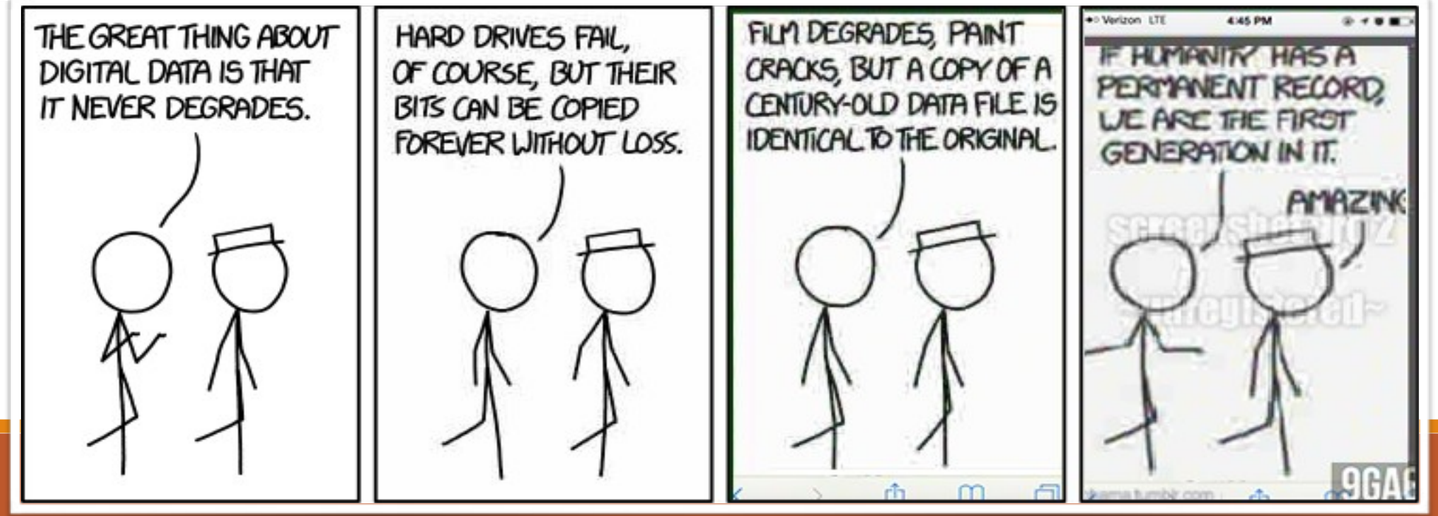
Структурное программирование

ЛЕКЦИЯ №6

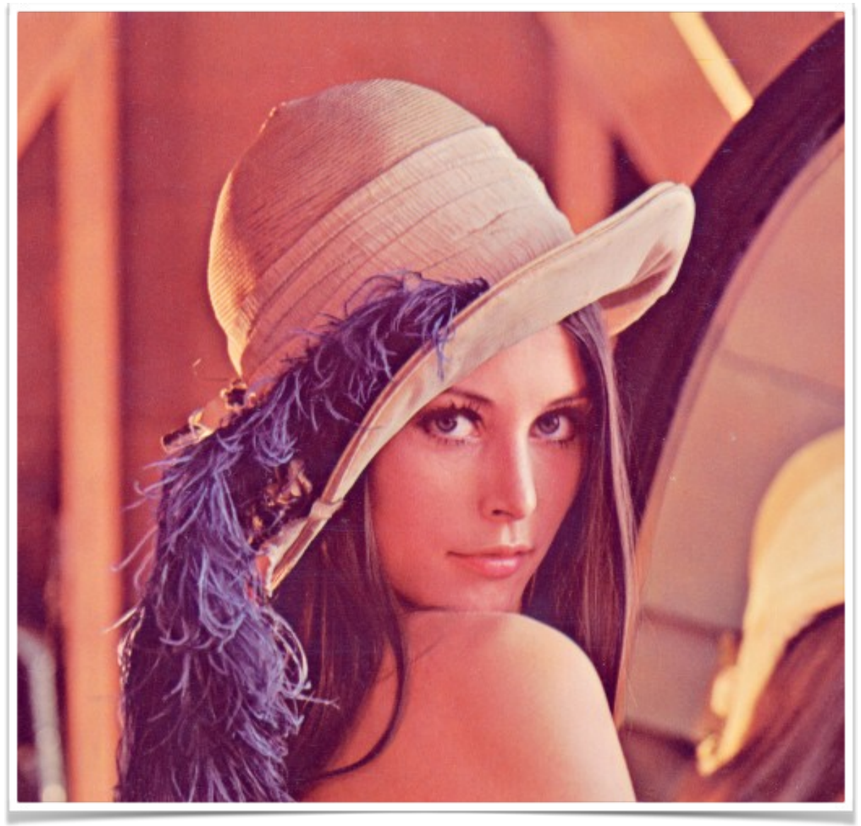
10 ОКТЯБРЯ 2023



Сжатие (Упаковка) информации

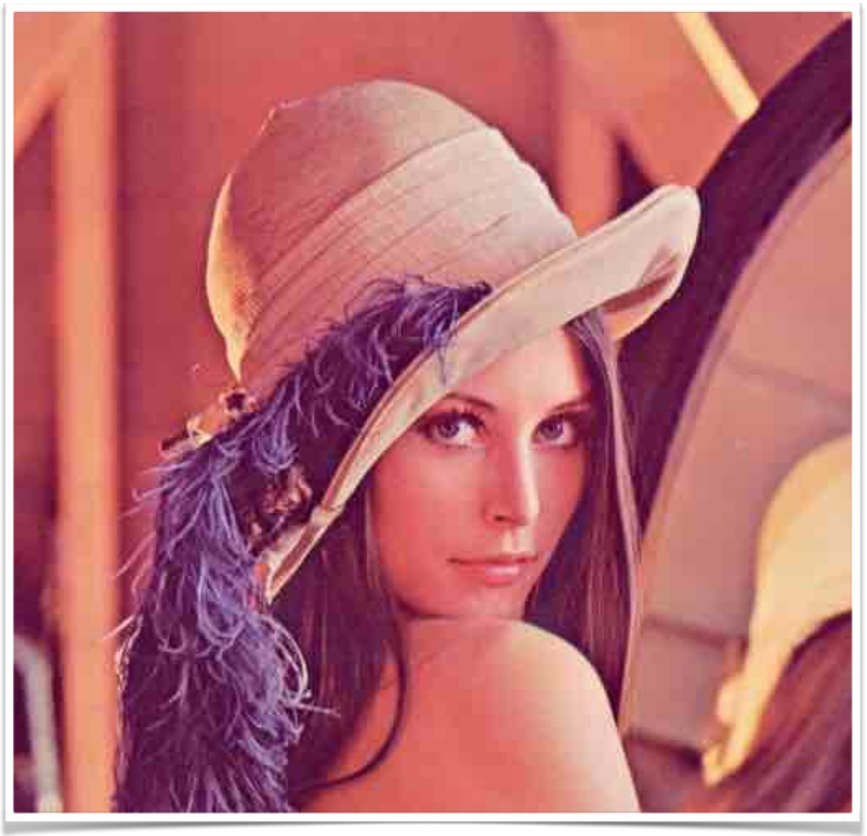
- Принцип: сжимаем битовую цепочку за счет специфического кодирования.
- Обычные данные: T_1 = время доступа к данным.
- Сжатые данные:
  T_2 = время доступа к сжатым данным + время на распаковку.
- Часто $T_2 < T_1$!
- Чем меньше размер, тем больше скорость доступа.
- Иногда места действительно мало.

Сжатие без потерь



Возможно точное (бит-в-бит) восстановление исходной битовой цепочки.

Сжатие с потерями



«А, все равно не видно (не слышно)…»
JPEG, MPEG и т.д.

