# Лекция 3: Префиксное кодирование

А. М. Шур

Кафедра алгебры и фундаментальной информатики УрФУ

6 марта 2017 г.

Даны алфавит $\Sigma = \{x_1, \ldots, x_k\}$ и случайная величина (кубик) $\xi = (x_{1\,|p_1}, \ldots, x_{k\,|p_k})$. Требуется найти функцию enc : $\Sigma^* \to \{0,1\}^*$ для кодирования текстов, сгенерированных $\xi$, такую что

1) enc — инъекция
2) матожидание длины закодированного текста enc($T$) по всем текстам любой фиксированной длины $n$ минимально среди всех функций некоторого класса
3) enc и enc$^{-1}$ можно вычислить за линейное от размера входа время, желательно онлайн

# Задача кодирования

Даны алфавит $\Sigma = \{x_1, \dots, x_k\}$ и случайная величина (кубик) $\xi = (x_{1\,|\,p_1}, \dots, x_{k\,|\,p_k})$.
Требуется найти функцию $\mathrm{enc} : \Sigma^* \to \{0,1\}^*$ для кодирования текстов, сгенерированных $\xi$, такую что

1) enc — инъекция
2) матожидание длины закодированного текста $\mathrm{enc}(T)$ по всем текстам любой фиксированной длины $n$ минимально среди всех функций некоторого класса
3) enc и $\mathrm{enc}^{-1}$ можно вычислить за линейное от размера входа время, желательно онлайн

На этой лекции мы будем решать данную задачу самым естественным образом — сопоставляя каждому $x_i$ двоичный код символа:

$$x_1 \to C_1, \dots, x_k \to C_k, \text{ где } C_i \in \{0,1\}^*,$$

и кодируя $T = x_{i_1} x_{i_2} \cdots x_{i_n}$ последовательностью кодов символов:
$\mathrm{enc}(T) = C_{i_1} C_{i_2} \cdots C_{i_n}$

Даны алфавит $\Sigma = \{x_1, \ldots, x_k\}$ и случайная величина (кубик) $\xi = (x_{1\,|p_1}, \ldots, x_{k\,|p_k})$. Требуется найти функцию $\mathrm{enc} : \Sigma^* \to \{0,1\}^*$ для кодирования текстов, сгенерированных $\xi$, такую что

1) enc — инъекция
2) матожидание длины закодированного текста $\mathrm{enc}(T)$ по всем текстам любой фиксированной длины $n$ минимально среди всех функций некоторого класса
3) enc и $\mathrm{enc}^{-1}$ можно вычислить за линейное от размера входа время, желательно онлайн

На этой лекции мы будем решать данную задачу самым естественным образом — сопоставляя каждому $x_i$ двоичный код символа:

$$x_1 \to C_1, \ldots, x_k \to C_k, \text{ где } C_i \in \{0,1\}^*,$$

и кодируя $T = x_{i_1} x_{i_2} \cdots x_{i_n}$ последовательностью кодов символов: $\mathrm{enc}(T) = C_{i_1} C_{i_2} \cdots C_{i_n}$

Дело за малым — научиться выбирать $C_1, \ldots, C_k$ так, чтобы выполнить условия 1)-3). При выбранном способе кодирования функция enc очевидно вычисляется за линейное время, так что нас будет интересовать только быстрое декодирование.

## Алгебраические коды

Требование, чтобы функция enc была инъективной, называется однозначностью декодирования. Это требование выполнено тогда и только тогда, когда множество $\{C_1, \ldots, C_k\}$ кодов символов является алгебраическим кодом. Формально,

# Алгебраические коды

Требование, чтобы функция enc была инъективной, называется однозначностью декодирования. Это требование выполнено тогда и только тогда, когда множество $\{C_1, \ldots, C_k\}$ кодов символов является алгебраическим кодом. Формально,

- множество строк $\{C_1, \ldots, C_k\}$ называется конечным алгебраическим кодом, если любое равенство вида $C_{i_1} C_{i_2} \cdots C_{i_n} = C_{j_1} C_{j_2} \cdots C_{j_m}$ влечет $n = m$, $C_{i_1} = C_{j_1}, \ldots, C_{i_n} = C_{j_n}$.
  - $\{0, 01, 10\}$ — не код, так как $0 \cdot 10 = 01 \cdot 0$
  - $\{0, 01\}$ и $\{0, 10\}$ — коды (докажите!)

# Алгебраические коды

Требование, чтобы функция enc была инъективной, называется однозначностью декодирования. Это требование выполнено тогда и только тогда, когда множество $\{C_1, \ldots, C_k\}$ кодов символов является алгебраическим кодом. Формально,

- множество строк $\{C_1, \ldots, C_k\}$ называется конечным алгебраическим кодом, если любое равенство вида $C_{i_1} C_{i_2} \cdots C_{i_n} = C_{j_1} C_{j_2} \cdots C_{j_m}$ влечет $n = m$, $C_{i_1} = C_{j_1}, \ldots, C_{i_n} = C_{j_n}$.
  - $\{0, 01, 10\}$ — не код, так как $0 \cdot 10 = 01 \cdot 0$
  - $\{0, 01\}$ и $\{0, 10\}$ — коды (докажите!)

Простейший класс кодов: равномерные коды, в которых все строки — одной длины

- декодировать можно очень быстро
- неэффективность: кодирование одного символа требует $\lceil \log k \rceil$ бит независимо от энтропии $\xi$

# Алгебраические коды

Требование, чтобы функция enc была инъективной, называется однозначностью декодирования. Это требование выполнено тогда и только тогда, когда множество $\{C_1, \ldots, C_k\}$ кодов символов является алгебраическим кодом. Формально,

- множество строк $\{C_1, \ldots, C_k\}$ называется конечным алгебраическим кодом, если любое равенство вида $C_{i_1} C_{i_2} \cdots C_{i_n} = C_{j_1} C_{j_2} \cdots C_{j_m}$ влечет $n = m$, $C_{i_1} = C_{j_1}, \ldots, C_{i_n} = C_{j_n}$.
  - $\{0, 01, 10\}$ — не код, так как $0 \cdot 10 = 01 \cdot 0$
  - $\{0, 01\}$ и $\{0, 10\}$ — коды (докажите!)

