Recherche d'un codage de Huffmann appliqué à une chaine de caractères

Points du programme officiel :

1) Représentation de données :

Contenus	Capacités attendues	Commentaires
Écriture d'un entier positif dans une base $b \geq 2$	Passer de la représentation d'une base dans une autre.	Les bases 2, 10 et 16 sont privilégiées.

Tableau indexé, tableau donné en compréhension	Lire et modifier les éléments d'un tableau grâce à leurs index. Construire un tableau par compréhension. Utiliser des tableaux de tableaux pour représenter des matrices : notation a [i] [j]. Itérer sur les éléments d'un tableau.	Seuls les tableaux dont les éléments sont du même type sont présentés. Aucune connaissance des tranches (*slices*) n'est exigible. L'aspect dynamique des tableaux de Python n'est pas évoqué. Python identifie listes et tableaux. Il n'est pas fait référence aux tableaux de la bibliothèque NumPy.
Dictionnaires par clés et valeurs	Construire une entrée de dictionnaire. Itérer sur les éléments d'un	Il est possible de présenter les données EXIF d'une image sous la forme d'un enregistrement. En Python, les p-uplets nommés

2) Langage et programmation :

Contenus	Capacités attendues	Commentaires
Constructions élémentaires	Mettre en évidence un corpus de constructions élémentaires.	Séquences, affectation, conditionnelles, boucles bornées, boucles non bornées, appels de fonction.

3) Algorithmique :

Contenus	Capacités attendues	Commentaires
Parcours séquentiel d'un tableau	Écrire un algorithme de recherche d'une occurrence sur des valeurs de type quelconque. Écrire un algorithme de recherche d'un extrémum, de calcul d'une moyenne.	On montre que le coût est linéaire.
Tris par insertion, par sélection	Écrire un algorithme de tri. Décrire un invariant de boucle qui prouve la correction des tris par	La terminaison de ces algorithmes est à justifier. On montre que leur coût est

Prérequis : Maîtrise de la représentation binaire
Constructions élémentaires : affectation, condition, boucles
Listes et tableaux

Tri et recherche

II. Algorithmes de Tri

1. Introduction.

- Les ordinateurs stockent et analyse de gros volumes de données
- L'analyse de ces données doit être rapide et efficace.
- Les moteurs de recherche doivent balayer de « big data » en un temps record.
- La recherche d'une information (un mot, un code-barres, image, etc.) se fait par de des « clés de recherche ».
  - ✓ Recherche *séquentiel* : méthode trop lente
  - ✓ Recherche *dichotomique : méthode* plus rapide
  - ✓ …

Une « figure »

Philippe Flajolet(1948-2011)

Un des pionniers de l'analyse de la complexité des algorithmes. Il a montré l'utilité, dans ce domaine, de plusieurs théories mathématiques : la combinatoire et le dénombrement, la théorie des probabilités et la théorie des fonctions d'une variable complexe.

Il a aussi eu conscience très tôt de l'apport des logiciels de calcul formel pour effectuer les calculs, parfois fastidieux, que cette analyse demande

Activité

Description de la méthode du tri
Taches élémentaires répétitives
Nombre d'opération effectuer ?

Département	Population	Dept
(1) Nord	2 579 208	
(2) Paris	2 249 975	
(3) Bouches-du-Rhône	1 975 896	
(4) Rhône	1 744 236	
(5) Hauts-de-Seine	1 581 628	
(6) Seine-Saint-Denis	1 529 928	
(7) Gironde	1 463 662	
(8) Pas-de-Calais	1 462 807	
(9) Yvelines	1 413 635	
(10) Seine-et-Marne	1 338 427	
(11) Val-de-Marne	1 333 702	
(12) Loire-Atlantique	1 296 364	
(13) Haute-Garonne	1 260 226	
(14) Seine-Maritime	1 251 282	
(15) Essonne	1 225 191	
(16) Isère	1 215 212	
(17) Val-d'Oise	1 180 365	
(18) Bas-Rhin	1 099 269	
(19) Alpes-Maritimes	1 081 244	
(20) Hérault	1 062 036	

