

Codages binaires optimaux. Corrigé

1) a) L est la liste des lettres apparaissant dans t .

M est la liste des couples (x, k) , où $k \in \mathbb{N}^*$ est le nombre d'occurrences de la lettre x dans t .

b) La complexité est en $O(n) + O(nm)$, où m est le nombre de lettres distinctes apparaissant dans t .

Dans le cas le pire, on a $m = n$. D'où une complexité $O(n^2)$.

2) a) `def occurrences(t) :`

```
w = {}
for x in t :
    if x in w : w[x] = 1
    else : w[x] += 1
return w
```

b) `def minima(w) :`

```
L = w.keys() ; n = len(L)
assert n > 1
# dans la suite, on utilise le principe du tri par sélections
for i in [0,1]
    m = w[L[i]] ; j = i
    for k in range(i+1,n) :
        if w[L[k]] < m : j = k
    L[i],L[j] = L[j],L[i]
return L[0],L[1]
```

3) `def alphabet(arbre) :`

```
lettres = {}
for m in arbre :
    if arbre[m] != None : lettre[arbre[m]] = m
return lettres
```

4) a) `def prefixe(u,v) :`

```
n = len(u) ; p = len(v)
if p < n : return False
for i in range(n) :
    if u[i] != v[i] : return False
return True
```

b) `def verif(lettres) :`

```
L = lettres.keys() ; n = len(L)
for i in range(n) :
    for j in range(n) :
        if i != j and prefixe(L[i],L[j]) : return False
return True
```

c) Une fois triée, il suffit de vérifier qu'aucun mot n'est préfixe du suivant dans la liste triée.

```
def verif_bis(lettres) :  
    L = sorted(lettres.keys()) ; n = len(L)  
    for i in range(n-1) :  
        if prefixe(L[i],L[i+1]) : return False  
    return True
```

d) def plus_petit(u,v) :

```
    n = len(u) ; m = len(v)  
    for i in range(min(n,m)) :  
        if u[i]<v[i] : return True  
        elif u[i]>v[i] : return False  
    # à ce stade, l'un des mots est préfixe de l'autre  
    return (n<=m)
```

5) def codage(T,arbre) :

```
    lettres = alphabet(arbre) ; M = ""  
    for x in T :  
        M = M + lettres[x]  
    return M
```

6) def decodage(M,arbre) :

```
    etat = "" ; T = ""  
    for x in M :  
        if arbre[etat] == None :  
            etat = etat + x  
        else :  
            T = T + arbre[etat] ; etat = ""  
    return T
```

7) def code_optimal(w) :

```
    n = len(w)  
    if n == 2 :  
        L = w.keys() ; lettres = { L[0] : "0" , L[1] : "1" }  
    else :  
        (x,y) = minima(w)  
        ww = w.copy() ; ww[x] = w[x] + w[y] ; del ww[y]  
        lettres = code_optimal(ww)  
        lettres[y] = lettres[x] + "1" ; lettres[x] = lettres[x] + "0"  
    return lettres
```