Простейший класс кодов: равномерные коды, в которых все строки — одной длины

- декодировать можно очень быстро
- неэффективность: кодирование одного символа требует $\lceil \log k \rceil$ бит независимо от энтропии $\xi$

Обобщением равномерных кодов, сочетающим эффективность кодирования и быстроту декодирования, являются префиксные коды:

- множество строк $\{C_1, \ldots, C_k\}$, в котором ни одна строка не является префиксом другой строки, называется префиксным кодом.
  - $\{0, 10\}$ — префиксный код, а $\{0, 01\}$ — нет
  - ★ декодируется за один проход: на каждой итерации читаем enc$(T)$ с текущей позиции слева направо, пока прочитанная строка не совпадет с кодом символа; добавляем символ к $T$ и обнуляем прочитанную строку, завершая итерацию

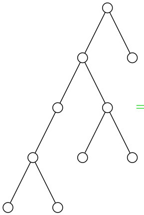
- Бинарным деревом называется корневое дерево, каждая вершина в котором имеет не более двух сыновей, причем любой из четырех вариантов (нет сыновей; только левый сын; только правый сын; оба сына) возможен
  - полное, если любая вершина имеет 0 или 2 сыновей

# Префиксные коды и бинарные деревья

- Бинарным деревом называется корневое дерево, каждая вершина в котором имеет не более двух сыновей, причем любой из четырех вариантов (нет сыновей; только левый сын; только правый сын; оба сына) возможен
  - полное, если любая вершина имеет 0 или 2 сыновей

Бинарные префиксные коды и бинарные деревья — это, фактически, одно и то же:

- Возьмем бинарное дерево, напишем на каждом ребре от отца к левому (правому) сыну 0 (соотв., 1) и сопоставим каждому листу $w$ строку $s(w)$ от корня до этого листа. Множество таких строк — префиксный код (если $s(u)$ — префикс $s(v)$, то $u$ — предок $v$ в дереве, что невозможно, так как $u$ — лист).



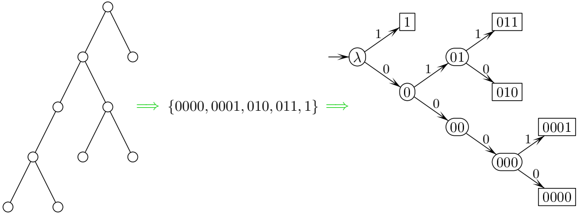
$\Longrightarrow \{0000, 0001, 010, 011, 1\}$

# Префиксные коды и бинарные деревья

- Бинарным деревом называется корневое дерево, каждая вершина в котором имеет не более двух сыновей, причем любой из четырех вариантов (нет сыновей; только левый сын; только правый сын; оба сына) возможен
  - полное, если любая вершина имеет 0 или 2 сыновей

Бинарные префиксные коды и бинарные деревья — это, фактически, одно и то же:

- Возьмем бинарное дерево, напишем на каждом ребре от отца к левому (правому) сыну 0 (соотв., 1) и сопоставим каждому листу $w$ строку $s(w)$ от корня до этого листа. Множество таких строк — префиксный код (если $s(u)$ — префикс $s(v)$, то $u$ — предок $v$ в дереве, что невозможно, так как $u$ — лист).




- Обратно, возьмем бинарный префиксный код и построим распознающий его бор (см. Лекцию 2). Этот бор является бинарным деревом: из каждой вершины выходит не больше одного "левого" ребра с меткой 0 и не больше одного "правого" с меткой 1.

# Декодирование префиксного кода

Представление префиксного кода деревом позволяет его очень быстро декодировать:

- Пометим каждый лист дерева символом, который он кодирует
- Назначим корень дерева текущей вершиной
- Читаем enc($T$) побитово, каждый раз назначая текущей вершиной сына текущей вершины, соответствующего прочитанному биту
- Если текущая вершина — лист, добавляем к $T$ символ, кодируемый этим листом, снова назначаем корень текущей вершиной и продолжаем чтение

# Декодирование префиксного кода

Представление префиксного кода деревом позволяет его очень быстро декодировать:

- Пометим каждый лист дерева символом, который он кодирует
- Назначим корень дерева текущей вершиной
- Читаем $\mathrm{enc}(T)$ побитово, каждый раз назначая текущей вершиной сына текущей вершины, соответствующего прочитанному биту
- Если текущая вершина — лист, добавляем к $T$ символ, кодируемый этим листом, снова назначаем корень текущей вершиной и продолжаем чтение

Таким образом, число операций на декодирование одного бита является константой, не зависящей от $|\Sigma|$

# Неравенство Крафта-Макмиллана

Одно из фундаментальных свойств алгебраических кодов выражает

## Теорема Макмиллана (неравенство Макмиллана)

Алгебраический код над $m$-буквенным алфавитом, состоящий из строк с длинами $\ell_1, \ldots, \ell_k$, существует тогда и только тогда, когда $\sum_{i=1}^{k} m^{-\ell_i} \leqslant 1$.

(Этот же результат верен и для бесконечных алгебраических кодов.)

# Неравенство Крафта-Макмиллана

Одно из фундаментальных свойств алгебраических кодов выражает

### Теорема Макмиллана (неравенство Макмиллана)

Алгебраический код над $m$-буквенным алфавитом, состоящий из строк с длинами $\ell_1, \ldots, \ell_k$, существует тогда и только тогда, когда $\sum_{i=1}^{k} m^{-\ell_i} \leqslant 1$.
(Этот же результат верен и для бесконечных алгебраических кодов.)

Нас интересуют бинарные префиксные коды; мы докажем более частный результат

### Теорема Крафта (неравенство Крафта)

Бинарный префиксный код, состоящий из строк с длинами $\ell_1, \ldots, \ell_k$, существует тогда и только тогда, когда $\sum_{i=1}^{k} 2^{-\ell_i} \leqslant 1$.

## Неравенство Крафта-Макмиллана

Одно из фундаментальных свойств алгебраических кодов выражает

**Теорема Макмиллана (неравенство Макмиллана)**

Алгебраический код над $m$-буквенным алфавитом, состоящий из строк с длинами $\ell_1, \ldots, \ell_k$, существует тогда и только тогда, когда $\sum\limits_{i=1}^{k} m^{-\ell_i} \leqslant 1$.

(Этот же результат верен и для бесконечных алгебраических кодов.)