Программа на сегодня

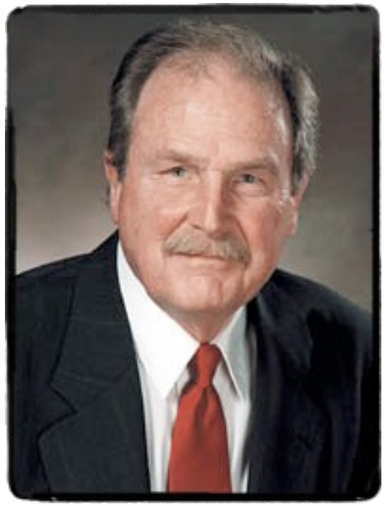
- Алгоритм RLE.
- Алгоритм Хаффмана.
- Алгоритм LZ77.
- Алгоритм LZW.

Алгоритм RLE

- Run-Length Encoding (сжатие повторяющихся цепочек). PCX, ILBM.
- Ч/б изображение: …БББББББББЧБББББЧЧБББББЧЧЧЧЧЧЧ…
- Упаковка: 9Б 1Ч 5Б 2Ч 5Б 9Ч.
- Общий случай: выбирается редко используемый байт-префикс (P).
  - XXXX → P 4 X.
  - X → X.
  - P P P P → P 4 P.
  - P → P 1 P.

Алгоритм хаффмана

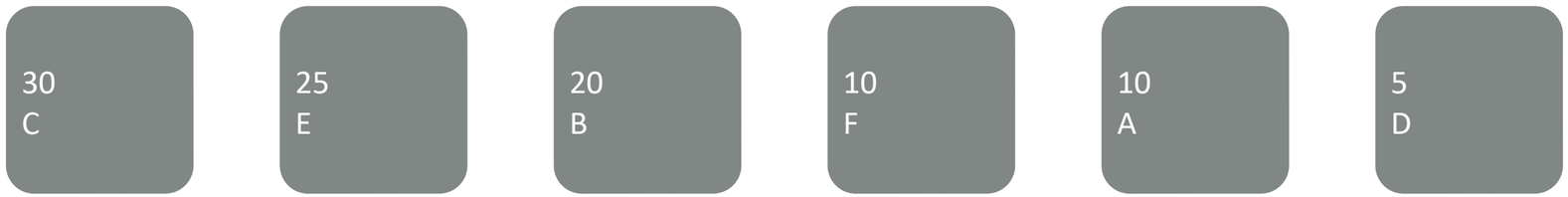
- В обычном файле все символы кодируются 8-битовыми цепочками, вне зависимости от частоты их появления.
- Идея: кодировать более часто встречающиеся символы более короткими цепочками.
- Префиксные коды: ни одно кодовое слово не является префиксом любого другого.
- Первый шаг: подсчет частоты вхождения символов в исходном файле.



Пример кодирования по хаффману

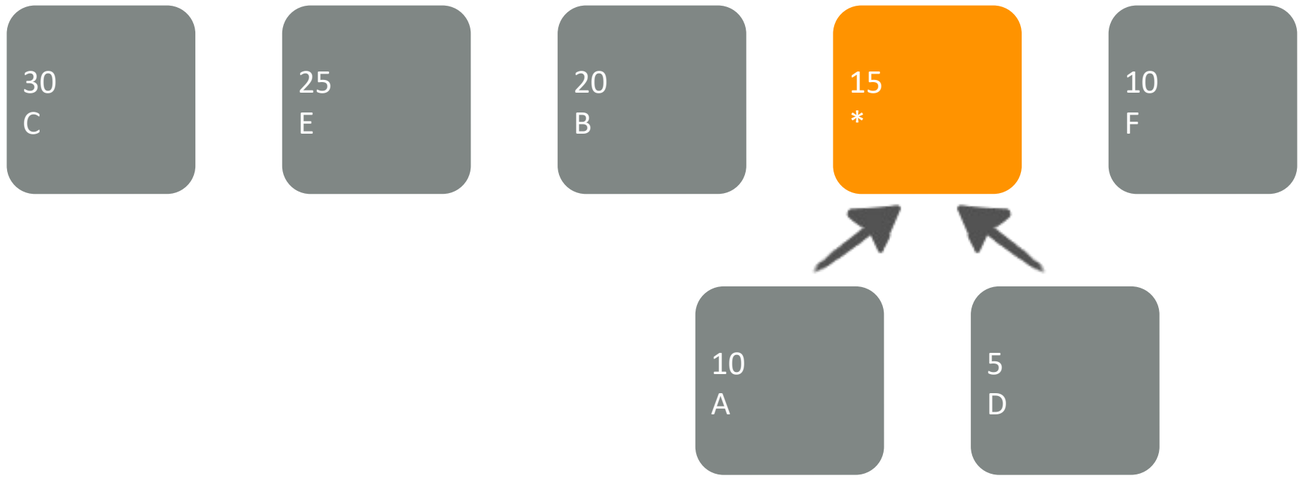
Символ	A	B	C	D	E	F
Частота вхождений	10	20	30	5	25	10

