Codes de Huffman

Méthode de compression des fichiers

BUT : coder les "lettres" de plus grandes fréquences sur moins de bits que celles de fréquences plus petites

Mauvais exemple

CAFE
010111

Quel est le problème ?

A	0
B	10
C	01
D	00
E	1
F	11

Mauvais exemple

CAFE
010111

Quel est le problème ?

A	0
B	10
C	01
D	00
E	1
F	11

LE HIC : savoir où un code se termine et où un nouveau code commence

Mauvais exemple

CAFE
010111

Quel est le problème ?

A	0
B	10
C	01
D	00
E	1
F	11

LE HIC : savoir où un code se termine et où un nouveau code commence

SOLUTION : le codage d'une "lettre" n'est jamais un préfixe du codage d'une autre "lettre"

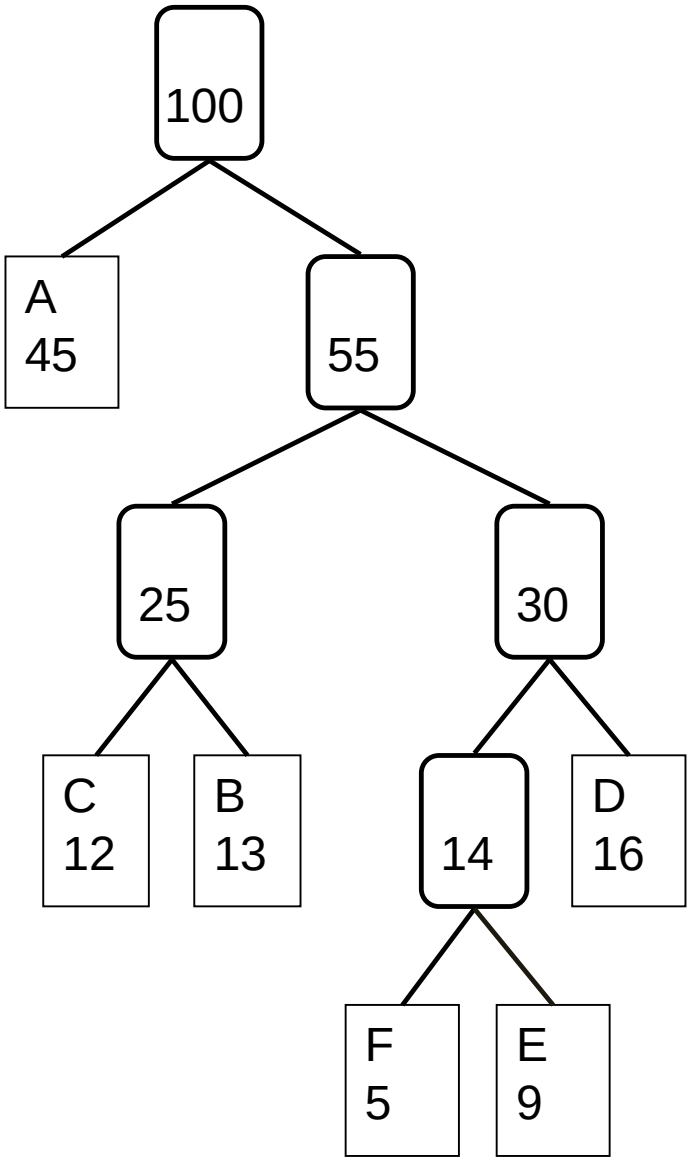
L'algorithme de Huffman construit un arbre binaire représentant le codage de chaque « lettre »

Exemple:

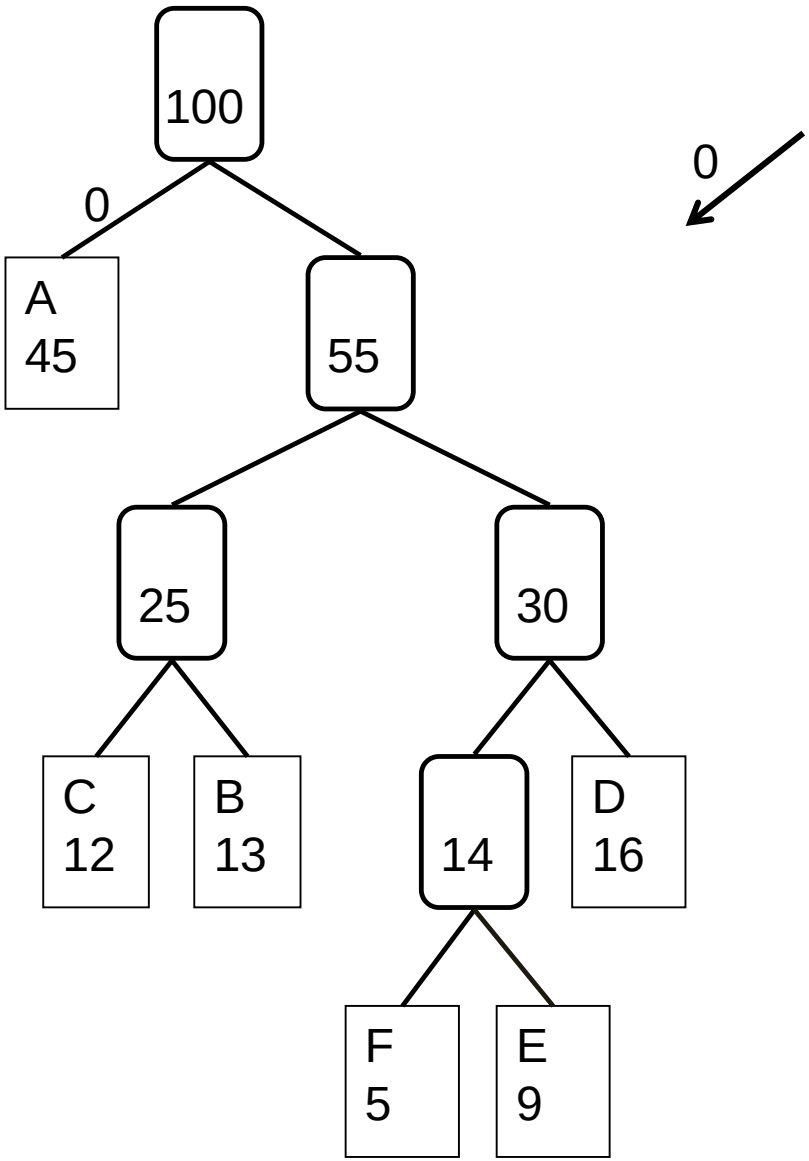
Supposons que notre texte ne contient que les lettres A, B, C, D, E et F et qu'on a la map des fréquences suivantes :

A	B	C	D	E	F
45	13	12	16	9	5

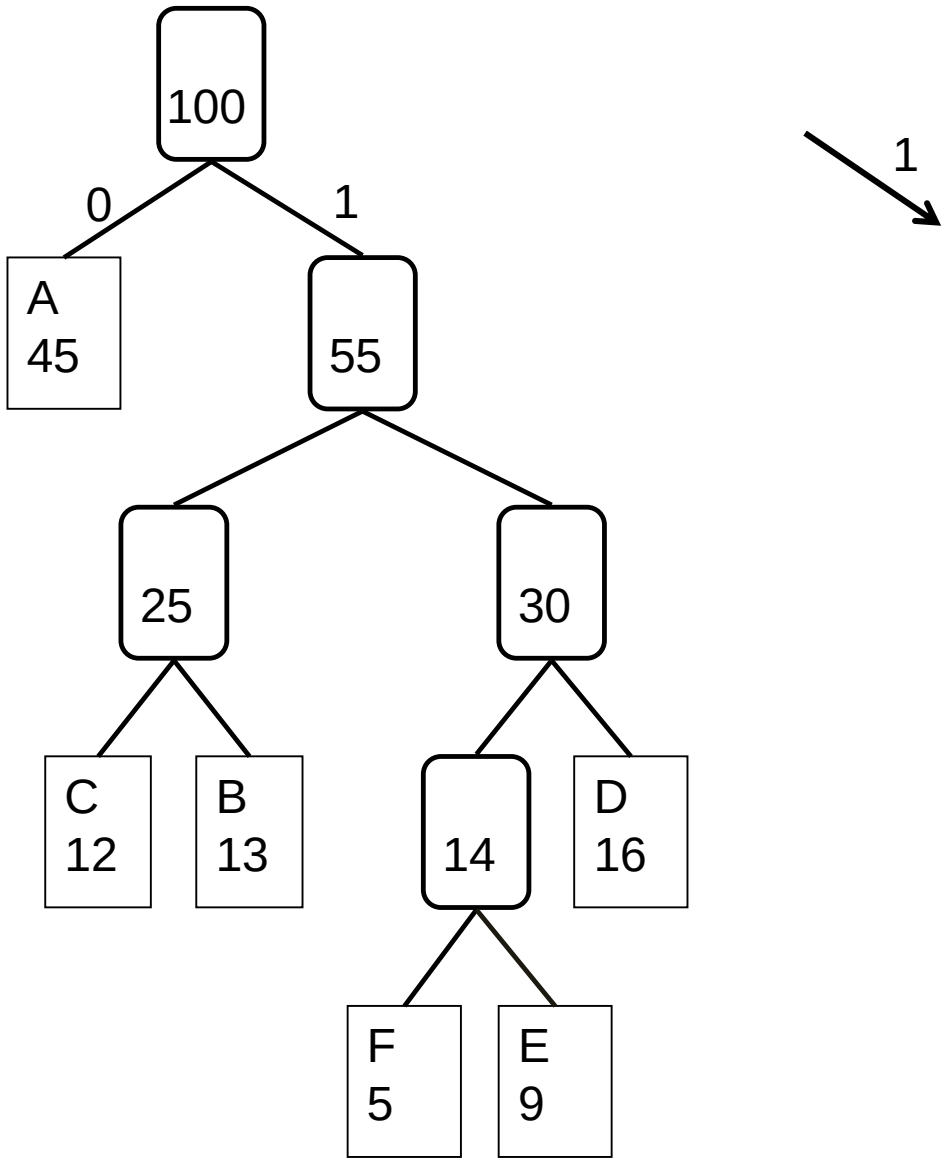
Voici l’arbre binaire construit par l’algorithme :



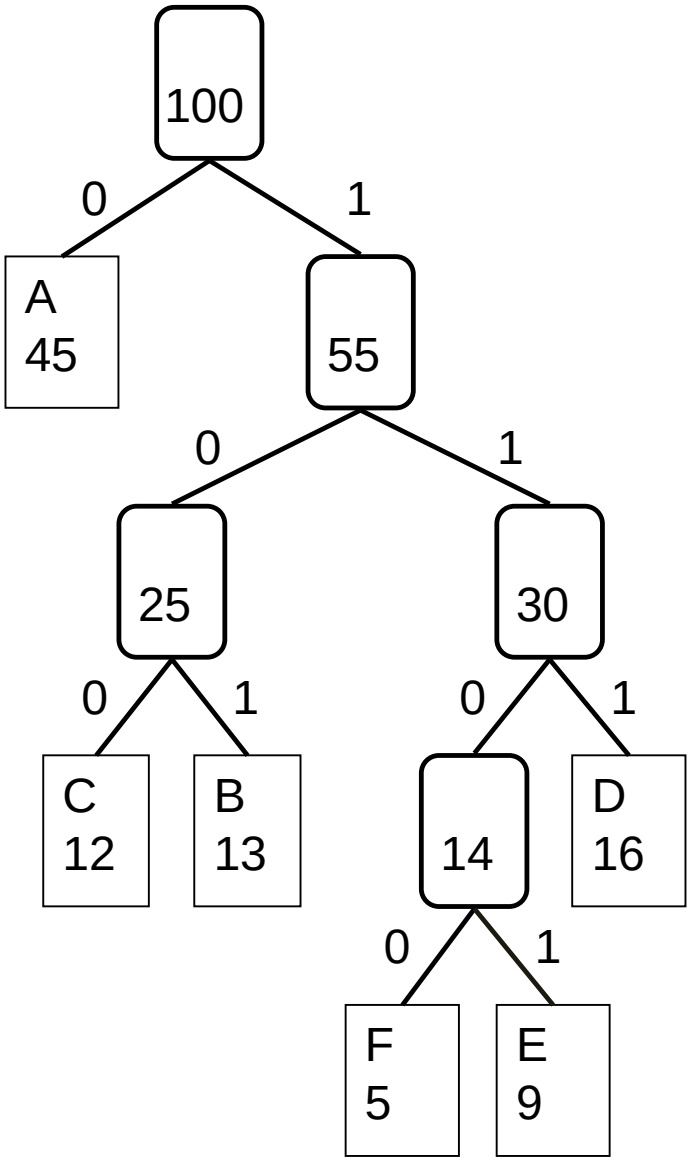
Voici l’arbre binaire construit par l’algorithme :