Нас интересуют бинарные префиксные коды; мы докажем более частный результат

**Теорема Крафта (неравенство Крафта)**

Бинарный префиксный код, состоящий из строк с длинами $\ell_1, \ldots, \ell_k$, существует тогда и только тогда, когда $\sum\limits_{i=1}^{k} 2^{-\ell_i} \leqslant 1$.

Ввиду соответствия между префиксными кодами и деревьями, достаточно показать, что

(1) если $\ell_1, \ldots, \ell_k$ — уровни всех листьев бинарного дерева, то $\sum_{i=1}^{k} 2^{-\ell_i} \leqslant 1$
  - уровень вершины есть число ребер на пути от корня до нее

(2) если числа $\ell_1, \ldots, \ell_k$ таковы, что $\sum_{i=1}^{k} 2^{-\ell_i} \leqslant 1$, то можно построить бинарное дерево с $k$ листьями на уровнях $\ell_1, \ldots, \ell_k$

(1) Пусть $D$ — дерево, $S = \sum_{u \in L} 2^{-\mathrm{lev}(u)}$, где $L$ — множество листьев $D$, а lev обозначает уровень вершины.

- Если $D$ состоит только из корня, то $S = 2^0 = 1$
- Если $D$ полное, возьмем вершину $u$ максимального уровня; это лист, и его брат $v$ (он существует, так как дерево полное) — тоже лист (иначе ниже есть вершины большего уровня)
- Удалим из дерева $u$ и $v$; чтобы пересчитать $S$, надо
  - вычесть вклад $u$, равный $2^{-\mathrm{lev}(u)}$, и такой же вклад $v$;
  - добавить вклад их отца, ставшего листом, равный $2^{-(\mathrm{lev}(u)-1)}$;

  таким образом, $S$ не изменился при данном преобразовании
- Повторяя удаление пары листьев нужное число раз, получим дерево, состоящее из корня, для которого $S = 1$; значит, $S = 1$ для любого полного дерева
- Если $D$ — неполное, добавим к каждой вершине с одним сыном новый лист в качестве второго сына; такое преобразование строго увеличивает $S$, а в результате получается полное дерево, для которого $S = 1$; значит, $S < 1$ для любого неполного дерева

# Доказательство теоремы Крафта (1)

(1) Пусть $D$ — дерево, $S = \sum_{u \in L} 2^{-\mathrm{lev}(u)}$, где $L$ — множество листьев $D$, а lev обозначает уровень вершины.

- Если $D$ состоит только из корня, то $S = 2^0 = 1$
- Если $D$ полное, возьмем вершину $u$ максимального уровня; это лист, и его брат $v$ (он существует, так как дерево полное) — тоже лист (иначе ниже есть вершины большего уровня)
- Удалим из дерева $u$ и $v$; чтобы пересчитать $S$, надо
  - вычесть вклад $u$, равный $2^{-\mathrm{lev}(u)}$, и такой же вклад $v$;
  - добавить вклад их отца, ставшего листом, равный $2^{-(\mathrm{lev}(u)-1)}$;

  таким образом, $S$ не изменился при данном преобразовании
- Повторяя удаление пары листьев нужное число раз, получим дерево, состоящее из корня, для которого $S = 1$; значит, $S = 1$ для любого полного дерева
- Если $D$ — неполное, добавим к каждой вершине с одним сыном новый лист в качестве второго сына; такое преобразование строго увеличивает $S$, а в результате получается полное дерево, для которого $S = 1$; значит, $S < 1$ для любого неполного дерева

(1) Пусть $D$ — дерево, $S = \sum_{u \in L} 2^{-\mathrm{lev}(u)}$, где $L$ — множество листьев $D$, а lev обозначает уровень вершины.

- Если $D$ состоит только из корня, то $S = 2^0 = 1$
- Если $D$ полное, возьмем вершину $u$ максимального уровня; это лист, и его брат $v$ (он существует, так как дерево полное) — тоже лист (иначе ниже есть вершины большего уровня)
- Удалим из дерева $u$ и $v$; чтобы пересчитать $S$, надо
  - вычесть вклад $u$, равный $2^{-\mathrm{lev}(u)}$, и такой же вклад $v$;
  - добавить вклад их отца, ставшего листом, равный $2^{-(\mathrm{lev}(u)-1)}$;

  таким образом, $S$ не изменился при данном преобразовании
- Повторяя удаление пары листьев нужное число раз, получим дерево, состоящее из корня, для которого $S = 1$; значит, $S = 1$ для любого полного дерева
- Если $D$ — неполное, добавим к каждой вершине с одним сыном новый лист в качестве второго сына; такое преобразование строго увеличивает $S$, а в результате получается полное дерево, для которого $S = 1$; значит, $S < 1$ для любого неполного дерева

(1) Пусть $D$ — дерево, $S = \sum_{u \in L} 2^{-\mathrm{lev}(u)}$, где $L$ — множество листьев $D$, а lev обозначает уровень вершины.

- Если $D$ состоит только из корня, то $S = 2^0 = 1$
- Если $D$ полное, возьмем вершину $u$ максимального уровня; это лист, и его брат $v$ (он существует, так как дерево полное) — тоже лист (иначе ниже есть вершины большего уровня)
- Удалим из дерева $u$ и $v$; чтобы пересчитать $S$, надо
  - вычесть вклад $u$, равный $2^{-\mathrm{lev}(u)}$, и такой же вклад $v$;
  - добавить вклад их отца, ставшего листом, равный $2^{-(\mathrm{lev}(u)-1)}$;

  таким образом, $S$ не изменился при данном преобразовании
- Повторяя удаление пары листьев нужное число раз, получим дерево, состоящее из корня, для которого $S = 1$; значит, $S = 1$ для любого полного дерева
- Если $D$ — неполное, добавим к каждой вершине с одним сыном новый лист в качестве второго сына; такое преобразование строго увеличивает $S$, а в результате получается полное дерево, для которого $S = 1$; значит, $S < 1$ для любого неполного дерева

(1) Пусть $D$ — дерево, $S = \sum_{u \in L} 2^{-\mathrm{lev}(u)}$, где $L$ — множество листьев $D$, а lev обозначает уровень вершины.

