности ДНК)


Шаг 1:
$T[1..0] = \lambda$, $\sigma = 0$
$T[1] = a$ (новый символ)
$\mathtt{enc}(T[1..1]) = a$ (исх. код $a$)

# Поддержание текущего дерева Хаффмана (пример)

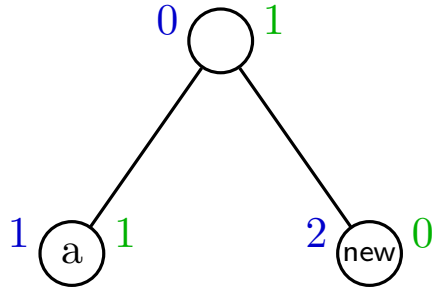
$T = acaggaatacac$ (фрагмент последовательности ДНК)




Шаг 1:
$T[1..0] = \lambda$, $\sigma = 0$
$T[1] = a$ (новый символ)
$\mathtt{enc}(T[1..1]) = a$ (исх. код $a$)

# Поддержание текущего дерева Хаффмана (пример)

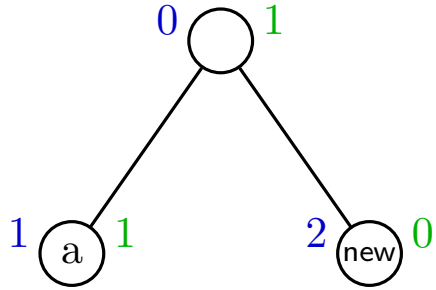
$T = acaggaatacac$ (фрагмент последовательности ДНК)




Шаг 1:
$T[1..0] = \lambda$, $\sigma = 0$
$T[1] = a$ (новый символ)
$\mathtt{enc}(T[1..1]) = a$ (исх. код $a$)

# Поддержание текущего дерева Хаффмана (пример)

$T = acaggaatacac$ (фрагмент последовательности ДНК)

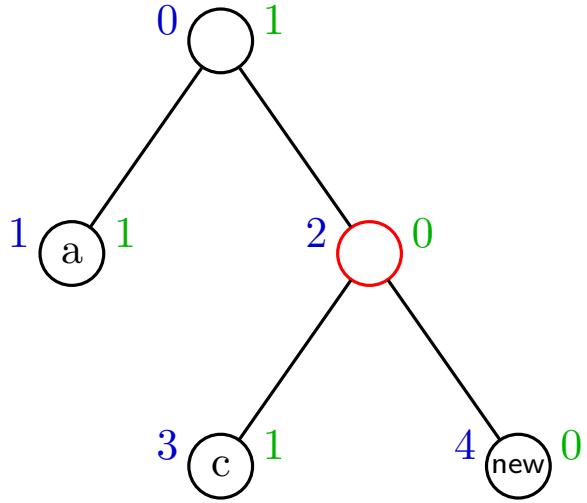



Шаг 2:
$T[1..1] = a$, $\sigma = 1$
$T[2] = c$ (новый символ)
$\mathtt{enc}(T[1..2]) = a1c$

# Поддержание текущего дерева Хаффмана (пример)

$T = acaggaatacac$ (фрагмент последовательности ДНК)

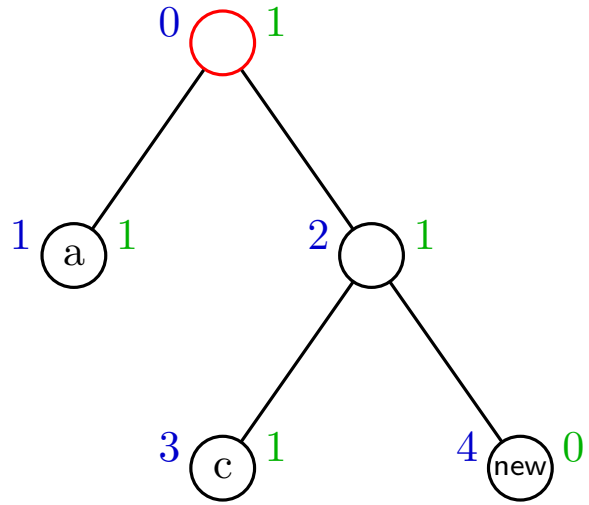
Шаг 2:
$T[1..1] = a$, $\sigma = 1$
$T[2] = c$ (новый символ)
$\mathtt{enc}(T[1..2]) = a1c$



# Поддержание текущего дерева Хаффмана (пример)

$T = acaggaatacac$ (фрагмент последовательности ДНК)

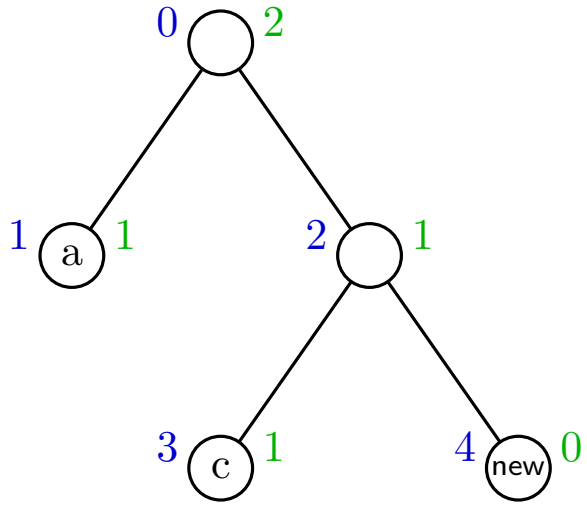
Шаг 2:
$T[1..1] = a$, $\sigma = 1$
$T[2] = c$ (новый символ)
$\mathtt{enc}(T[1..2]) = a1c$



# Поддержание текущего дерева Хаффмана (пример)

$T = acaggaatacac$ (фрагмент последовательности ДНК)

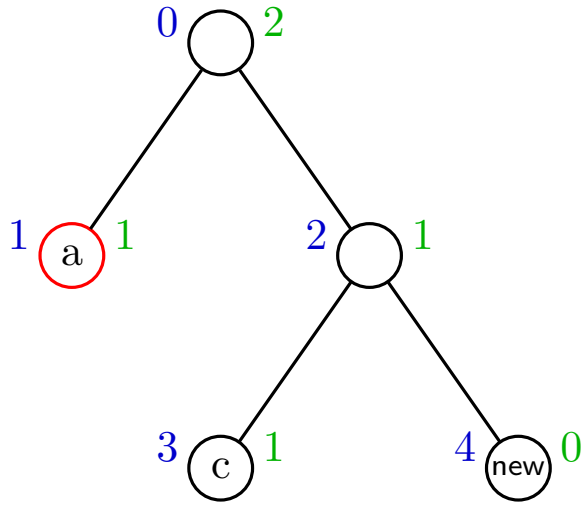
Шаг 2:
$T[1..1] = a$, $\sigma = 1$
$T[2] = c$ (новый символ)
$\mathtt{enc}(T[1..2]) = a1c$



# Поддержание текущего дерева Хаффмана (пример)

$T = acaggaatacac$ (фрагмент последовательности ДНК)