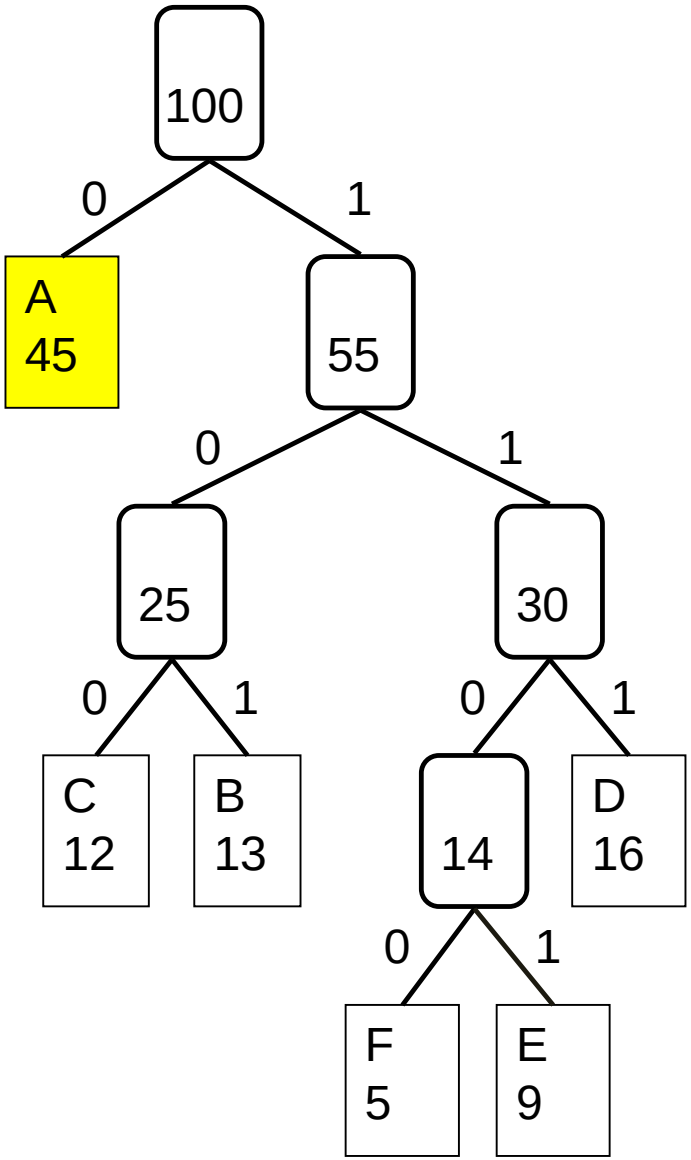
Voici l’arbre binaire construit par l’algorithme :



Voici l’arbre binaire construit par l’algorithme :

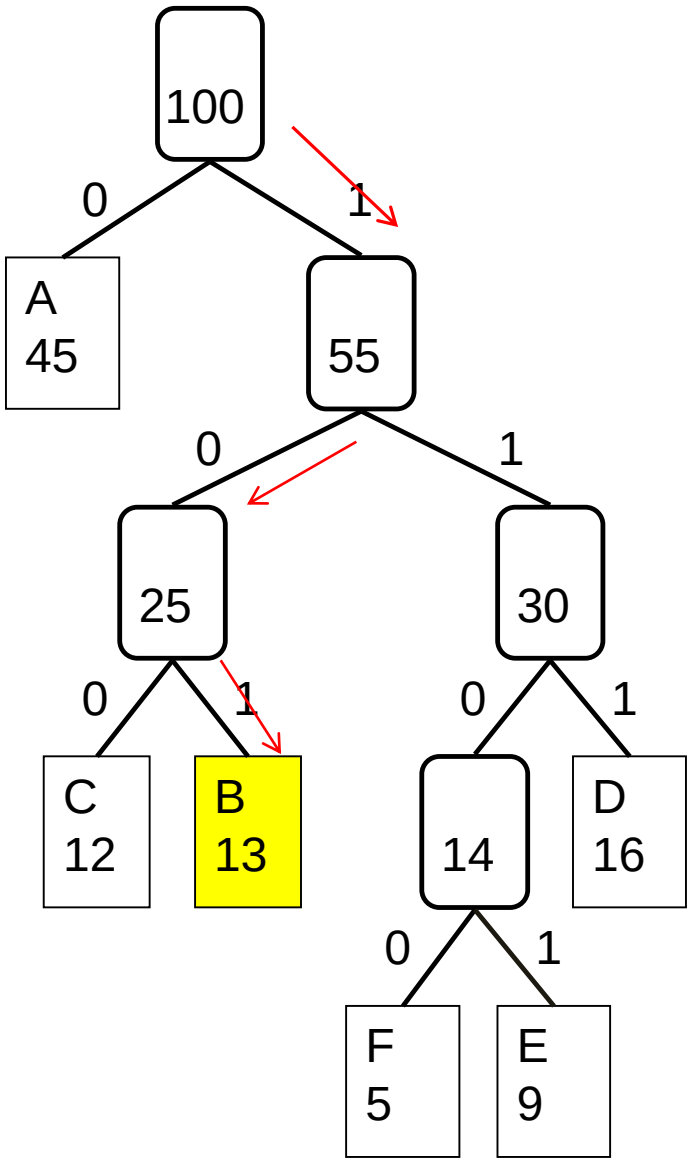


CODAGE :



A 0

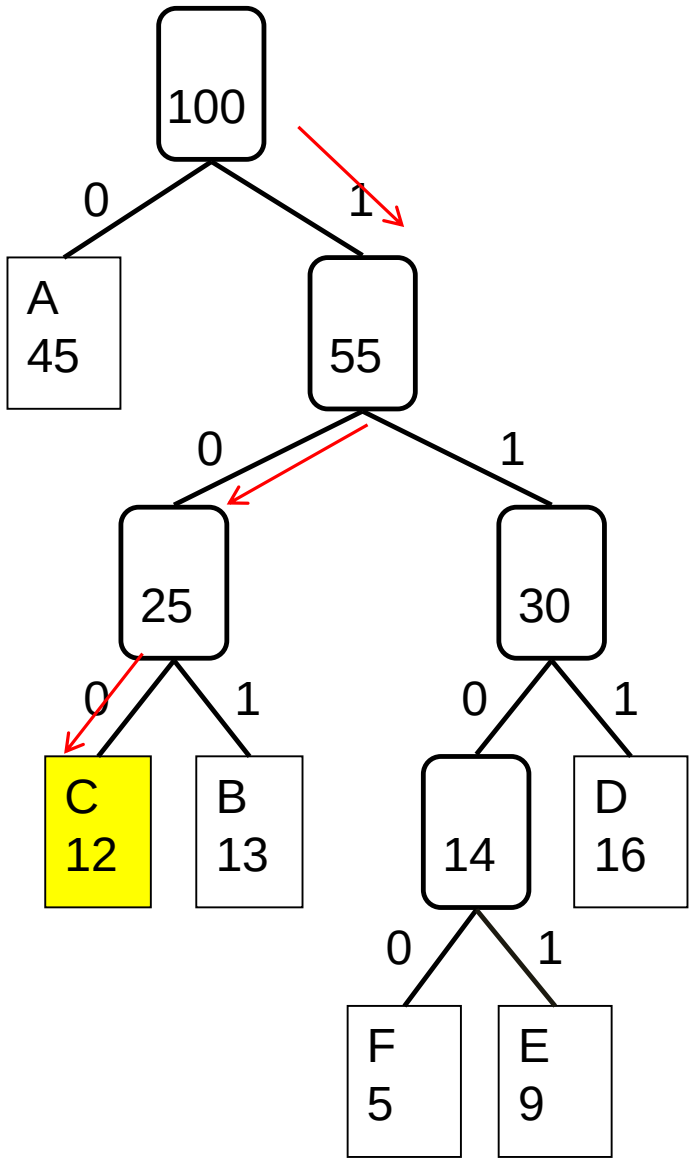
CODAGE :



A 0

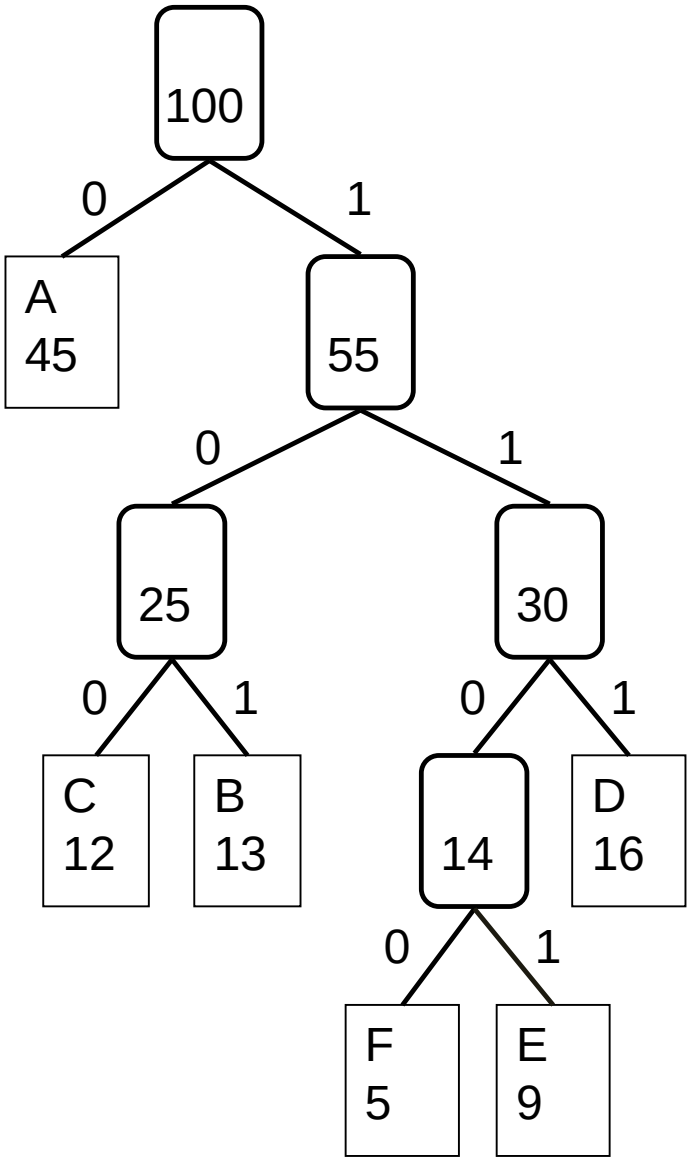
B 101

CODAGE :



A	0
B	101
C	100

CODAGE :



A	0
B	101
C	100
D	111
E	1101
F	1100

CODAGE :

A	0
B	101
C	100
D	111
E	1101
F	1100

CAFE

CODAGE :

A	0
B	101
C	100
D	111
E	1101
F	1100

CAFE

100

CODAGE :

A	0
B	101
C	100
D	111
E	1101
F	1100

CAFE

1000

CODAGE :

A	0
B	101
C	100
D	111
E	1101
F	1100

CAFE

10001100

CODAGE :