- Если $D$ состоит только из корня, то $S = 2^0 = 1$
- Если $D$ полное, возьмем вершину $u$ максимального уровня; это лист, и его брат $v$ (он существует, так как дерево полное) — тоже лист (иначе ниже есть вершины большего уровня)
- Удалим из дерева $u$ и $v$; чтобы пересчитать $S$, надо
  - вычесть вклад $u$, равный $2^{-\mathrm{lev}(u)}$, и такой же вклад $v$;
  - добавить вклад их отца, ставшего листом, равный $2^{-(\mathrm{lev}(u)-1)}$;

  таким образом, $S$ не изменился при данном преобразовании
- Повторяя удаление пары листьев нужное число раз, получим дерево, состоящее из корня, для которого $S = 1$; значит, $S = 1$ для любого полного дерева
- Если $D$ — неполное, добавим к каждой вершине с одним сыном новый лист в качестве второго сына; такое преобразование строго увеличивает $S$, а в результате получается полное дерево, для которого $S = 1$; значит, $S < 1$ для любого неполного дерева

# Доказательство теоремы Крафта (2)

(2) Упорядочим числа $\ell_1, \ldots, \ell_k$ по возрастанию; далее считаем, что $\ell_1 \leqslant \ldots \leqslant \ell_k$.

# Доказательство теоремы Крафта (2)

(2) Упорядочим числа $\ell_1, \ldots, \ell_k$ по возрастанию; далее считаем, что $\ell_1 \leqslant \ldots \leqslant \ell_k$.

- Рассмотрим множество $B$ всех бинарных строк длины $\leqslant \ell_k$
- Выберем произвольную строку длины $\ell_1$ в качестве $C_1$
- Вычеркнем из $B$ все строки с префиксом $C_1$ — они не могут быть в префиксном коде вместе с $C_1$
  - будет вычеркнуто $2^{\ell-\ell_1}$ строк длины $\ell$ для любого $\ell \in [\ell_1, \ell_n]$
- Выберем произвольную невычеркнутую строку длины $\ell_2$ в качестве $C_2$ и вычеркнем из $B$ все строки с префиксом $C_2$
  - Ни одна из вычеркиваемых строк не вычеркнута ранее, поскольку у ранее вычеркнутых строк длины $\geqslant \ell_2$ префиксы длины $\ell_2$ тоже были вычеркнуты
- Повторяем предыдущий шаг, выбирая $C_3, \ldots, C_k$

# Доказательство теоремы Крафта (2)

(2) Упорядочим числа $\ell_1, \ldots, \ell_k$ по возрастанию; далее считаем, что $\ell_1 \leqslant \ldots \leqslant \ell_k$.

- Рассмотрим множество $B$ всех бинарных строк длины $\leqslant \ell_k$
- Выберем произвольную строку длины $\ell_1$ в качестве $C_1$
- Вычеркнем из $B$ все строки с префиксом $C_1$ — они не могут быть в префиксном коде вместе с $C_1$
  - будет вычеркнуто $2^{\ell-\ell_1}$ строк длины $\ell$ для любого $\ell \in [\ell_1, \ell_n]$
- Выберем произвольную невычеркнутую строку длины $\ell_2$ в качестве $C_2$ и вычеркнем из $B$ все строки с префиксом $C_2$
  - Ни одна из вычеркиваемых строк не вычеркнута ранее, поскольку у ранее вычеркнутых строк длины $\geqslant \ell_2$ префиксы длины $\ell_2$ тоже были вычеркнуты
- Повторяем предыдущий шаг, выбирая $C_3, \ldots, C_k$

Для каждого $j = 2, \ldots, k$ в момент выбора $C_j$ вычеркнуто

$$\sum_{i=1}^{j-1} 2^{\ell_j - \ell_i} < 2^{\ell_j} \sum_{i=1}^{k} 2^{-\ell_i} \leqslant 2^{\ell_j}$$

строк длины $\ell_j$, т.е. гарантированно существует невычеркнутая строка на роль $C_j$. Значит, все $k$ строк можно выбрать, и они образуют префиксный код по построению (каждая строка не содержит в качестве префиксов ранее выбранных, т.е. более коротких, строк).

# Оптимизация ожидаемой длины кодовой строки

Мы уже знаем, что префиксные коды позволяют быстро и однозначно декодировать закодированный текст. Осталось научиться выбирать префиксный код, минимизирующий ожидаемую длину кода символа для текстов, порожденных заданным кубиком $\xi = (x_{1\,|p_1}, \ldots, x_{k\,|p_k})$.

# Оптимизация ожидаемой длины кодовой строки

Мы уже знаем, что префиксные коды позволяют быстро и однозначно декодировать закодированный текст. Осталось научиться выбирать префиксный код, минимизирующий ожидаемую длину кода символа для текстов, порожденных заданным кубиком $\xi = (x_{1\,|p_1}, \ldots, x_{k\,|p_k})$.

- Будем считать без ограничения общности, что $p_1 \geqslant p_2 \geqslant \cdots \geqslant p_k > 0$
- Пусть $\ell_1, \ell_2, \ldots, \ell_k$ — (неизвестные) длины кодов символов $x_1, \ldots, x_k$ соответственно
- Тогда матожидание длины кода символа — величина, которую мы хотим минимизировать — равна $\sum_{i=1}^{k} p_i \ell_i$
- По теореме Крафта, префиксный код с длинами строк $\ell_1, \ell_2, \ldots, \ell_k$ существует тогда и только тогда, когда $\sum_{i=1}^{k} 2^{-\ell_i} \leqslant 1$
- Как следует из доказательства теоремы Крафта, если префиксный код с данными длинами строк существует, его можно эффективно построить

# Оптимизация ожидаемой длины кодовой строки

Мы уже знаем, что префиксные коды позволяют быстро и однозначно декодировать закодированный текст. Осталось научиться выбирать префиксный код, минимизирующий ожидаемую длину кода символа для текстов, порожденных заданным кубиком $\xi = (x_{1\,|p_1}, \dots, x_{k\,|p_k})$.