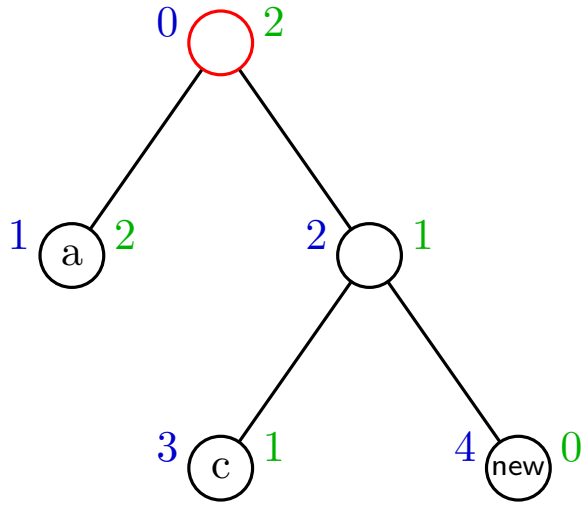
Шаг 3:
$T[1..2] = ac$, $\sigma = 2$
$T[3] = a$ (старый символ)
$\mathtt{enc}(T[1..3]) = a1c0$



# Поддержание текущего дерева Хаффмана (пример)

$T = acaggaatacac$ (фрагмент последовательности ДНК)

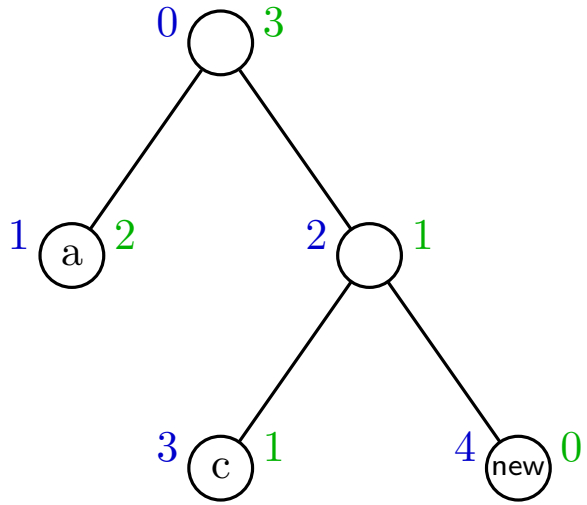
Шаг 3:
$T[1..2] = ac$, $\sigma = 2$
$T[3] = a$ (старый символ)
$\texttt{enc}(T[1..3]) = a1c0$



# Поддержание текущего дерева Хаффмана (пример)

$T = acaggaatacac$ (фрагмент последовательности ДНК)

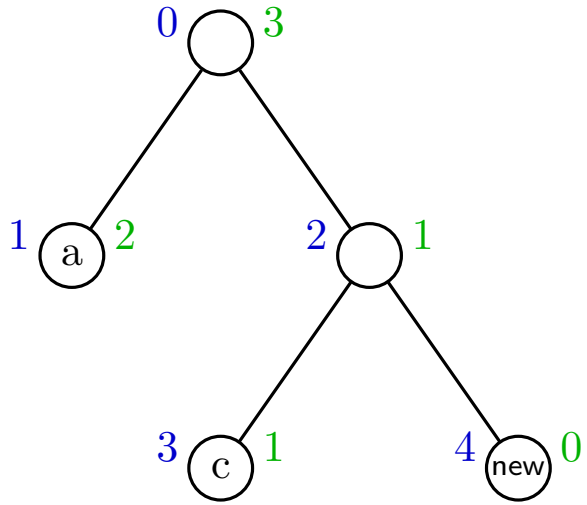
Шаг 3:
$T[1..2] = ac$, $\sigma = 2$
$T[3] = a$ (старый символ)
$\texttt{enc}(T[1..3]) = a1c0$



# Поддержание текущего дерева Хаффмана (пример)

$T = acaggaatacac$ (фрагмент последовательности ДНК)

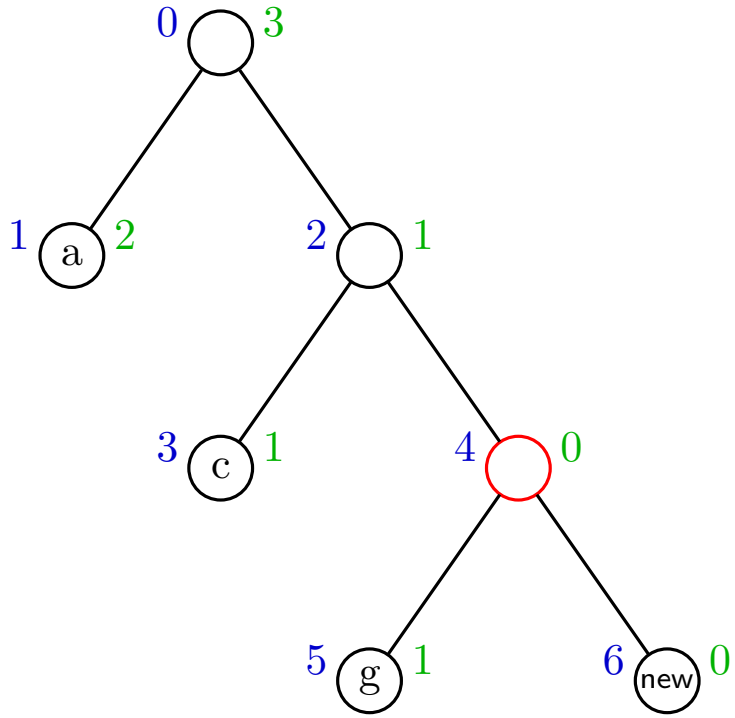
Шаг 4:
$T[1..3] = aca$, $\sigma = 2$
$T[4] = g$ (новый символ)
$\mathtt{enc}(T[1..4]) = a1c011g$



# Поддержание текущего дерева Хаффмана (пример)

$T = acaggaatacac$ (фрагмент последовательности ДНК)

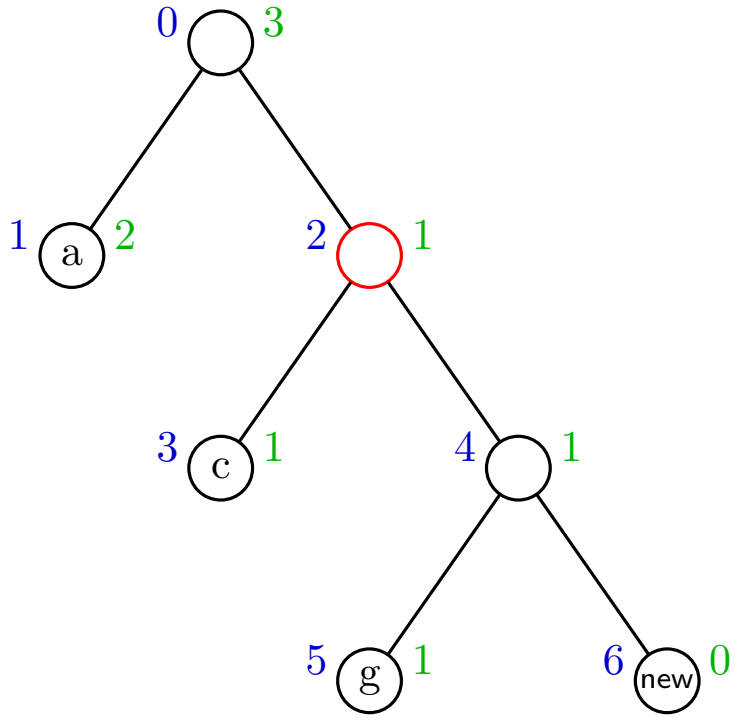
Шаг 4:
$T[1..3] = aca$, $\sigma = 2$
$T[4] = g$ (новый символ)
$\mathtt{enc}(T[1..4]) = a1c011g$



