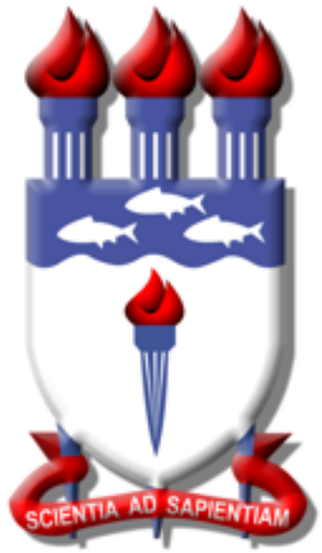


Estruturas de Dados / Programação 2
Codificação de Huffman

Márcio Ribeiro
marcio@ic.ufal.br
twitter.com/marciomribeiro

Fixed-Length Character Encodings

- ASCII (American Standard Code for Information Interchange)

- 8 bits per character

Character	Decimal	Hexadecimal	Binary
a	97	61	01100001
b	98	62	01100010
c	99	63	01100011
d	100	64	01100100
…	…	…	…
z	122	7A	01111010

Variable-Length Character Encodings

- Problem: fixed-length encodings waste space

"test … tested … test"

- Solution:
  - encodings of different lengths for different characters; and
  - assign shorter encodings to frequently occurring characters

Character	Binary
e	01
o	100
s	111
t	00

"test"

00 01 111 00 —> 000111100

test **01110100011001010111001101110100**

Character	Binary
e	01
o	100
s	111
t	00

000111100

71% smaller!

Requirement

- No character's encoding can be the prefix of another character's encoding

- Example: we cannot have

Character	Binary
e	00
t	001

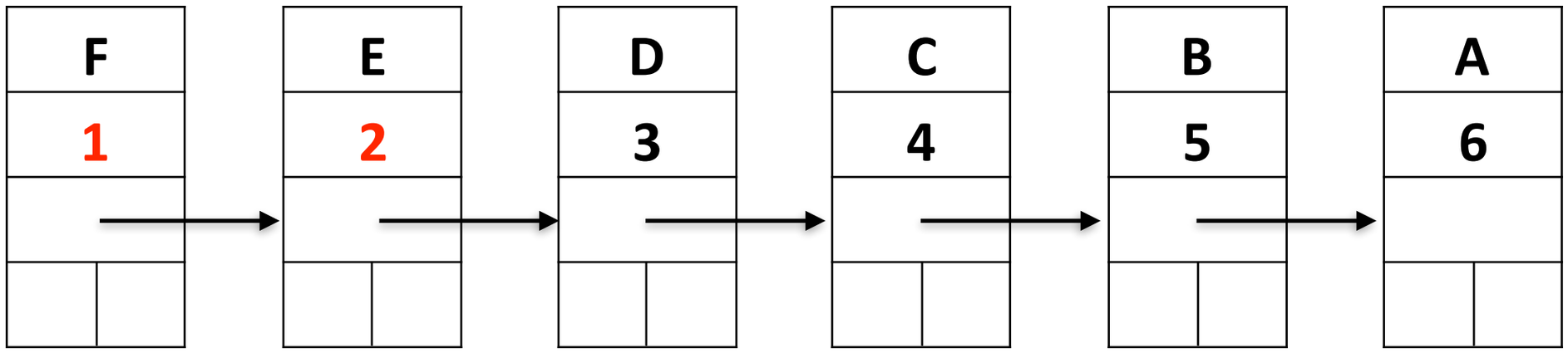
001111100

"e" or "t"?