2. Méthode de tris

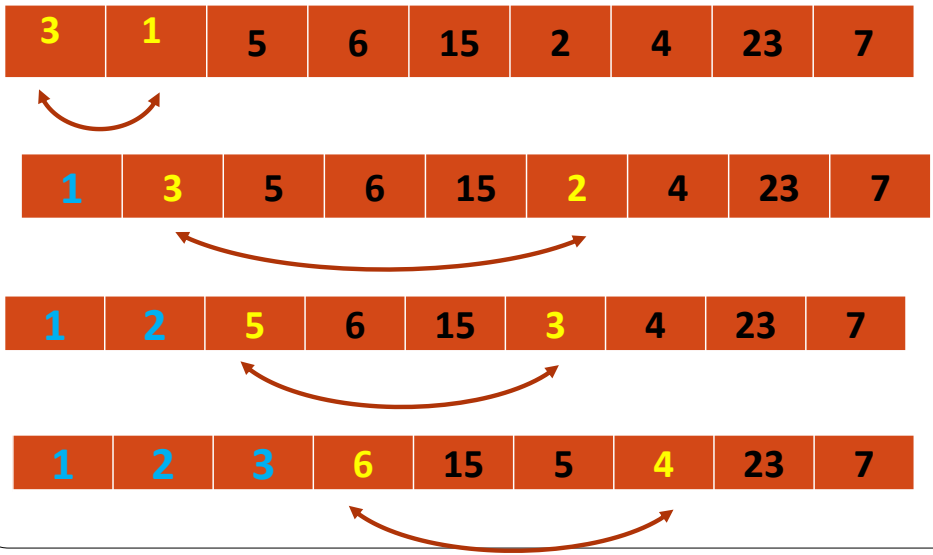
a. **Tri par sélection.**

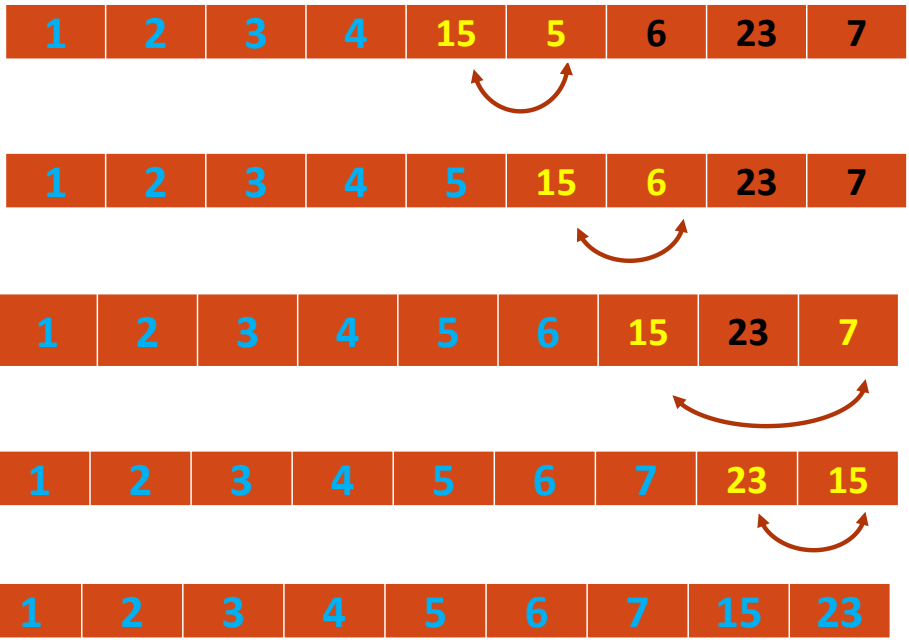
- **Le principe :**
  - On cherche le plus petit
  - On l'échange avec le premier élément
  - On cherche le plus petit élément parmi les éléments non triés
  - On l'échange avec le deuxième élément ;

Pour résumer : On compare l'élément en cour avec les suivants jusqu'à trouver un candidat pour la pour sa palace.

Remarque : On peut se chercher le plus grand élément pour le placer à la fin du tableau.

Exemple : A chaque étape : On compare l'élément en cour avec les suivants jusqu'à trouver un candidat pour la pour sa palace.




1 2 3 4 15 5 6 23 7
1 2 3 4 5 15 6 23 7
1 2 3 4 5 6 15 23 7
1 2 3 4 5 6 7 23 15
1 2 3 4 5 6 7 15 23

Mise en place d'un algorithme

- **Les deux taches répétitives utilisées dans cet algorithme ?**
  - **Comparer**
  - **Echanger**
- **L'algorithme ressemblera à :**

DEBUT

POUR i = 1 JUSQU'A N-1 FAIRE

min ← i // On initialise l'indice du minimum

Comparer

echanger

Fin pour

Fin

```
algorithme TRI_SELECTION
VARIABLE
        min, i, j, taille sont des  entiers;
        mon_tab est un tableau[1..taille ] d'entiers;
DEBUT  POUR i = 1 JUSQU'A N-1 FAIRE
   min ← i
   POUR j=i+1 JUSQU'A N FAIRE
        SI  T[j] < T[min] ALORS
                min ← j
        FINSI
   FINPOUR
echanger(Ton_tab[],i,min) ;
FINPOUR
FIN
```

La fonction echanger

FONCTION echanger(Tab[],i,j)

VARIABLES temp : nombre ;

DEBUT

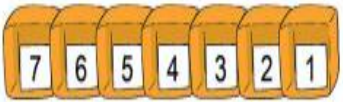
temp ← Tab[j];

Tab[j] ← T[i];

Tab[i] ← temp

FIN.

- **Exercice**
- Effectuer à la main un tri par sélection des tableaux :



- Combien de place mémoire a-t-on besoin pour trier ce tableau ?
- Déterminer le nombre de comparaisons nécessaires pour trier ce tableau.
- Traduire cet algorithme en Python, puis le tester.