# Поддержание текущего дерева Хаффмана (пример)

$T = acaggaatacac$ (фрагмент последовательности ДНК)

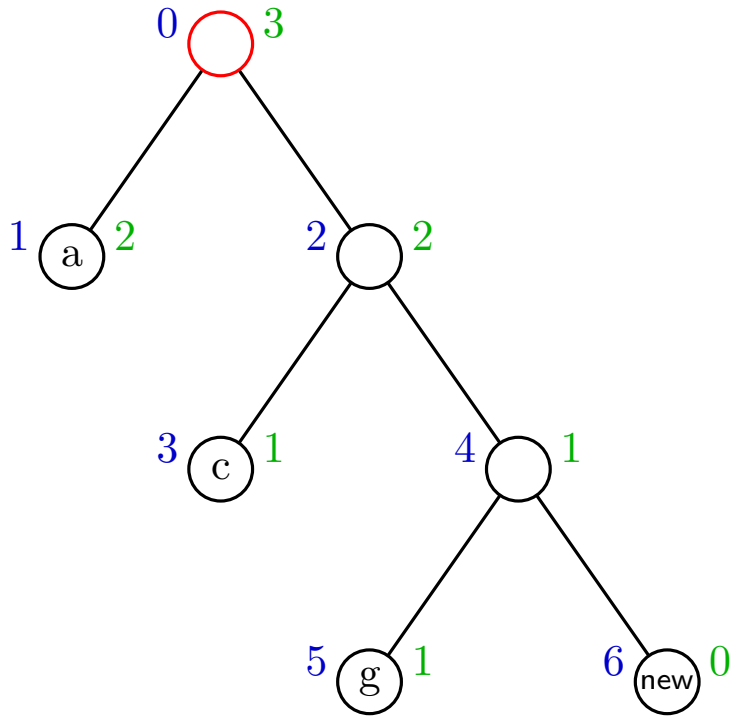
Шаг 4:
$T[1..3] = aca$, $\sigma = 2$
$T[4] = g$ (новый символ)
$\mathtt{enc}(T[1..4]) = a1c011g$



# Поддержание текущего дерева Хаффмана (пример)

$T = acaggaatacac$ (фрагмент последовательности ДНК)

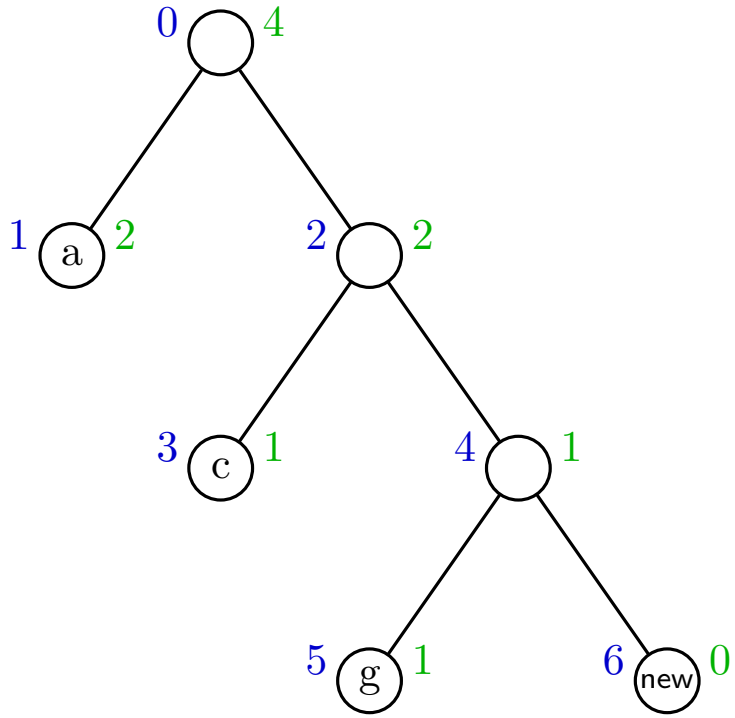
Шаг 4:
$T[1..3] = aca$, $\sigma = 2$
$T[4] = g$ (новый символ)
$\mathtt{enc}(T[1..4]) = a1c011g$



# Поддержание текущего дерева Хаффмана (пример)

$T = acaggaatacac$ (фрагмент последовательности ДНК)

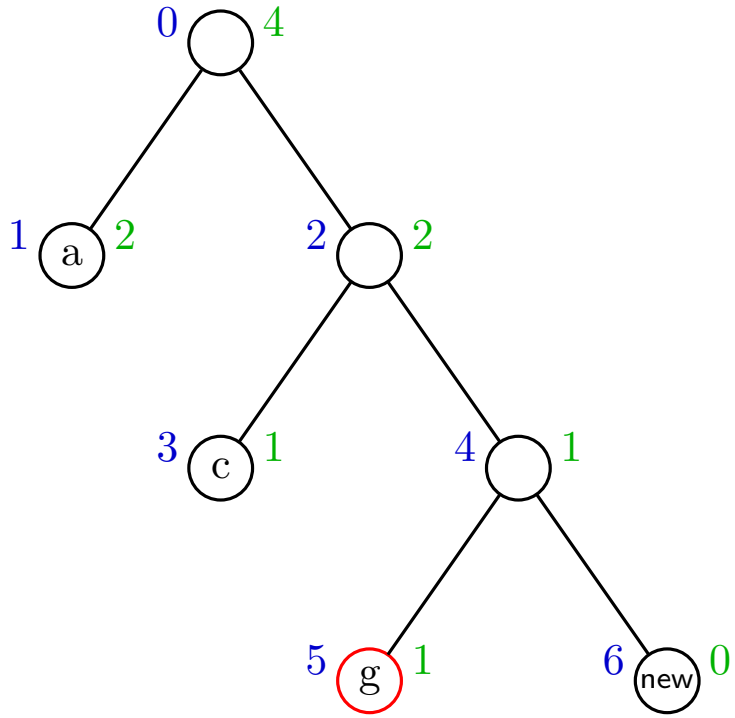
Шаг 4:
$T[1..3] = aca$, $\sigma = 2$
$T[4] = g$ (новый символ)
$\mathtt{enc}(T[1..4]) = a1c011g$



# Поддержание текущего дерева Хаффмана (пример)

$T = acaggaatacac$ (фрагмент последовательности ДНК)

Шаг 5:
$T[1..4] = acag$, $\sigma = 3$
$T[5] = g$ (старый символ)
$\mathtt{enc}(T[1..5]) = a1c011g110$




Проблема!
При увеличении $w(5)$ нарушится монотонность ($w(3) < w(5)$),
Нужно перестраивать дерево

# Структура данных

Будем представлять текущее дерево Хаффмана набором массивов с диапазоном индексов $[0..2\sigma]$

- если $k$-символьный алфавит, над которым задан текст $T$, не очень велик, то можно задать статические массивы с индексами $[0..2k]$, иначе потребуются какие-то динамические структуры; мы оставляем этот вопрос за скобками

# Структура данных

Будем представлять текущее дерево Хаффмана набором массивов с диапазоном индексов $[0..2\sigma]$

- если $k$-символьный алфавит, над которым задан текст $T$, не очень велик, то можно задать статические массивы с индексами $[0..2k]$, иначе потребуются какие-то динамические структуры; мы оставляем этот вопрос за скобками