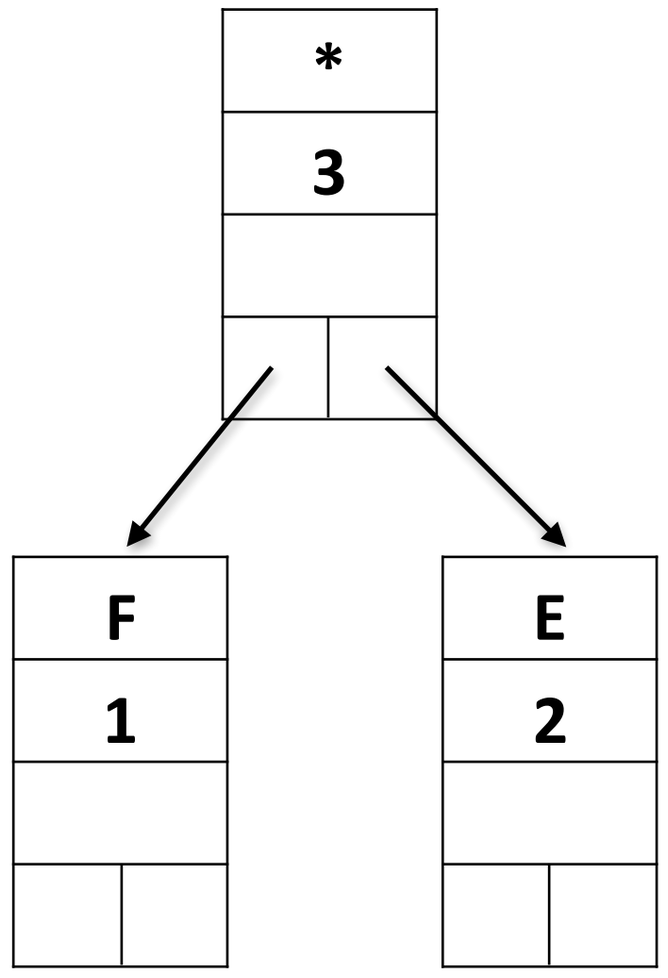
Huffman Coding

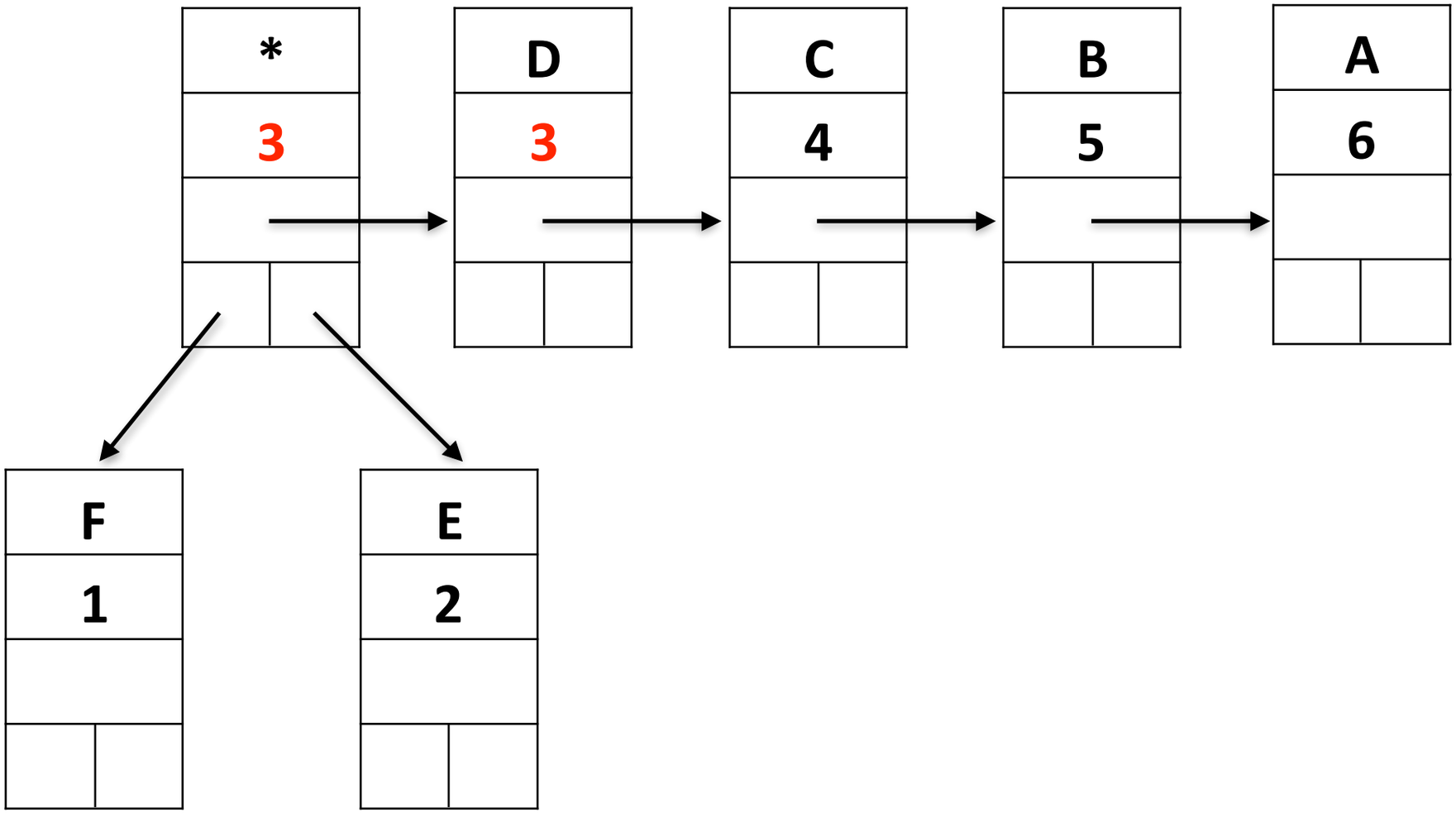
AAAAAABBBBBCCCCDDDEEF

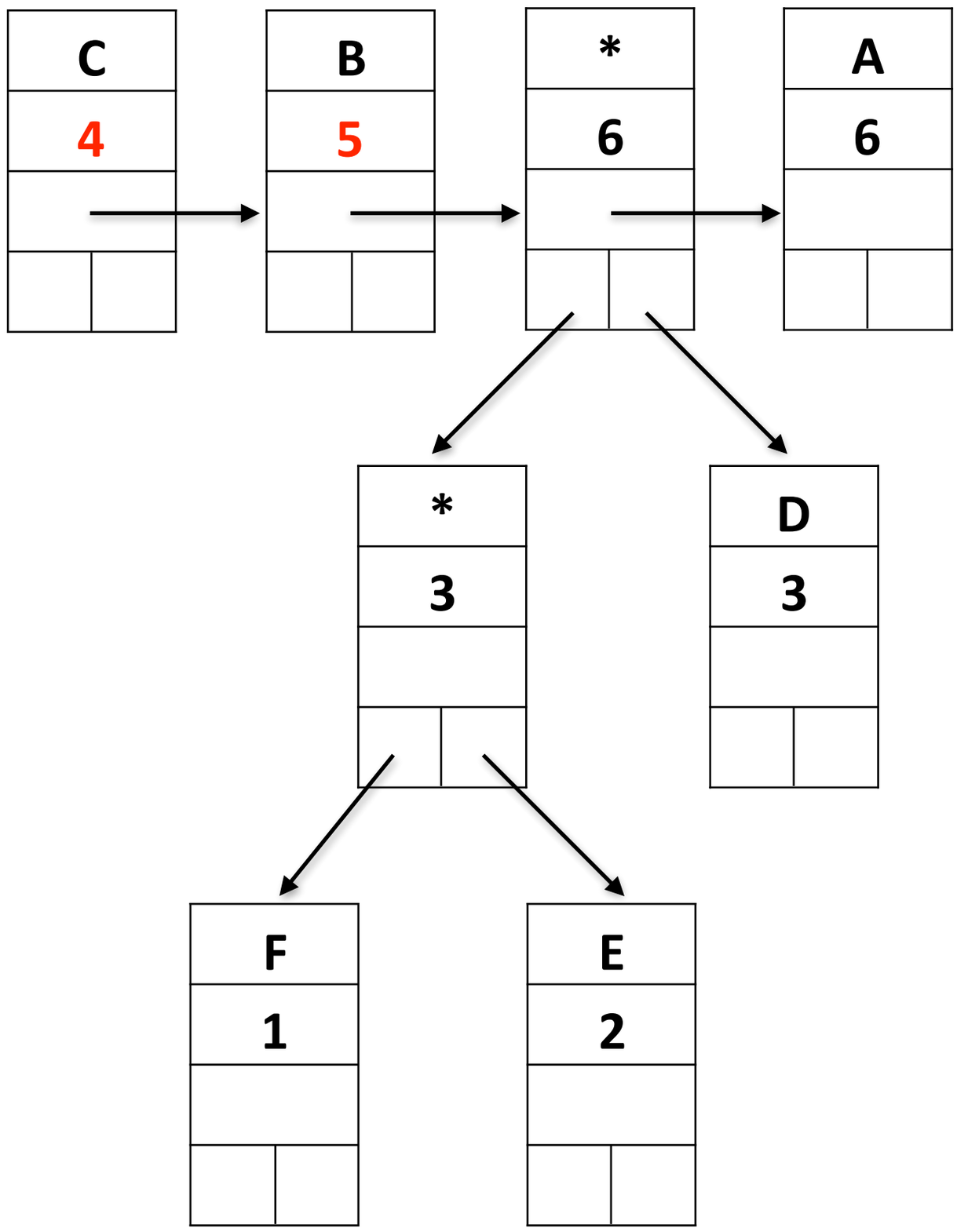
1) Read the text to determine the frequencies and create a list

