

Devoir surveillé 2 - - Corrigé

Ce devoir reprenait en grande partie le sujet MINES-PONTS 2024 (questions SQL modifiées, et les 3 questions finales ont été supprimées).

1. Si l'on code a par 0, b par 10 et c par 11, il n'y aura pas d'ambiguïté de lecture : si l'on lit un 1, on sait que l'on commence à lire un b ou un c , le bit suivant indiquant lequel. Les 1 sont donc toujours au début d'un bloc de longueur 2 et ne sont pas interprétés seuls, jamais les 0.

La chaîne s est alors codée par 01000100110, donc sur 11 bits.

2.

```
1 def nbCaracteres(c, s) :
2     compteur = 0
3     for lettre in s :
4         if lettre == c :
5             compteur += 1
6     return compteur
```

3. Sur cet exemple, la fonction renvoie la liste ['a', 'b', 'c'].

En effet, cette fonction parcourt les caractères de la chaîne s , et pour chaque caractère :

- s'il n'a pas déjà été rencontré, donc s'il n'est pas dans la liste `listCar`, on l'ajoute à `listCar` ;
- sinon, on le passe.

Cette fonction renvoie ensuite la liste `listCar`, qui contient bien la liste des caractères de s , sans doublon.

4. Les instructions des lignes 2, 3, 5 et 7 s'effectuent en temps $O(1)$.

Le test de la ligne 6 s'effectue en temps (au plus) $O(k)$. Ainsi, un tour de la boucle `for` s'effectue en temps $O(k) + O(1) = O(k)$. Cette boucle effectue n tours, donc a une complexité en $nO(k) = O(nk)$.

La complexité totale de la fonction est donc en $O(nk) + O(1) = O(nk)$.

5. Cette fonction calcule la liste des caractères de s (ligne 3), et pour chacun de ces caractères c elle calcule le nombre n_c d'occurrences de c dans s (ligne 6), puis renvoie la liste des (c, n_c) .

Cette fonction renvoie donc l'*histogramme* de la liste s , c'est-à-dire la liste des couples (c, n_c) , où les c sont triés par ordre d'apparition dans s .

Sur cet exemple, la fonction renvoie la liste [('b', 3), ('a', 6), ('c', 1)].

6. La ligne 3 a une complexité en $O(kn)$. La ligne 6 a une complexité en $O(n)$ (appel de la fonction `nbCaracteres`), et est itérée k fois. Ainsi, la boucle `for` de cette fonction s'effectue en temps au plus $kO(n) = O(kn)$.

La complexité totale de cette fonction est donc $O(kn) + O(kn) = O(kn)$.

7.

```
7 def analyseTexte(s) :
8     dico = {}
9     for lettre in s :
10        if lettre in dico :
11            dico[lettre] += 1
12        else :
13            dico[lettre] = 1
14    return dico
```

8.

```
15 SELECT DISTINCT auteur
16 FROM corpus
```

9.

```
17 SELECT COUNT(*)
18 FROM caractere WHERE typeCaractère='lettre'
```

10.

```
19 SELECT auteur, SUM(nombreCaractères) AS nb_total_caractères FROM corpus
    ↪ GROUP BY auteur
```

11. Lettre b : intervalle $[0, 2; 0, 3[$, de largeur $\frac{1}{10}$.

Pour la lettre a , on considère ensuite le premier cinquième de cet intervalle : $[0, 2; 0, 22[$, de largeur $\frac{1}{5} \times \frac{1}{10} = \frac{1}{50}$.

Pour la lettre c , on subdivise cet intervalle en dix intervalles :

$$[0, 2; 0, 202; 0, 204; 0, 206; 0, 208; 0, 210; 0, 212; 0, 214; 0, 216; 0, 218; 0, 220],$$

et l'on considère les quatrième et cinquième intervalles, ce qui donne $[0, 206; 0, 210]$. Cet intervalle est bien de largeur $\frac{1}{5} \times \frac{1}{50} = \frac{1}{250} = 0,004$.

12.

```
20 def codage(s) :
21     g, d = 0, 1
22     for caractere in s :
23         g, d = codCar(caractere, g, d)
24     return g, d
```

13. Le caractère a donne l'intervalle $[0; 0, 2[$. Le caractère d donne ensuite l'intervalle $[0, 1; 0, 18[$. On le subdivise ensuite en 10 intervalles de largeur $\frac{8}{100}$, ce qui donne la subdivision

$$[0, 1; 0, 108; 0, 116; 0, 124; 0, 132; 0, 140; 0, 148; 0, 156; 0, 164; 0, 172; 0, 18].$$

Comme x appartient au troisième intervalle, le troisième caractère est un b .

14. Les chaînes b et ba sont toutes les deux représentées par $0, 2$.

Cela vient du fait que la borne de gauche est autorisée pour a . Si on prenait le codage de la chaîne à l'intérieur de l'intervalle considéré, il n'y aurait pas ce problème.