Для каждой вершины $v$

| | |
|---|---|
| `parent`$[v]$ | родитель $v$ в дереве; `null` если $v = 0$ (корень) |
| `lson`$[v]$, `rson`$[v]$ | дети $v$ в дереве; `null` для листа |
| `symbol`$[v]$ | символ, сопоставленный $v$; `null` для внутренней вершины |
| `weight`$[v]$ | вес $v$ |

# Структура данных

Будем представлять текущее дерево Хаффмана набором массивов с диапазоном индексов $[0..2\sigma]$

- если $k$-символьный алфавит, над которым задан текст $T$, не очень велик, то можно задать статические массивы с индексами $[0..2k]$, иначе потребуются какие-то динамические структуры; мы оставляем этот вопрос за скобками

Для каждой вершины $v$

| | |
|---|---|
| $\texttt{parent}[v]$ | родитель $v$ в дереве; $\texttt{null}$ если $v = 0$ (корень) |
| $\texttt{lson}[v]$, $\texttt{rson}[v]$ | дети $v$ в дереве; $\texttt{null}$ для листа |
| $\texttt{symbol}[v]$ | символ, сопоставленный $v$; $\texttt{null}$ для внутренней вершины |
| $\texttt{weight}[v]$ | вес $v$ |

★ Декодер после декодирования $T[i]$ автоматически окажется в листе $v$ таком, что $T[i] = \texttt{symbol}[v]$

★ Кодеру для попадания в нужный лист после прочтения $T[i]$ требуется дополнительный массив указателей $\texttt{ptr}$ длины $k+1$:

- $\texttt{ptr}[x] = v$, если $\texttt{symbol}[v] = x$; $\texttt{ptr}[x] = \texttt{null}$ если такой вершины $v$ нет
- "лишний" символ — это $\texttt{new}$: если $\texttt{ptr}[x] = \texttt{null}$, нужный лист — это $\texttt{ptr}[\texttt{new}]$
- если $k$ очень велико, можно использовать хэши вместо массива

★ Для обновления дерева нужен массив (или словарь) $\texttt{head}$, где $\texttt{head}[w]$ — минимальный номер вершины с весом $w$ ($\texttt{null}$, если таких нет)

# Структура данных

Будем представлять текущее дерево Хаффмана набором массивов с диапазоном индексов $[0..2\sigma]$

- если $k$-символьный алфавит, над которым задан текст $T$, не очень велик, то можно задать статические массивы с индексами $[0..2k]$, иначе потребуются какие-то динамические структуры; мы оставляем этот вопрос за скобками

Для каждой вершины $v$

| | |
|---|---|
| $\texttt{parent}[v]$ | родитель $v$ в дереве; $\texttt{null}$ если $v = 0$ (корень) |
| $\texttt{lson}[v]$, $\texttt{rson}[v]$ | дети $v$ в дереве; $\texttt{null}$ для листа |
| $\texttt{symbol}[v]$ | символ, сопоставленный $v$; $\texttt{null}$ для внутренней вершины |
| $\texttt{weight}[v]$ | вес $v$ |

★ Декодер после декодирования $T[i]$ автоматически окажется в листе $v$ таком, что $T[i] = \texttt{symbol}[v]$

★ Кодеру для попадания в нужный лист после прочтения $T[i]$ требуется дополнительный массив указателей $\texttt{ptr}$ длины $k+1$:

- $\texttt{ptr}[x] = v$, если $\texttt{symbol}[v] = x$; $\texttt{ptr}[x] = \texttt{null}$ если такой вершины $v$ нет
- "лишний" символ — это $\texttt{new}$: если $\texttt{ptr}[x] = \texttt{null}$, нужный лист — это $\texttt{ptr}[\texttt{new}]$
- если $k$ очень велико, можно использовать хэши вместо массива

★ Для обновления дерева нужен массив (или словарь) $\texttt{head}$, где $\texttt{head}[w]$ — минимальный номер вершины с весом $w$ ($\texttt{null}$, если таких нет)

★ $\texttt{weight}[v] < \texttt{weight}[\texttt{parent}[v]]$ и $\texttt{head}[\texttt{weight}[v]] > \texttt{parent}[v]$; исключение:

- $v = 2\sigma - 1$ (брат $v$ имеет вес 0); тогда $\texttt{head}[\texttt{weight}[v]] \leqslant \texttt{parent}[v]$

# Алгоритм обновления дерева Хаффмана

К сожалению, без псевдокода не обойтись:

```
 1: function UpdateHuffTree(i, σ)                ▷ (позиция в T, текущий размер алфавита)
 2:     v_0 ← ptr[T[i]];                          ▷ две строки для кодера; декодер уже в v_0
 3:     if v_0 = null then v_0 ← parent[ptr[new]]
 4:     for (v ← v_0; v > 0; v ← parent[v]) do     ▷ пока не дойдем до корня
 5:         w ← weight[v]; u ← head[w]
 6:         if u < v and u ≠ parent[v] then        ▷ перевесим поддеревья с корнями u, v
 7:             lson[v] ↔ lson[u]; rson[v] ↔ rson[u]; symbol[v] ↔ symbol[u]
 8:             if symbol[v] ≠ null then ptr[symbol[v]] ← v      ▷ две строки для кодера
 9:             if symbol[u] ≠ null then ptr[symbol[u]] ← u
10:             v ← u
11:         if u ≠ parent[v] then                  ▷ исправим head под будущий вес v
12:             if w < weight[v−1] − 1 then head[w+1] ← v
13:             if v = 2σ−2 or weight[v+1] < w then head[w] ← null
14:         weight[v] ← w + 1                      ▷ наконец, увеличим вес v
15:     weight[0] ← weight[0] + 1                  ▷ увеличим вес корня
```

# Алгоритм обновления дерева Хаффмана

К сожалению, без псевдокода не обойтись:

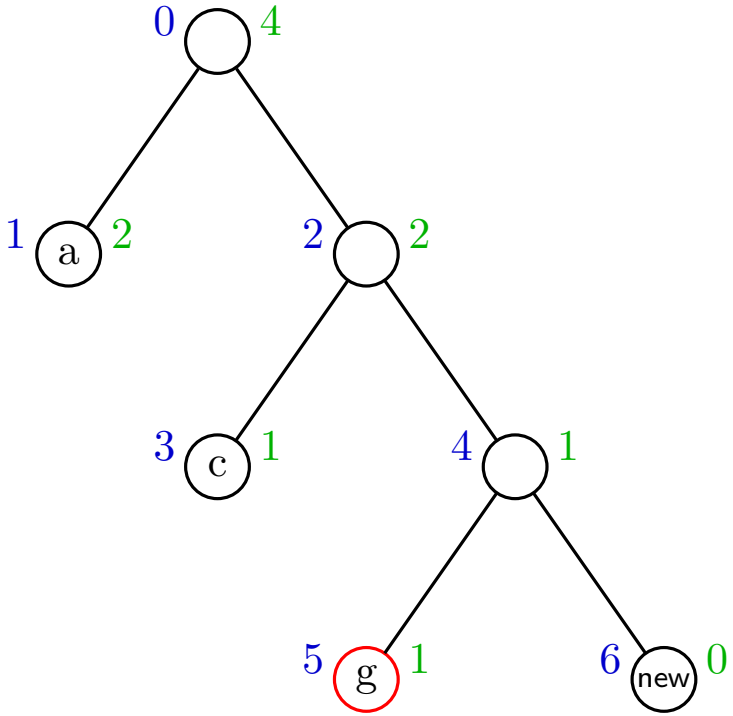
1: **function** *UpdateHuffTree*$(i, \sigma)$ ▷ (позиция в $T$, текущий размер алфавита)
2: $v_0 \leftarrow \texttt{ptr}[T[i]]$; ▷ две строки для кодера; декодер уже в $v_0$
3: **if** $v_0 = \texttt{null}$ **then** $v_0 \leftarrow \texttt{parent}[\texttt{ptr}[\texttt{new}]]$
4: **for** $(v \leftarrow v_0;\ v > 0;\ v \leftarrow \texttt{parent}[v])$ **do** ▷ пока не дойдем до корня
5: $w \leftarrow \texttt{weight}[v]$; $u \leftarrow \texttt{head}[w]$
6: **if** $u < v$ and $u \neq \texttt{parent}[v]$ **then** ▷ перевесим поддеревья с корнями $u$, $v$
7: $\texttt{lson}[v] \leftrightarrow \texttt{lson}[u]$; $\texttt{rson}[v] \leftrightarrow \texttt{rson}[u]$; $\texttt{symbol}[v] \leftrightarrow \texttt{symbol}[u]$
8: **if** $\texttt{symbol}[v] \neq \texttt{null}$ **then** $\texttt{ptr}[\texttt{symbol}[v]] \leftarrow v$ ▷ две строки для кодера
9: **if** $\texttt{symbol}[u] \neq \texttt{null}$ **then** $\texttt{ptr}[\texttt{symbol}[u]] \leftarrow u$
10: $v \leftarrow u$
11: **if** $u \neq \texttt{parent}[v]$ **then** ▷ исправим head под будущий вес $v$
12: **if** $w < \texttt{weight}[v-1] - 1$ **then** $\texttt{head}[w+1] \leftarrow v$
13: **if** $v = 2\sigma - 2$ or $\texttt{weight}[v+1] < w$ **then** $\texttt{head}[w] \leftarrow \texttt{null}$
14: $\texttt{weight}[v] \leftarrow w + 1$ ▷ наконец, увеличим вес $v$
15: $\texttt{weight}[0] \leftarrow \texttt{weight}[0] + 1$ ▷ увеличим вес корня

Обратите внимание на перевешивание поддеревьев (строка 7): на месте $u$ оказывается вершина с тем же родителем, но с детьми $v$ и символом $v$, т.е. на место $u$ перевешено поддерево с корнем $v$ (и наоборот); веса у вершин совпадали, так что меняться ими не требовалось.

# Поддержание текущего дерева Хаффмана (продолжение примера)

$T = acaggaatacac$

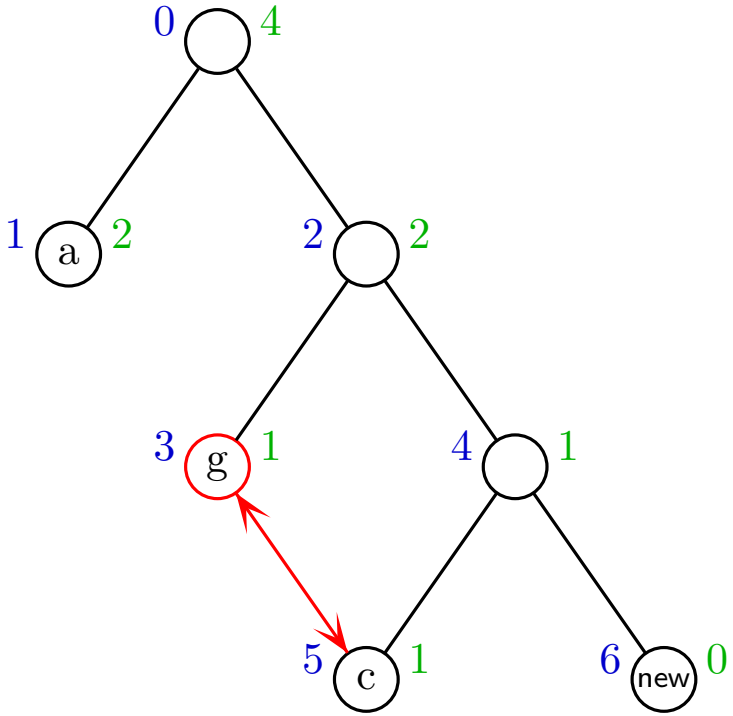
Шаг 5:
$T[1..4] = acag$, $\sigma = 3$
$T[5] = g$ (старый символ)
$\mathtt{enc}(T[1..5]) = a1c011g110$



# Поддержание текущего дерева Хаффмана (продолжение примера)

$T = acaggaatacac$

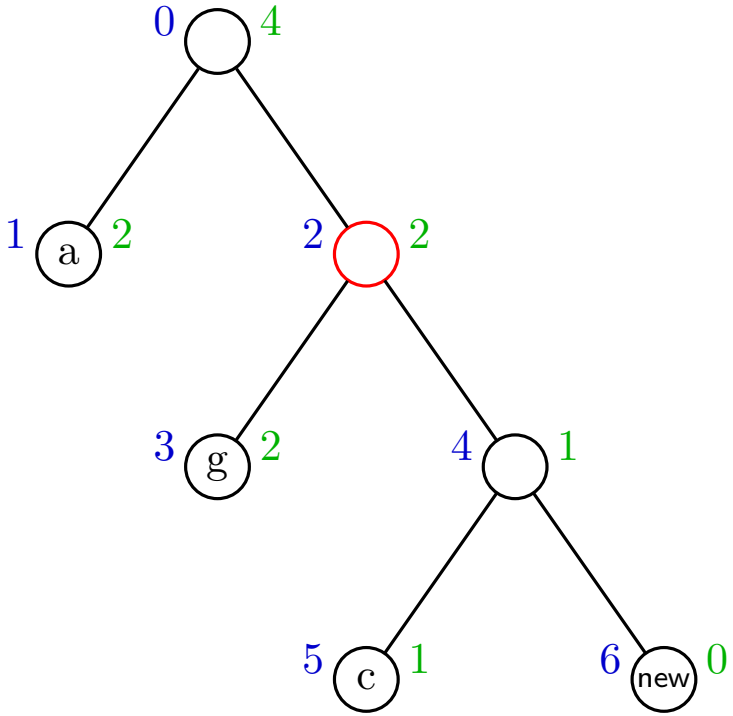
Шаг 5:
$T[1..4] = acag$, $\sigma = 3$
$T[5] = g$ (старый символ)
$\mathtt{enc}(T[1..5]) = a1c011g110$



# Поддержание текущего дерева Хаффмана (продолжение примера)

$T = acaggaatacac$

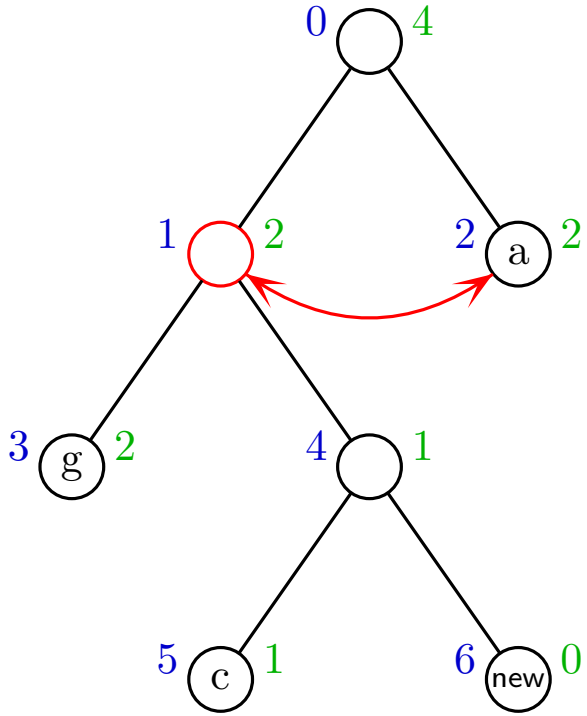
Шаг 5:
$T[1..4] = acag$, $\sigma = 3$
$T[5] = g$ (старый символ)
$\mathtt{enc}(T[1..5]) = a1c011g110$