A	0
B	101
C	100
D	111
E	1101
F	1100

CAFE

10001100**1101**

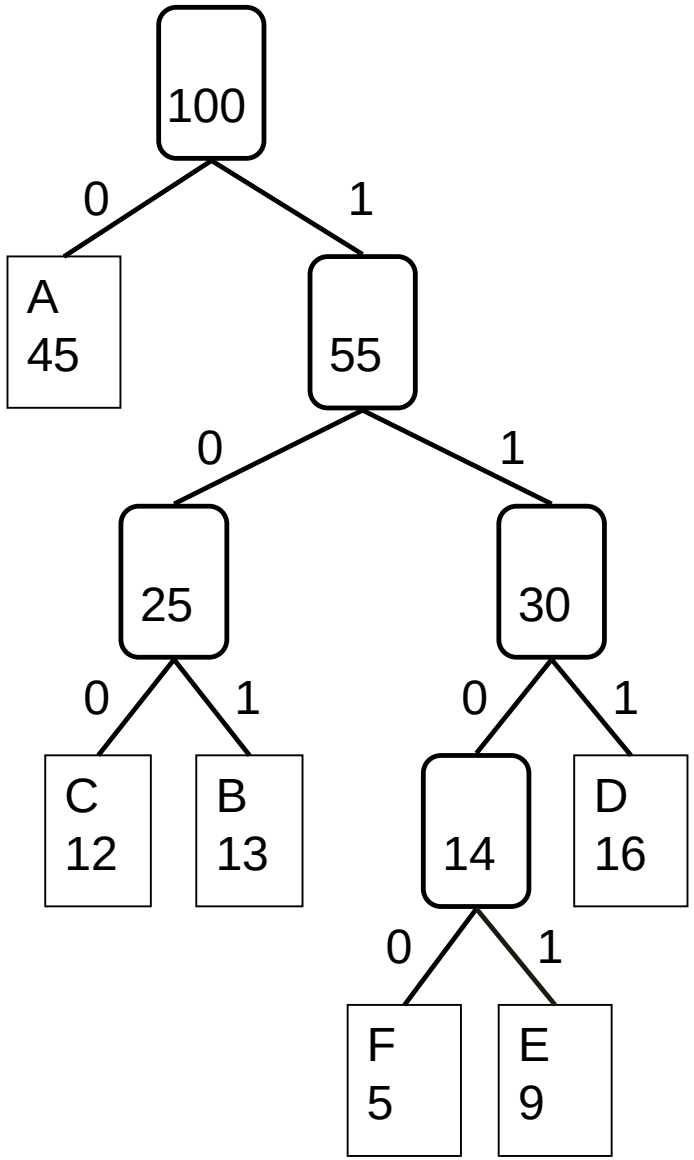
CODAGE :

A	0
B	101
C	100
D	111
E	1101
F	1100

CAFE

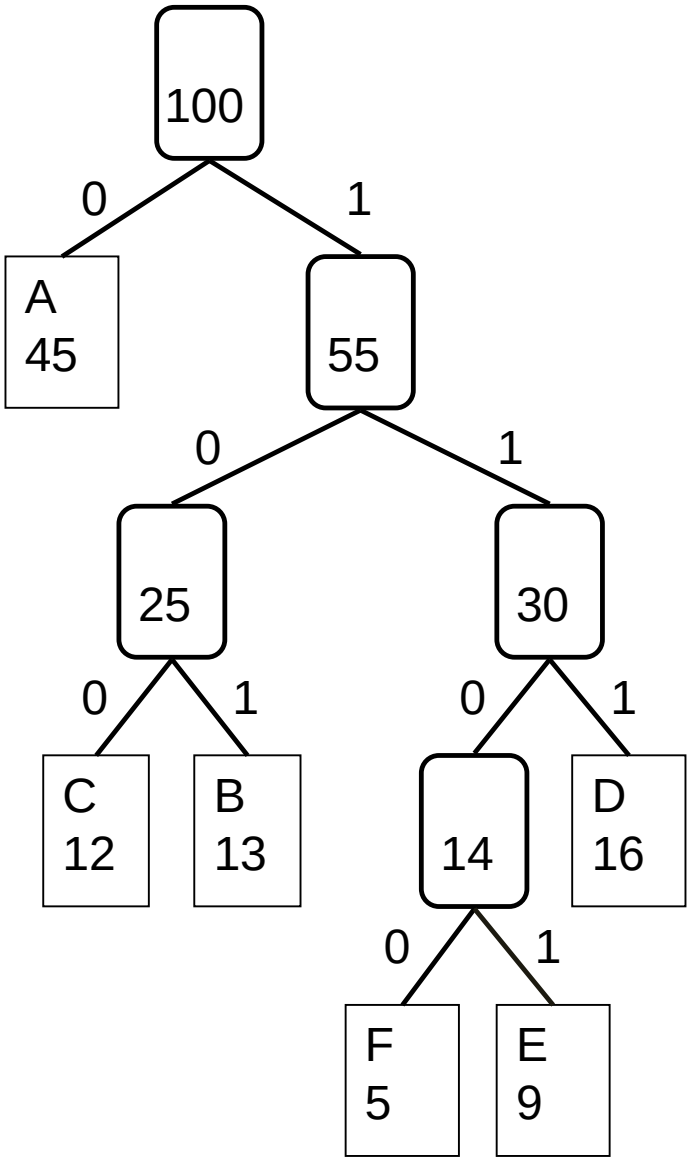
100011001101

DECODAGE :



A	0
B	101
C	100
D	111
E	1101
F	1100

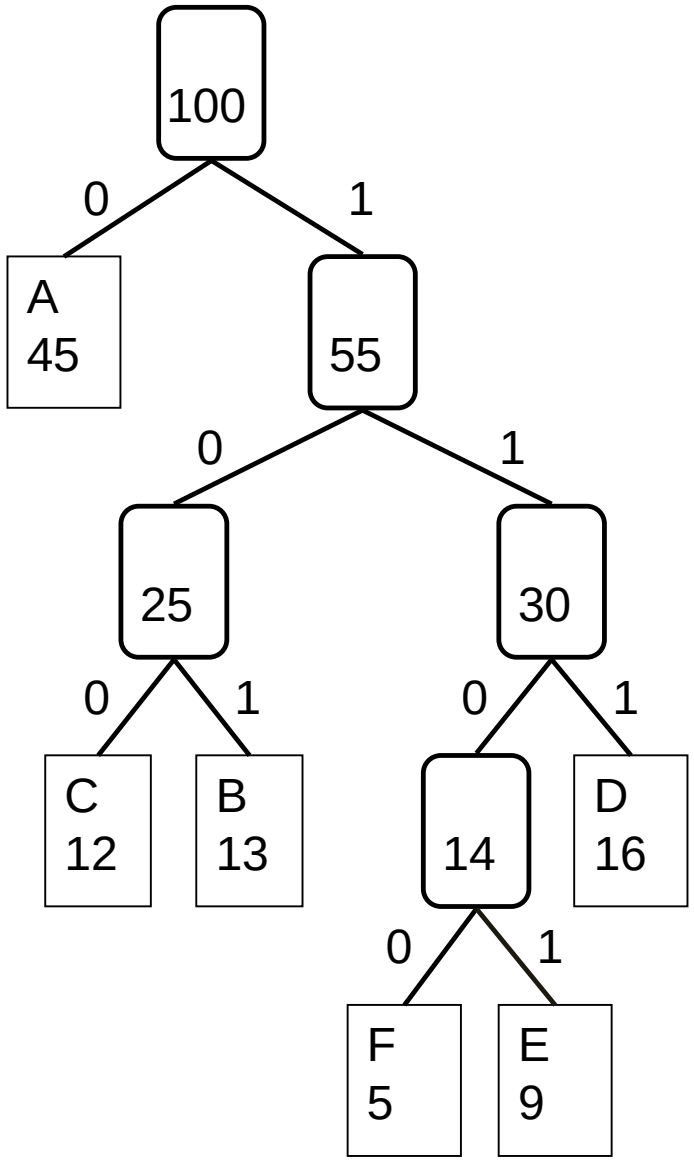
DECODAGE :



A	0
B	101
C	100
D	111
E	1101
F	1100

11111011000

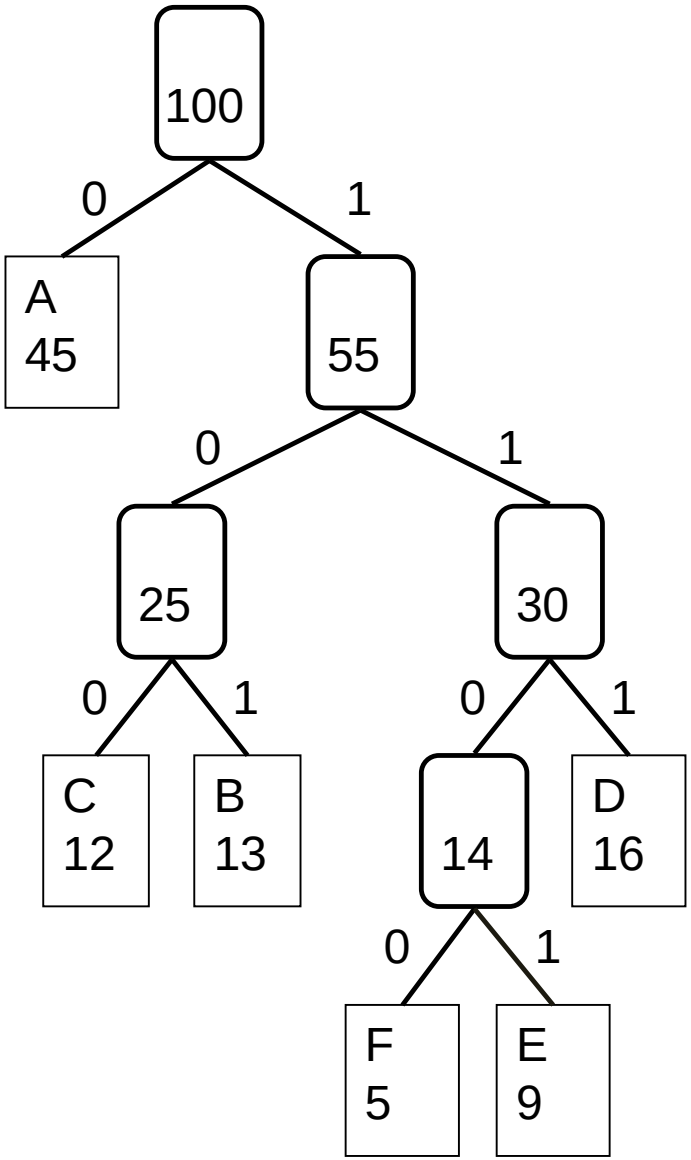
DECODAGE :



A	0
B	101
C	100
D	111
E	1101
F	1100

11111011000

DECODAGE :



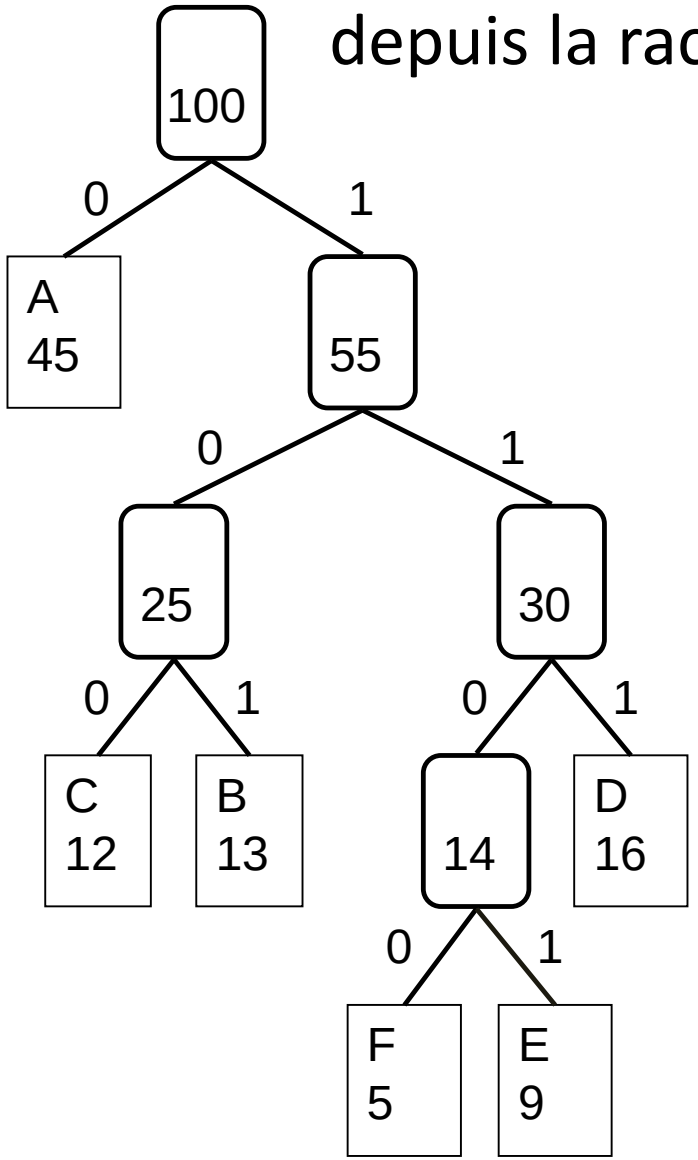
11111011000

DECODAGE :

