CORRIGÉ DE L'ÉPREUVE ITC CCMP 2024

1

Question 1 – On peut proposer les représentations suivantes : `'a'` par `0`, `'b'` par `10` et `'c'` par `11`.
Le problème avec une représentation des lettres par des suites de `0` et `1` de longueurs différentes est de savoir quand commence la représentation de chaque lettre (problème qui n'existe pas lorsque, comme ci-dessus, toutes les lettres ont une représentation de longueur 2).
Cependant, avec la représentation proposée, lorsque l'on rencontre un `1`, c'est que l'on est en train de représenter un `'b'` ou un `'c'` (et que l'on doit prendre en considération 2 chiffres).
Exemple : `1011010010` sera déchiffré sans ambiguïté. On commence par un `1`, donc la première lettre est représentée par 2 chiffres (`10`) ; c'est donc un `'b'`. Puis on tombe sur un `1` ; même raisonnement : on prend en compte les deux chiffres `11`, qui représentent un `'c'`. Puis on tombe sur un `0` : `'a'` ; puis `10` : `'b'` ; puis `0` : `'a'` ; puis `10` : `'b'`.
Au final, `1011010010` représente `'bcabab'`.

1.1 Analyse du texte source

Question 2 –

```
def nbCaracteres(c,s):
    nb = 0
    for car in s :
        if car == c:
            nb += 1
    return nb
```

Question 3 – Lorsque `s='abaabaca`, cette fonction renvoie `['a','b','c']`.
Cette fonction parcourt l'ensemble des lettres de `s`. Si la lettre lue n'a pas encore été rencontrée, elle ne figure pas encore dans `listeCar` et y est rajoutée. Ainsi cette fonction renvoie donc la liste des lettres contenues dans `s`, sans redondance.

Question 4 – la boucle externe présente dans la fonction est de longueur n.
À chaque tour dans cette boucle il y a une affectation, un test d'appartenance à la liste `listeCar` et éventuellement une instruction `append`. Le test d'appartenance nécessite, dans le pire des cas (lorsque toutes les lettres distinctes ont été rencontrées), de l'ordre de k opérations.
Ainsi la complexité de cette fonction est en $\mathcal{O}(n \times k)$.

Question 5 – Cette fonction prend en entrée une chaîne de caractères `s`, détermine la liste `l` des caractères contenus dans `s` et renvoie une liste `R` de même taille que `l`. L'élément `R[i]` de cette liste est un couple constitué du caractère `l[i]` et de son nombre d'occurences dans `s`.
La commande `analyseTexte('babaaaabca')` renvoie la valeur `[('b',3),('a',6),('c',1)]`.

Question 6 – L'exécution de la fonction `analyseTexte` nécessite l'appel à la fonction `listeCarracteres`, dont la complexité est de l'ordre de $\mathcal{O}(n \times k)$.
La boucle est ensuite de longueur k. À chaque tour dans cette boucle, on fait appel à la fonction `nbCarracteres`, de complexité linéaire en n. L'exécution de cette boucle a donc une complexité de l'ordre de $\mathcal{O}(n \times k)$.
Ainsi la complexité de cette fonction est de l'ordre de $\mathcal{O}(n \times k)$.

Question 7 –

```
def analyseTexte(s):
    dico = {}
    for c in s:
        if c in dico:
            dico[c] += 1
        else :
            dico[c] = 1
    return dico
```

L'accès à coût constant aux clefs du dictionnaire assure la complexité linéaire de cette fonction.

1.2 Exploitation d'analyses existantes

Question 8 –

```
SELECT DISTINCT auteur
FROM corpus
```

Question 9 –

```
SELECT car.symbole , SUM(occ.nombreOccurences) / SUM(cor.nombreCaracteres)
FROM caractere AS car JOIN corpus AS cor JOIN occurences AS occ
                      ON car.idCar = occ.idCar AND occ.idLivre = cor.idLivre
WHERE cor.langue = 'Français'
GROUP BY car.idCar
```

Remarque : ne fonctionne que si toutes les lettres apparaissent dans chaque livre.
On pouvait pallier à cet inconvénient par :

```
SELECT car.symbole , SUM(occ.nombreOccurences) /
         (SELECT SUM(nombreCaracteres) FROM corpus WHERE langue = 'Français')
FROM caractere AS car JOIN corpus AS cor JOIN occurences AS occ
                      ON car.idCar = occ.idCar AND occ.idLivre = cor.idLivre
WHERE cor.langue = 'Francais'
GROUP BY car.idCar
```

Mais le rapport du jury indique : « Le rapport du jury indique : « Plusieurs candidats proposent des sous-requêtes alors que le sujet demande explicitement UNE requête »...