# Поддержание текущего дерева Хаффмана (продолжение примера)

$T = acaggaatacac$

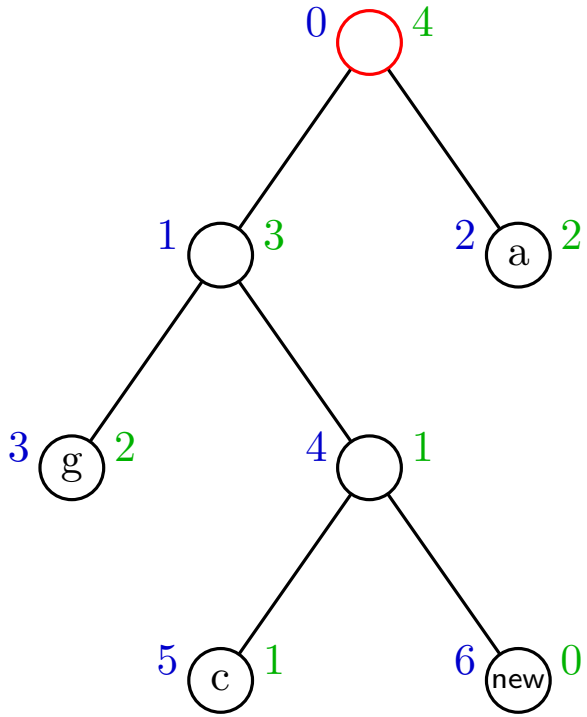
Шаг 5:
$T[1..4] = acag$, $\sigma = 3$
$T[5] = g$ (старый символ)
$\mathtt{enc}(T[1..5]) = a1c011g110$



# Поддержание текущего дерева Хаффмана (продолжение примера)

$T = acaggaatacac$

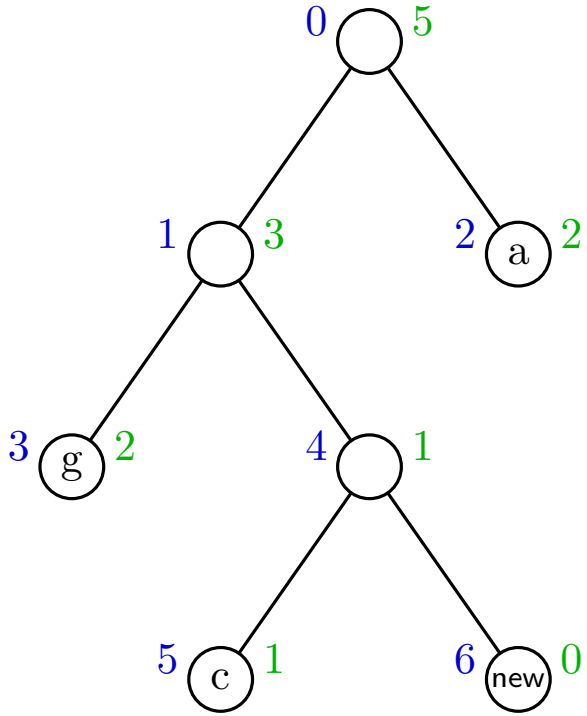
Шаг 5:
$T[1..4] = acag$, $\sigma = 3$
$T[5] = g$ (старый символ)
$\mathtt{enc}(T[1..5]) = a1c011g110$



# Поддержание текущего дерева Хаффмана (продолжение примера)

$T = acaggaatacac$

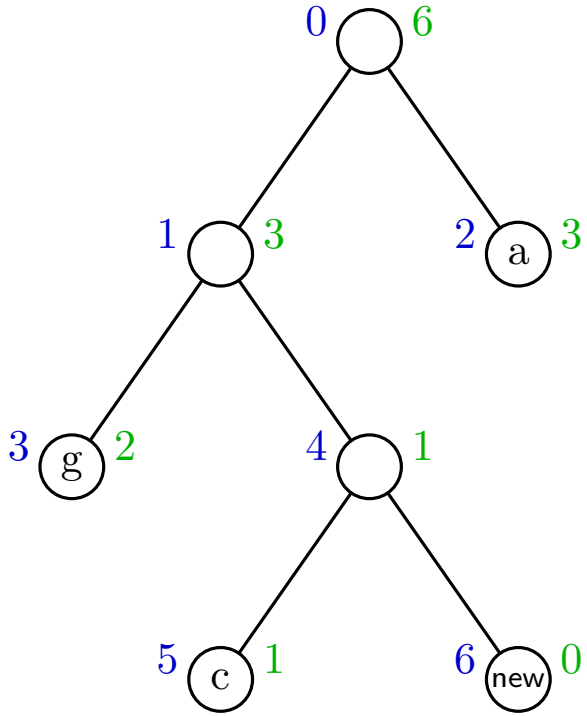
Шаг 5:
$T[1..4] = acag$, $\sigma = 3$
$T[5] = g$ (старый символ)
$\mathtt{enc}(T[1..5]) = a1c011g110$



# Поддержание текущего дерева Хаффмана (продолжение примера)

$T = acaggaatacac$

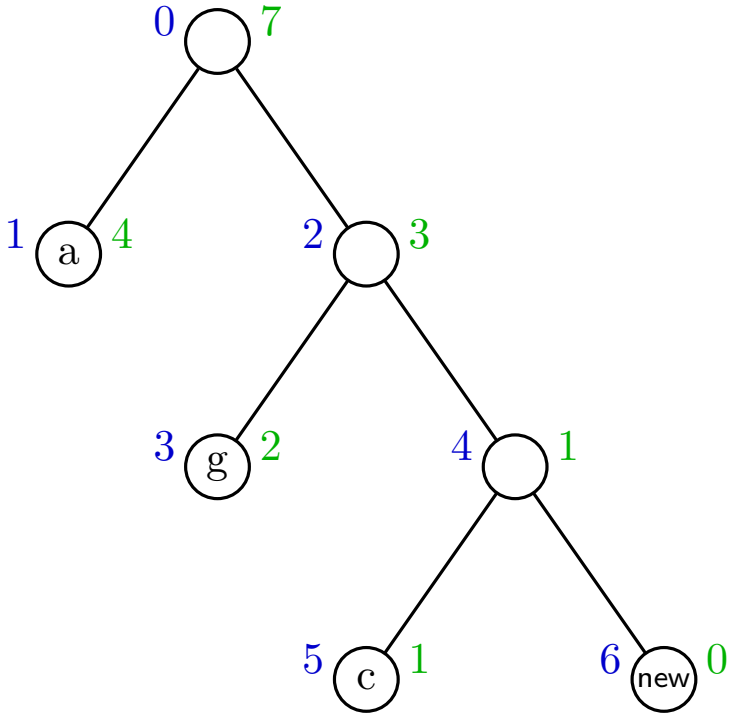
Шаг 6:
$T[1..5] = acagg$, $\sigma = 3$
$T[6] = a$ (старый символ)
$\mathtt{enc}(T[1..6]) = a1c011g1101$