Plusieurs parcours de l'arbre depuis la racine jusqu'une feuille

0 → à gauche

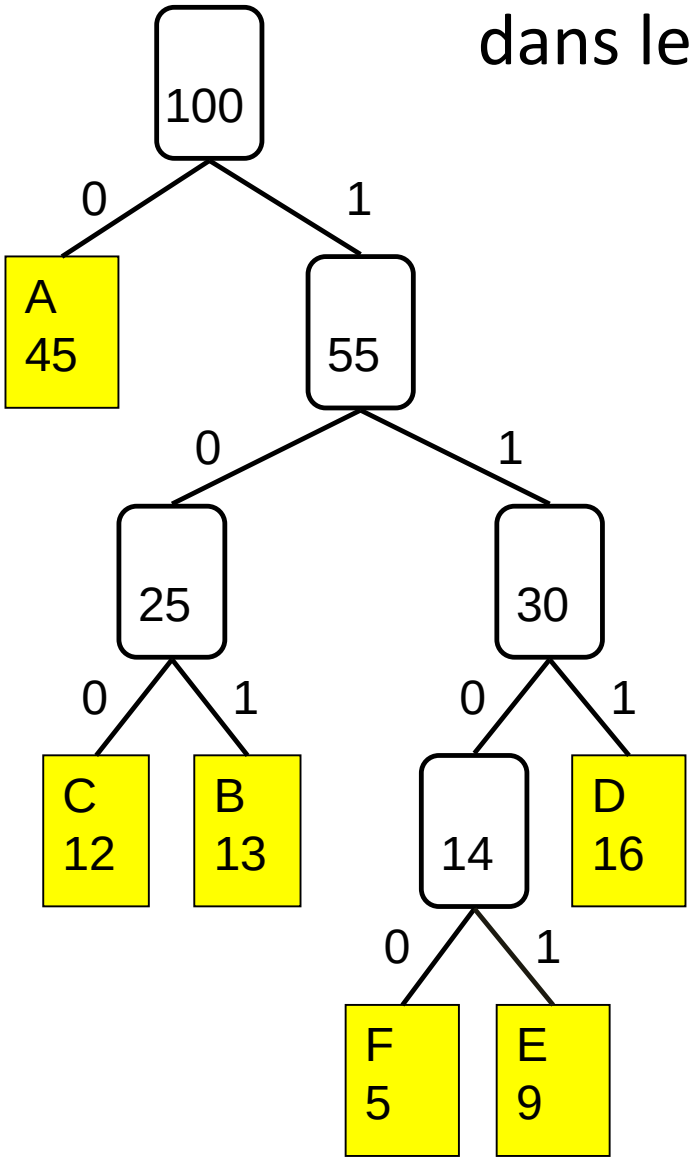
1 → à droite



11111011000

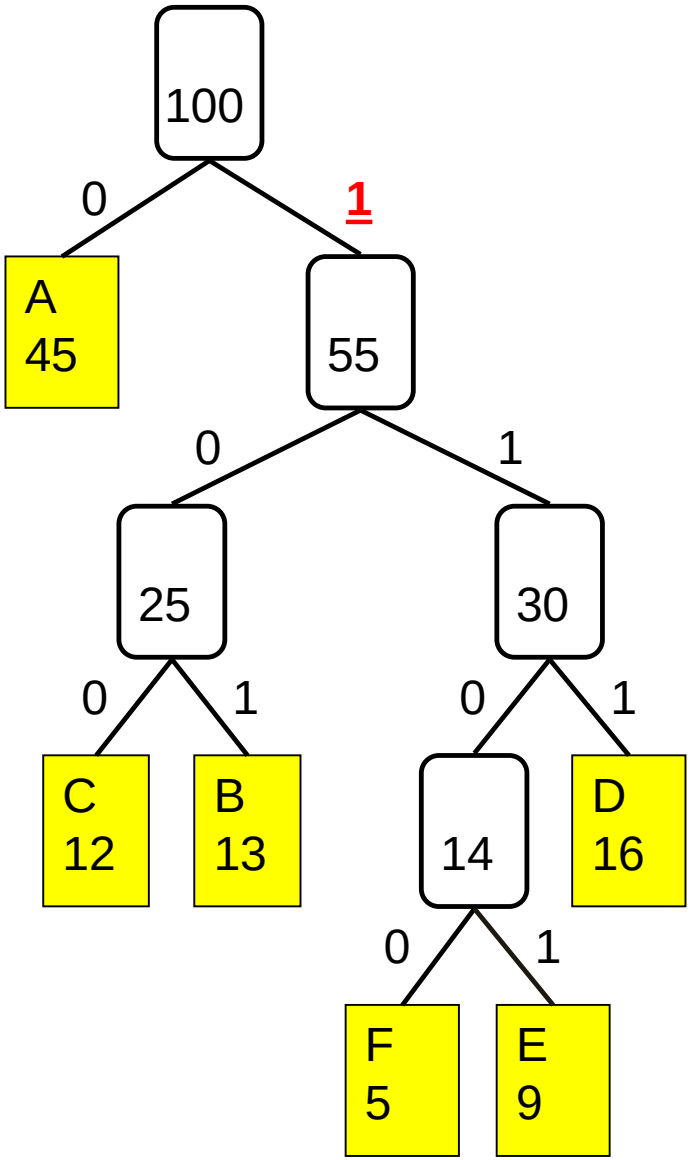
DECODAGE :

Les caractères se trouvent dans les feuilles!



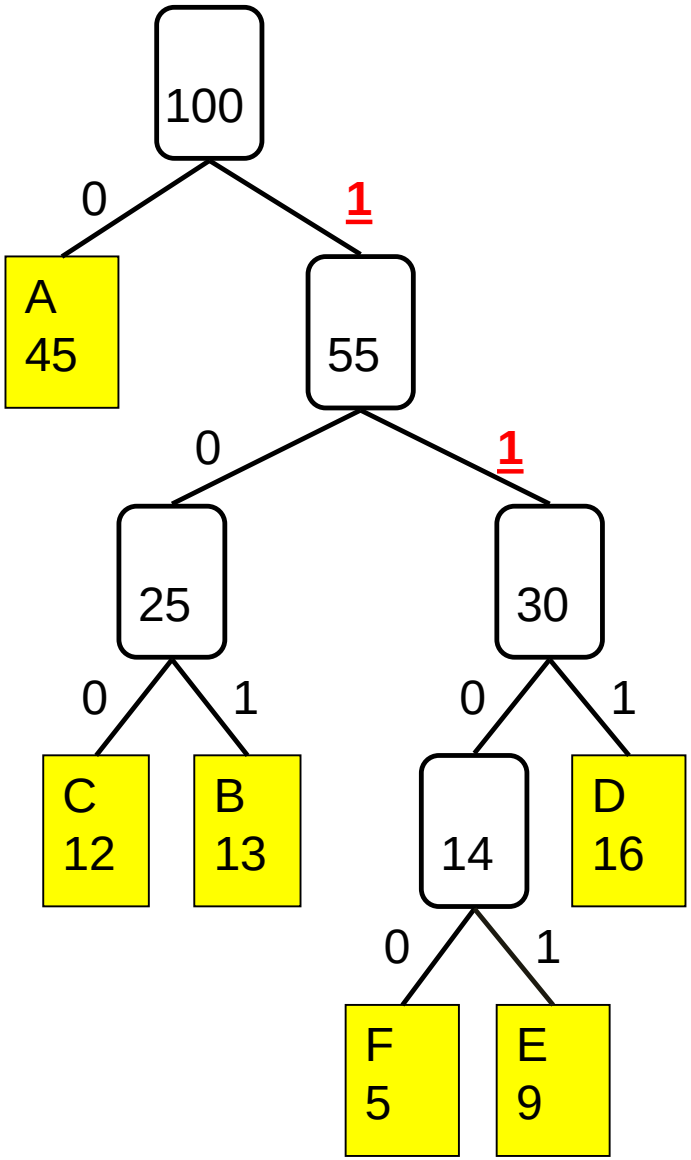
11111011000

DECODAGE :



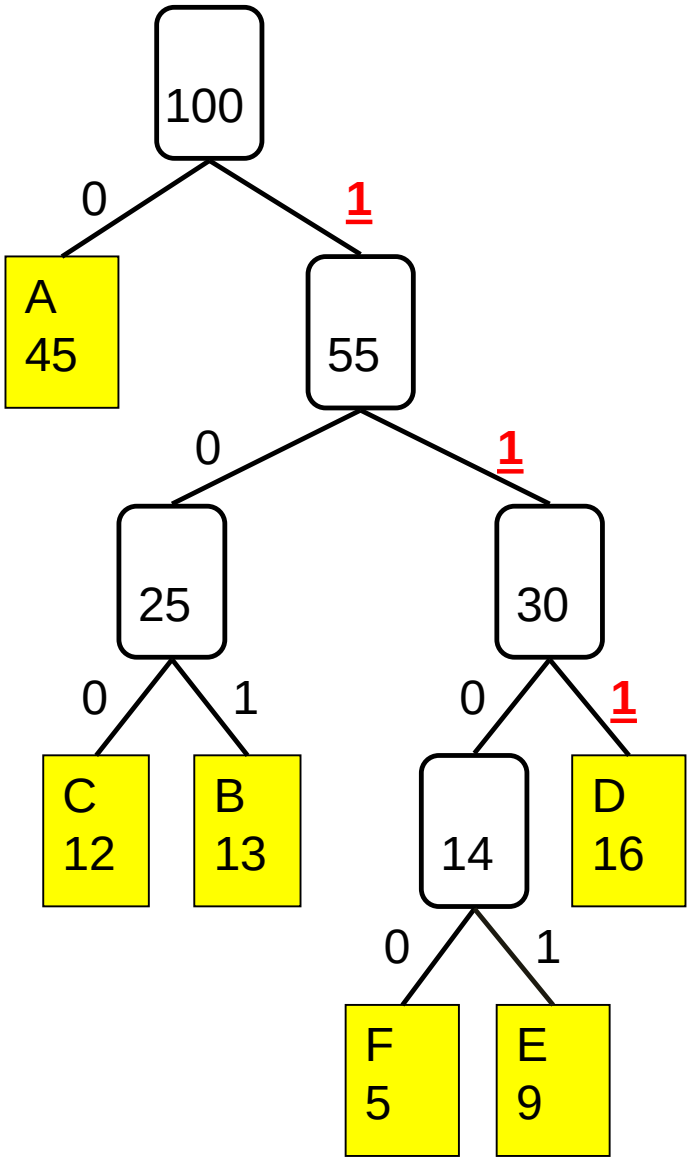
11111011000

DECODAGE :