Сортируем по числу вхождений:



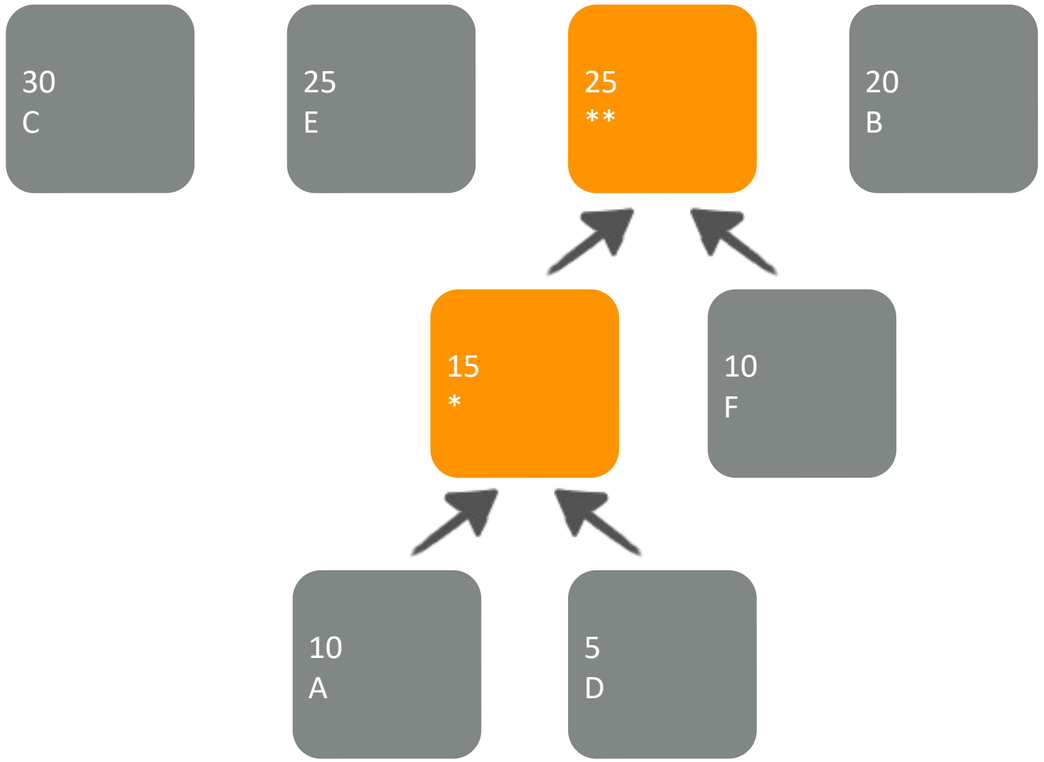
Объединение (Шаг 1)

Два символа с минимальной частотой (A и D) формируют «суммарный» узел (*) с частотой 15.



