

# Алгоритм Шеннона — Фано

Сирота Александр  
[github.com/nukefluke](https://github.com/nukefluke)

ГУАП  
5 факультет  
Группа 5511

Санкт-Петербург, 2016

# План

- 1 Основные сведения
  - Коды переменной длины
  - Префиксный код
  - Пример префиксного кодирования
- 2 Основные этапы
- 3 Построение кодового дерева
  - Алгоритм
  - Пример кодового дерева
- 4 Реализация
  - Особенности
  - Интерфейс
  - Эффективность
- 5 Оценка сложности

# Основные сведения

**Алгоритм Шеннона — Фано** — один из первых алгоритмов сжатия, который сформулировали американские учёные Клод Шеннон и Роберт Фано.

- Относится к вероятностным методам сжатия
- Алгоритм использует коды переменной длины
- Коды Шеннона — Фано префиксные

# Коды переменной длины

При использовании **кодов переменной длины** символы кодируются набором бит различной длины. Часто встречающийся символ кодируется кодом меньшей длины, редко встречающийся — кодом большей длины.

# Префиксный код

**Префиксный код** (англ. prefix code) — код, в котором никакое кодовое слово не является префиксом какого-то другого кодового слова.

## Пример префиксного кодирования

$$U = \{a, b, c\}$$

$$Z = \{0, 1\}$$

$$c(a) = 00 \quad c(b) = 01 \quad c(c) = 1$$

Закодируем строку *abacaba* :

$$c^*(abacaba) = 0001001000100$$

Такой код можно однозначно разбить на слова:

00 01 00 1 00 01 00

# Алгоритм Фано: Основные этапы

Алгоритм Фано:

- 1 Выписать символы по убыванию вероятностей.
- 2 Разделить список на две части с равными долями вероятности.
- 3 Для первой части добавить к коду «0», для второй — «1».
- 4 Повторить шаги (1–3) для каждой части.

# Алгоритм Фано: Этап 1

Символы первичного алфавита выписывают по убыванию вероятностей.

Символ	Вероятность
a	0.36
b	0.18
c	0.18
d	0.12
e	0.09
f	0.07



## Алгоритм Фано: Этап 2

Символы полученного алфавита делят на две части, суммарные вероятности символов которых максимально близки друг другу.

Символ	Вероятность
a	0.36
b	0.18
c	0.18
d	0.12
e	0.09
f	0.07

## Алгоритм Фано: Этап 3

В префиксном коде для первой части алфавита присваивается двоичная цифра «0», второй части — «1».

Символ	Вероятность	1
a	0.36	0
b	0.18	
c	0.18	1
d	0.12	
e	0.09	
f	0.07	

## Алгоритм Фано: Этап 4

Полученные части рекурсивно делятся и их частям назначаются соответствующие двоичные цифры в префиксном коде.

Символ	Вероятность	1	2	3	4	Итог
a	0.36	0	00			00
b	0.18		01			01
c	0.18	1	10			10
d	0.12		11	110		110
e	0.09			111	1110	1110
f	0.07				1111	1111

# Построение кодового дерева

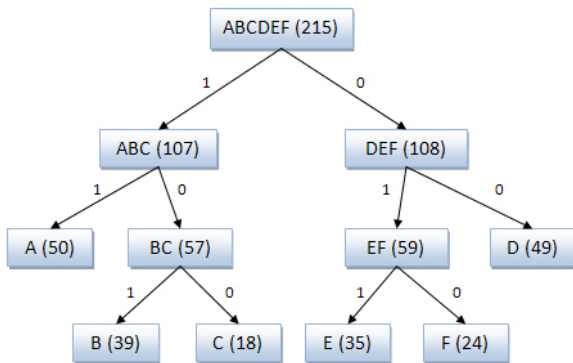
Алгоритм:

- Код Шеннона — Фано строится с помощью бинарного дерева
- Всё множество кодируемых элементов соответствует корню дерева
- Множество разбивается на два подмножества с примерно одинаковыми суммарными вероятностями
- Если подмножество содержит единственный элемент, то такое подмножество последующему разбиению не подлежит
- Ветви кодового дерева размечаются символами 1 и 0

# Пример кодового дерева

Исходные символы:

Символ	Частота Встречаемости
A	50
B	39
C	18
D	49
E	35
F	24



Полученные коды:

A	B	C	D	E	F
11	101	100	00	011	010

# Реализация

## Репозиторий на github

<https://github.com/nukefluke/text-compressor>

Особенности реализации:

- Простота
- Низкая сложность
- Иногда коды строятся неоптимально
- Необходимо дописывать шапку в файл

# Интерфейс

## Запуск программы

```
$ ./fano <mode> <input> [ -o <output> ]
```

### <mode>

'-e' — кодирование. Расширение входного файла **.txt**

'-d' — раскодирование. Расширение входного файла **.fano**

### <input>

Входной файл.