Complexité d'un algorithme

- Un algorithme répond à un problème.
- Il est composé d'un ensemble d'étapes nécessaires à la résolution.
- Le nombre de ces étapes varie en fonction du nombre d'éléments à traiter.
- Une réponse à un problème peut être donnée très efficacement ou au contraire être inacceptable. Elle se fonde sur :
  - La complexité en temps : une estimation des temps de calcul
  - La complexité en espace : une estimation des besoins en mémoire
- L'algorithme TRI-SELECTION est
  - Econome en mémoire : Il trie les éléments du tableau dans le tableau lui-même
  - Non économe en temps : en effet, il utilise deux boucles ***for*** imbriqués

- La première boucle doit être exécutée N-1 fois et elle comprend, entre autres, la deuxième boucle ***for*** qui effectue *N-1* comparaisons, à sa première exécution, puis *N-2*, *N-3*, … etc. Ainsi,
- Le nombre total de comparaisons pour chaque valeur de j : (N-j-1) soit au total :

$$(N-1)+(N-2)+\cdots+1=\frac{N(N-1)}{2}=\frac{N^2-N}{2}$$

comparaisons : On dira qu'il est d'ordre de N^2

Autre méthodes : Tri Par Insertion

Principe : Le tri par insertion ou l'algorithme du joueur de carte :

- On parcourt la liste en plaçant les valeurs à leurs places au fur et à mesure.
- En bleu la valeur à insérer et en souligné les valeurs à décaler

3	**1**	5	6	8	2	4	9	7
1	3	5	6	8	**2**	4	9	7
1	2	3	5	6	8	4	9	7
1	2	3	4	5	6	8	9	**7**

…

Exercice

Trier la liste suivante par insertion :

[61, 1, 25, 33, 70, 20, 34, 19, 68, 79, 64, 24]

Combien de comparaisons a-t-on effectué ?

L'algorithme Tri par insertion

```
TriInsertion(liste,n)
DEBUT
DONNÉES

VARIABLES
   i,j: entier
DEBUT
    i <- 1 # erreur remplacer 2 par 1;
    TantQue (i ≤ n) FAIRE
       j <- i
       TantQue (j >=  1) ET (liste[j - 1] > liste[j])
FAIRE
          echanger(liste, j, j - 1)
          j <- j - 1
       FinTQ
       i <-  i + 1
   FinTQ
FIN
```

Traduire en python cet algorithme

Tri par fusion

Fusion de deux paquets de cartes triées

- On pose devant soi deux paquets de cartes triées, la plus petite carte au-dessus.
- Pour fusionner les deux paquets en un seul paquet triée :
  - ✓ on prend la plus petite carte que l'on voit sur les deux tas
  - ✓ puis on prend la plus petite carte que l'on voit sur les deux tas et on la glisse sous la carte que l'on a déjà dans la main
  - ✓ et on recommence au point précédent. . .

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

$3, 5,$

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

3, 5, 7,

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

3, 5, 7, 10,

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

3, 5, 7, 10, 12,

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

3, 5, 7, 10, 12, 13,

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

3, 5, 7, 10, 12, 13, 15,

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

3, 5, 7, 10, 12, 13, 15, 16,

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $3, 7, 12, 16, 25, 38, 40$

3, 5, 7, 10, 12, 13, 15, 16, 19,

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

3, 5, 7, 10, 12, 13, 15, 16, 19, 20,

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

3, 5, 7, 10, 12, 13, 15, 16, 19, 20, 25,

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $3, 7, 12, 16, 25, 38, 40$

3, 5, 7, 10, 12, 13, 15, 16, 19, 20, 25, 35,

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $3, 7, 12, 16, 25, 38, 40$

3, 5, 7, 10, 12, 13, 15, 16, 19, 20, 25, 35, 38,

Le tri **fusion**

idée : fusionner deux tableaux triés pour former un unique tableau trié se fait facilement :

$5, 10, 13, 15, 19, 20, 35$ $\qquad$ $3, 7, 12, 16, 25, 38, 40$

3, 5, 7, 10, 12, 13, 15, 16, 19, 20, 25, 35, 38, 40

Tri **fusion**

Étant donné un tableau (ou une liste) de $T[1, \ldots, n]$:

- si $n = 1$, retourner le tableau T !
- sinon :
  - Trier le sous-tableau $T[1 \ldots \frac{n}{2}]$
  - Trier le sous-tableau $T[\frac{n}{2} + 1 \ldots n]$
  - Fusionner ces deux sous-tableaux...
- Il s'agit d'un algorithme "diviser-pour-régner".
- $O(n \log n)$ opérations (au pire).

Principe du tri : Diviser pour régner !