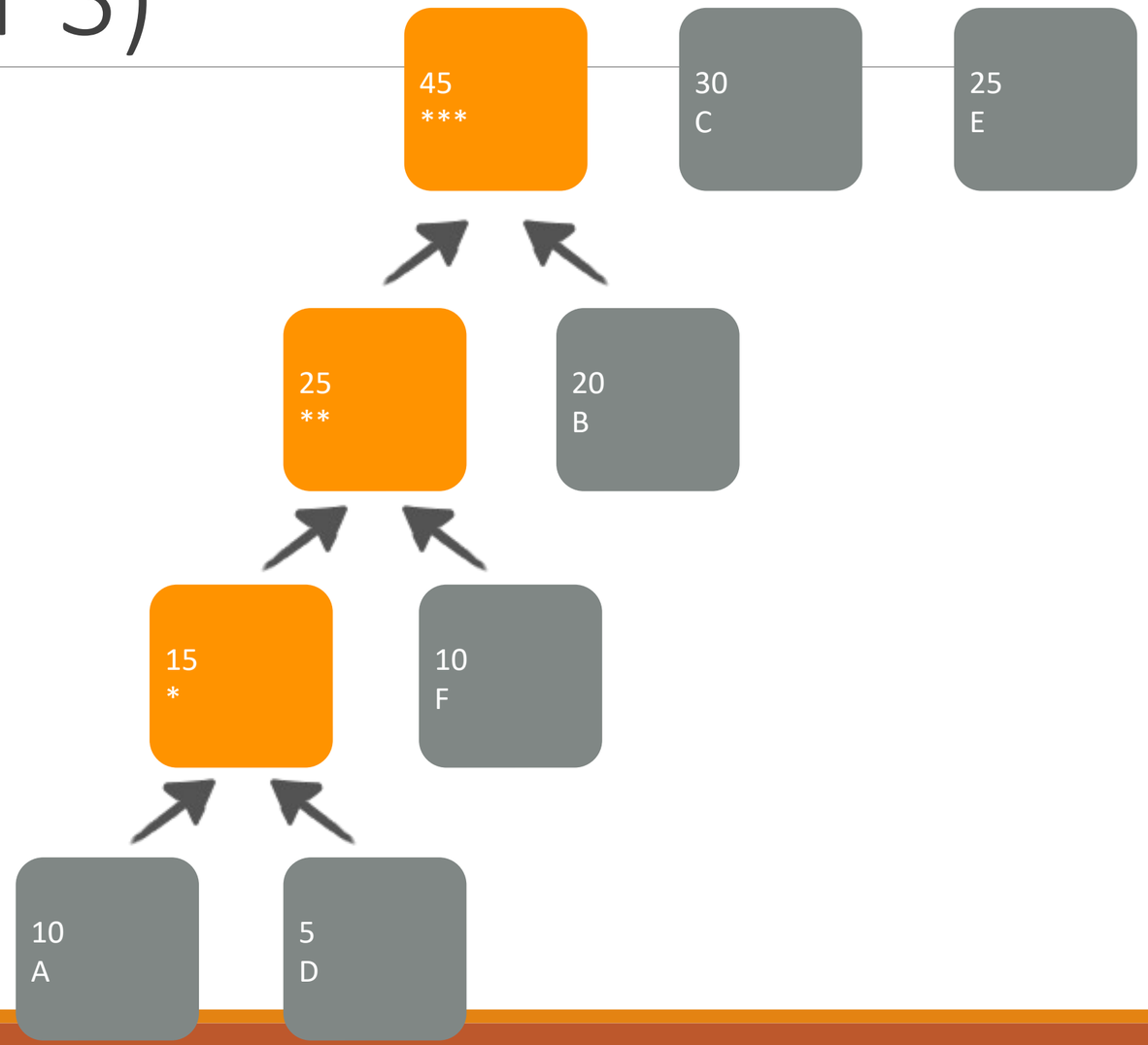
Объединение (шаг 2)

Узлы F и * формируют узел ** (частота 25).