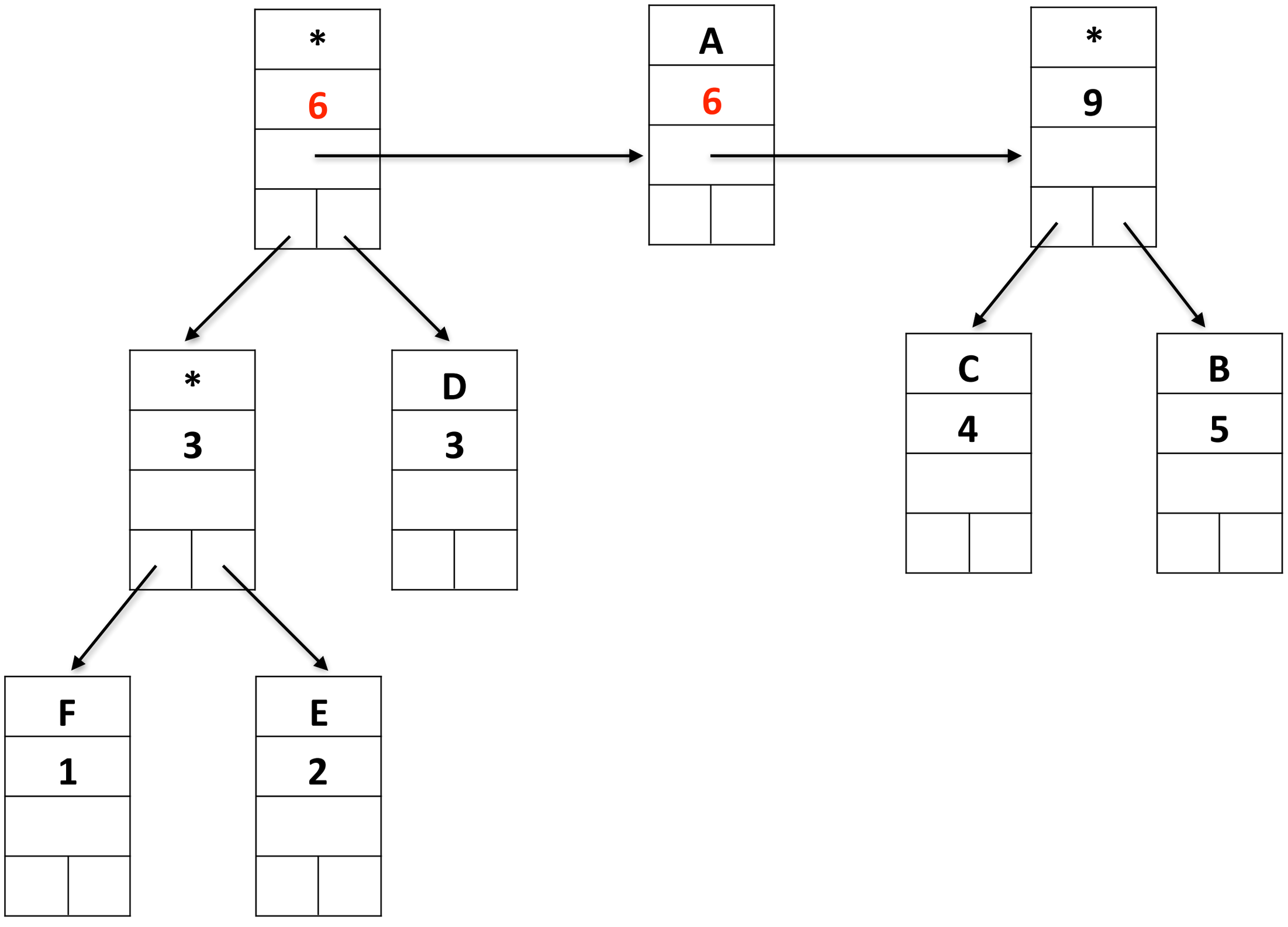


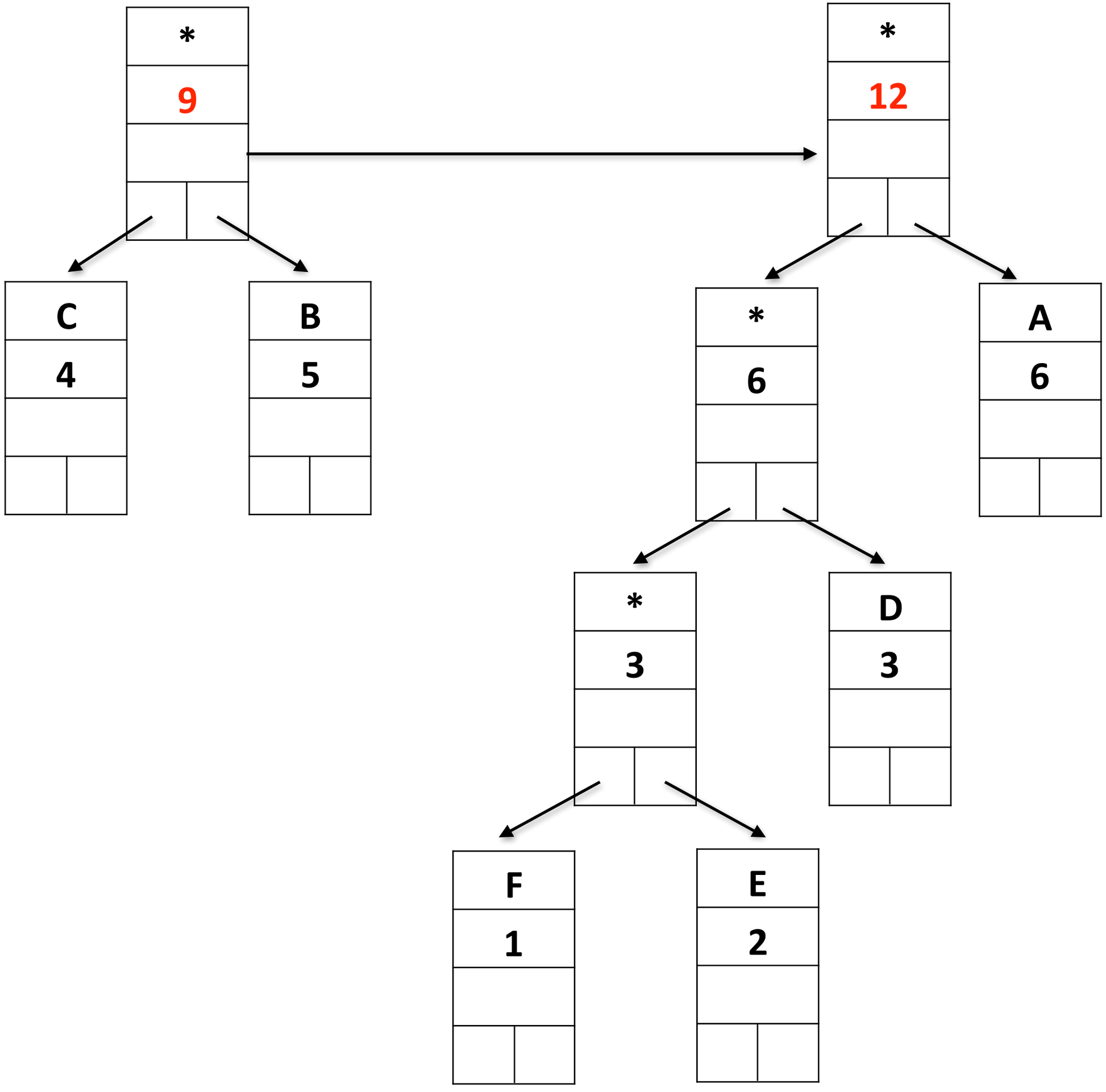
2) Remove and “merge” the nodes with the two lowest frequencies, forming a new node that is their parent. Frequency of parent = sum of the frequencies of its children

