

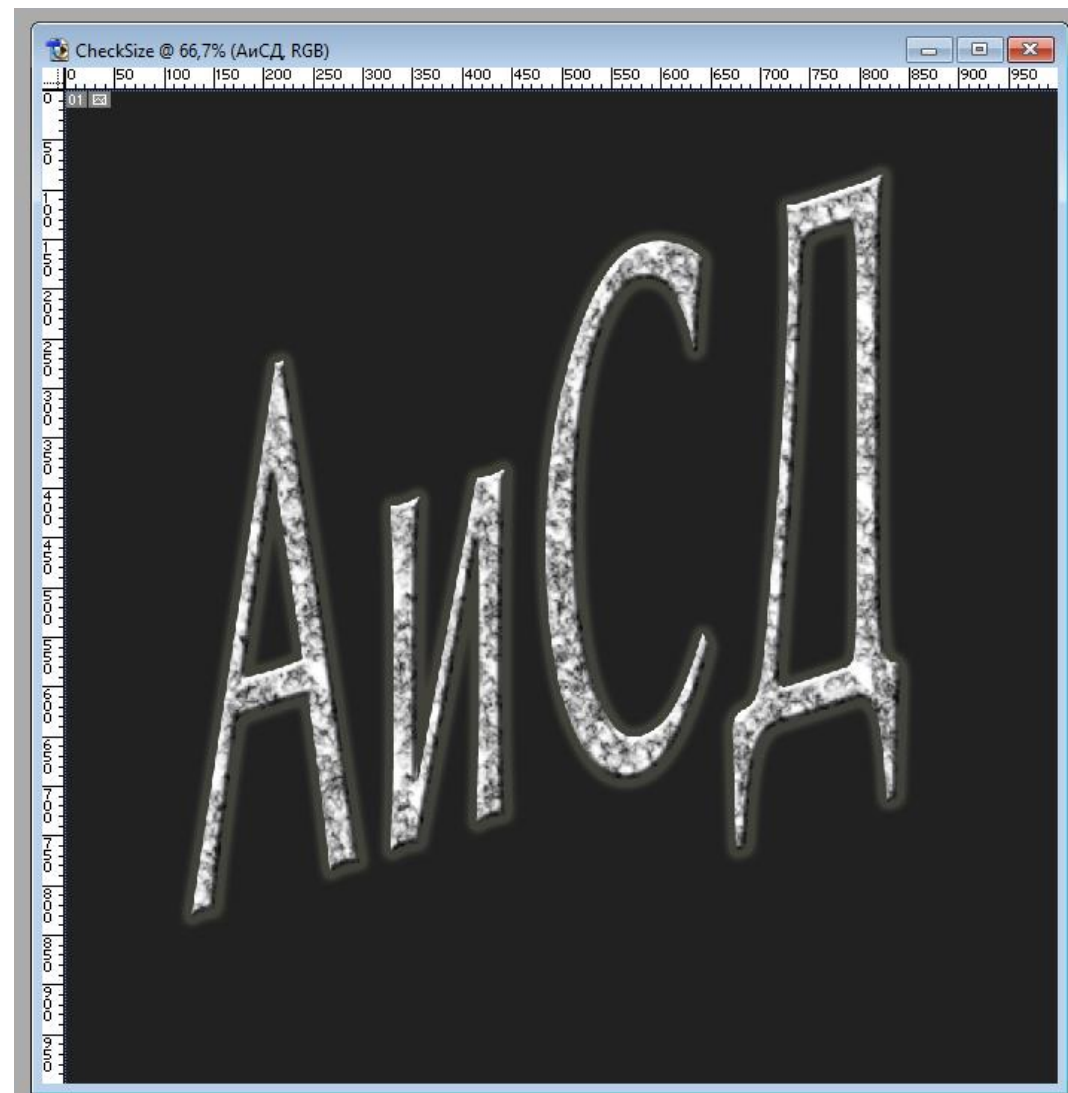
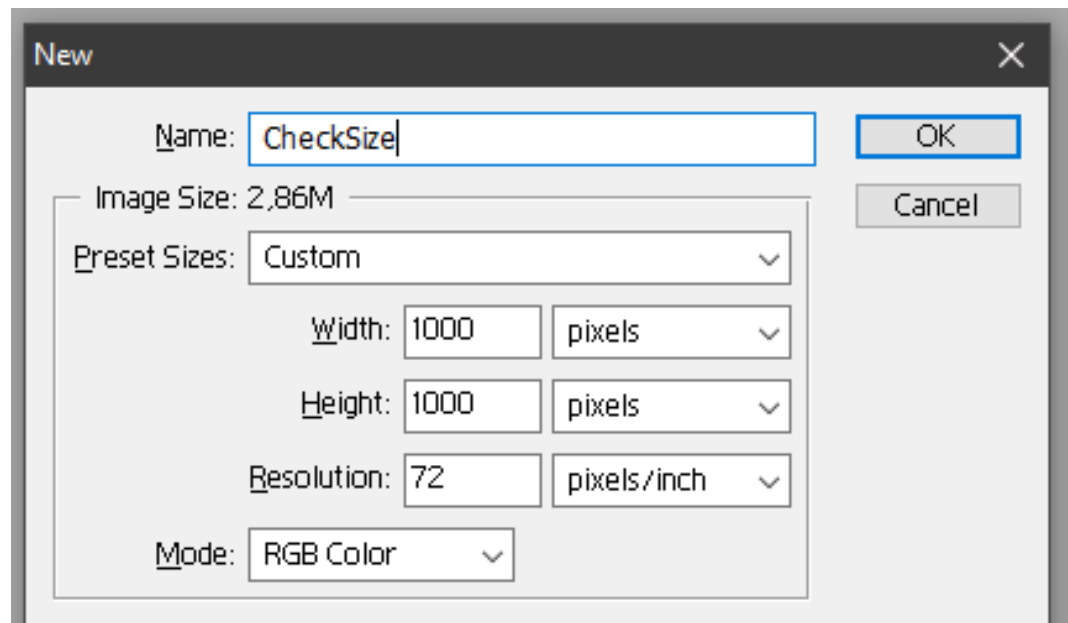
Сжимающее кодирование

Зачем нужно сжатие данных

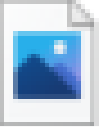
Пример из обычной жизни:

Зачем нужно сжатие данных

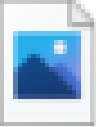
Пример из обычной жизни:





Зачем нужно сжатие данных

	CheckSize.bmp
Тип файла:	Файл "BMP" (.bmp)
Приложение:	Фотографии
Расположение:	C:\Users\Максим\Desktop
Размер:	2,86 МБ (3 000 056 байт)
На диске:	2,86 МБ (3 002 368 байт)

Зачем нужно сжатие данных

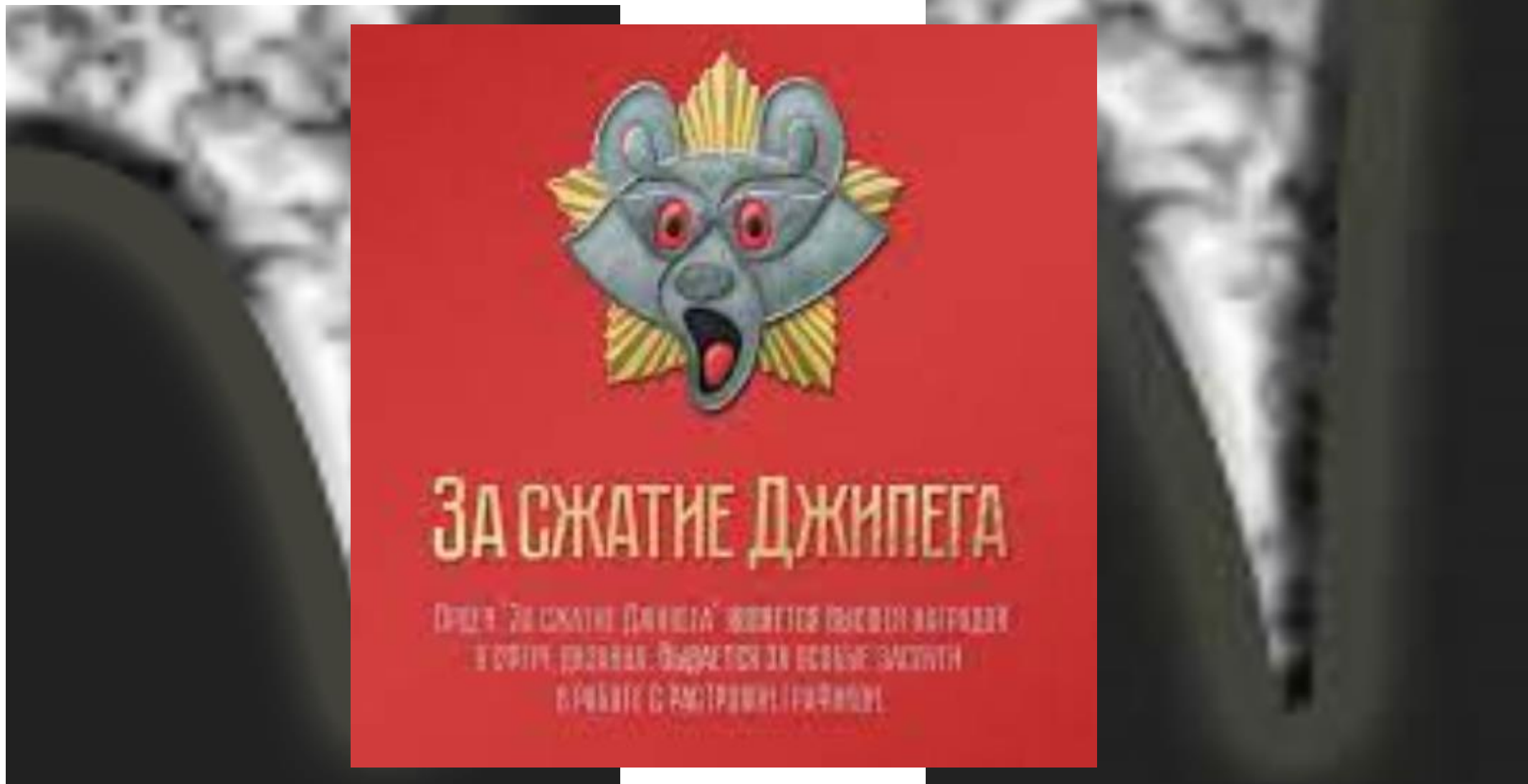
	CheckSize.bmp
Тип файла:	Файл "BMP" (.bmp)
Приложение:	Фотографии
Расположение:	C:\Users\Максим\Desktop
Размер:	2,86 МБ (3 000 056 байт)
На диске:	2,86 МБ (3 002 368 байт)

	CheckSize.jpg
Тип файла:	IrfanView JPG File (.jpg)
Приложение:	 IrfanView 64-bit
Расположение:	C:\Users\Максим\Desktop
Размер:	50,1 КБ (51 332 байт)
На диске:	52,0 КБ (53 248 байт)

Но здесь есть одно но...

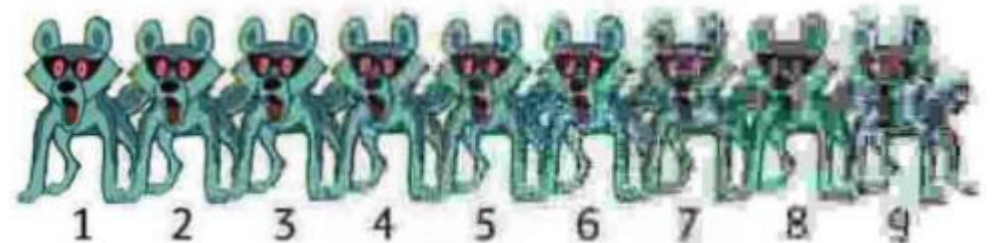


Но здесь есть одно но...



Особенности сжатия

На примере сжатия картинок мы увидели так называемое **сжимание с потерей** (качества). Теряем мы изначальную информацию, отбрасывая её. За счёт такого действия мы выигрываем в требуемом объёме данных для хранения, но при этом проигрываем в их полноте.



Особенности сжатия

На примере сжатия картинок мы увидели так называемое **сжимание с потерей** (качества). Теряем мы изначальную информацию, отбрасывая её. За счёт такого действия мы выигрываем в требуемом объёме данных для хранения, но при этом проигрываем в их полноте.