- Будем считать без ограничения общности, что $p_1 \geqslant p_2 \geqslant \dots \geqslant p_k > 0$
- Пусть $\ell_1, \ell_2, \dots, \ell_k$ — (неизвестные) длины кодов символов $x_1, \dots, x_k$ соответственно
- Тогда матожидание длины кода символа — величина, которую мы хотим минимизировать — равна $\sum_{i=1}^{k} p_i \ell_i$
- По теореме Крафта, префиксный код с длинами строк $\ell_1, \ell_2, \dots, \ell_k$ существует тогда и только тогда, когда $\sum_{i=1}^{k} 2^{-\ell_i} \leqslant 1$
- Как следует из доказательства теоремы Крафта, если префиксный код с данными длинами строк существует, его можно эффективно построить

Итак, нужно решить следующую задачу математического программирования:

### Задача

Найти целые числа $\ell_1, \ell_2, \dots, \ell_k$, минимизирующие значение линейной комбинации $\sum_{i=1}^{k} p_i \ell_i$ при (нелинейном) ограничении $\sum_{i=1}^{k} 2^{-\ell_i} \leqslant 1$.

# Оптимизация ожидаемой длины кодовой строки

Мы уже знаем, что префиксные коды позволяют быстро и однозначно декодировать закодированный текст. Осталось научиться выбирать префиксный код, минимизирующий ожидаемую длину кода символа для текстов, порожденных заданным кубиком $\xi = (x_{1\,|p_1}, \ldots, x_{k\,|p_k})$.

- Будем считать без ограничения общности, что $p_1 \geqslant p_2 \geqslant \cdots \geqslant p_k > 0$
- Пусть $\ell_1, \ell_2, \ldots, \ell_k$ — (неизвестные) длины кодов символов $x_1, \ldots, x_k$ соответственно
- Тогда матожидание длины кода символа — величина, которую мы хотим минимизировать — равна $\sum_{i=1}^{k} p_i \ell_i$
- По теореме Крафта, префиксный код с длинами строк $\ell_1, \ell_2, \ldots, \ell_k$ существует тогда и только тогда, когда $\sum_{i=1}^{k} 2^{-\ell_i} \leqslant 1$
- Как следует из доказательства теоремы Крафта, если префиксный код с данными длинами строк существует, его можно эффективно построить

Итак, нужно решить следующую задачу математического программирования:

**Задача**

Найти целые числа $\ell_1, \ell_2, \ldots, \ell_k$, минимизирующие значение линейной комбинации $\sum_{i=1}^{k} p_i \ell_i$ при (нелинейном) ограничении $\sum_{i=1}^{k} 2^{-\ell_i} \leqslant 1$.

★ Решение всегда является полным префиксным кодом (докажите!)

# Коды Шеннона-Фано

Рассмотрим следующий алгоритм (Шеннона-Фано) построения полного бинарного дерева по набору вероятностей $p_1 \geqslant p_2 \geqslant \cdots \geqslant p_k > 0$ кубика $\xi$:

- Начать с дерева из одного корня, пометив его списком $p_1, \ldots, p_k$
- Пока в дереве существуют листья, помеченные более чем одним числом
  - выбрать любой такой лист $v$
  - разбить помечающий $v$ список $p_i, \ldots, p_j$ на два списка $p_i, \ldots, p_{i+r}$ и $p_{i+r+1}, \ldots, p_j$ так, чтобы суммы списков были максимально близки, и создать двух сыновей $v$, пометив их данными списками
- Каждый лист построенного дерева помечен некоторым $p_i$ и соответствует коду символа $x_i$

Рассмотрим следующий алгоритм (Шеннона-Фано) построения полного бинарного дерева по набору вероятностей $p_1 \geqslant p_2 \geqslant \cdots \geqslant p_k > 0$ кубика $\xi$:

- Начать с дерева из одного корня, пометив его списком $p_1, \dots, p_k$
- Пока в дереве существуют листья, помеченные более чем одним числом
  - выбрать любой такой лист $v$
  - разбить помечающий $v$ список $p_i, \dots, p_j$ на два списка $p_i, \dots, p_{i+r}$ и $p_{i+r+1}, \dots, p_j$ так, чтобы суммы списков были максимально близки, и создать двух сыновей $v$, пометив их данными списками
- Каждый лист построенного дерева помечен некоторым $p_i$ и соответствует коду символа $x_i$

$$\boxed{\frac{5}{13}, \frac{2}{13}, \frac{2}{13}, \frac{2}{13}, \frac{2}{13}}$$

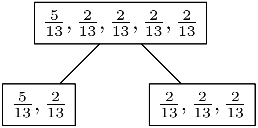
Пример:
$p_1 = \frac{5}{13}$, $p_2 = p_3 = p_4 = p_5 = \frac{2}{13}$.

# Коды Шеннона-Фано

Рассмотрим следующий алгоритм (Шеннона-Фано) построения полного бинарного дерева по набору вероятностей $p_1 \geqslant p_2 \geqslant \cdots \geqslant p_k > 0$ кубика $\xi$:

- Начать с дерева из одного корня, пометив его списком $p_1, \ldots, p_k$
- Пока в дереве существуют листья, помеченные более чем одним числом
  - выбрать любой такой лист $v$
  - разбить помечающий $v$ список $p_i, \ldots, p_j$ на два списка $p_i, \ldots, p_{i+r}$ и $p_{i+r+1}, \ldots, p_j$ так, чтобы суммы списков были максимально близки, и создать двух сыновей $v$, пометив их данными списками
- Каждый лист построенного дерева помечен некоторым $p_i$ и соответствует коду символа $x_i$

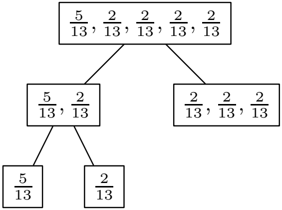
Пример:
$p_1 = \frac{5}{13}$, $p_2 = p_3 = p_4 = p_5 = \frac{2}{13}$.



Рассмотрим следующий алгоритм (Шеннона-Фано) построения полного бинарного дерева по набору вероятностей $p_1 \geqslant p_2 \geqslant \dots \geqslant p_k > 0$ кубика $\xi$:

- Начать с дерева из одного корня, пометив его списком $p_1, \dots, p_k$
- Пока в дереве существуют листья, помеченные более чем одним числом
  - выбрать любой такой лист $v$
  - разбить помечающий $v$ список $p_i, \dots, p_j$ на два списка $p_i, \dots, p_{i+r}$ и $p_{i+r+1}, \dots, p_j$ так, чтобы суммы списков были максимально близки, и создать двух сыновей $v$, пометив их данными списками
- Каждый лист построенного дерева помечен некоторым $p_i$ и соответствует коду символа $x_i$

Пример:
$p_1 = \frac{5}{13}$, $p_2 = p_3 = p_4 = p_5 = \frac{2}{13}$.