- Etant donné un tableau (ou une liste) de Tab[1, … , N],
- N est une puissance de 2, sinon on le compète
- ✓ Si N = 1, retourner le tableau Tab (déjà trié)
- ✓ sinon
  - ✓ On découpe le tableau de 2 de même nombre
  - ✓ Trier le sous-tableau $\text{Tab}\left[1 \ldots \frac{N}{2}\right]$
  - ✓ Trier le sous-tableau $\text{Tab}\left[\frac{N}{2} \ldots N\right]$
- ✓ Fusionner ces deux sous-tableaux
- Il s'agit d'un algorithme « diviser pour régner ».

- Exercice

Appliquer à la main l'algorithme de tri par fusion aux listes d'entiers suivantes :

1. [19,5,18,17,6,5,2,13]
2. [13,6,6,12,17,10,16,20]
3. [4,1,14,18,14,1,7,4]
4. [6,10,7,16,6,5,17,15]

Implémentation

La fonction principale du tri par fusion est la fusion de de sous tableaux triés. Voici un programme en python qui réalise cette fonction :

```
from random import randint
liste1 = [randint(0,20) for i in range(8)]
liste1.sort()
print("1ère liste : ", liste1)
liste2 = [randint(0,20) for i in range(8)]
liste2.sort()
print("2ème liste : ", liste2)
liste = liste1+liste2
print ("Fusion de deux liste : ",liste)
liste_Trie= []
x = 0
y = 8
for i in range(0,16):
    if(x<=7 and y<=15 and
liste[x]<=liste[y]) or (y==16):
        liste_Trie.append(liste[x])
        x = x+1
    else:
        liste_Trie.append(liste[y])
        y = y+1
print("Liste trié  : ",liste_Trie)
```

CODAGE HUFFMAN

Codage d'un texte sous forme d'une chaîne de caractères

Codage ACII classique : 1 caractère occupe 8 bits, quel qu'il soit.

Si on veut réduire la taille d'un texte sans perdre d'information, une idée est de coder chaque caractère avec un nombre variable de bits. Les caractères les plus utilisés seront codés sur un nombre de bits petit, ceux qui ne sont que peu utilisés le seront avec un grand nombre de bits.

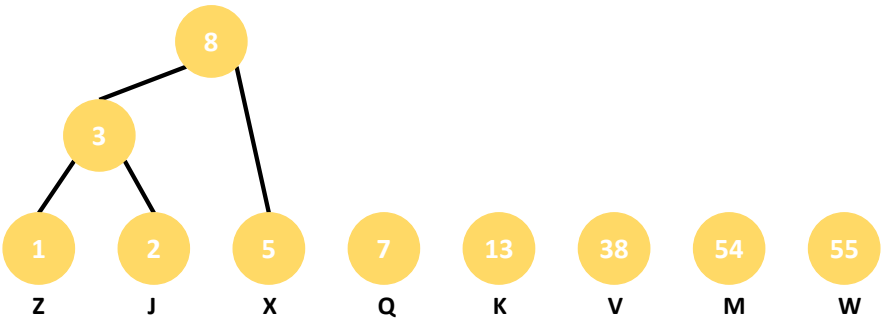
Le codage utilisé dépendra de la langue du texte à coder si on privilégie l'efficacité du codage.

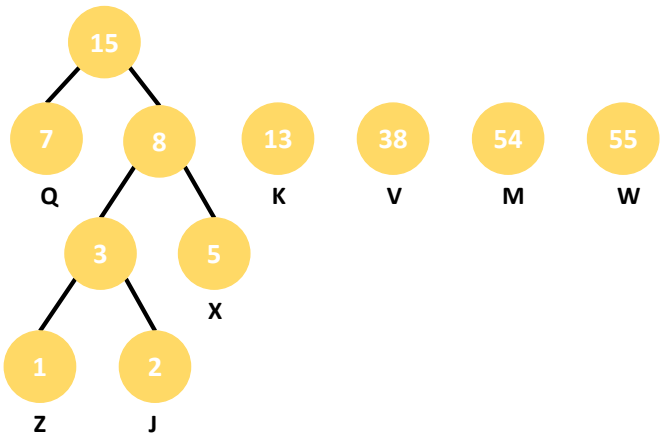
On parcourt le texte source, et on compte le nombre d'occurrences de chaque caractère rencontré. On peut alors définir l'effectif de chaque caractère (nombre de fois où il apparaît dans le texte), ou sa fréquence (effectif divisé par le nombre de caractère total du texte).

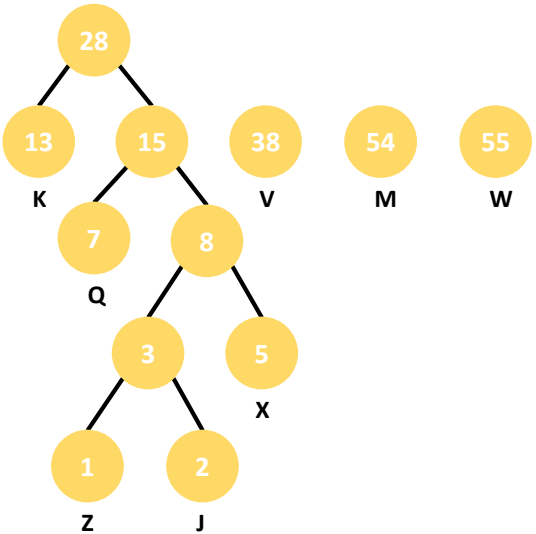
Ce travail préliminaire a été réalisé sur un texte en anglais. Voici un extrait des effectifs pour les lettres les moins utilisées :