Можно ли такой фокус провернуть с музыкой?



Особенности сжатия

На примере сжатия картинок мы увидели так называемое **сжимание с потерей** (качества). Теряем мы изначальную информацию, отбрасывая её. За счёт такого действия мы выигрываем в требуемом объёме данных для хранения, но при этом проигрываем в их полноте.

Можно ли такой фокус провернуть с музыкой?

А с текстом?



Сжатие текста

Со сжатием текста так не получится, т.к. ценность текстовой информации заключается именно в последовательности букв и символов, а сжимающее кодирование будет их нарушать, превращая текст в кашу.

Сжатие текста

Со сжатием текста так не получится, т.к. ценность текстовой информации заключается именно в последовательности букв и символов, а сжимающее кодирование будет их нарушать, превращая текст в кашу.

Поэтому для текста требуется **сжатие без потерь**.

Такое сжатие пытается уменьшить «избыточность» информации.

Основной способ:

Сжатие текста

Со сжатием текста так не получится, т.к. ценность текстовой информации заключается именно в последовательности букв и символов, а сжимающее кодирование будет их нарушать, превращая текст в кашу.

Поэтому для текста требуется **сжатие без потерь**.
Такое сжатие пытается уменьшить «избыточность» информации.

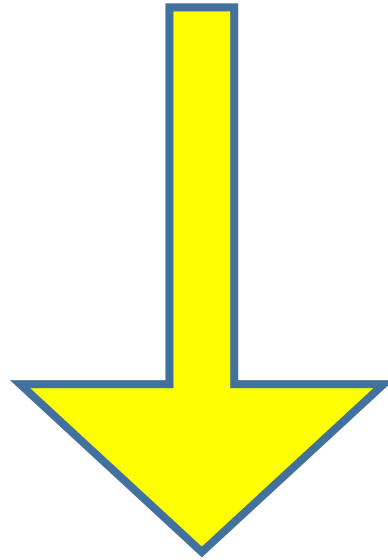
Основной способ:

замена длинных последовательностей на более короткие

Как ищем такие последовательности?

Анализируем
сочетания букв

Анализируем
текст побуквенно

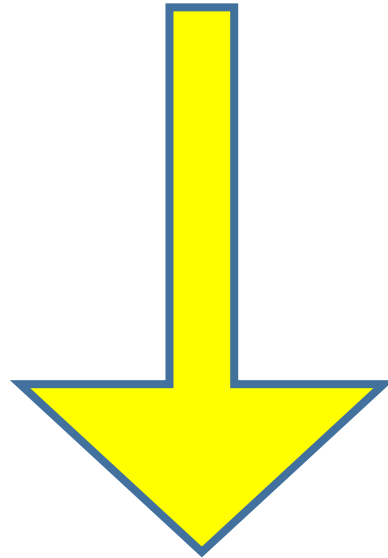


Статистика

Как ищем такие последовательности?

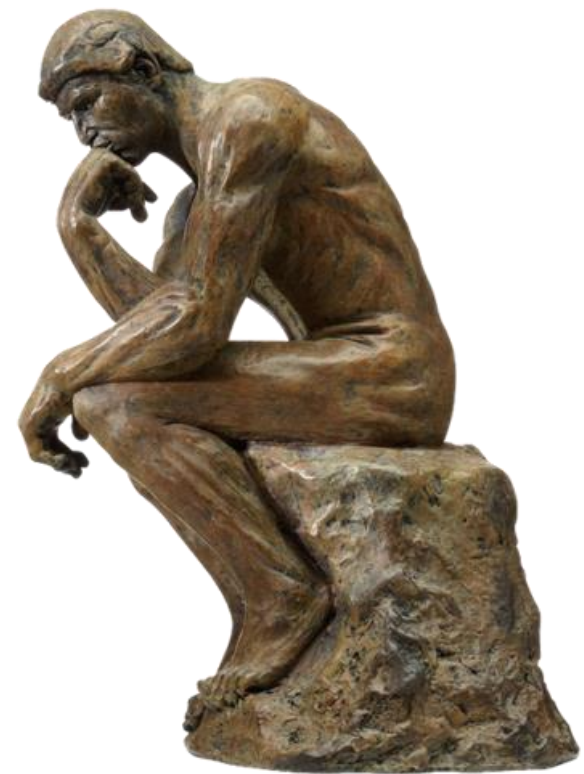
Анализируем
сочетания букв

Анализируем
текст побуквенно



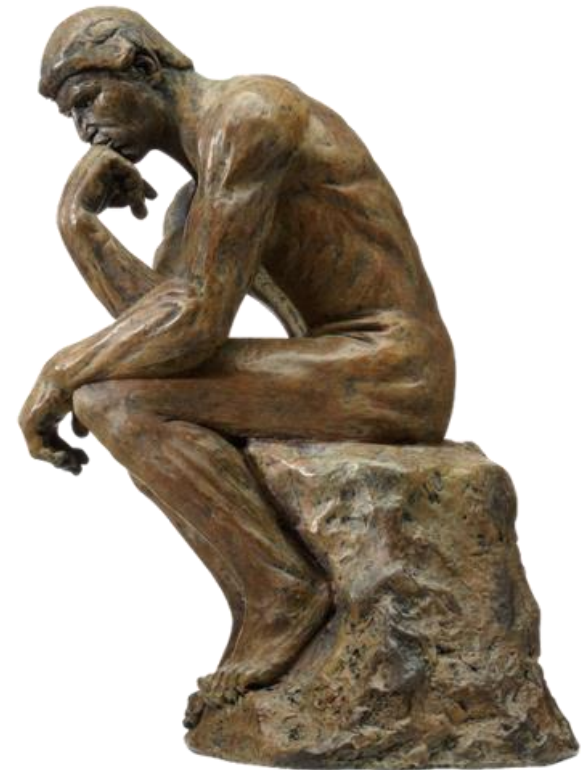
Статистика

Что такое текст?



Что такое текст?

С первого семестра помним, что буквы в тексте кодируются определённым набором бит. За такие наборы отвечают таблицы кодировки.



Что такое текст?

С первого семестра помним, что буквы в тексте кодируются определённым набором бит. За такие наборы отвечают таблицы кодировки.