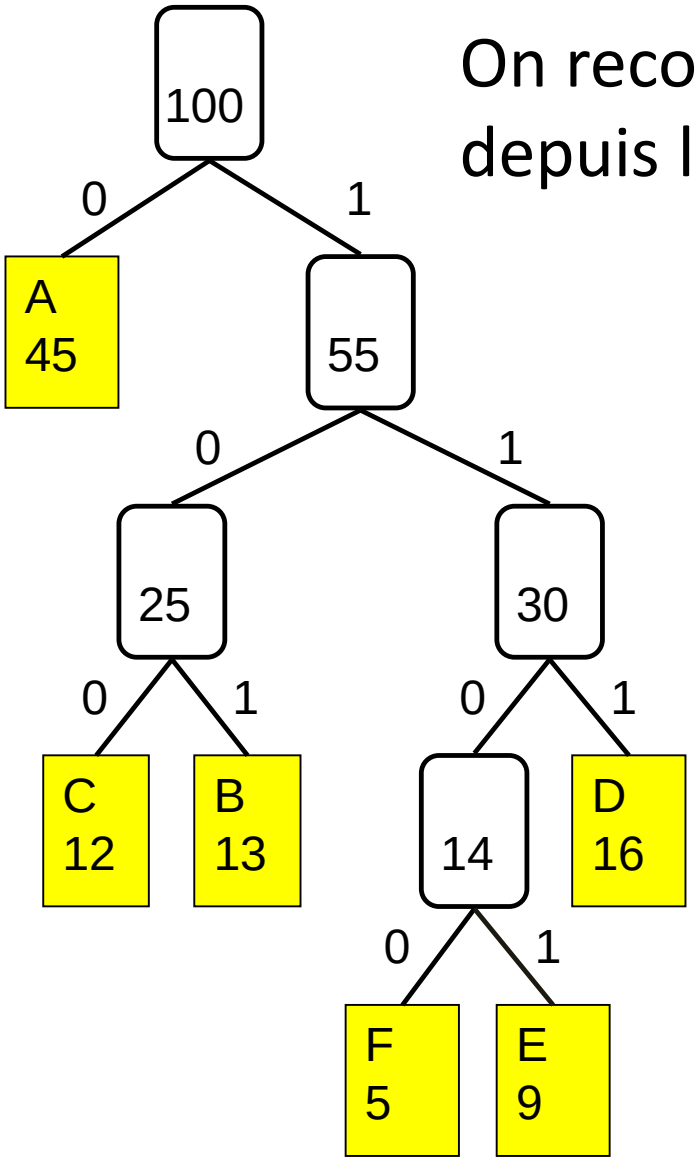
11111011000

DECODAGE :



111 11011000

DECODAGE :

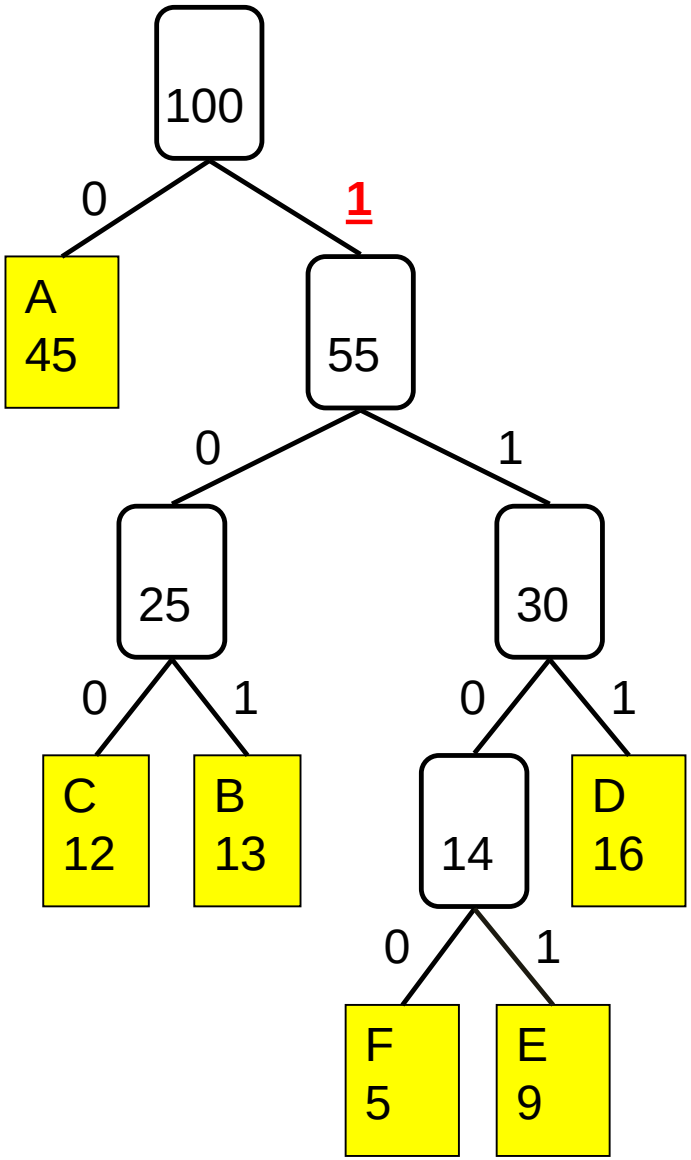


On recommence un parcours depuis la racine

111 11011000

D

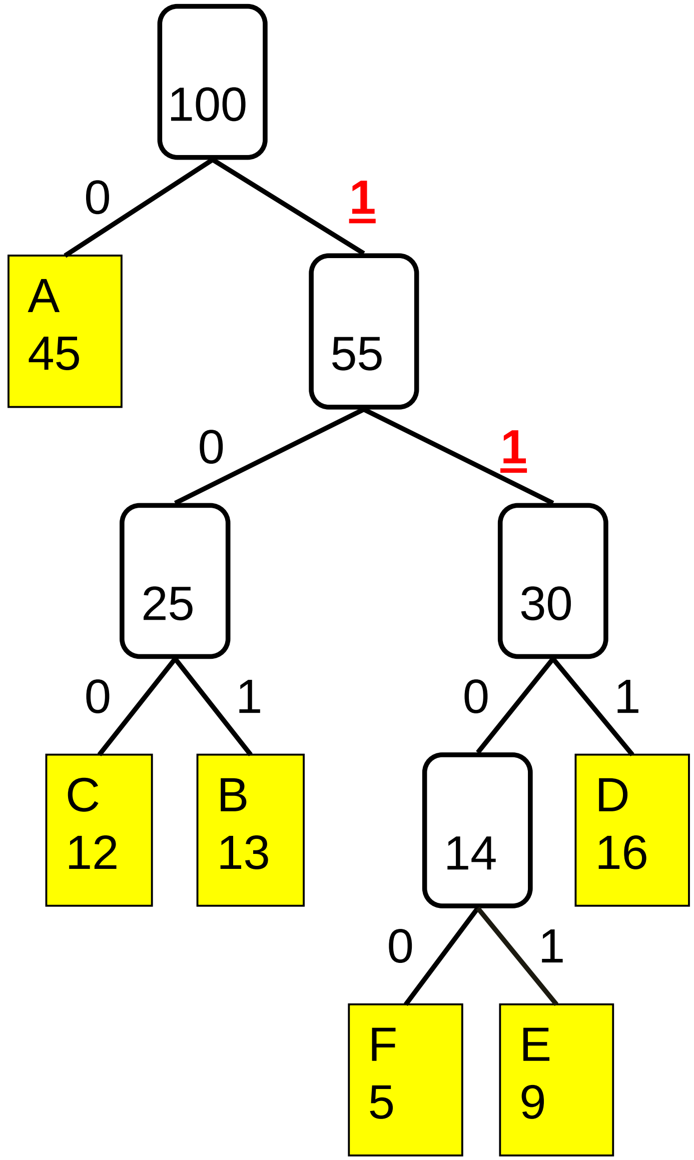
DECODAGE :



111110110000

D

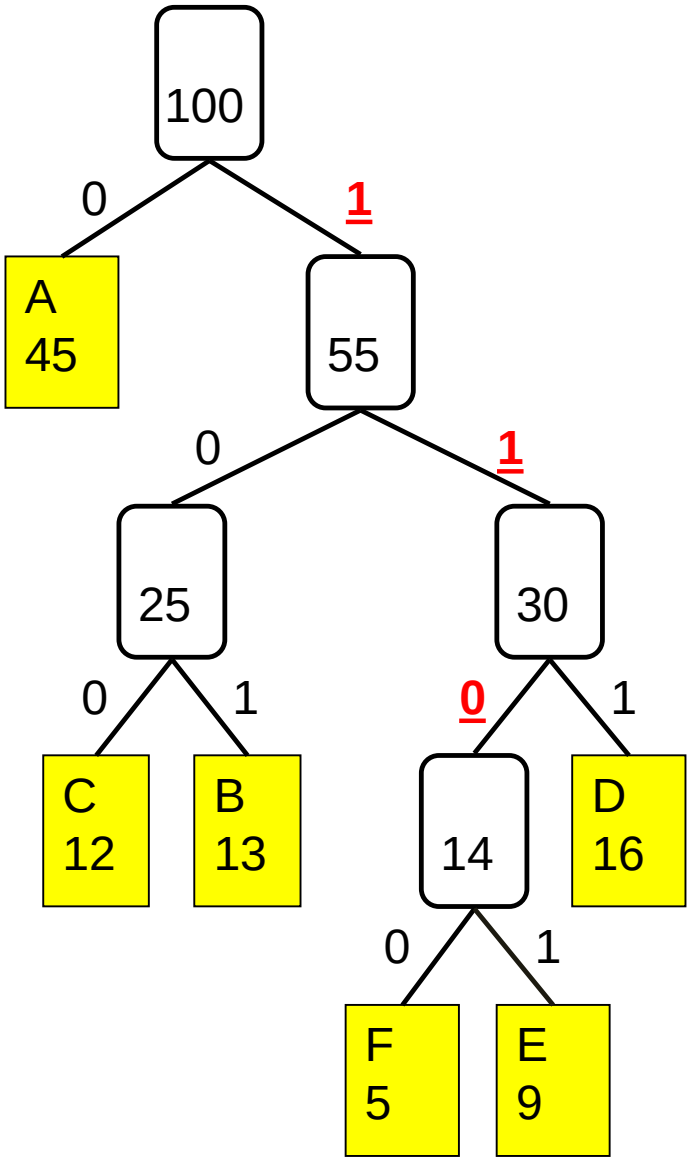
DECODAGE :



111<u>11</u>011000

D

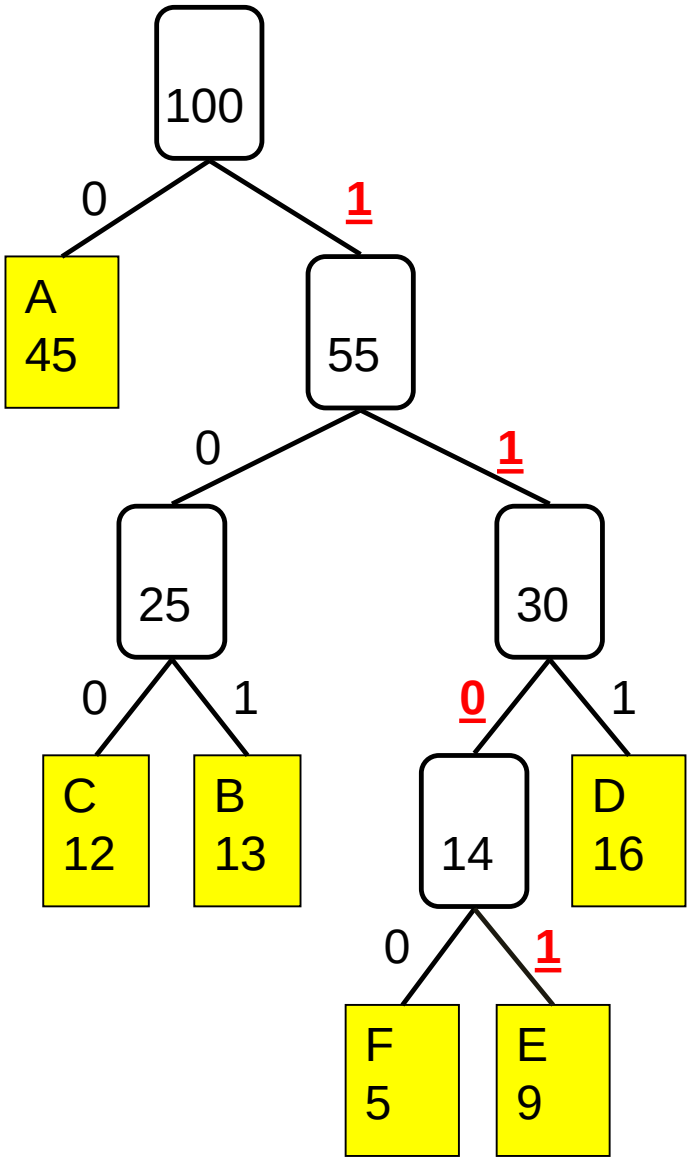
DECODAGE :