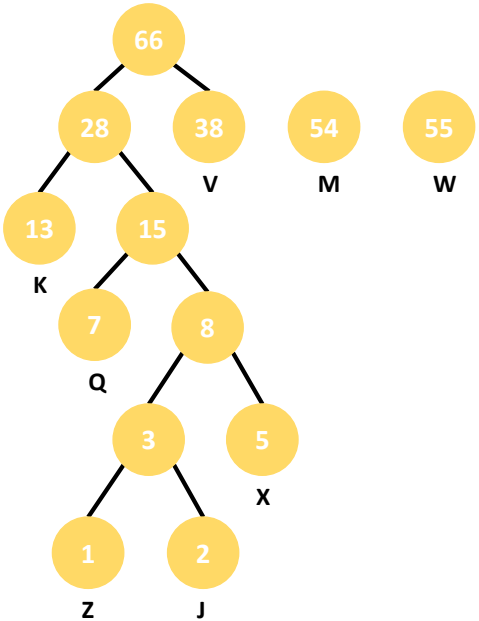
Z : 1	**J : 2**	**X : 5**	**Q : 7**
K : 13	**V : 38**	**M : 54**	**W : 55**

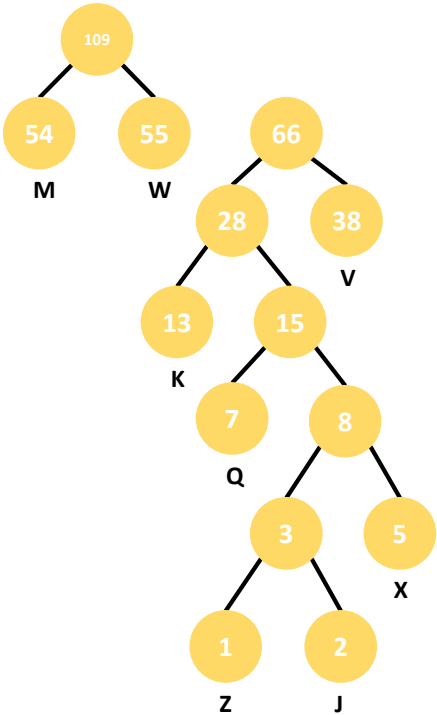

1
2
5
7
13
38
54
55
Z
J
X
Q
K
V
M
W

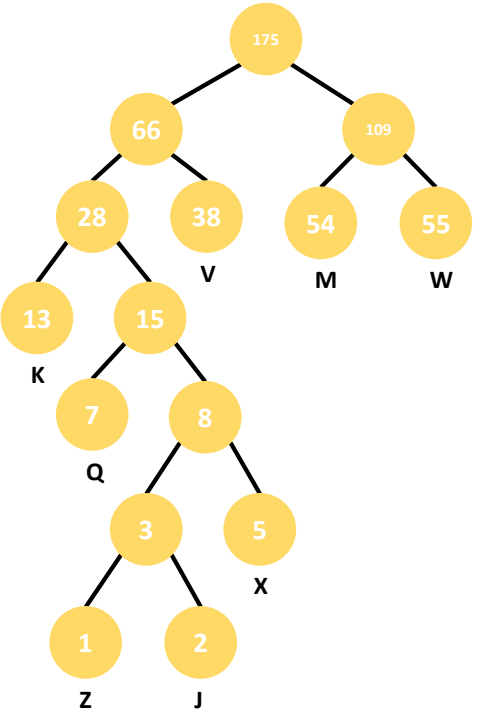

8
3
1
2
5
7
13
38
54
55
Z
J
X
Q
K
V
M
W

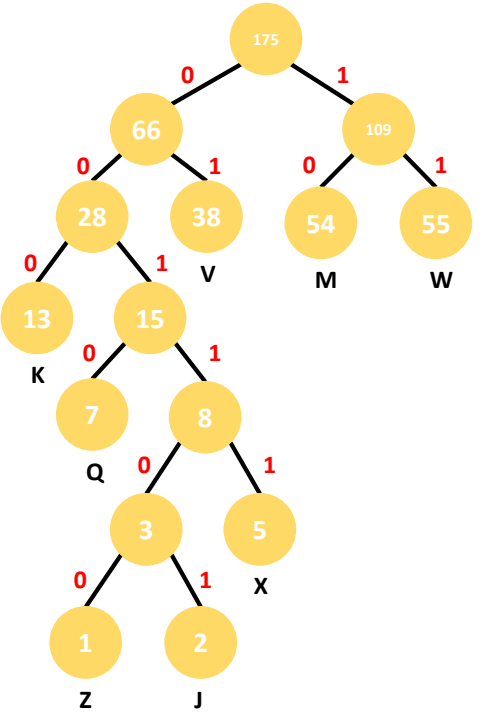

15
7
8
13
38
54
55
Q
K
V
M
W
3
5
X
1
2
Z
J


28
13
15
38
54
55
K
V
M
W
7
8
Q
3
5
X
1
2
Z
J


66
28
38
54
55
V
M
W
13
15
K
7
8
Q
3
5
X
1
2
Z
J


109
54
55
66
M
W
28
38
V
13
15
K
7
8
Q
3
5
X
1
2
Z
J


175
66
109
28
38
54
55
V
M
W
13
15
K
7
8
Q
3
5
X
1
2
Z
J


175
0
1
66
109
0
1
0
1
28
38
54
55
V
M
W
0
1
13
15
K
0
1
7
8
Q
0
1
3
5
X
0
1
1
2
Z
J

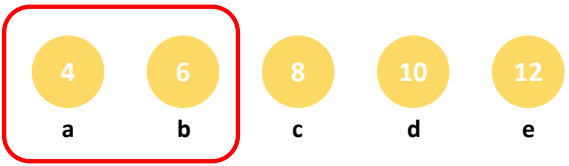
