****

****

**Merge Sort**

3.2. Слияние двух подмассивов в третий результирующий массив.

На каждом шаге мы берём меньший из двух первых элементов подмассивов и записываем его в результирующий массив. Счётчики номеров элементов результирующего массива и подмассива, из которого был взят элемент, увеличиваем на 1.

3.3. «Прицепление» остатка.

Когда один из подмассивов закончился, мы добавляем все оставшиеся элементы второго подмассива в результирующий массив.

**Merge**

2 8 1 4

\_ \_

сравниваем 2 и 1 и записываем 1 как меньшее число, индекс в правом массиве сдвигаем вправо

1

сравниваем 2 и 4, записываем 2, как меньшее число и индекс в левом массиве сдвигаем вправо, получаем

1 2

сравниваем 8 и 4, записываем 4, как меньшее, правый массив закончился, следовательно после этого все, что было слева дописываем к результату

1 2 4 8

**Another merge**

1 2 4 8 3 5 6 7

\_ \_

сравниваем 1 и 3 и записываем 1 как меньшее число, индекс в левом массиве сдвигаем вправо

1

сравниваем 2 и 3 и записываем 2 как меньшее число, индекс в левом массиве сдвигаем вправо

1 2

сравниваем 4 и 3 и записываем 3 как меньшее число, индекс в правом массиве сдвигаем вправо

1 2 3

сравниваем 4 и 5 и записываем 4 как меньшее число, индекс в левом массиве сдвигаем вправо

1 2 3 4

сравниваем 8 и 5 и записываем 5 как меньшее число, индекс в правом массиве сдвигаем вправо

1 2 3 4 5

сравниваем 8 и 6 и записываем 6 как меньшее число, индекс в правом массиве сдвигаем вправо

1 2 3 4 5 6

сравниваем 8 и 7 и записываем 7 как меньшее число, индекс в правом массиве сдвигаем вправо

1 2 3 4 5 6 7 8

(Когда один из подмассивов закончился, мы добавляем все оставшиеся элементы второго подмассива в результирующий массив, т.е. не только 7, но и 8 сразу добавляем)

**Insertion sort**

1) Есть входные данные

2) Считаем что в выходных нету ничего

3) Берем из входных данных текущий символ (первый, потом второй, третий и тд) и ищем ему подходящее место в выходных

4) Вставляем символ в подходящую позицию, выходной поток увеличился на 1 символ

5) Сдвигаем указатель во входящем потоке

6) Повторяем с 3

начальные данные:

dacxszy

входной поток:

dacxszy

\_ - это будет указатель текущего символа

выходной поток:

пусто

входной поток:

dacxszy

\_

выходной поток

d

входной поток:

dacxszy

\_

выходной поток

ad

входной поток:

dacxszy

\_

выходной поток

acd

входной поток:

dacxszy

\_

выходной поток

acdx

входной поток:

dacxszy

\_

выходной поток

acdsx

входной поток:

dacxszy

\_

выходной поток

acdsxz

входной поток:

dacxszy

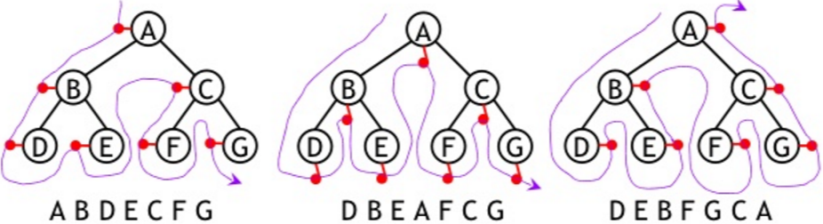
\_

выходной поток

acdsxyz







**Coding tree**

To generate a tree from \*a\*\*d\*c!\*rb we have to remember that here we don't have a principle "smaller in the left and bigger in the right".

No need to worry about it. Instead, here's the principle:

1) first star (or a point, if we look in the lecture slides) is always the root

2) then we fill symbols(a, \_, -, whatever) into left and right (in such order) leaves

3) if there's a star after a star, then it's put in the left side

4) if there's a star after a letter - we go up until we find a free right leaf

Now we build a tree \*a\*\*d\*c!\*rb

1) \* as a root

2) a -> into the left leaf

3) \* -> we go up and then from root star go to right leaf and put a star there

3) \* -> into left leaf after second \*

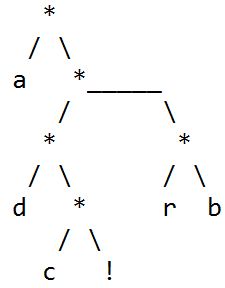
4) d -> into left leaf after third \*

5) \* -> go up and then from third star go to right leaf and put a star there

6) c and ! -> into left and right leaves

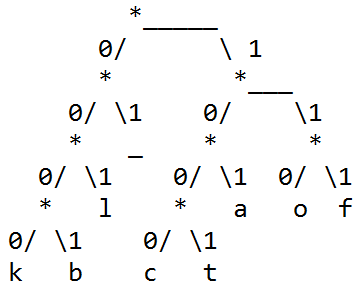
7) \* -> go up to the second star (it has free right leaf)