111110111000

D

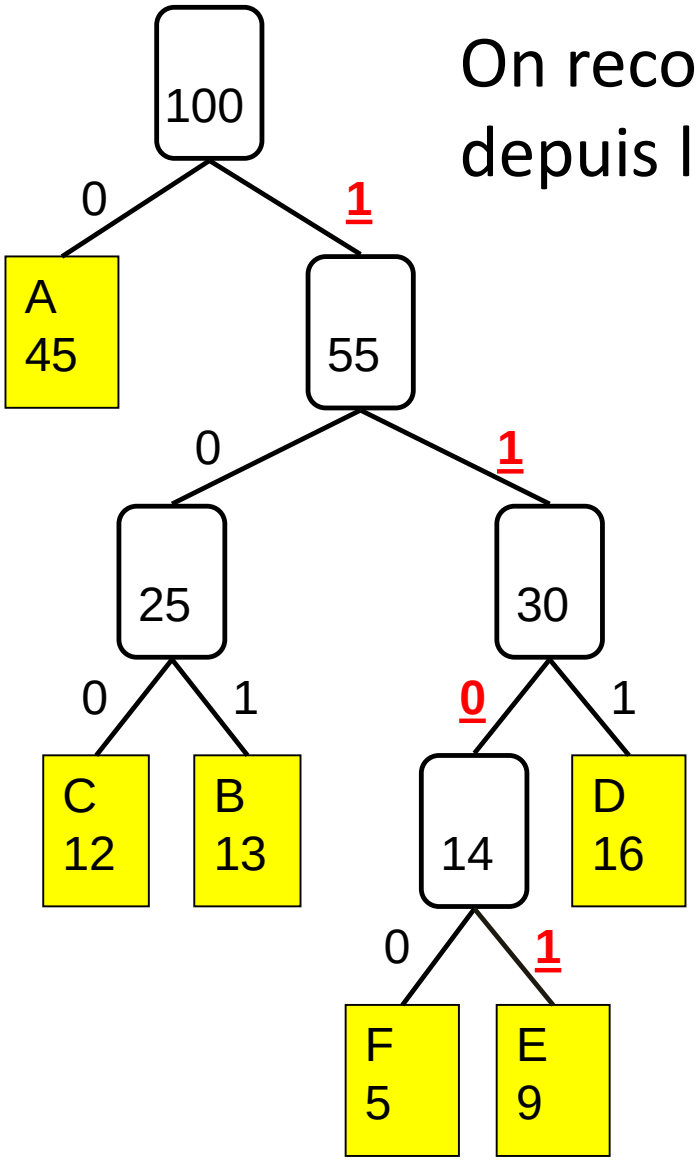
DECODAGE :



111**11101**1000

D

DECODAGE :

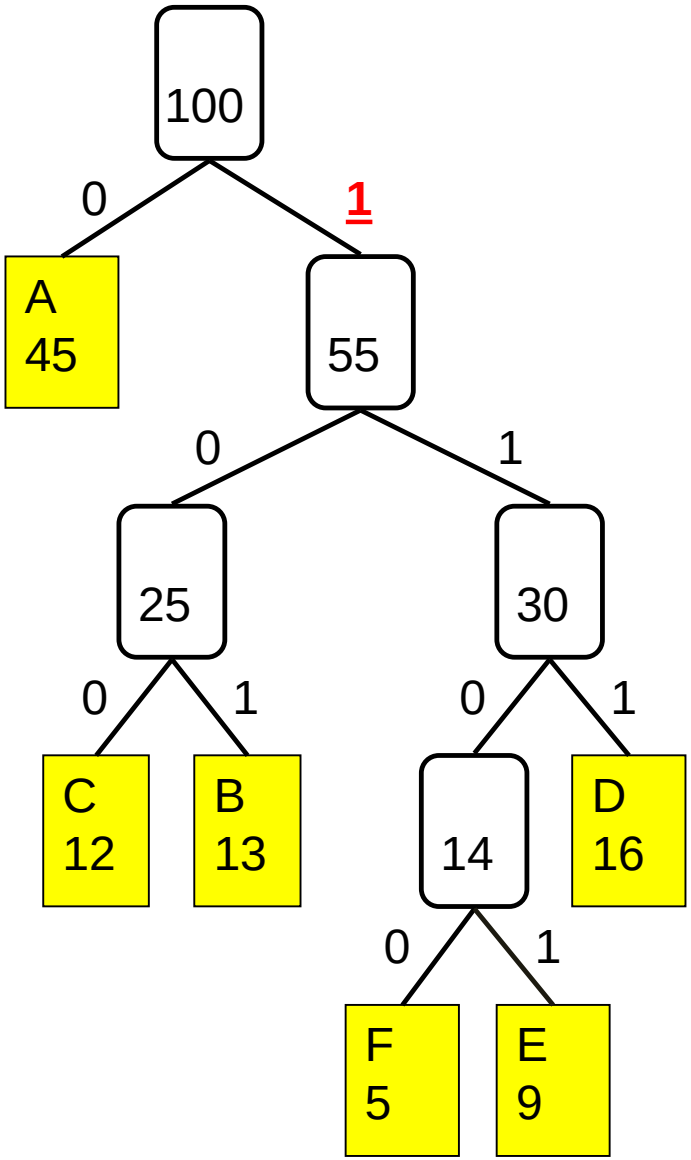


On recommence un parcours depuis la racine

111<u>1101</u>1000

D E

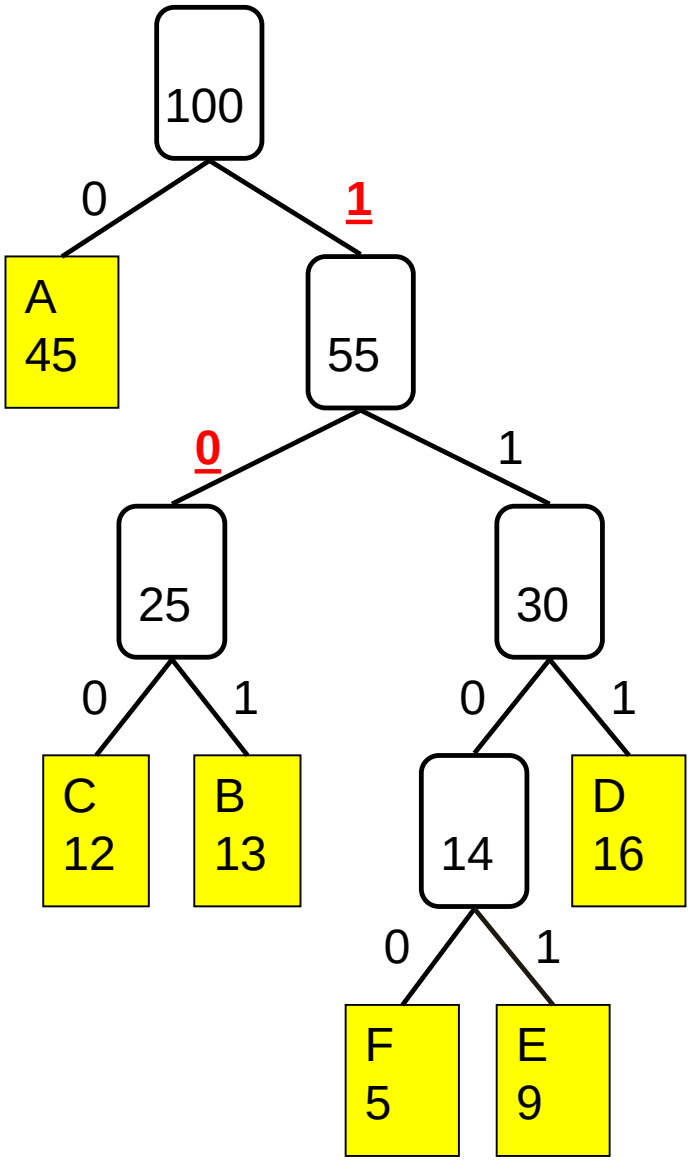
DECODAGE :



11111011000

D E

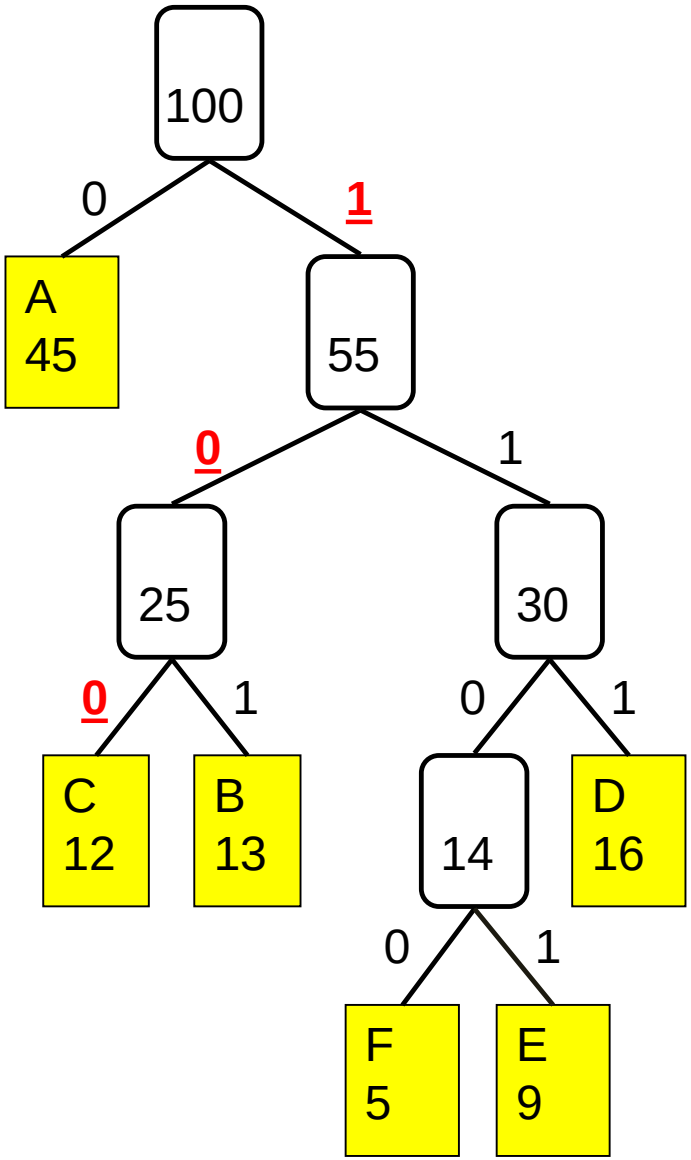
DECODAGE :