On arrive au codage suivant :

M : 10	**W : 11**	**V : 01**
K : 000	**Q : 0010**	**X : 00111**
Z : 001100	**J : 001101**	

Il faudrait faire le même travail sur toutes les caractères utilisables (majuscules, minuscules, chiffres, signes de ponctuation) afin de coder tout type de texte. L'arbre obtenu sera bien entendu beaucoup plus complexe.

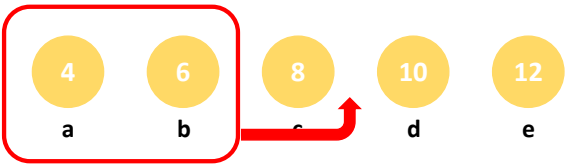
Attention, l’arbre n’est pas forcément unique !

Attention, l'arbre n'est pas forcément unique !



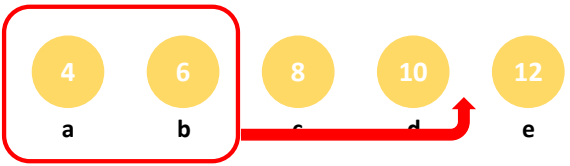
4 + 6 = 10

Attention, l'arbre n'est pas forcément unique !



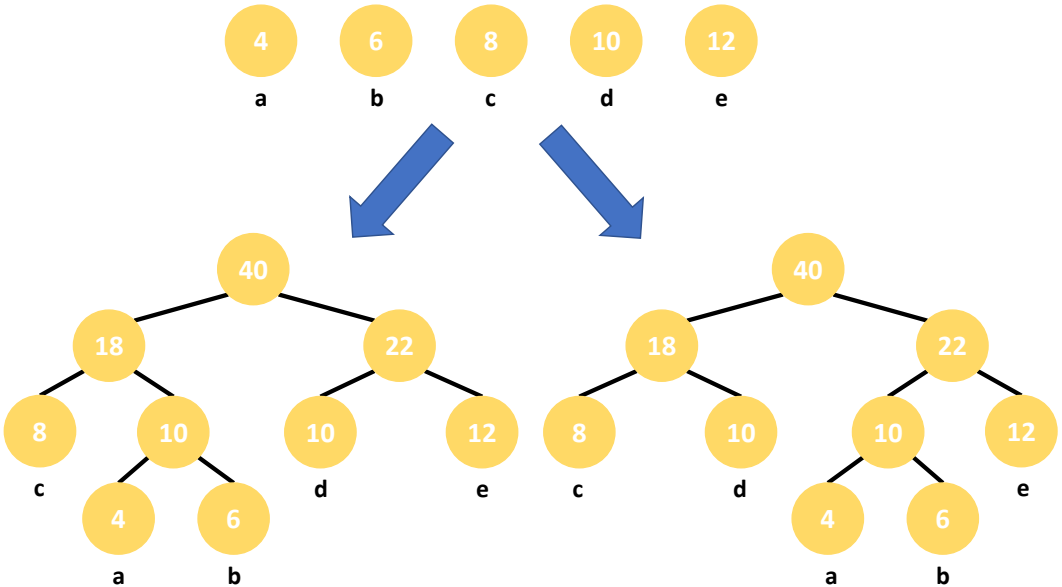
4 + 6 = 10

Attention, l'arbre n'est pas forcément unique !