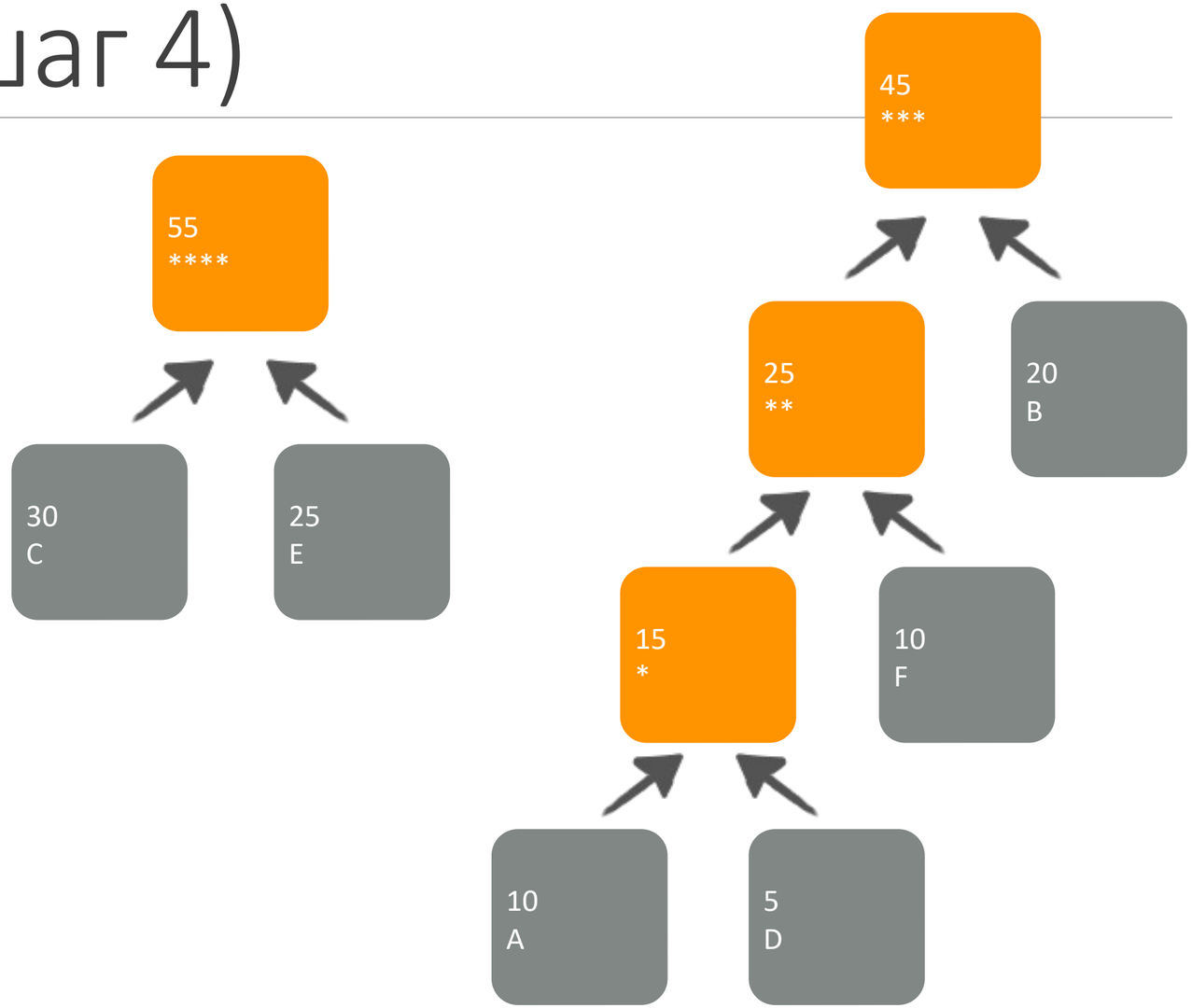
Объединение (шаг 3)

В и ** формируют *** (частота 45).