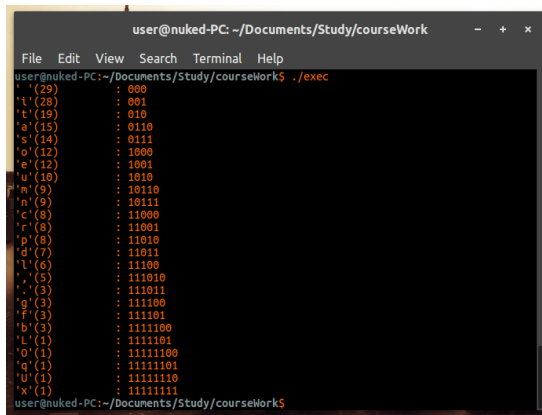
### <output>

Выходной файл. Если не указан, то будет использовано имя входного файла + расширение.

# Пример сгенерированного словаря

Для фразы:

*«Lorem ipsum dolor sit amet, consectetur adipiscing elit. Odit sint cupiditate magni, illo officia facere magnam, ad pariatur ipsum explicabo sit nostrum aliquid nisi necessitatibus natus temporibus. Ut, optio, odit.»*

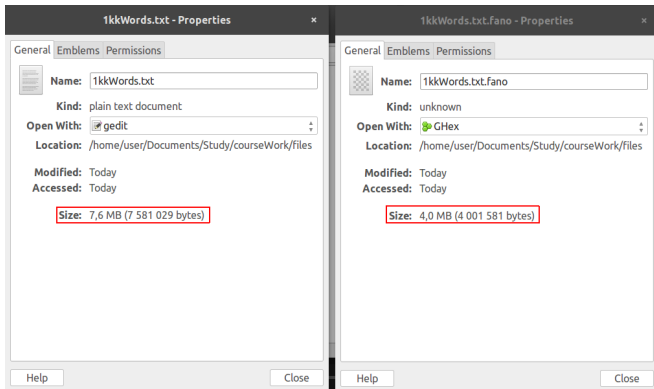


```
user@nuked-PC: ~/Documents/Study/courseWork
File Edit View Search Terminal Help
user@nuked-PC:~/Documents/Study/courseWork$ ./exec
' '(29) : 000
' '(28) : 001
' '(19) : 010
' a'(15) : 0110
' s'(14) : 0111
' o'(12) : 1000
' e'(12) : 1001
' u'(10) : 1010
' m'(9) : 10110
' n'(9) : 10111
' c'(8) : 11000
' r'(8) : 11001
' p'(8) : 11010
' d'(7) : 11011
' l'(6) : 11100
' ,'(5) : 111010
' .'(3) : 111011
' g'(3) : 111100
' f'(3) : 111101
' b'(3) : 1111100
' l'(1) : 1111101
' o'(1) : 11111100
' q'(1) : 11111101
' U'(1) : 11111110
' x'(1) : 11111111
user@nuked-PC:~/Documents/Study/courseWork$
```



# Эффективность

При сжатии текстового файла, содержащего один миллион слов (7,6 Мб), получился файл размером 4Мб. Эффективность сжатия - 47%.



# Оценка сложности

Кодирование файла:

- Подсчёт символов:  $O(n)$
- Построение дерева:  $O(c)$
- Создание шапки для закодированного файла:  $O(c)$
- Запись закодированных данных в файл:  $O(n)$

Раскодирование файла:

- Чтение шапки:  $O(c)$
- Создание таблицы кодировки:  $O(c)$
- Раскодирование:  $O(n)$
- Запись раскодированных данных в файл:  $O(n)$

Итоговая сложность — линейная  $O(n)$

Спасибо за внимание!