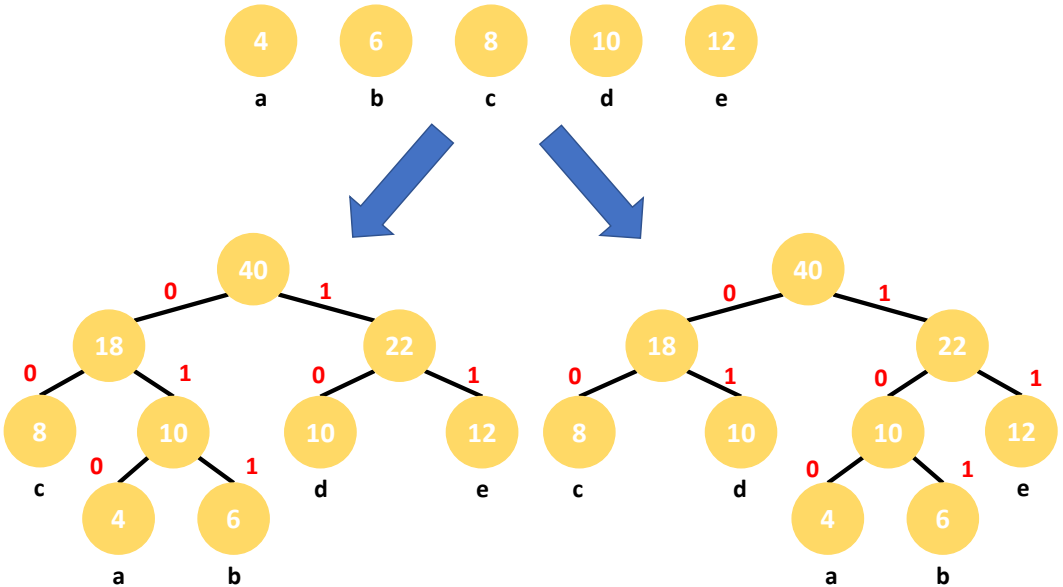


4 + 6 = 10

Attention, l'arbre n'est pas forcément unique !



Attention, l'arbre n'est pas forcément unique !



Deux codages possibles :

a : 010 b : 011 c : 00 d : 10 e : 11

a : 100 b : 101 c : 00 d : 01 e : 11

$\Rightarrow$ Nécessité de se mettre d'accord sur l'arbre utilisé pour le codage afin que le décodage soit possible !

Trois possibilités :

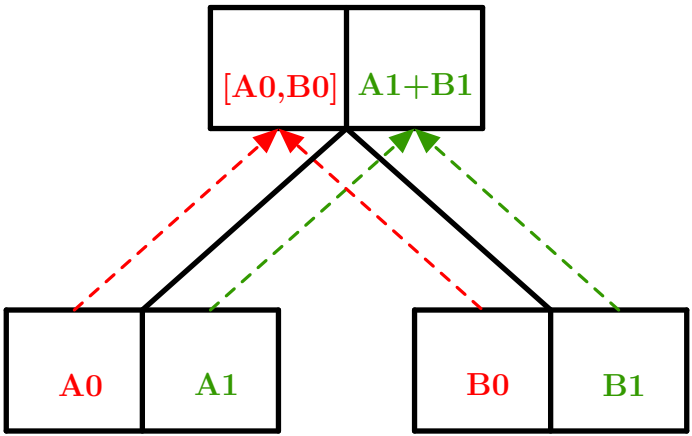
- Utilisation d'un arbre fixe (Huffman statique)
- Transmission de l'arbre en plus des données (Huffman semi-adaptatif)
- Construction de l'arbre « à la volée » (Huffman adaptatif)

ACTIVITE

I. Correspondance Listes imbriquées et arbres binaires

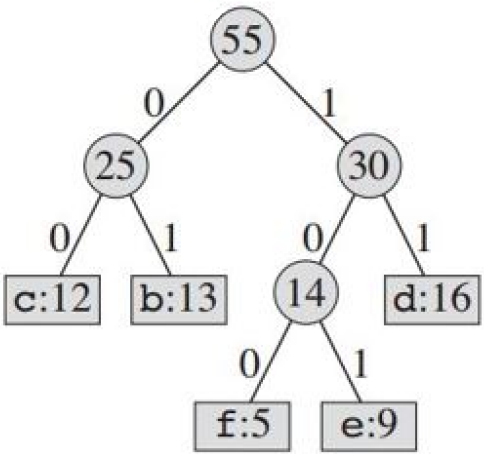
1. De l'arbre vers la liste imbriquée

Le principe : connaissant l'arbre, on le remonte en appliquant le principe suivant :



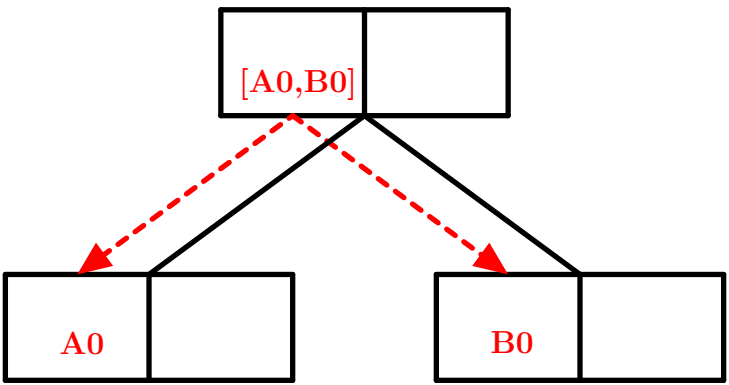
On applique le principe précédent tant qu'on n'a encore des sous arbres.