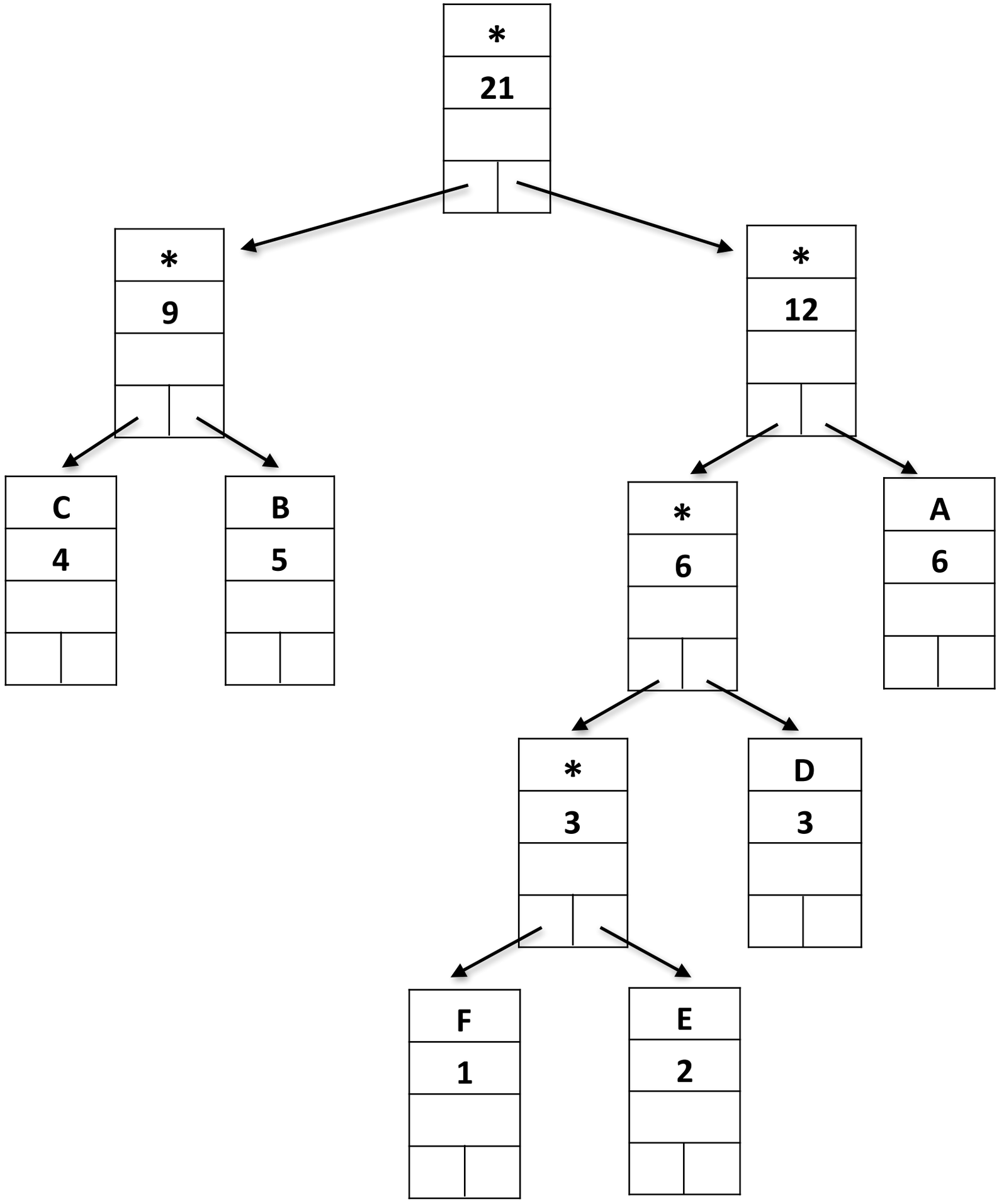



*
3
D
3
C
4
B
5
A
6
F
1
E
2

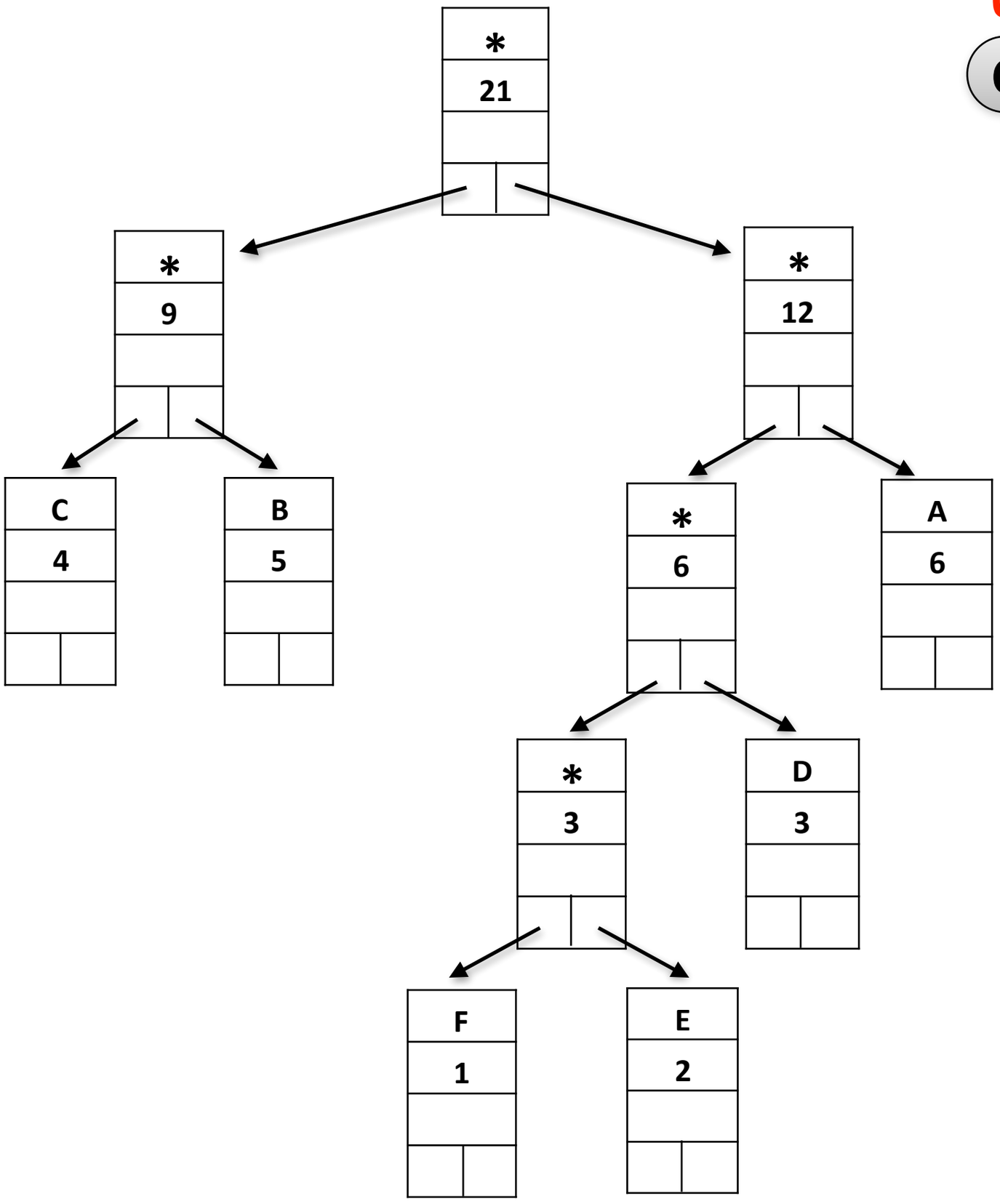

C
4
B
5
*
6
A
6
*
3
D
3
F
1
E
2

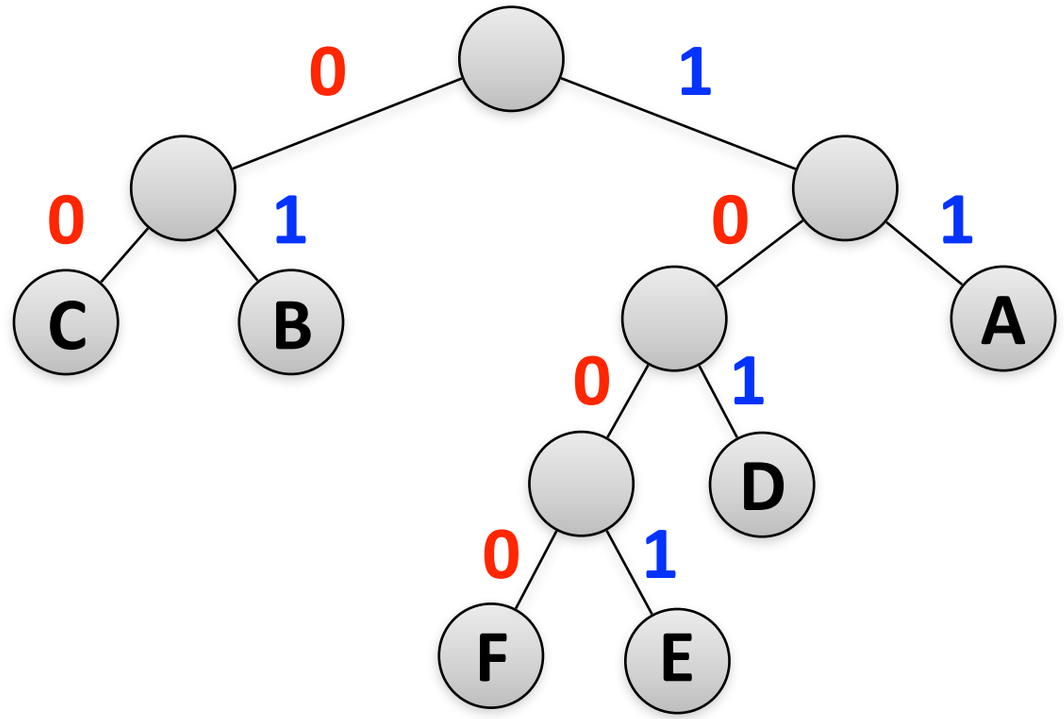

*
6
A
6
*
9
*
3
D
3
C
4
B
5
F
1
E
2


*
9
C
4
B
5
*
12
*
6
A
6
*
3
D
3
F
1
E
2


*
21
*
9
*
12
C
4
B
5
*
6
A
6
*
3
D
3
F
1
E
2

Huffman Tree





Character	Binary
A	11
B	01
C	00
D	101
E	1001
F	1000

Helpful functions

Bit shift Operators

x << n

Shifts the value of x left by n bits.

x >> n

Shifts the value of x right by n bits.

Note: here, we consider unsigned!