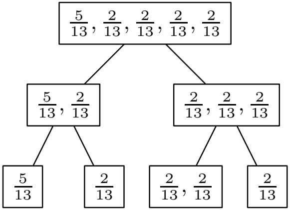


# Коды Шеннона-Фано

Рассмотрим следующий алгоритм (Шеннона-Фано) построения полного бинарного дерева по набору вероятностей $p_1 \geqslant p_2 \geqslant \dots \geqslant p_k > 0$ кубика $\xi$:

- Начать с дерева из одного корня, пометив его списком $p_1, \dots, p_k$
- Пока в дереве существуют листья, помеченные более чем одним числом
  - выбрать любой такой лист $v$
  - разбить помечающий $v$ список $p_i, \dots, p_j$ на два списка $p_i, \dots, p_{i+r}$ и $p_{i+r+1}, \dots, p_j$ так, чтобы суммы списков были максимально близки, и создать двух сыновей $v$, пометив их данными списками
- Каждый лист построенного дерева помечен некоторым $p_i$ и соответствует коду символа $x_i$

Пример:
$p_1 = \frac{5}{13}$, $p_2 = p_3 = p_4 = p_5 = \frac{2}{13}$.



# Коды Шеннона-Фано

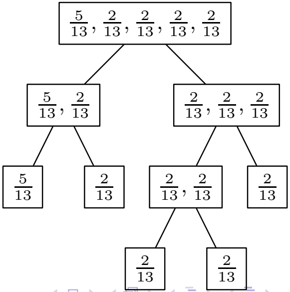
Рассмотрим следующий алгоритм (Шеннона-Фано) построения полного бинарного дерева по набору вероятностей $p_1 \geqslant p_2 \geqslant \cdots \geqslant p_k > 0$ кубика $\xi$:

- Начать с дерева из одного корня, пометив его списком $p_1, \ldots, p_k$
- Пока в дереве существуют листья, помеченные более чем одним числом
  - выбрать любой такой лист $v$
  - разбить помечающий $v$ список $p_i, \ldots, p_j$ на два списка $p_i, \ldots, p_{i+r}$ и $p_{i+r+1}, \ldots, p_j$ так, чтобы суммы списков были максимально близки, и создать двух сыновей $v$, пометив их данными списками
- Каждый лист построенного дерева помечен некоторым $p_i$ и соответствует коду символа $x_i$

Пример:
$p_1 = \frac{5}{13}$, $p_2 = p_3 = p_4 = p_5 = \frac{2}{13}$.

$\ell_1 = \ell_2 = \ell_5 = 2$, $\ell_3 = \ell_4 = 3$.

Ожидаемая длина кода символа:
$2 \cdot \frac{5}{13} + 2 \cdot 2 \cdot \frac{2}{13} + 2 \cdot 3 \cdot \frac{2}{13} = \frac{30}{13}$



# Коды Хаффмана

Рассмотрим другой алгоритм (Хаффмана) построения полного бинарного дерева по набору вероятностей $p_1 \geqslant p_2 \geqslant \cdots \geqslant p_k > 0$ кубика $\xi$:

- Начать с пенькового леса из $k$ пней, присвоив им веса $p_1, \ldots, p_k$ соответственно
- Пока текущий лес не связен
  - выбрать два дерева $u$ и $v$ с самыми легкими корнями
  - создать новую вершину $x$ с весом $\text{вес}(u) + \text{вес}(v)$ и сделать $u$ и $v$ сыновьями $x$
- Каждый лист построенного дерева помечен некоторым $p_i$ и соответствует коду символа $x_i$

# Коды Хаффмана

Рассмотрим другой алгоритм (Хаффмана) построения полного бинарного дерева по набору вероятностей $p_1 \geqslant p_2 \geqslant \cdots \geqslant p_k > 0$ кубика $\xi$:

- Начать с пенькового леса из $k$ пней, присвоив им веса $p_1, \ldots, p_k$ соответственно
- Пока текущий лес не связен
  - выбрать два дерева $u$ и $v$ с самыми легкими корнями
  - создать новую вершину $x$ с весом $\text{вес}(u) + \text{вес}(v)$ и сделать $u$ и $v$ сыновьями $x$
- Каждый лист построенного дерева помечен некоторым $p_i$ и соответствует коду символа $x_i$

Пример:
$p_1 = \frac{5}{13}$, $p_2 = p_3 = p_4 = p_5 = \frac{2}{13}$.

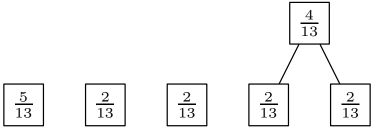
$$\boxed{\frac{5}{13}} \quad \boxed{\frac{2}{13}} \quad \boxed{\frac{2}{13}} \quad \boxed{\frac{2}{13}} \quad \boxed{\frac{2}{13}}$$

# Коды Хаффмана

Рассмотрим другой алгоритм (Хаффмана) построения полного бинарного дерева по набору вероятностей $p_1 \geqslant p_2 \geqslant \cdots \geqslant p_k > 0$ кубика $\xi$:

- Начать с пенькового леса из $k$ пней, присвоив им веса $p_1, \ldots, p_k$ соответственно
- Пока текущий лес не связен
  - выбрать два дерева $u$ и $v$ с самыми легкими корнями
  - создать новую вершину $x$ с весом $\text{вес}(u) + \text{вес}(v)$ и сделать $u$ и $v$ сыновьями $x$
- Каждый лист построенного дерева помечен некоторым $p_i$ и соответствует коду символа $x_i$

Пример:
$p_1 = \frac{5}{13}$, $p_2 = p_3 = p_4 = p_5 = \frac{2}{13}$.