8) r and b -> into left and right leaves



///////////////////////////----------------------------------------

\*\*\*\*kbl\_\*\*\*cta\*of



0 for every left node and 1 for every right node

doesn't matter if we make a table in alphabet order, but OK, we do it in such way

for "b" start the route from root star 0, then 0, then 0, then riiiight, it's 1 and thus 0001

a 101

b 0001

c 1000

k 0000

f 111

l 001

o 110

t 1001

\_ 01

Calculate in a table the frequency of the characters {\_, a, b, ..., o } in the s1="a\_flock\_of\_bat\_of\_a\_block\_of\_flat\_of\_lot\_of\_a\_cat\_flab\_a\_flat\_cab"

a = 10 (0 = x1 1 = x2) (we wrote how many 0 and 1 for each symbol)

b = 4 (0 = x3 1 = x1) (don't mind the "x", just three 0 (x3) and one 1 (x1) )

c = 4 (0 = x3 1 = x1)

k = 2 (0 = x4 1 = x0)

f = 9 (0 = x0 1 = x3)

l = 6 (0 = x2 1 = x1)

o = 8 (0 = x1 1 = x2)

t = 5 (0 = x2 1 = x2)

\_ = 17 (0 = x1 1 = x1)

65

The number of bits needed to encode the String s1 = 10\*3 + 4\*4 + 4\*4 + 2\*4 + 9\*3 + 6\*3 + 8\*3 + 5\*4 + 17\*2 = 30 + 16 + 16 + 8 + 27 + 18 + 24 + 20 + 34 = 30 + 40 + 45 + 78 = 115 + 78 = 193 bits

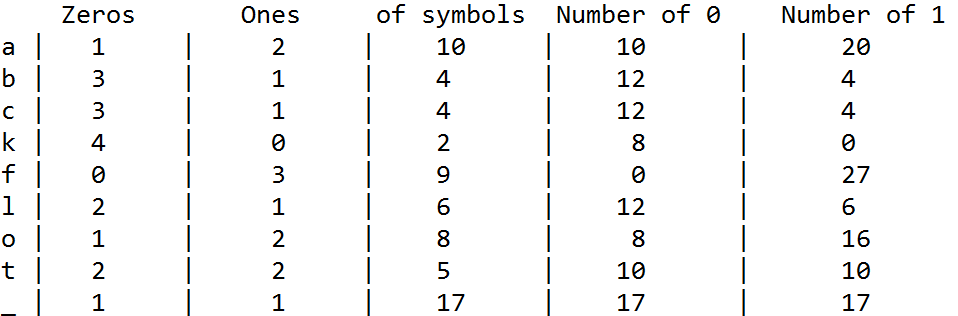
as well as the number of 0's and 1's that will appear

0 => 10\*1 + 4\*3 + 4\*3 + 2\*4 + 9\*0 + 6\*2 + 8\*1 + 5\*2 + 17\*1 = 89 bits

1 => 10\*2 + 4\*1 + 4\*1 + 2\*0 + 9\*3 + 6\*1 + 8\*2 + 5\*2 + 17\*1 = 104 bits

193 bits

**Frequency**



-----------------------------------------------------------

Provide a coding tree for the String s2 = "Sam's\_shop\_stocks\_short\_spotted\_socks"

First we count the amount of every unique symbol

s

a

m

'

\_

h

o

p

t

c

k

r

e

d

14 symbols

Now we know that for coding this string we need 5 more symbols than in previous example.

We can take the tree from previous example and add these 5 symbols.

\*\_\_\_\_\_

0/ \ 1

\* \*\_\_\_

0/ \1 0/ \1

\* p \* \*

0/ \1 0/ \1 0/ \1

\* o \* k r e

0/ \1 0/ \1

\* h \* c

/ \ / \

\* \_ t d

/ \

\* '

/ \

\* m

/ \

s a

\*\*\*\*\*\*\*\*sam'\_hop\*\*\*\*tdck\*re

Of course, it's not the most optimal tree (could be better), but it will work for sure.

------Huffman coding:

For Huffman coding we don't get something like \*\*\*\*\*\*\*\*sam'\_hop\*\*\*\*tdck\*re from a program.

This method needs an example (a string). It takes and counts the frequency of each symbol.

For example:

a b r a c a d a b r a !

symbol "c" can be found 1 time, symbol "!" - 1 time too, but "a" is found 5 times (the max possible amount).

Thus it's more optimal to insert "a" in the tree as close to the first root as possible to get the shortest code.

While symbols with min frequency (like "c") - as far from first root as possible (well, it's considered as a sacrifice in favor of optimality).