# Поддержание текущего дерева Хаффмана (продолжение примера)

$T = acaggaatacac$

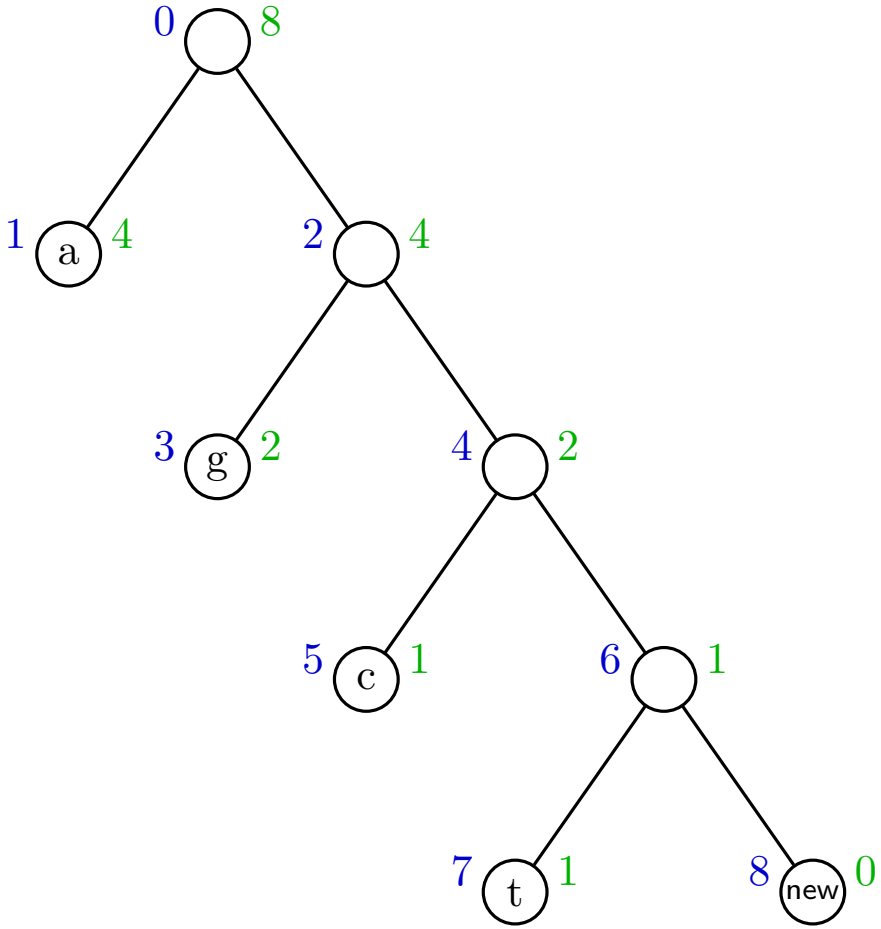
Шаг 7:
$T[1..6] = acagga$, $\sigma = 3$
$T[7] = a$ (старый символ)
$\mathtt{enc}(T[1..7]) = a1c011g11011$



# Поддержание текущего дерева Хаффмана (продолжение примера)

$T = acaggaatacac$

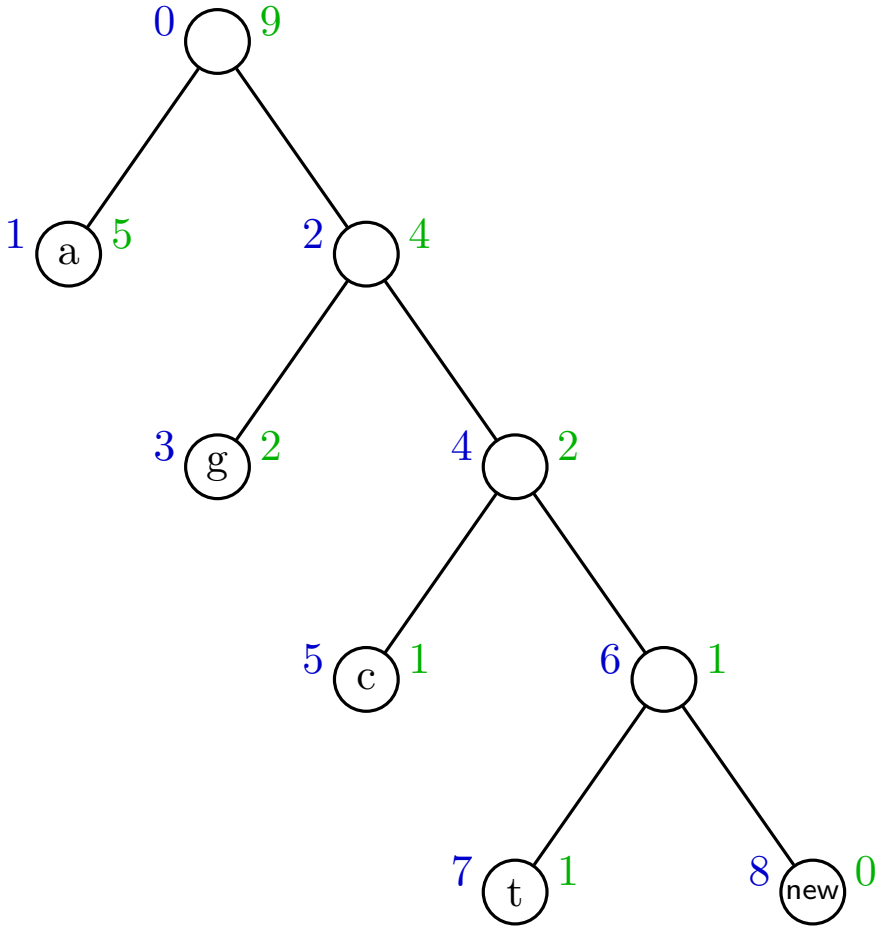
Шаг 8:
$T[1..7] = acaggaa$, $\sigma = 3$
$T[8] = t$ (новый символ)
$\mathtt{enc}(T[1..8]) = a1c011g1101111t$



# Поддержание текущего дерева Хаффмана (продолжение примера)

$T = acaggaatacac$

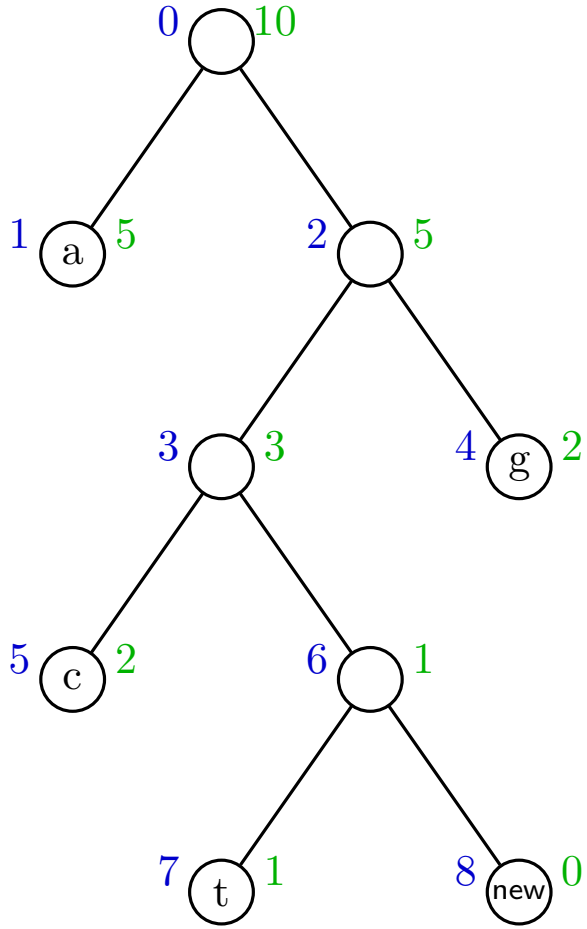
Шаг 9:
$T[1..8] = acaggaat$, $\sigma = 4$
$T[9] = a$ (старый символ)
$\mathtt{enc}(T[1..9]) = a1c011g1101111t0$