Left shift

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
1	1	0	0	0	1	1	1

x = x << 2

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	0	0	1	1	1	0	0

Right shift

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
1	1	0	0	0	1	1	1

x = x >> 2

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	0	1	1	0	0	0	1

Is bit "*i*" set?

```
int is_bit_i_set(unsigned char c, int i)
{
  unsigned char mask = 1 << i;
  return mask & c;
}
```

i = 4

	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
	1	1	0	0	0	1	1	1
1 << 4	0	0	0	1	0	0	0	0
&	0	0	0	0	0	0	0	0

4th bit is not set

Setting a bit

```
unsigned char set_bit(unsigned char c, int i)
{
  unsigned char mask = 1 << i;
  return mask | c;
}
```

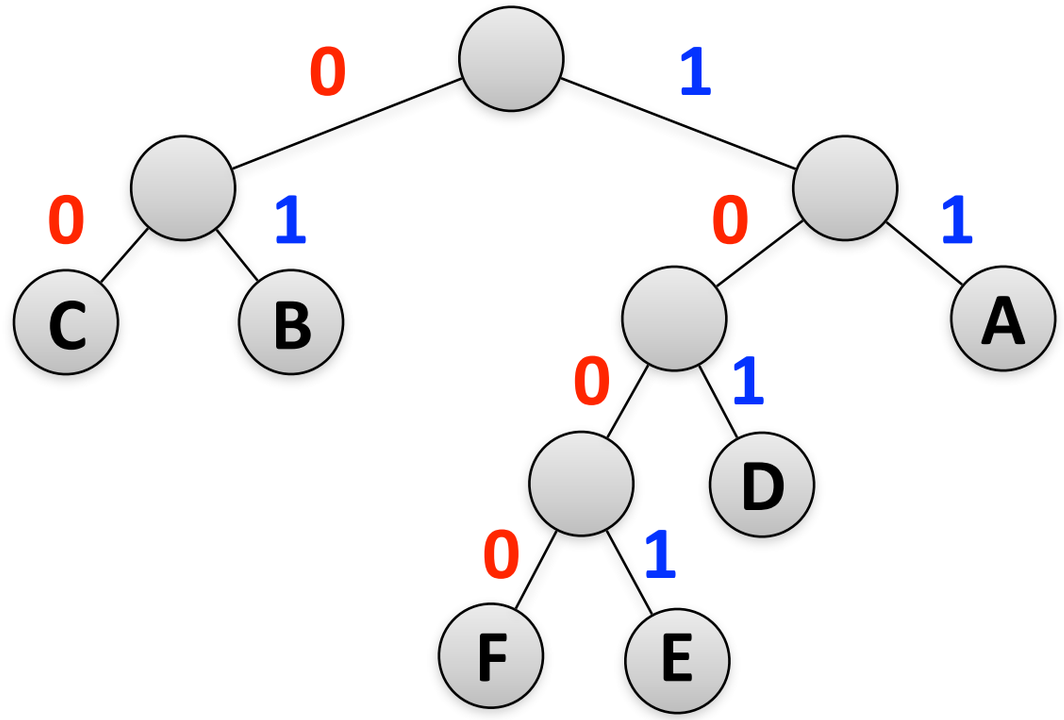
i = 4

	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
	1	1	0	0	0	1	1	1
1 << 4	0	0	0	1	0	0	0	0
|	1	1	0	1	0	1	1	1