1111101**10**00

D E

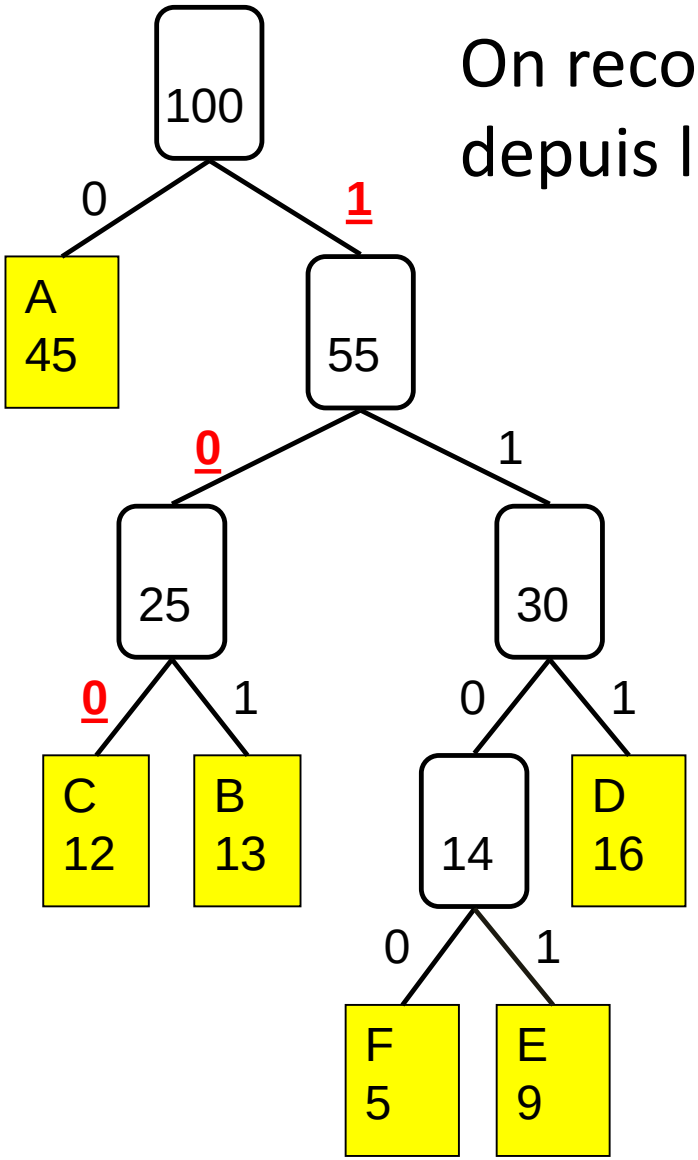
DECODAGE :



1111101<u>1000</u>

D E

DECODAGE :

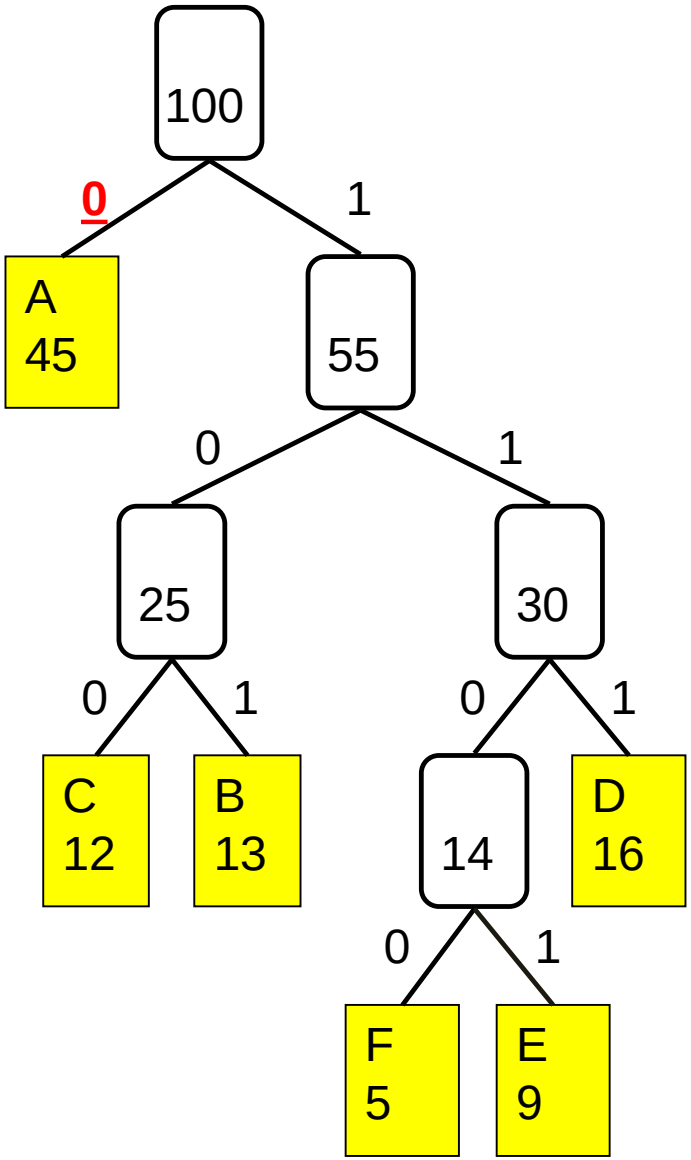


On recommence un parcours depuis la racine

1111101<u>1000</u>

D E C

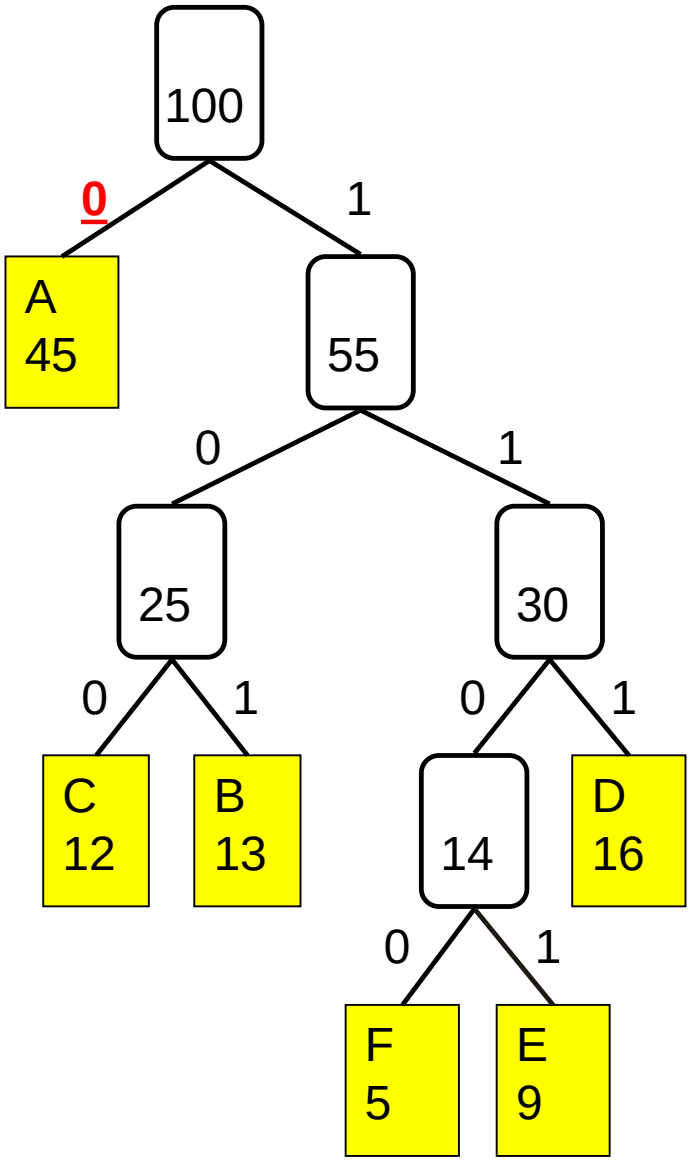
DECODAGE :



11111011000

D E C

DECODAGE :



11111011000

D E C A

ALGORITHME :

L'algorithme de Huffman utilise une file de priorité.

Cette file de priorité contient des arbres.

La priorité correspond à la fréquence.

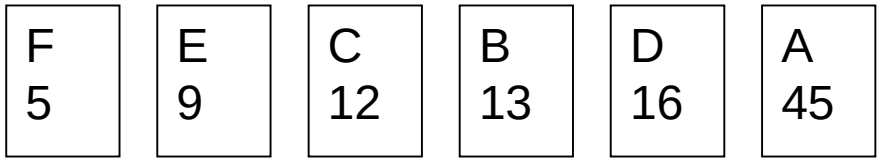
ALGORITHME :

L'algorithme de Huffman utilise une file de priorité.

Cette file de priorité contient des arbres.

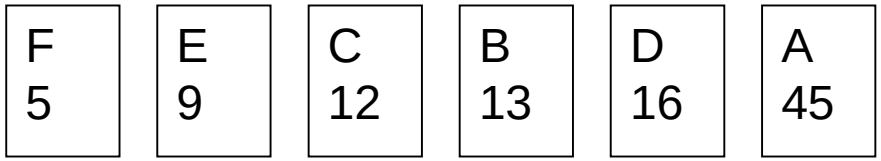
La priorité correspond à la fréquence.

Dans les figures suivantes, la file est triée pour faciliter la compréhension de l'algorithme!



Etape initiale

n insère()

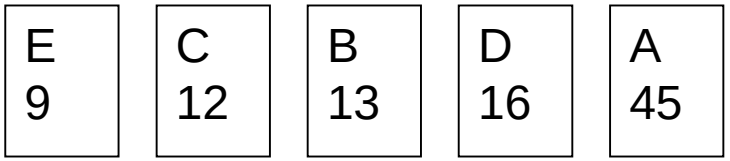


A chaque étape :

2 x supprimeMax()

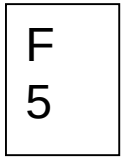
Σ fréquences

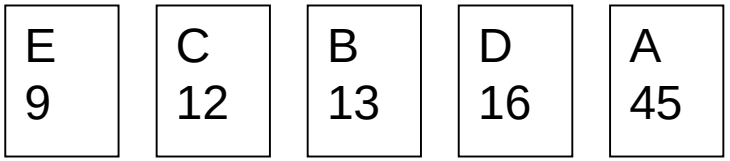
insère()



Etape 1

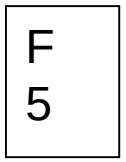
supprimeMax()