Самый простой пример: ASCII-таблица

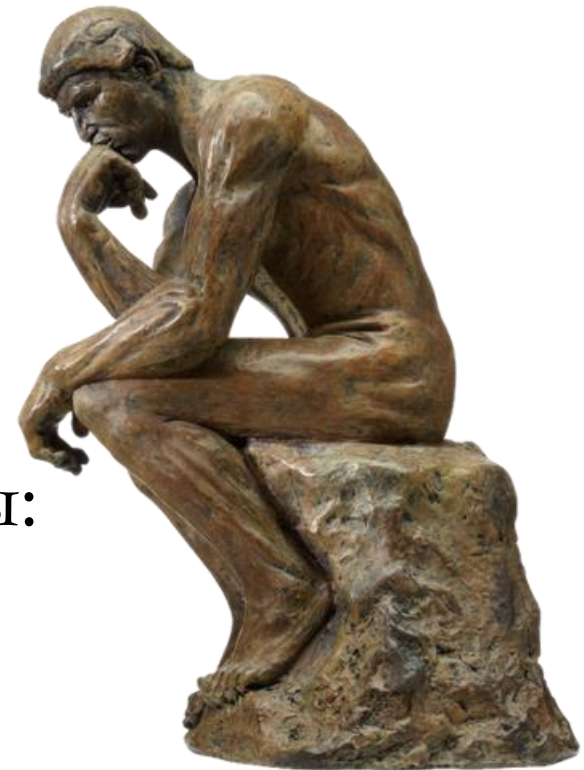
A – 0x41

B – 0x42

Z – 0x5A

Такой код везде один — код **фиксированной** длины:

На каждый символ — 1 байт.



Что такое текст?

С первого семестра помним, что буквы в тексте кодируются определённым набором бит. За такие наборы отвечают таблицы кодировки.

Самый простой пример: ASCII-таблица

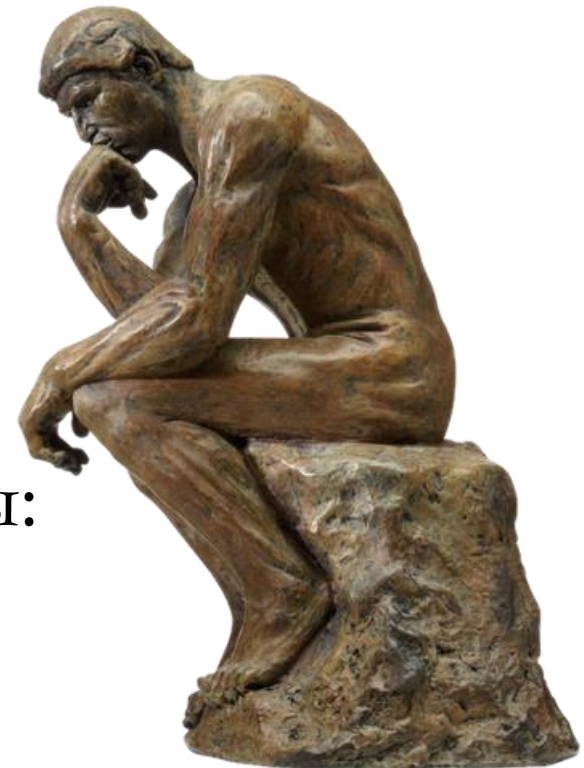
A – 0x41

B – 0x42

Z – 0x5A

Такой код везде один — код **фиксированной** длины:
На каждый символ — 1 байт.

А можно ли лучше? Код переменной длины?



Алгоритм Хаффмана. Основная идея

Можем задавать код символа в зависимости от частоты его встречи в тексте

Чем чаще встречаем символ, тем меньше бит будем тратить на кодирование этого символа.

Пример:

Текст: АААСССС. А будем нулем (0), а С – единицей (1).

ASCII-кодировка: $7 * 1 \text{ байт/символ} = 7 \text{ байтов}$.

Haffman-encode: $1 \text{ бит/символ} * 7 = 1 \text{ байт}$: **0001111**

Закодируем же быстрее!

Рассмотрим алфавит $\Sigma = \{A, B, C, D\}$

Код фиксированной длины

Символ	Кодирование
<i>A</i>	00
<i>B</i>	01
<i>C</i>	10
<i>D</i>	11

Код переменной длины

Символ	Кодирование
<i>A</i>	0
<i>B</i>	01
<i>C</i>	10
<i>D</i>	1

Сколько тратим на хранение сообщения «BCCDAAD» ???

Закодируем же быстрее!

Рассмотрим алфавит $\Sigma = \{A, B, C, D\}$

Код фиксированной длины

Символ	Кодирование
<i>A</i>	00
<i>B</i>	01
<i>C</i>	10
<i>D</i>	11

Код переменной длины

Символ	Кодирование
<i>A</i>	0
<i>B</i>	01
<i>C</i>	10
<i>D</i>	1

Сколько тратим на хранение сообщения «BCCDAAD» ???

На 25% меньше!

Первая проблема

Закодировать-то мы закодировали. Но как теперь раскодировать?

Сообщение с прошлого слайда:

011010101001

Символ	Кодирование
<i>A</i>	0
<i>B</i>	01
<i>C</i>	10
<i>D</i>	1

Первая проблема

Закодировать-то мы закодировали. Но как теперь декодировать?

Сообщение с прошлого слайда:

011010101001

Символ	Кодирование
<i>A</i>	0
<i>B</i>	01
<i>C</i>	10
<i>D</i>	1

Никак! Потому что возникла неоднозначность при декодировании.

Первая проблема

Закодировать-то мы закодировали. Но как теперь декодировать?

Сообщение с прошлого слайда:

011010101001

Символ	Кодирование
<i>A</i>	0
<i>B</i>	01
<i>C</i>	10
<i>D</i>	1

Никак! Потому что возникла неоднозначность при декодировании.

Всё, приплыли?

Решение – беспрефиксный код

Такой код, в котором для любых
символов $a, b \in \Sigma$ их коды не являются
префиксами друг друга.

Решение – беспрефиксный код

Такой код, в котором для любых символов $a, b \in \Sigma$ их коды не являются префиксами друг друга.

Внимательнее посмотрим на таблицу кодировки.
Это префиксный код?

Символ	Кодирование
<i>A</i>	0
<i>B</i>	01
<i>C</i>	10
<i>D</i>	1

Решение – беспрефиксный код

Такой код, в котором для любых символов $a, b \in \Sigma$ их коды не являются префиксами друг друга.