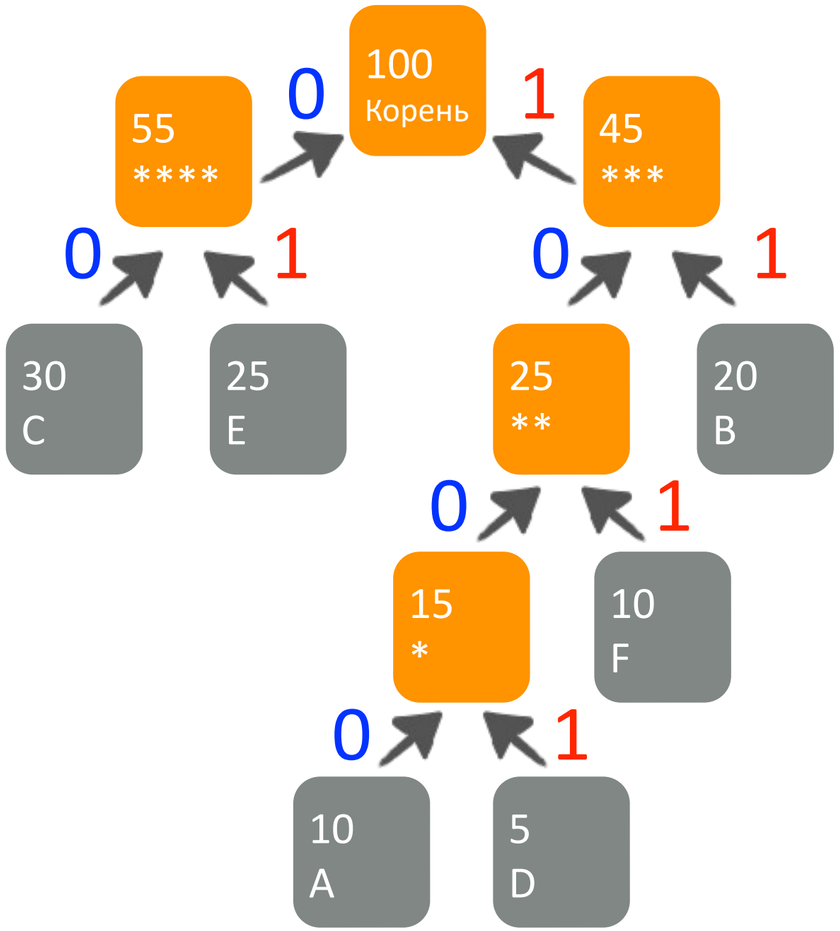


Объединение (шаг 4)

C и E формируют **** (частота 55).



Итоговое дерево



Символ	Частота	К. слово
C	30	00
E	25	01
B	20	11
F	10	101
A	10	1000
D	5	1001

Канонический код Хаффмана

Получившуюся таблицу кодов сортируем по парному ключу (длина цепочки, ASCII код символа)

Первый символ: цепочка «все нули».

Последующие символы: прибавляем 1 и дополняем нулями справа до нужной длины.

```
A = 1000
B = 11
C = 00
D = 1001
E = 01
F = 101
```

Распаковка

- Тривиальна при известном дереве (префиксные коды расшифровываются однозначно).
- Канонический код Хаффмана: дерево восстанавливается из таблицы соответствий «символ – длина кодового слова» при известной максимальной длине кода.
- Для упаковки массива длин размером 256 можно воспользоваться RLE.

На сколько сжалось?

- Было: 100 байт × 8 бит = 800 бит.
- Стало: (30+25+20) × 2 + 10 × 3 + (10+5) × 4 = 240 бит.
- Набор длин кодов: 6 букв × 2 бита + 16 (RLE) + 8 (макс. длина) = 36 бит.
- Итого: (240+36) / 800 = 34,5 %.

Программа на сегодня

- ✓ Алгоритм RLE.
- ✓ Алгоритм Хаффмана.
- Алгоритм LZ77.
- Алгоритм LZW.

Алгоритм LZ77

В окне ищется максимальная последовательность, соответствующая содержимому упреждающего буфера.

В выходной поток пишется тройка (смещение, длина, след. символ в буфере).

Окно сдвигается на длину найденной последовательности + 1.

пример LZ77

Исходные данные: «abracadabra».

Красным цветом отмечен буфер поиска.

Синим цветом отмечена совпадающаю последовательность.