4th bit is now set

AB 2

Character	Binary
A	11
B	01
C	00
D	101
E	1001
F	1000



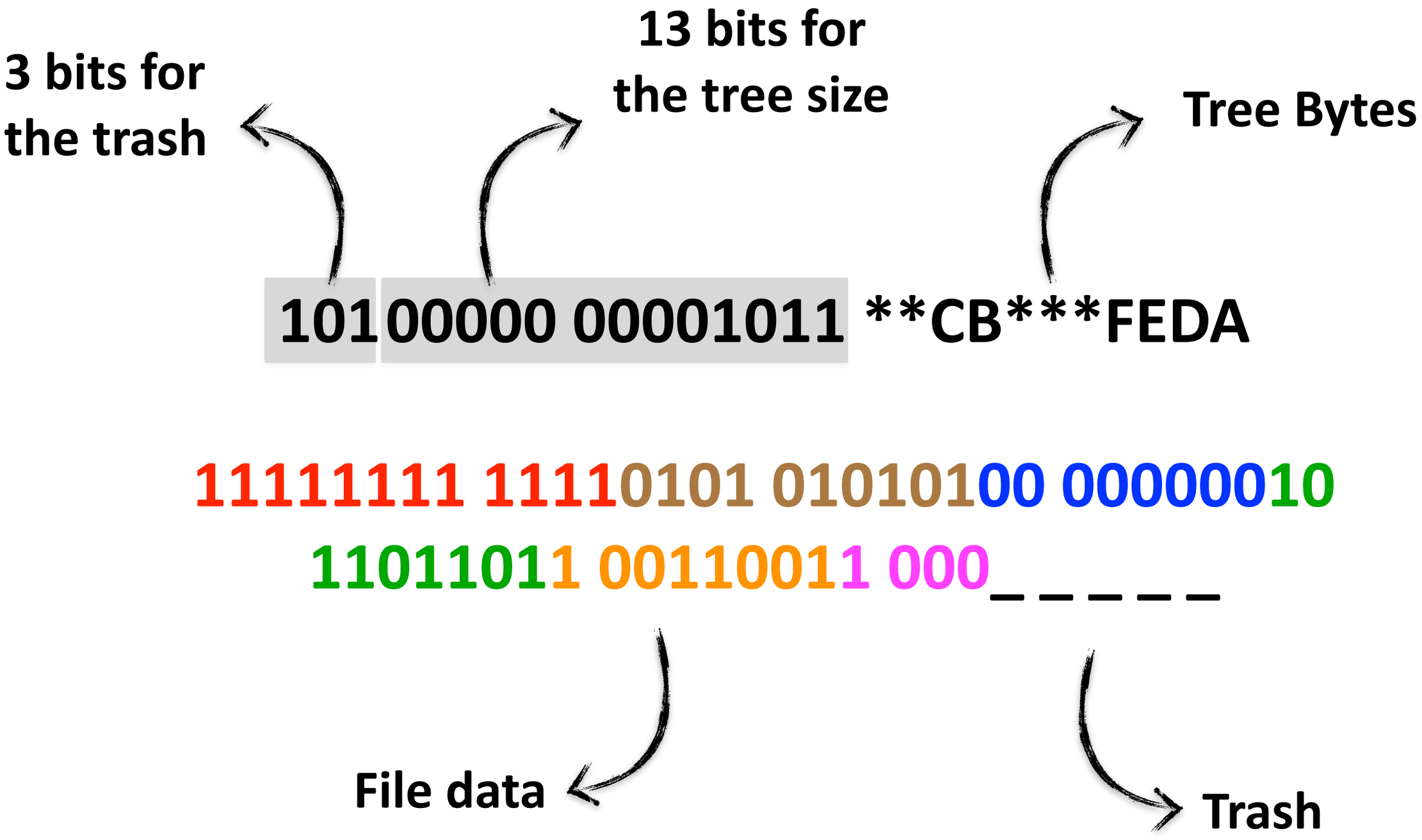
****CB***FEDA**

AAAAAABBBBBCCCCDDDEEF

11111111 11110101 01010100 00000010

11011011 00110011 000_ _ _ _ _

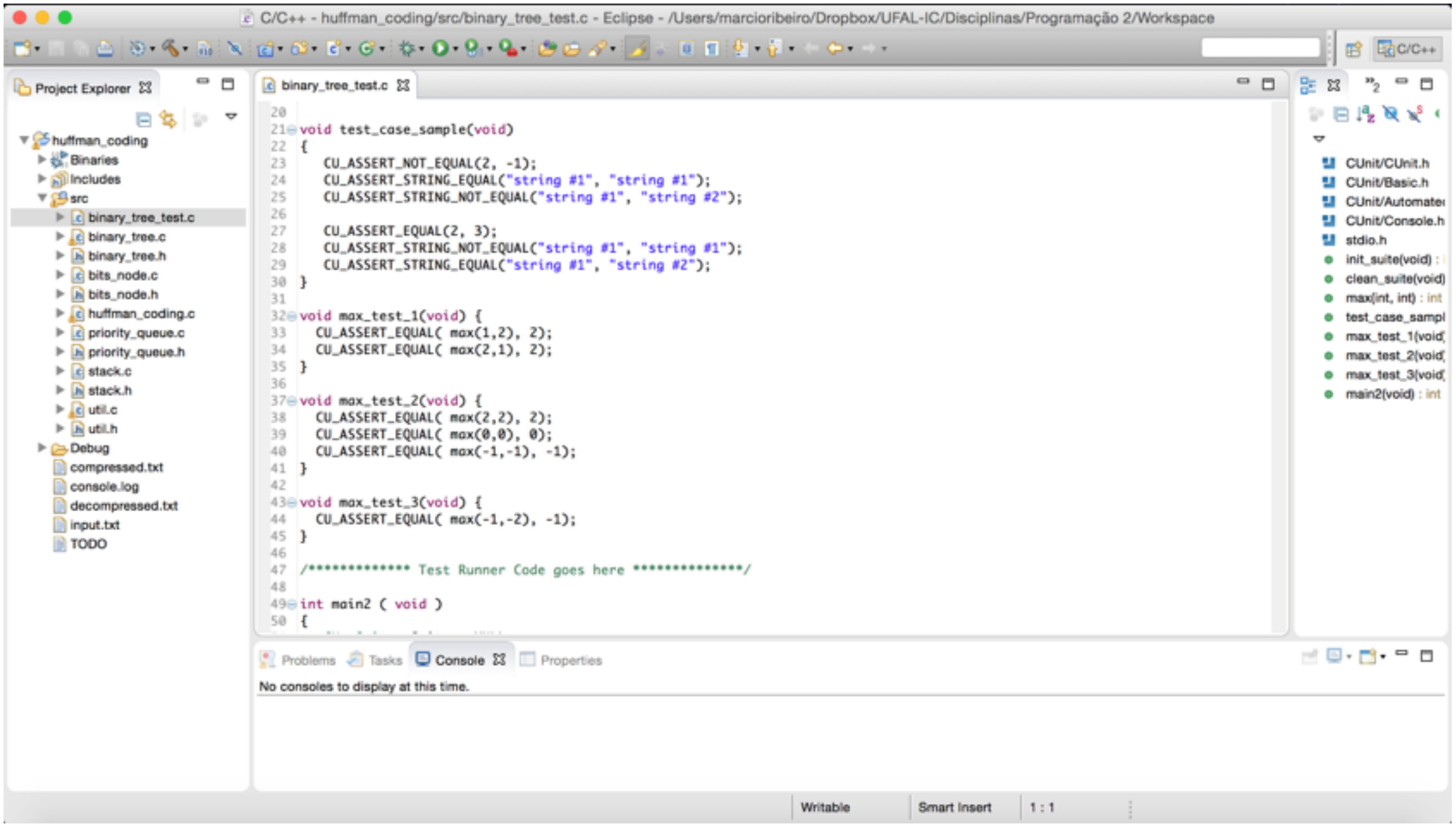
Header: filename.huff



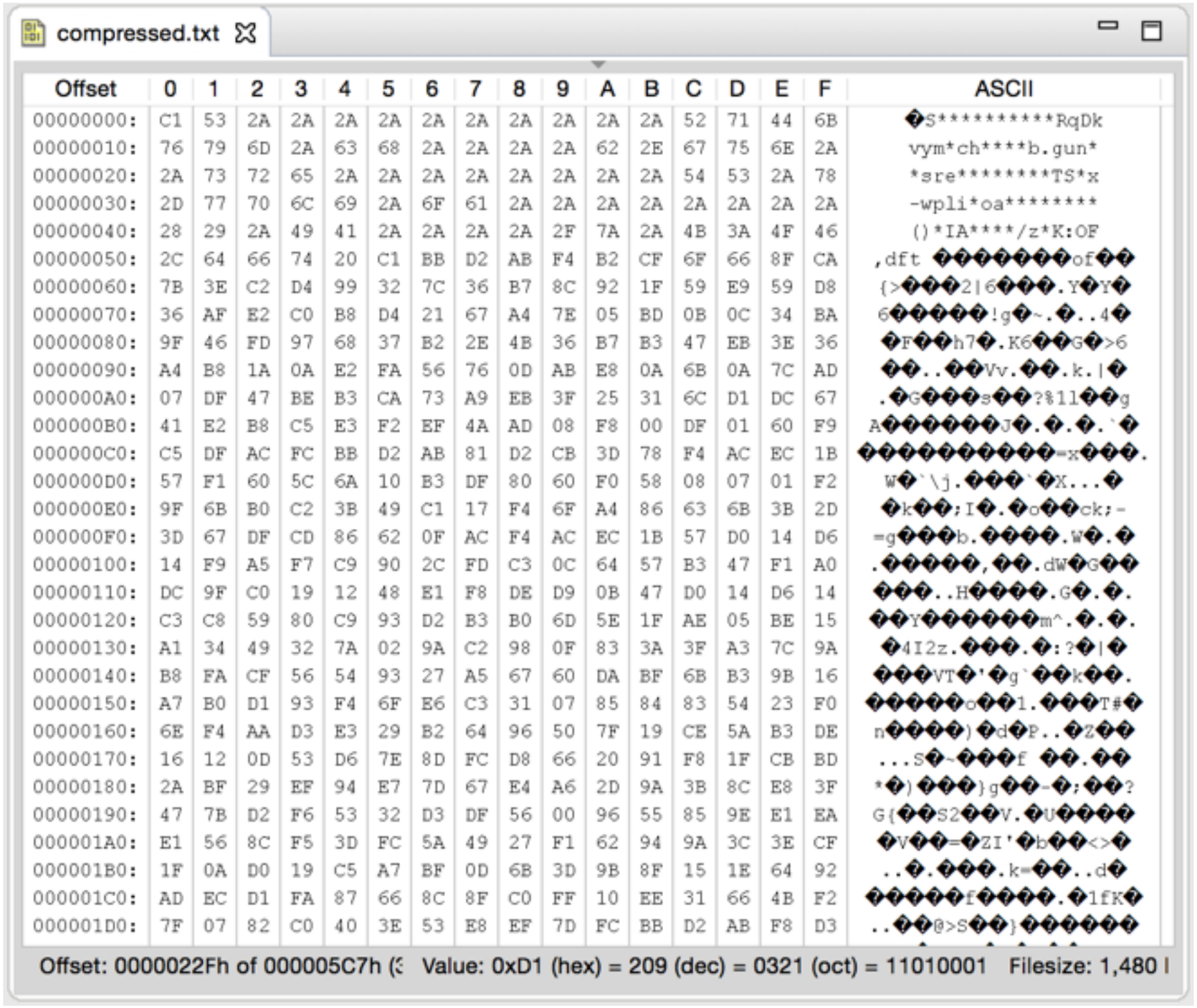
Mandatory Rules

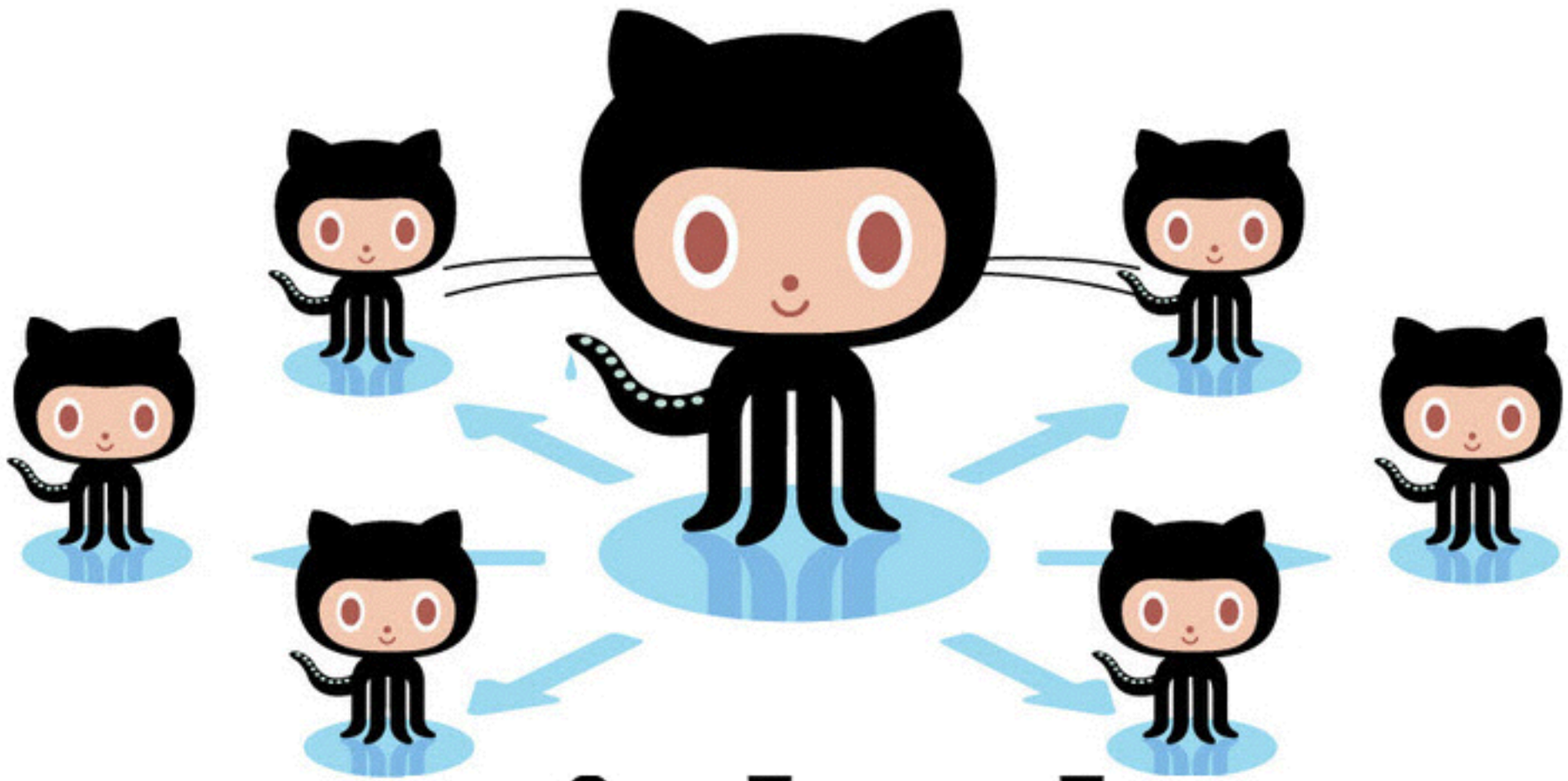