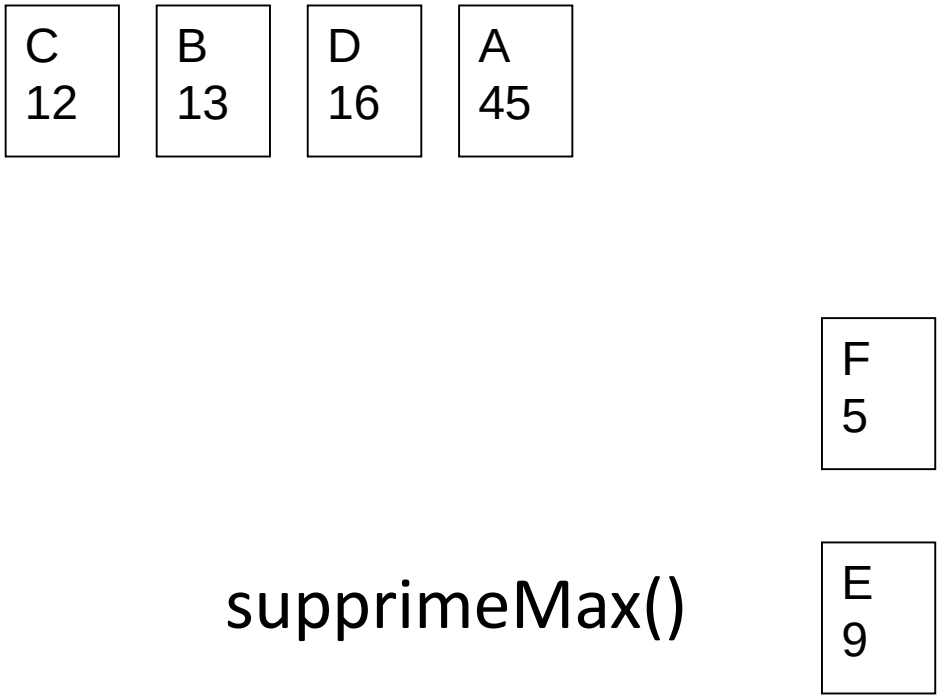


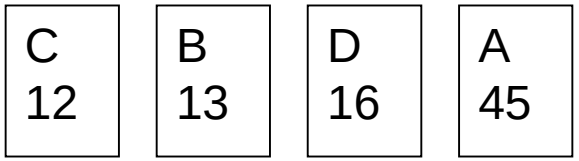


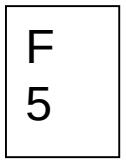
Etape 1

Fils gauche

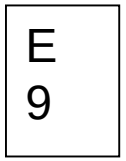


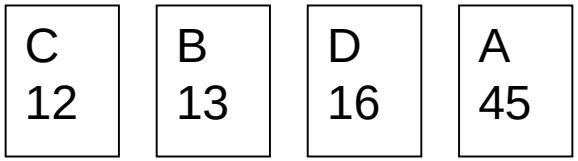

C
12
B
13
D
16
A
45
F
5
supprimeMax()
E
9

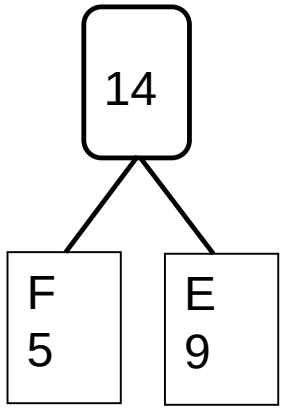




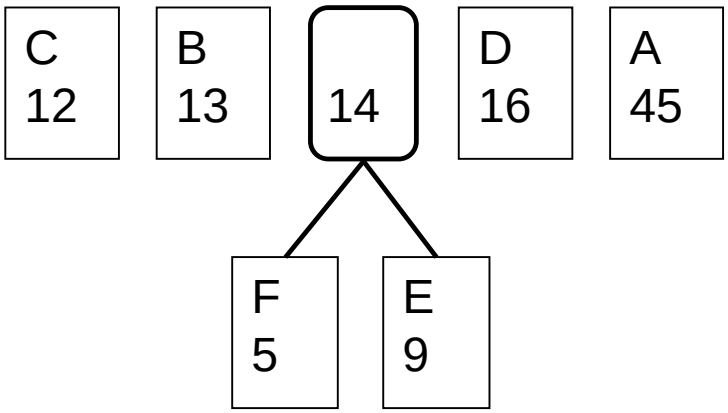
Fils droit



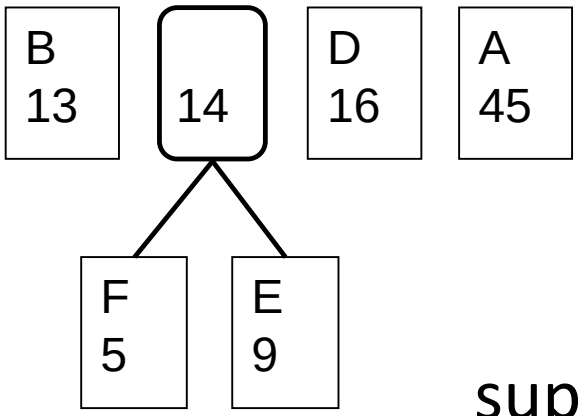




Σ des 2 fréquences
→ racine



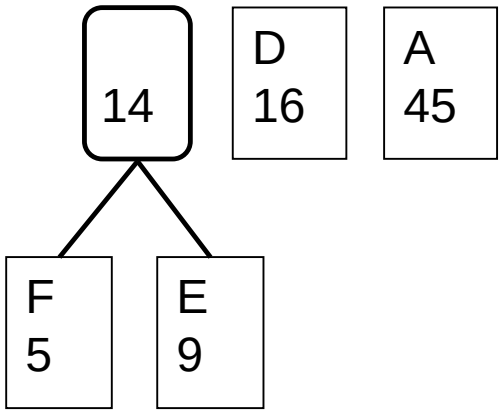
insère()



Etape 2

supprimeMax()

C
12



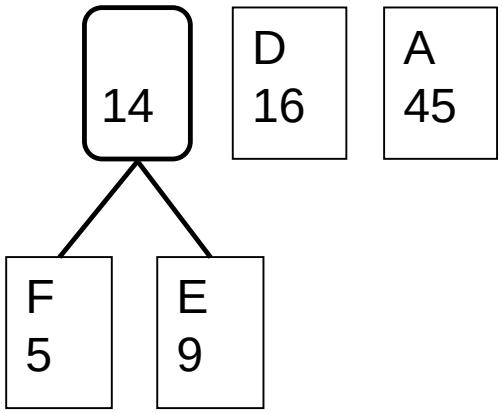
D
16

A
45

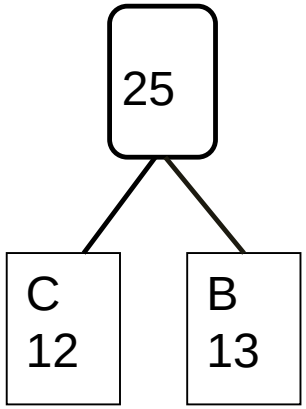
C
12

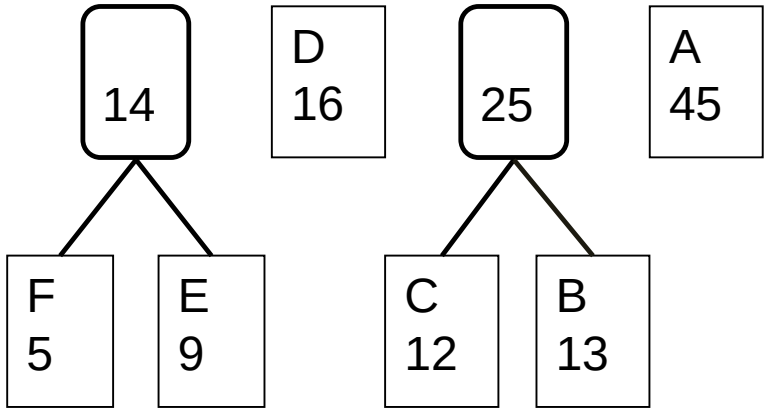
supprimeMax()

B
13

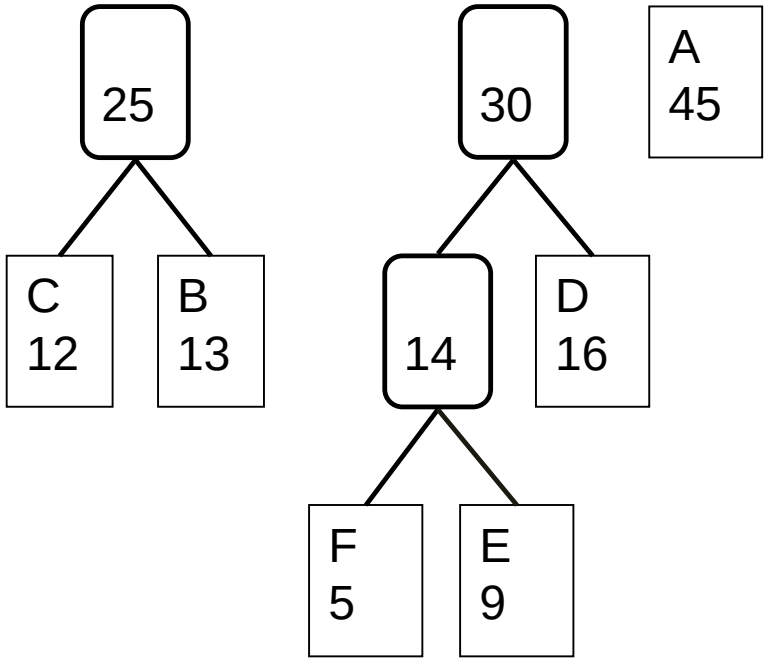


Σ des 2 fréquences

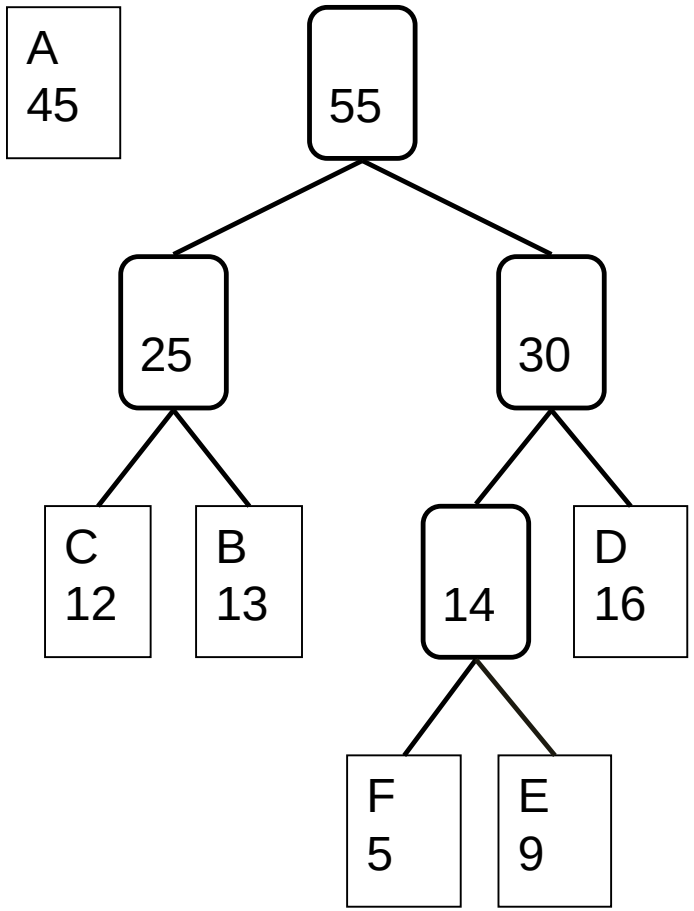




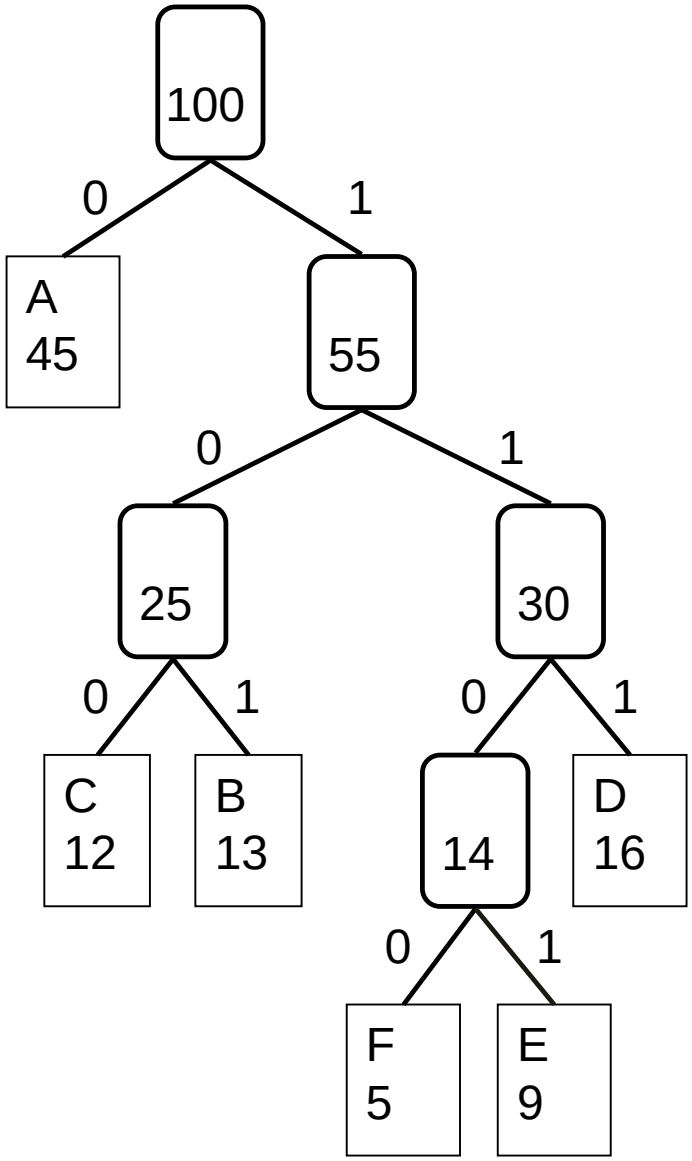
insère()



A la fin de
L'étape3



A la fin de
L'étape4



A la fin de L'étape5

Fin

PERFORMANCE :

L'utilisation d'une file de priorité donne un algorithme en $O(n*log(n))$.

En effet les insertions et suppressions y sont en $O(log(n))$.