Окно	Позиция	Длина	Символ
abracadabra	0	0	a
abracadabra	0	0	b
abracadabra	0	0	r
abracadabra	3	1	c
abracadabra	2	1	d
abracadabra	7	4	нет

Код: (0, 0, a), (0, 0, b), (0, 0, r), (3, 1, c), (2, 1, d), (7, 4, -)

пример LZ77

Исходные данные: «abababcab».
Красным цветом отмечен буфер поиска.
Синим цветом отмечена совпадающаю последовательность.

Окно	Позиция	Длина	Символ
abababcab	0	0	a
ababababcab	0	0	b
abababcab	2	4	c
abababcab	3	2	нет

Код: (0, 0, a), (0, 0, b), (2, 4, c), (3, 2, -)

Алгоритм LZW

Заводим таблицу строк (например, размером 4096). Первые 256 кодов соответствуют символам ASCII.

Алгоритм.

- Ищем с текущей позиции строку S максимальной длины, существующую в таблице.
- Выводим в файл индекс S в массиве (код).
- Если файл кончился, то выход, иначе добавляем в таблицу строку S+C (C – следующий символ) и устанавливаем текущую позицию на символ C.

пример LZW

X A B C X A B X A B B X A B B X A B D

s	c	Выходной поток	Новые коды
X	A	X	256 = XA
A	B	A	257 = AB
B	C	B	258 = BC
C	X	C	259 = CX
XA	B	256	260 = XAB
B	X	B	261 = BX
XAB	B	260	262 = XABB
BX	A	261	263 = BXA
AB	B	257	264 = ABB
BXA	B	263	265 = BXAB
B	D	B	266 = BD
D	Конец файла	D	нет

LZW распаковка

Прочитать $Код_0$ из файла и вывести $Таблица[Код_0]$.

Пока входной поток не пуст:

- читаем $Код_1$ из файла;
- выводим $Таблица[Код_1]$;
- добавляем $Таблица[Код_0]$ + ПервыйСимвол($Таблица[Код_1]$);
- заменяем $Код_0 \leftarrow Код_1$.

Есть проблема с распаковкой

Пусть строка XYZ есть в таблице (код 333). Пусть далее: ……XYZXYZXYZ.

S=XYZ, C=X, выводим 333, добавляем код 400=XYZX.

На следующем шаге выводим код 400 (XYZX).

Распаковщик увидит код 400 раньше, чем он определен.

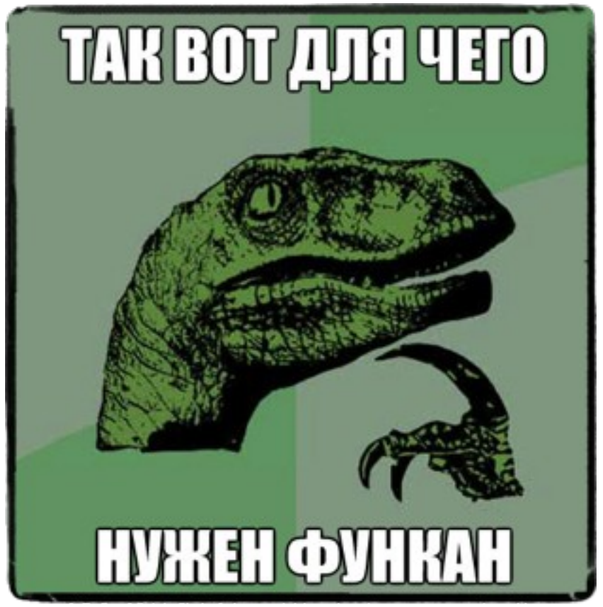
Решение: если считан неизвестный код, он будет соответствовать цепочке $Код_0$ +ПервыйСимвол($Код_0$).

Сжатие с потерями

$$\hat{f}(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) e^{-ix\omega}\, dx$$

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn}$$

$$y[n] = (x * g)[n] = \sum_{k=-\infty}^{\infty} x[k] g[n-k]$$



На сегодня все