Символ	Кодирование
A	0
B	01
C	10
D	1



Символ	Кодирование
A	0
B	10
C	110
D	111

Как генерировать такие коды?

Требуется алгоритмически генерировать беспрефиксные коды минимальной длины. На помощь приходят бинарные деревья. Почему?

Как генерировать такие коды?

Требуется алгоритмически генерировать беспрефиксные коды минимальной длины. На помощь приходят бинарные деревья. Почему?

Символ	Кодирование
<i>A</i>	00
<i>B</i>	01
<i>C</i>	10
<i>D</i>	11

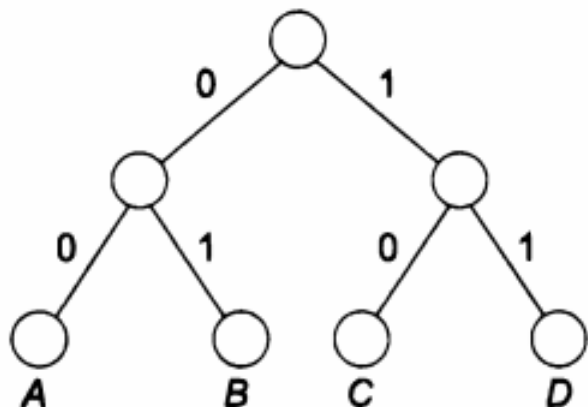
Символ	Кодирование
<i>A</i>	0
<i>B</i>	01
<i>C</i>	10
<i>D</i>	1

Символ	Кодирование
<i>A</i>	0
<i>B</i>	10
<i>C</i>	110
<i>D</i>	111

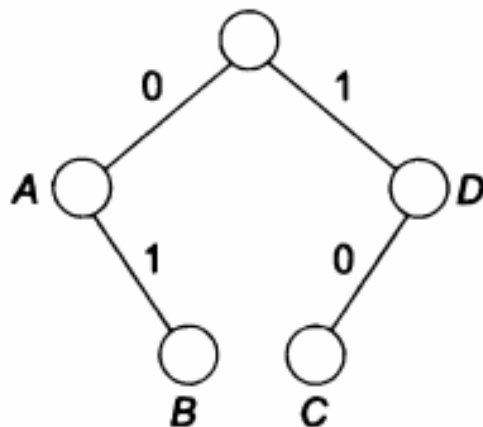
Как генерировать такие коды?

Требуется алгоритмически генерировать беспрефиксные коды минимальной длины. На помощь приходят бинарные деревья. Почему?

Символ	Кодирование
<i>A</i>	00
<i>B</i>	01
<i>C</i>	10
<i>D</i>	11



Символ	Кодирование
<i>A</i>	0
<i>B</i>	01
<i>C</i>	10
<i>D</i>	1

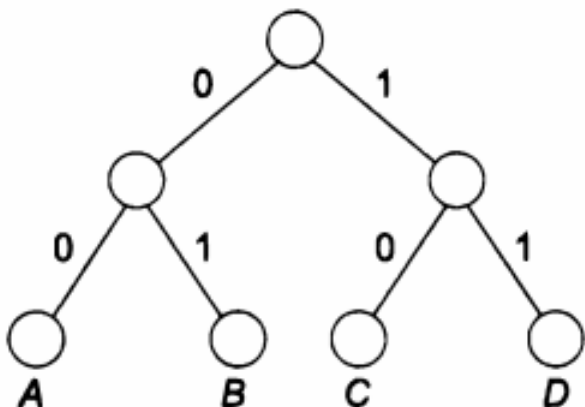


Символ	Кодирование
<i>A</i>	0
<i>B</i>	10
<i>C</i>	110
<i>D</i>	111

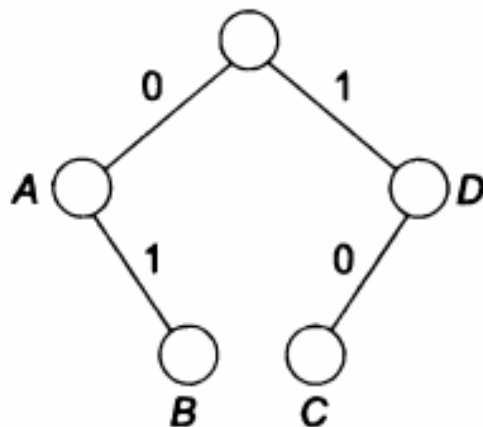
Как генерировать такие коды?

Требуется алгоритмически генерировать беспрефиксные коды минимальной длины. На помощь приходят бинарные деревья. Почему?

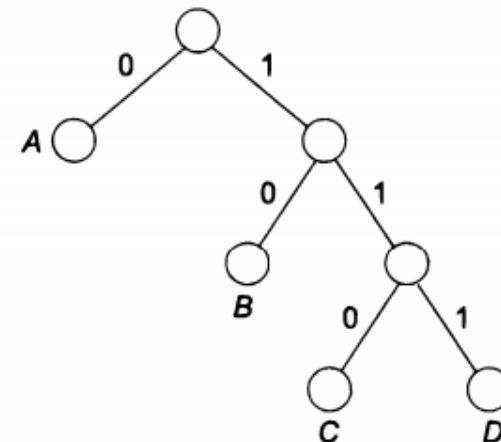
Символ	Кодирование
<i>A</i>	00
<i>B</i>	01
<i>C</i>	10
<i>D</i>	11



Символ	Кодирование
<i>A</i>	0
<i>B</i>	01
<i>C</i>	10
<i>D</i>	1



Символ	Кодирование
<i>A</i>	0
<i>B</i>	10
<i>C</i>	110
<i>D</i>	111



Формулируем определение

Каждый двоичный код может быть представлено в виде двоичного дерева, в котором левое и правое дочерние рёбра помечены соответственно 0 и 1, и каждый символ алфавита используется в качестве метки только для одного узла. И наоборот.

Формулируем определение

Каждый двоичный код может быть представлено в виде двоичного дерева, в котором левое и правое дочерние рёбра помечены соответственно 0 и 1, и каждый символ алфавита используется в качестве метки только для одного узла. И наоборот.

Вводим ограничение: Помеченными могут быть только листья.