- Follow the header
- Save the tree using “*” and the pre-order traversal
- Use “\” as the scape character
- Encapsulation
- Documented ADTs
- ***void** in at least one Data Structure or Huffman to any file format

Eclipse + CUnit



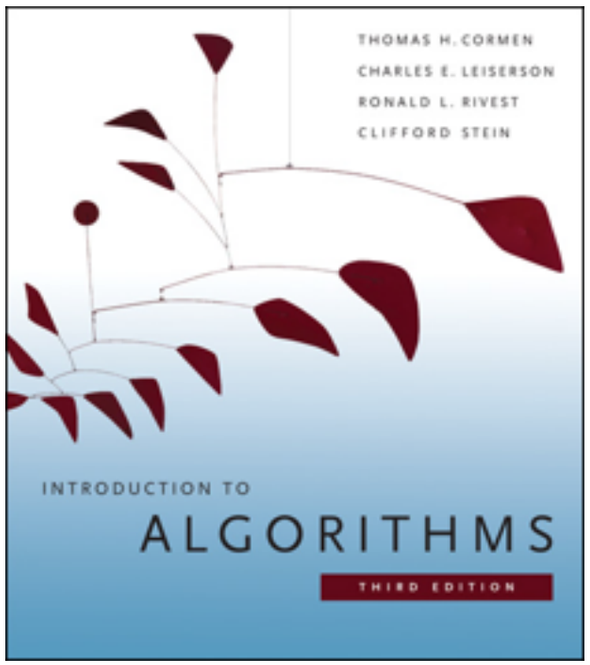
Eclipse Hex Editor Plugin





github
SOCIAL CODING

References



Chapter 16

Chapter 11