1.3 Compression

Question 10 – Si la chaîne à coder est `'bac'`, on obtient successivement :
- l'intervalle $[0.2; 0.3[$ correspondant à la première lettre `'b'` ;
- le caractère `'a'` détermine alors le sous intervalle $[0.20; 0.22[$;
- le caractère `'c'` détermine enfin l'intervalle $[0.206; 0.210[$.

Question 11 –

```
def codage(s):
    n = len(s)
    g , d = 0 , 1
    for i in range(n):
        g , d = codeCar(s[i],g,d)
    return (g,d)
```

1.4 Décodage

Question 12 – Comme `x` est dans l'intervalle $[0.116; 0.124[$, la lettre suivant `'ad'` est un `'b'`.

Question 13 – Les chaînes `'b'` et `'ba'` peuvent correspondre au nombre 0.2.
L'origine de cette ambiguïté est que le fait de rajouter des `'a'` ne change pas la borne gauche de l'intervalle. Comme on ne connaît pas la longueur de la chaîne, cela crée une ambiguïté.

Question 14 – Pour la fonction décodage, on a besoin à chaque étape du caractère obtenu mais aussi de l'intervalle `[g,d[` dans lequel on l'a trouvé.

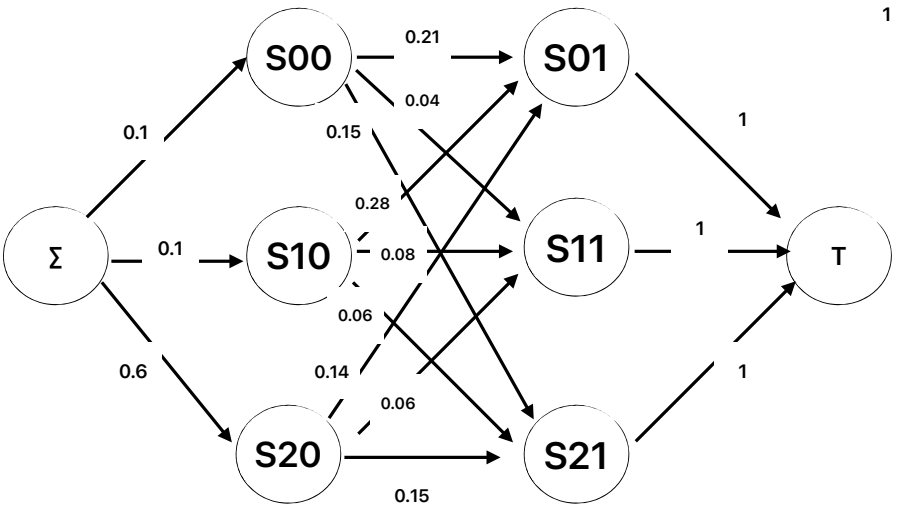
```
def decodage(x):
    g , d = 0 , 1
    s = ''
    car = decodeCar(x,g,d)
    while car != '#':
        s = s + car
        g  , d  = codeCar(car,g,d)
        car = decodeCar(x,g,d)
    s = s + '#'
    return s
```

2

2.1 Modélisation du canal de communication par un graphe

Question 15 – Il y a $N \times K$ sommets.
Il y $N-1$ transitions entre deux couches (verticales). À chaque transition il y a K^2 arcs. Il y a donc en tout $(N-1) \times K^2$ arcs.

Question 16 – Faisons un exemple de calcul : celui de la pondération de l'arc allant de $S_{1,0}$ à $S_{2,1}$. D'après l'énoncé, la pondération de cet arc est égale à $E_{obs_1,2}P_{1,2}$, c'est à dire $E_{0,2}P_{1,2}$, c'est à dire $0.3 \times 0.2 = 0.06$.