15. On peut récupérer le premier caractère de la chaîne par la fonction `decodeCar`, puis calculer les bornes suivantes par la fonction `codage`. On itère ensuite ce procédé, tant que le dernier caractère décodé n'est pas $\#$. On peut donc écrire la fonction suivante.

```
25 def decodage(x) :
26     s = decodeCar(x, 0, 1)
27     g, d = codage(s)
28     while s[-1] != '#' :
29         s = s + decodeCar(x, g, d) #ATTENTION pas de append avec les
    ↪ chaînes de caractères
30         g, d = codage(s)
31     return s
```

On peut aussi en donner une version récursive.

```

32 def decodage(x):
33     def aux(s, g, d):
34         if s[-1] == '#':
35             return s
36         else:
37             c = decodeCar(x, g, d)
38             s2 = s+c
39             g, d = codage(s2)
40             return aux(s2, g, d)
41     return aux("", 0, 1)

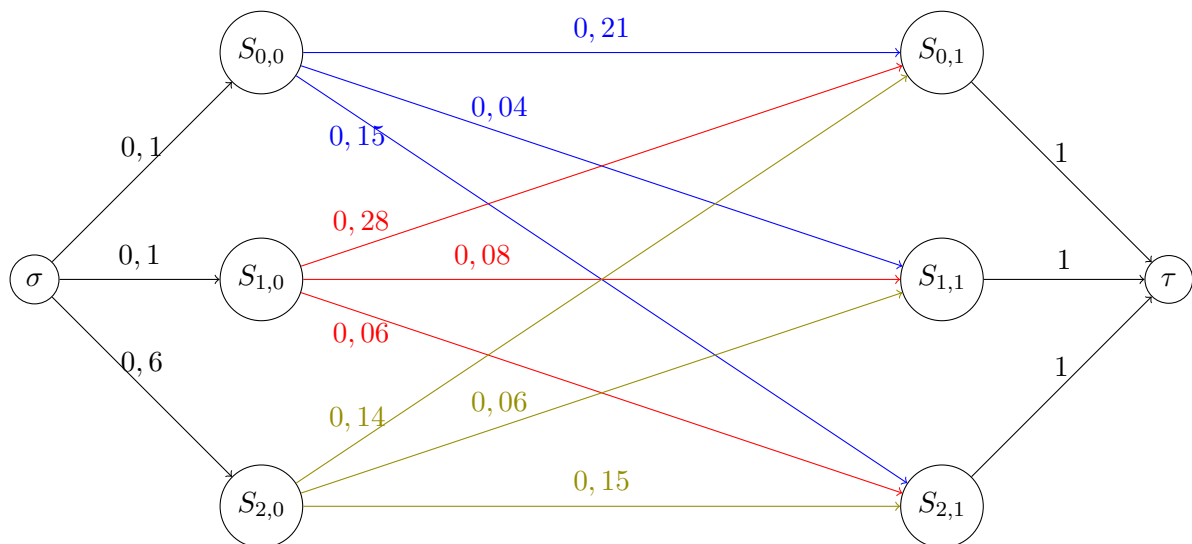
```

16. Il y a N observations, et pour chaque observation un sommet pour chacun des K caractères : il y a donc KN sommets.

Pour chacune des $N - 1$ observations, sauf la dernière, et chacun des K caractères, il y a K arêtes dans le graphe. Il y a donc $(N - 1)K^2$ arête dans le graphe.

17. Le premier symbole est 2, les probabilités partant de la source sont donc données par la dernière ligne de E : $(0, 1; 0, 1; 0, 6)$.

Le deuxième symbole est 0. Pour calculer la probabilité de la transition de $S_{i,0}$ vers $S_{k,1}$, on multiplie coefficient par coefficient chaque ligne de P par la première ligne de E .



18. Choisir un chemin entre σ et τ revient, pour chacune des N observations, à choisir un symbole. Il y a donc K^N tels chemins exactement (pas besoin de $O...$).

Il n'est pas possible de mettre en œuvre un algorithme exhaustif dès que K ou N sont grands : il y a trop de chemins.

19.

```

42 def maximumListe(liste) :
43     imax = 0
44     for i in range(1, len(liste)) :
45         if liste[i] > liste[imax] :
46             imax = i
47     return liste[imax], imax

```

20.

```

48 def glouton(Obs, P, E, K, N):
49     message = [initialiserGlouton(Obs, E, K)]

```

```

50     for i in range(1,N):
51         i = message[-1]
52         probas = [E[Obs[j]][k] * P[i][k] for k in range(K)]
53         _, k = maximumListe(probas)
54         message.append(k)
55     return message

```

21. La fonction `initialiserGlouton` a *a priori* une complexité en $O(K)$, vu qu'elle consiste en un parcours simple d'une liste de longueur K .

Le calcul de la liste `probas` s'effectue en temps $O(K)$, pour la même raison, et le calcul de k aussi. Ainsi, un tour de la boucle `for` s'effectue en temps $O(K)$, et cette boucle est réalisée $N - 1 \leq N$ fois.

La complexité de la fonction `glouton` est donc en $O(K) + NO(K) = O(NK)$.

22. L'algorithme glouton choisit d'abord le caractère 0, puis encore le caractère 0. Le produit des probabilités sur ce chemin donne 0,3.

Or, le chemin 10 a une probabilité de 0,36. L'algorithme glouton n'est donc pas optimal.

23. En transformant chaque probabilité de transition $p \in [0, 1]$ en $1 - p \in [0, 1]$, on transforme le plus long chemin en le plus court chemin, toujours dans un graphe pondéré à poids positifs.

On pourrait alors appliquer l'algorithme de Dijkstra.