Формулируем определение

Каждый двоичный код может быть представлено в виде двоичного дерева, в котором левое и правое дочерние рёбра помечены соответственно 0 и 1, и каждый символ алфавита используется в качестве метки только для одного узла. И наоборот.

Вводим ограничение: Помеченными могут быть только листья.

**ЗАДАЧА: ОПТИМАЛЬНЫЕ БЕСПРЕФИКСНЫЕ КОДЫ
(В НОВОЙ ФОРМУЛИРОВКЕ)**

Вход: неотрицательная частота p_a для каждого символа a алфавита Σ размера $n \geq 2$.

Выход: Σ -дерево с минимально возможной средней глубиной листа (14.1).

Строим дерево – алгоритм Хаффмана

Суть алгоритма Хаффмана заключается в использовании восходящего подхода, начиная с листьев дерева.

Символ	Частота
<i>A</i>	0,60
<i>B</i>	0,25
<i>C</i>	0,10
<i>D</i>	0,05

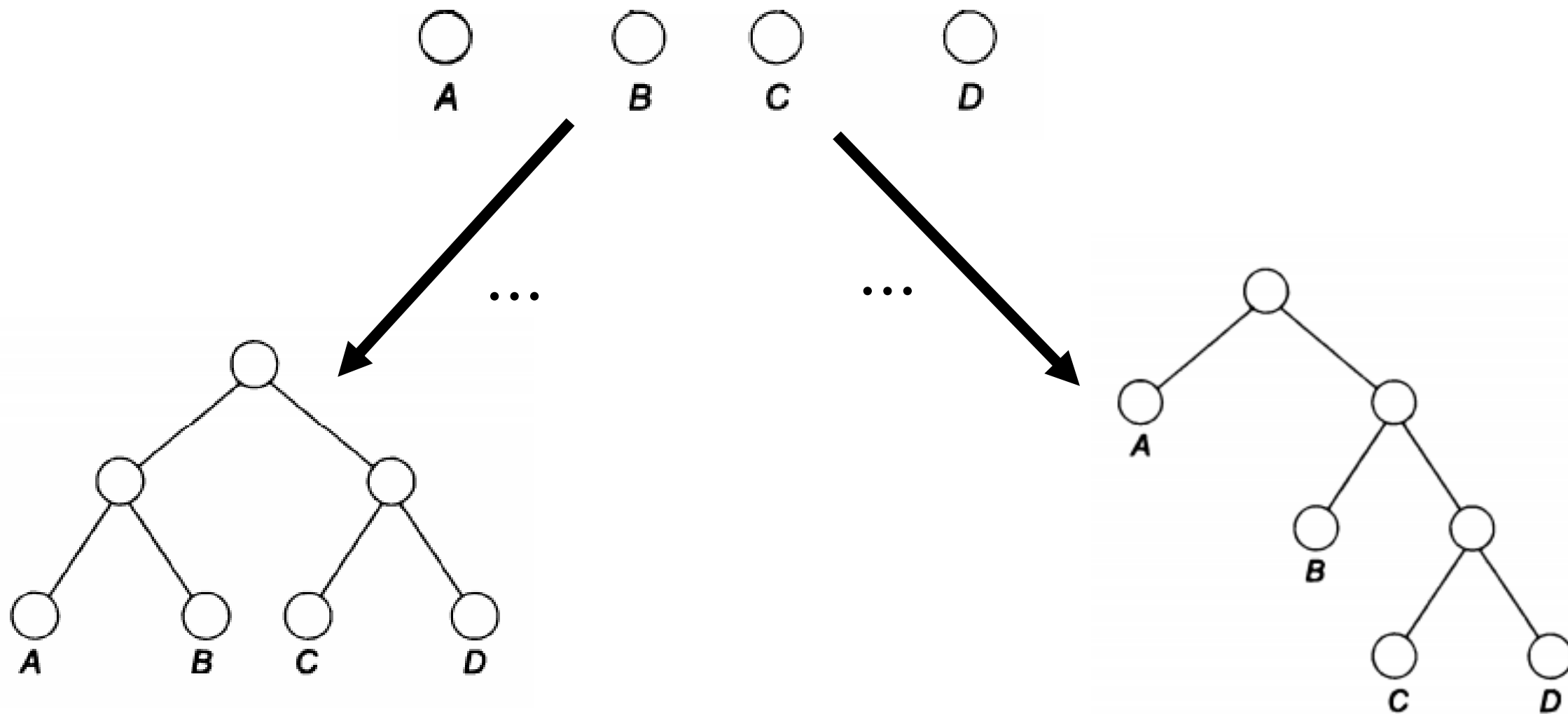
○
A

○
B

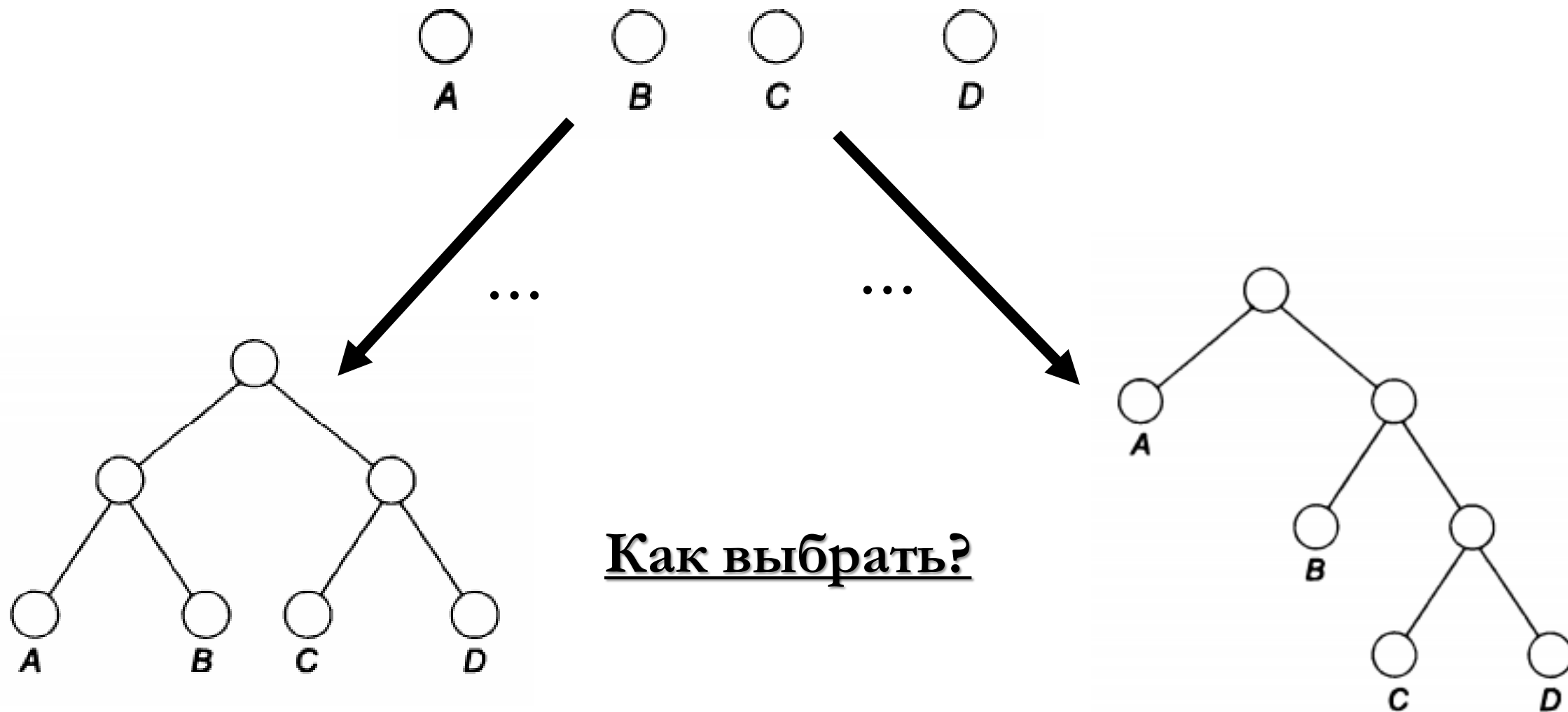
○
C

○
D

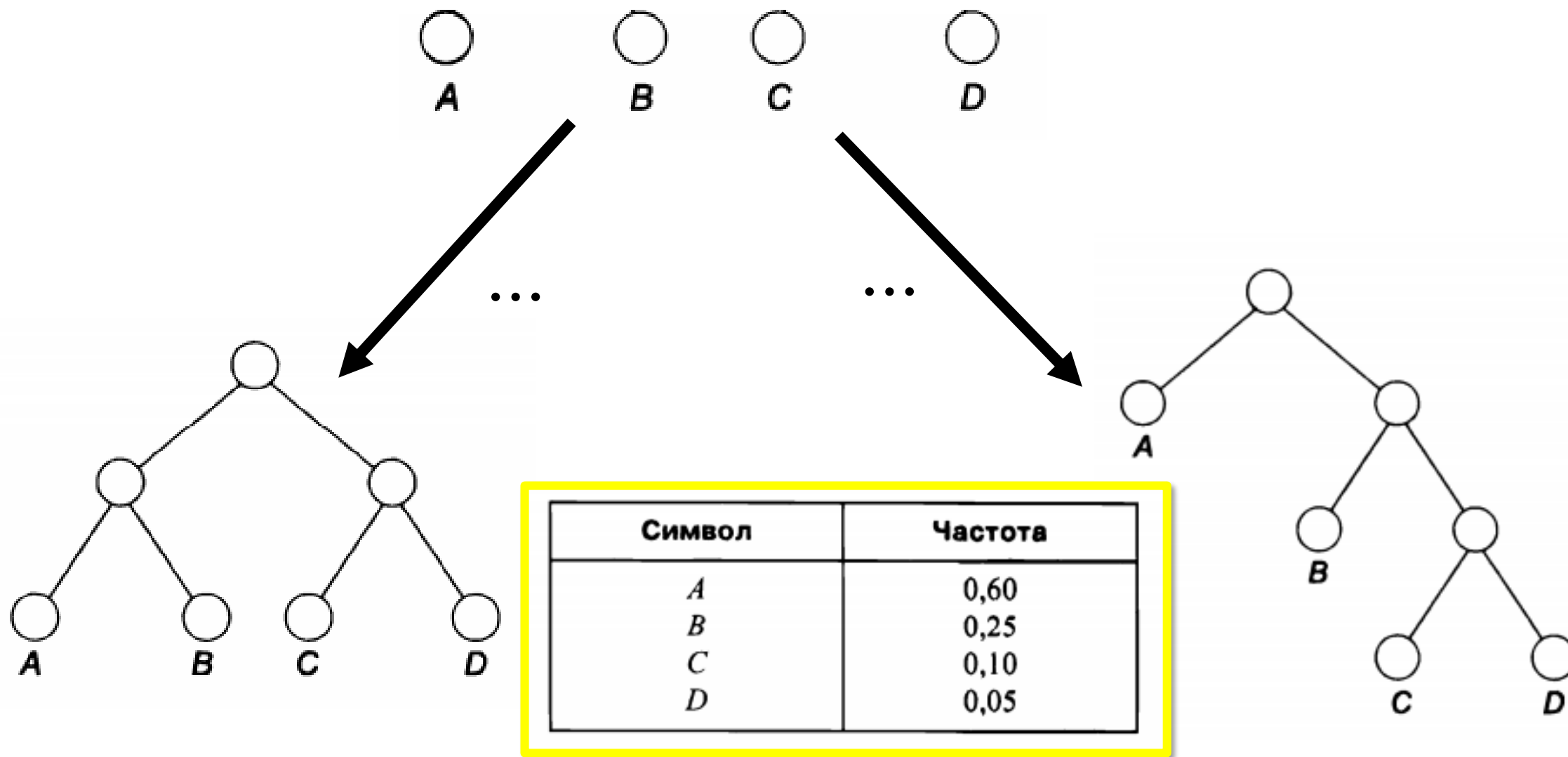
Строим дерево – алгоритм Хаффмана



Строим дерево – алгоритм Хаффмана



Строим дерево – алгоритм Хаффмана



Жадный критерий хаффмана

Шаг алгоритма: слияние деревьев.

Какие деревья выбрать?

Жадный критерий хаффмана

Шаг алгоритма: слияние деревьев.

Какие деревья выбрать?