Question 17 – À chacune des N étapes il y a K choix pour le chemin. Il y a donc en tout (de l'ordre de) K^N chemins.
Un algorithme d'exploration exhaustive aurait donc une complexité exponentielle, ce qui n'est pas envisageable dès que la taille des données est importante.

2.2 Stratégie gloutonne

Question 18 –

```
def maximumListe(L):
    n = len(L)
    m = L[0]
    im = 0
    for i in range(1,n):
        if L[i] > m:
            im = i
            m = L[i]
    return m , im
```

Question 19 – Remarque : contrairement à ce qui est écrit dans l'énoncé, la première ligne de la fonction `initialiserGlouton(Obs,E,K)` doit être : `probaInitiales = [E[Obs[0]][i] for i in range(K)]`.

```
def glouton(Obs,P,E,K,N):
        symb = initialiserGlouton(Obs,E,K) # symb d\'esigne la lettre
                                           #venant d'\^etre d\'ecod\'ee
        message = [symb] # liste contenant les lettres du message d\'ecod\'e
        for j in range(0,N-1):
                probas = [E[Obs[j+1]][k] *P[symb][k] for k in range(K)]
                s , symb = maximumListe(probas)
                message.append(symb)
        return message
```

Question 20 – À chaque étape de la boucle contenue dans cette fonction, la construction de la liste `probas` ainsi que la recherche du maximum se font en $\mathcal{O}(K)$. La complexité de la fonction est donc en $\mathcal{O}(K \times N)$.

Question 21 – Dans cet exemple le message décodé est `[0,0]`.
On obtient un produit des probabilité égal à $0.6 \times 0.5 = 0.30$.
Or il existe un chemin donnant $0.4 \times 0.9 = 0.36$.
Le décodage obtenue par la stratégie gloutonne n'est donc pas optimale.

2.3 Stratégie de programmation dynamique

Question 22 – On cherche un chemin $\sigma S_{i_0,0} \dots S_{i_{N-1},N-1}\tau$ tel que le produit des probabilités qui le composent soit maximal.
Cela revient à maximiser la somme des logarithmes de ces probabilités, ou encore à minimiser la somme des opposés des logarithmes de ces probabilités (qui sont bien positifs).
On se ramène ainsi à un problème de plus court chemin dans un graphe pondéré à poids positifs.
Ce problème pourrait être résolu par l'algorithme de Dijkstra.

Question 23 –

```
def construireTableauViterbi(Obs,P,E,K,N):
        T , argT = initialiserViterbi(E,Obs[0],K,N)
        for j in range(1,N):
                for i in range(K):
                        listeTPE = [T[k][j-1]*P[k][i]*E[Obs[j]][i] for k in range(K)]
                        T[i][j] , argT[i][j] = maximumListe(listeTPE)
        return T , argT
```

Question 24 – On cherche le chemin de probabilité maximale de la manière suivante :
- On détermine le maximum des $T_{k,N-1}$, pour $0 \leq k < K$; on trouve ici $1.8e-05$, obtenu pour $k = 0$.
- On remonte à partir de ce sommet $S_{0,N-1}$ en lisant les antécédents successifs dans le tableau `argT` ; point de départ : 0 , puis 0,1,1,2,0,0,2. On obtient ainsi le message décodé : `[2,0,0,2,1,1,0,0]`.

Question 25 – On a dans cette fonctions deux boucles imbriquées de longueurs K et N.
À chaque tour dans la boucle interne, on construit une liste de longueur K et on fait appel à la fonction `maximumListe` sur cette liste, ce qui se fait en $\mathcal{O}(K)$.
On obtient ainsi une complexité en $\mathcal{O}(N \times K^2)$.
Concernant la complexité en espace, on construit un tableau d'entiers et un tableau de flottants de taille $K \times N$ (on ne tient pas compte des listes intermédiaires). Si l'on considère que chacun de ces nombres est codé sur 8 octets (64 bits), cela donne une complexité spatiale égale à $16 \times K \times N$ octets.