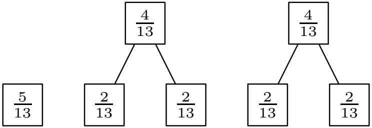


# Коды Хаффмана

Рассмотрим другой алгоритм (Хаффмана) построения полного бинарного дерева по набору вероятностей $p_1 \geqslant p_2 \geqslant \cdots \geqslant p_k > 0$ кубика $\xi$:

- Начать с пенькового леса из $k$ пней, присвоив им веса $p_1, \ldots, p_k$ соответственно
- Пока текущий лес не связен
  - выбрать два дерева $u$ и $v$ с самыми легкими корнями
  - создать новую вершину $x$ с весом $\text{вес}(u) + \text{вес}(v)$ и сделать $u$ и $v$ сыновьями $x$
- Каждый лист построенного дерева помечен некоторым $p_i$ и соответствует коду символа $x_i$

Пример:
$p_1 = \frac{5}{13}$, $p_2 = p_3 = p_4 = p_5 = \frac{2}{13}$.

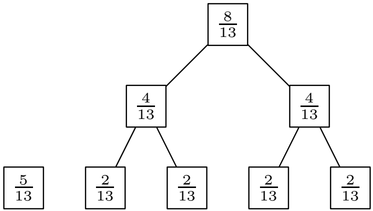


# Коды Хаффмана

Рассмотрим другой алгоритм (Хаффмана) построения полного бинарного дерева по набору вероятностей $p_1 \geqslant p_2 \geqslant \cdots \geqslant p_k > 0$ кубика $\xi$:

- Начать с пенькового леса из $k$ пней, присвоив им веса $p_1, \ldots, p_k$ соответственно
- Пока текущий лес не связен
  - выбрать два дерева $u$ и $v$ с самыми легкими корнями
  - создать новую вершину $x$ с весом $\text{вес}(u) + \text{вес}(v)$ и сделать $u$ и $v$ сыновьями $x$
- Каждый лист построенного дерева помечен некоторым $p_i$ и соответствует коду символа $x_i$

Пример:
$p_1 = \frac{5}{13}$, $p_2 = p_3 = p_4 = p_5 = \frac{2}{13}$.



# Коды Хаффмана

Рассмотрим другой алгоритм (Хаффмана) построения полного бинарного дерева по набору вероятностей $p_1 \geqslant p_2 \geqslant \cdots \geqslant p_k > 0$ кубика $\xi$:

- Начать с пенькового леса из $k$ пней, присвоив им веса $p_1, \ldots, p_k$ соответственно
- Пока текущий лес не связен
  - выбрать два дерева $u$ и $v$ с самыми легкими корнями
  - создать новую вершину $x$ с весом $\mathrm{вес}(u) + \mathrm{вес}(v)$ и сделать $u$ и $v$ сыновьями $x$
- Каждый лист построенного дерева помечен некоторым $p_i$ и соответствует коду символа $x_i$

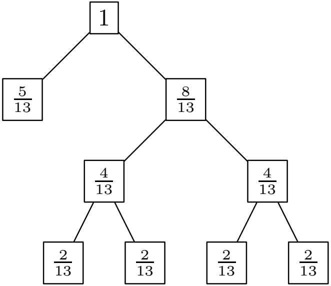
Пример:

$p_1 = \frac{5}{13}$, $p_2 = p_3 = p_4 = p_5 = \frac{2}{13}$.

$\ell_1 = 1$, $\ell_2 = \ell_3 = \ell_4 = \ell_5 = 3$.

Ожидаемая длина кода символа:

$\frac{5}{13} + 4 \cdot 3 \cdot \frac{2}{13} = \frac{29}{13}$



## Коды Хаффмана

Рассмотрим другой алгоритм (Хаффмана) построения полного бинарного дерева по набору вероятностей $p_1 \geqslant p_2 \geqslant \cdots \geqslant p_k > 0$ кубика $\xi$:

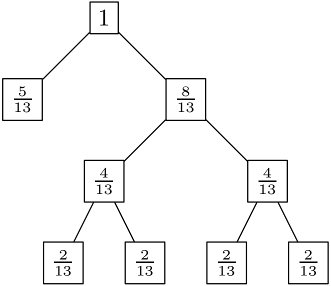
- Начать с пенькового леса из $k$ пней, присвоив им веса $p_1, \ldots, p_k$ соответственно
- Пока текущий лес не связен
  - выбрать два дерева $u$ и $v$ с самыми легкими корнями
  - создать новую вершину $x$ с весом $\text{вес}(u) + \text{вес}(v)$ и сделать $u$ и $v$ сыновьями $x$
- Каждый лист построенного дерева помечен некоторым $p_i$ и соответствует коду символа $x_i$

Пример:
$p_1 = \frac{5}{13}$, $p_2 = p_3 = p_4 = p_5 = \frac{2}{13}$.

$\ell_1 = 1$, $\ell_2 = \ell_3 = \ell_4 = \ell_5 = 3$.

Ожидаемая длина кода символа:
$\frac{5}{13} + 4 \cdot 3 \cdot \frac{2}{13} = \frac{29}{13}$

Меньше, чем у Шеннона-Фано $\Rightarrow$
для некоторых кубиков $\xi$ код Шеннона-Фано неоптимален



# Оптимальность кода Хаффмана

Удивительно, но верна

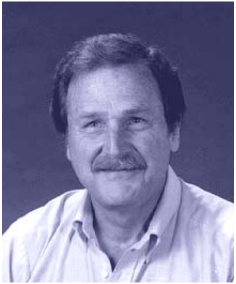
**Теорема**

Для любого кубика $\xi$ код Хаффмана, построенный по $\xi$, минимизирует ожидаемую длину кода символа в текстах, порожденных $\xi$.

Удивительно, но верна

**Теорема**

Для любого кубика $\xi$ код Хаффмана, построенный по $\xi$, минимизирует ожидаемую длину кода символа в текстах, порожденных $\xi$.



Дэвид А. Хаффман

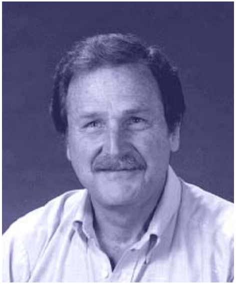
Придумал свои коды в 1951, будучи аспирантом в MIT

Впоследствии стал довольно известным математиком

Удивительно, но верна

**Теорема**

Для любого кубика $\xi$ код Хаффмана, построенный по $\xi$, минимизирует ожидаемую длину кода символа в текстах, порожденных $\xi$.