Критерий: слияние должно приводить к минимального возможному увеличению средней глубины листа.

Жадный критерий хатфмана

Шаг алгоритма: слияние деревьев.

Какие деревья выбрать?

Критерий: слияние должно приводить к минимального возможному увеличению средней глубины листа.

Для каждого символа a в одном из двух участвующих деревьев глубина соответствующего листа увеличивается на 1, а вклад соответствующего члена в сумму (14.1) увеличивается на p_a . Таким образом, слияние двух деревьев T_1 и T_2 увеличивает среднюю глубину листа на сумму частот участвующих символов:

$$\sum_{a \in T_1} p_a + \sum_{a \in T_2} p_a, \quad (14.2)$$

Жадный критерий хатфмана

Шаг алгоритма: слияние деревьев.

Какие деревья выбрать?

Критерий: слияние должно приводить к минимального возможному увеличению средней глубины листа.

Символ	Частота
<i>A</i>	0,60
<i>B</i>	0,25
<i>C</i>	0,10
<i>D</i>	0,05

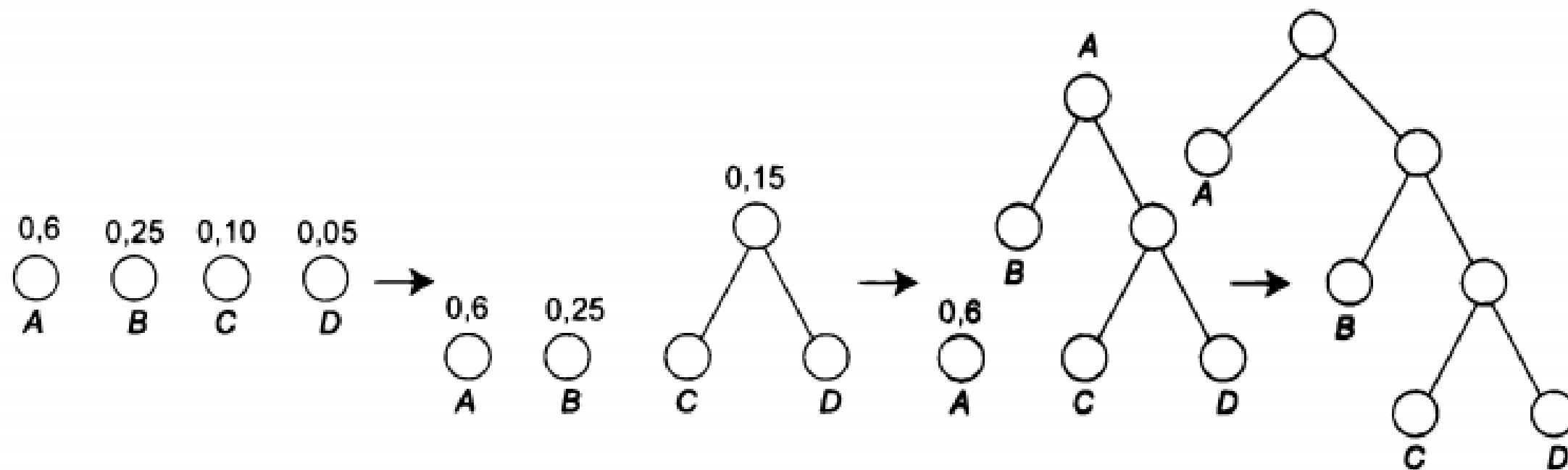
Для каждого символа a в одном из двух участвующих деревьев глубина соответствующего листа увеличивается на 1, а вклад соответствующего члена в сумму (14.1) увеличивается на p_a . Таким образом, слияние двух деревьев T_1 и T_2 увеличивает среднюю глубину листа на сумму частот участвующих символов:

$$\sum_{a \in T_1} p_a + \sum_{a \in T_2} p_a,$$

(14.2)



Шаги построения дерева



Анализ сложности

Анализ сложности

Построение таблицы распределения символов

Пока больше одного узла:

ищем два минимальных

объединяем

кладем обратно

Анализ сложности

Построение таблицы распределения символов

Построение кучи из массива деревьев

Пока больше одного узла:

ищем два минимальных

объединяем

кладем обратно

ВЫВОДЫ

- ★ Беспрефиксные двоичные коды переменной длины могут иметь меньшие средние длины кодирования, чем коды фиксированной длины, когда разные символы алфавита имеют разные частоты.
- ★ Жадный алгоритм Хаффмана поддерживает лес, где листья находятся в соответствии с символами алфавита, и на каждой итерации жадно выполняет слияние пары деревьев, вызывая минимально возможное увеличение средней глубины листа.
- ★ Алгоритм Хаффмана гарантированно вычисляет беспрефиксный код с минимально возможной средней длиной кодирования.
- ★ Алгоритм Хаффмана может быть реализован с работой за время $O(n \log n)$, где n — это число символов.


```
func buildHuffmanTree(charFreq freqTable) *huffmanBTNode {  
    nodes := make(heapOfNodes, 0, len(charFreq))  
    for char, freq := range charFreq {  
        nodes = append(nodes, &huffmanBTNode{chars: []rune{char}, weight: freq})  
    }  
    heap.Init(&nodes)  
  
    for len(nodes) > 1 {  
        leftNode := heap.Pop(&nodes).(*huffmanBTNode)  
        rightNode := heap.Pop(&nodes).(*huffmanBTNode)  
        newNode := mergeHuffmanBTNodes(leftNode, rightNode)  
        heap.Push(&nodes, newNode)  
    }  
    return heap.Pop(&nodes).(*huffmanBTNode)  
}
```

```
199 func generateCodesByTreeTraverse(root *huffmanBTNode, codes encodeTable) {
200     if root.IsLeaf() {
201         codes[root.chars[0]] = "0"
202         return
203     }
204
205     var traverse func(rootNode *huffmanBTNode, prevCode string)
206     traverse = func(rootNode *huffmanBTNode, prevCode string) {
207         if rootNode.IsLeaf() {
208             if len(rootNode.chars) != 1 {
209                 panic("Leaf has != 1 lenght of chars")
210             }
211             codes[rootNode.chars[0]] = prevCode
212             return
213         }
214         traverse(rootNode.left, prevCode+"0")
215         traverse(rootNode.right, prevCode+"1")
216     }
217     traverse(root, "")
218 }
```