Exemple : A partir de l'arbre suivant, reconstruire la liste imbriquée correspondante :



2. Des Listes Imbriquées vers l'arbre

Le principe : connaissant la liste imbriquée, on sépare les deux éléments de la liste en deux listes comme sur le schéma suivant :



On applique ce principe tant que toutes les sous listes contiennent encore des listes et pas une unique lettre.

Exemple : A partir de la liste imbriquée suivante, recontruire l'arbre correspondant.

$$\Big[z, \Big[v, \Big[[x, d], s\Big]\Big]\Big]$$

3. Conclusion

On vient de voir qu'on a correspondance ("bijection") entre une liste imbriquée et un arbre binaire.

$$\text{Liste imbriquée} \iff \text{Arbre binaire}$$

Ainsi, pour programmer la construction d'un arbre binaire pour le codage de Huffman, on va pouvoir construire la liste imbriquée correspondante à l'arbre. Les étapes :

1. Texte
2. Liste de [lettre,effectif]
3. Liste triée selon les effectifs de [lettre,effectif]
4. - sur papier : passage à l'arbre
   - avec Python : construction à la liste imbriquée

II. Programmation

Objectif : programmer une fonction Python qui prend en argument un texte et qui renvoie la liste imbriquée associée à l'arbre binaire.

1. Ecrire une fonction python CreerListe.
   Cette fonction :
   - prend en arguments une lettre ℓ et un nombre k ;
   - retourne la liste $[\ell,k]$.

Correction 1 :

```
def CreerListe(l,k):
    L=[l,k]
    return(L)
```

2. Créer une fonction python ListeOccurence.
   Cette fonction :
   - prend en argument un texte ;
   - retourne une liste contenant des sous listes de la forme $[\ell,k]$, où ℓ correspond à une lettre du texte et k le nombre d'apparition de cette lettre dans le texte.

   Aide : on pourra utiliser le résultat de la fonction CréerListe.

Correction 2 :

```
def ListeOccurence(texte):
    L=[[texte[0],0]]
    for l in texte[1:]:
        i=0
        while (i<len(L) and l!=L[i][0]):
            i+=1
        if i==len(L):
            L=L+[CreerListe(str(l),1)]
        else :
            L[i][1]=L[i][1]+1
    return(L)
```

3. Créer une fonction python ListeTriParBulle.
   Cette fonction :
   - prend en argument une liste contenant des sous listes de la forme $[\ell,k]$;
   - retourne une liste contenant les sous listes de la forme $[\ell,k]$ rangées de manière croissante suivant les valeurs de k.

   Aide : on pourra utiliser la fonction TriABulle déjà étudiée.

Correction 3 :

```
def ListeTriParBulle(T):
    for i in range(len(T)-1,0,-1):
        for j in range(i):
            if T[j][1]>T[j+1][1]:
                aux=T[j+1]
                T[j+1]=T[j]
                T[j]=aux
    return(T)
```

4. Créer une fonction python FusionDeListe.
   Cette fonction :
   - prend en argument 2 listes de la forme $[\ell,k]$;
   - retourne une liste C de deux élements :
     — $C[0]$ étant une liste de 2 éléments $A[0]$ et $B[0]$;
     — $C[1]$ contient la somme de $A[1]$ et $B[1]$.

Correction 4 :

```
def FusionDeListes(A,B):
    C=[0,0]
    C[0]=[A[0],B[0]]
    C[1]=A[1]+B[1]
    return(C)
```

5. Créer une fonction python CréationDeLArbre.
   Cette fonction :
   - prend en argument une liste contenant des sous listes de la forme $[\ell,k]$;
   - retourne une liste C de deux élements :
     — $C[0]$ étant une liste de taille 2 contenant les éléments $A[0]$ et $B[0]$;
     — $C[1]$ contient la somme de $A[1]$ et $B[1]$.

   Aide : on pourra utiliser les fonctions ListeTriParBulle et FusionDeListes.

Correction 5 :

```
def CreationDeLArbre(T) :
    while len(T)>1 :
        T[1]=FusionDeListes(T[0],T[1])
        del T[0]
        ListeTriParBulle(T)
    return(T)
```

6. Répondre alors à l'objectif donné en début de partie.

7. Application :
   (a) Tester la fonction écrite en question précédente avec la chaine de caractère 'aabbbcddef'.
   (b) Construire alors l'arbre binaire associé à la liste imbriquée renvoyée.
   (c) Donner le codage de chacune des lettres présentes dans la chaine de caractère.

Pour aller plus loin ...

- Reprendre les différentes fonctions écrites dans la partie précédente en utilisant le principe de la programmation défensive.
- Ecrire un algorithme qui, à partir d'une liste imbriquée, code chacun des caractères présents dans celle-ci, par la méthode décrite dans le codage de Huffman.