Дэвид А. Хаффман

Придумал свои коды в 1951, будучи аспирантом в MIT

Впоследствии стал довольно известным математиком

Нужно доказать, что длины $\ell_1, \ldots, \ell_k$ строк кода Хаффмана минимизируют сумму $\sum_{i=1}^{k} p_i \ell_i$ при условии $\sum_{i=1}^{k} 2^{-\ell_i} \leqslant 1 \Longrightarrow$

# Доказательство теоремы

Индукция по $k$ База индукции: $k = 2$ (при $k = 1$ задача не имеет смысла)

- есть единственный полный префиксный код $\{0, 1\}$, на котором и достигается оптимум; очевидно, алгоритм Хаффмана его построит

# Доказательство теоремы

Индукция по $k$     База индукции: $k = 2$ (при $k = 1$ задача не имеет смысла)

- есть единственный полный префиксный код $\{0, 1\}$, на котором и достигается оптимум; очевидно, алгоритм Хаффмана его построит

Шаг индукции

- Пусть для всех кубиков с $< k$ исходов алгоритм Хаффмана строит оптимальный код
- Рассмотрим кубик $\xi = (x_{1|p_1}, \ldots, x_{k|p_k})$, $p_1 \geqslant p_2 \geqslant \cdots \geqslant p_k > 0$
- Пусть по $\xi$ построен код Хаффмана с длинами строк $l_1, \ldots, l_k$
  - $l_i$ соответствует исходу $x_i$, т.е. $l_i$ есть уровень листа с весом $p_i$ в итоговом дереве
- В ходе алгоритма уровень листа увеличивается на единицу каждый раз, когда дерево, содержащее этот лист, на очередном шаге сливается с другим деревом

$\Rightarrow$ Если $u_i, u_j$ — листья с весами $p_i > p_j$ соответственно, и текущий уровень $u_j$ равен текущему уровню $v_i$, то корень дерева $X$, содержащего $u_j$, легче корня дерева $Y$, содержащего $u_i$; значит, в дальнейшем $X$ сольется не позже $Y$, откуда $l_i \leqslant l_j \Rightarrow l_1 \leqslant l_2 \leqslant \cdots \leqslant l_k$

- Далее, $l_{k-1} = l_k$, так как соответствующие листья сливаются на первом шаге и далее находятся в одном дереве
- После этого первого шага алгоритм Хаффмана работает, как для кубика $\xi'$ с $k-1$ исходами, имеющими вероятности $p'_1 = p_1, \ldots, p'_{k-2} = p_{k-2}, p'_{k-1} = p_{k-1} + p_k$
  - $p_{k-1} + p_k$ — вес корня дерева, полученного на первом шаге
- Длины строк, которые при этом получатся $(l_1, \ldots, l_{k-2}, l_k - 1)$, минимизируют сумму $\sum_{i=1}^{k-1} p'_i \ell_i$ по предположению индукции $\Longrightarrow$

- Теперь рассмотрим оптимальный код для $\xi$; в нем строки длины $L_1, \dots, L_k$
  - $L_i$ соответствует $x_i, p_i$
- $L_1 \leqslant \dots \leqslant L_k$
  - если $L_i > L_{i+1}$, то можно было бы уменьшить значение целевой функции, поменяв коды $x_i$ и $x_{i+1}$ местами

# Доказательство теоремы (окончание)

- Теперь рассмотрим оптимальный код для $\xi$; в нем строки длины $L_1, \ldots, L_k$
  - $L_i$ соответствует $x_i, p_i$
- $L_1 \leqslant \ldots \leqslant L_k$
  - если $L_i > L_{i+1}$, то можно было бы уменьшить значение целевой функции, поменяв коды $x_i$ и $x_{i+1}$ местами
- Если $L_{k-1} < L_k$, то $2^{L_k} \cdot \sum_{i=1}^{k} 2^{-L_i}$ — нечетное целое число

$\Rightarrow$ $\sum_{i=1}^{k} 2^{-L_i} \leqslant 1 - 1/2^{L_k}$ из неравенства Крафта

$\Rightarrow$ Набор длин $L_1, \ldots, L_{k-1}, L_k - 1$ тоже удовлетворяет неравенству Крафта (т.е. позволяет построить префиксный код), но дает меньшее значение целевой функции, что противоречит оптимальности взятого набора $\Rightarrow$ $L_{k-1} = L_k$

# Доказательство теоремы (окончание)

- Теперь рассмотрим оптимальный код для $\xi$; в нем строки длины $L_1, \ldots, L_k$
  - $L_i$ соответствует $x_i, p_i$
- $L_1 \leqslant \ldots \leqslant L_k$
  - если $L_i > L_{i+1}$, то можно было бы уменьшить значение целевой функции, поменяв коды $x_i$ и $x_{i+1}$ местами
- Если $L_{k-1} < L_k$, то $2^{L_k} \cdot \sum_{i=1}^{k} 2^{-L_i}$ — нечетное целое число

$\Rightarrow$ $\sum_{i=1}^{k} 2^{-L_i} \leqslant 1 - 1/2^{L_k}$ из неравенства Крафта

$\Rightarrow$ Набор длин $L_1, \ldots, L_{k-1}, L_k{-}1$ тоже удовлетворяет неравенству Крафта (т.е. позволяет построить префиксный код), но дает меньшее значение целевой функции, что противоречит оптимальности взятого набора $\Rightarrow$ $L_{k-1} = L_k$

★ Имеем $\sum_{i=1}^{k} p_i L_i = \sum_{i=1}^{k-2} p_i L_i + (p_{k-1}{+}p_k)(L_k - 1) + (p_{k-1}{+}p_k)$ $\Rightarrow$ набор $(L_1, \ldots, L_{k-2}, L_k{-}1)$ минимизирует целевую функцию для $\xi'$ (см. предыдущий слайд), при этом достигнутое минимальное значение отличается от минимального значения для $\xi$ на $p_{k-1}{+}p_k$

- Кроме того, знаем, что эта же целевая функция минимизируется набором $(l_1, \ldots, l_{k-2}, l_k{-}1)$, и полученное значение отличается от значения целевой функции для $\xi$ на наборе $(l_1, \ldots, l_k)$ на те же $p_{k-1}{+}p_k$

$\Rightarrow$ $(l_1, \ldots, l_k)$ минимизирует целевую сумму для $\xi$; шаг индукции доказан

Уф. Мы доказали, что для любого кубика $\xi$ можно эффективно построить один из оптимальных по “экономичности” кодирования символов префиксный код. Как этим пользоваться — начнем обсуждать на следующей лекции.