# Поддержание текущего дерева Хаффмана (продолжение примера)

$T = acaggaatacac$

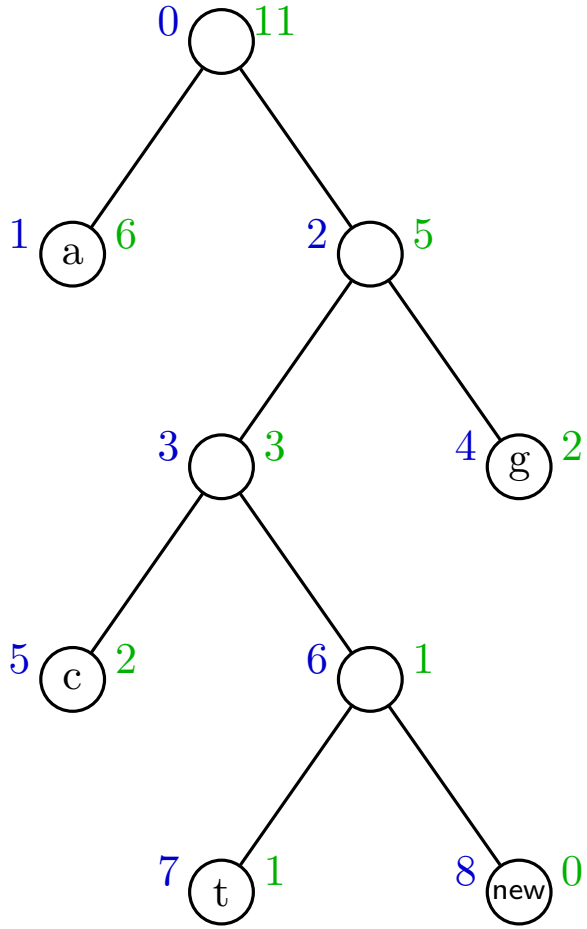
Шаг 10:
$T[1..9] = acaggaata$, $\sigma = 4$
$T[10] = c$ (старый символ)
$\mathtt{enc}(T[1..10]) =$
$a1c011g1101111t0110$



# Поддержание текущего дерева Хаффмана (продолжение примера)

$T = acaggaatacac$

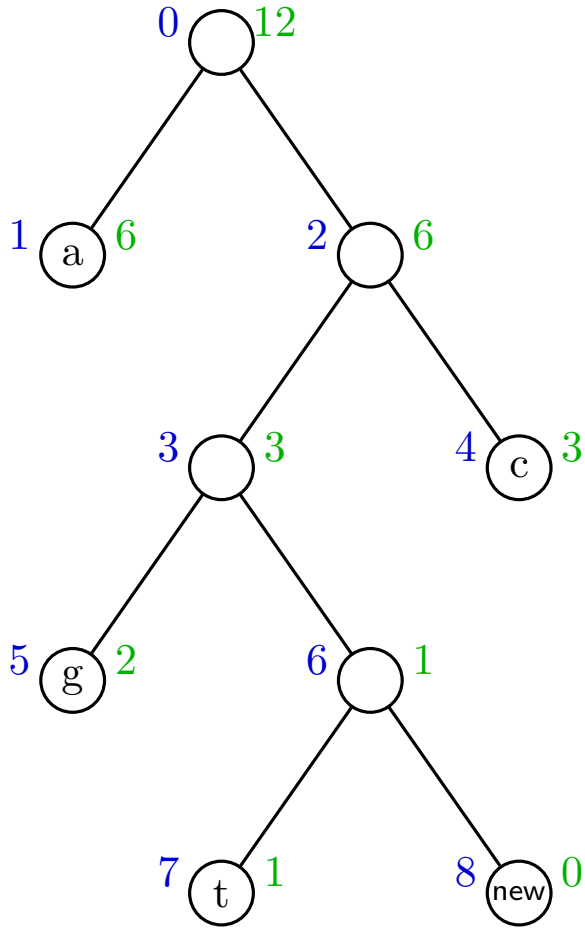
Шаг 11:
$T[1..10] = acaggaatac$, $\sigma = 4$
$T[11] = a$ (старый символ)
$\mathtt{enc}(T[1..11]) =$
$a1c011g1101111t01100$



# Поддержание текущего дерева Хаффмана (продолжение примера)

$T = acaggaatacac$

Шаг 12:
$T[1..11] = acaggaataca$, $\sigma = 4$
$T[12] = c$ (старый символ)
$\mathtt{enc}(T[1..12]) =$
$a1c011g1101111t01100100$



# Заключительные замечания

- Кодирование/декодирование символа $T(i)$ требует времени, пропорционального уровню листа $\texttt{ptr}[T(i)]$ в текущем дереве
  - декодирование делается проходом по дереву сверху вниз, а кодирование — снизу вверх (ввиду свойства братьев четность номера вершины говорит, левым или правым сыном она является; таким образом, подъем от листа до корня позволяет восстановить код символа “задом наперед”)
- Обновление дерева также требует времени, пропорционального уровню листа $\texttt{ptr}[T(i)]$
  - в предположении, что элементы структур `head` и `ptr` обновляются за время $O(1)$
- Поскольку уровень листа равен длине кода символа, общее время работы пропорционально $|\texttt{enc}(T)|$
- ★ На реальных данных динамическое сжатие работает немногим медленнее статического
  - совпадения весов встречаются редко: при длинном тексте возможных значений весов много больше, чем символов; т.е. операций “перевешивания” поддеревьев очень мало в общем объеме

# Заключительные замечания

- Кодирование/декодирование символа $T(i)$ требует времени, пропорционального уровню листа $\texttt{ptr}[T(i)]$ в текущем дереве
  - декодирование делается проходом по дереву сверху вниз, а кодирование — снизу вверх (ввиду свойства братьев четность номера вершины говорит, левым или правым сыном она является; таким образом, подъем от листа до корня позволяет восстановить код символа "задом наперед")
- Обновление дерева также требует времени, пропорционального уровню листа $\texttt{ptr}[T(i)]$
  - в предположении, что элементы структур `head` и `ptr` обновляются за время $O(1)$
- Поскольку уровень листа равен длине кода символа, общее время работы пропорционально $|\texttt{enc}(T)|$
- ★ На реальных данных динамическое сжатие работает немногим медленнее статического
  - совпадения весов встречаются редко: при длинном тексте возможных значений весов много больше, чем символов; т.е. операций "перевешивания" поддеревьев очень мало в общем объеме

- ★ Динамическое сжатие по Хаффману можно сделать адаптивным, подстраивающимся под изменения распределения символов; один из простых способов сделать это — масштабирование счетчиков: через каждые $N$ итераций веса всех листьев делят на 2 (с округлением вверх)
  - текущее распределение приближается лучше, так как свежая статистика вносит больший вклад, чем старая
  - дополнительно, можно сделать `head` массивом (веса достаточно малы)