Sujet Mines-Ponts 2024

Q1 Pour qu'il n'y ait pas d'ambiguïtë dans le décodage, il est nécessaire de choisir un bit pour coder `'a'` différent du premier bit qui code `'b'` ou `'c'`.

On choisit donc `0` pour coder `'a'`, `10` pour coder `'b'` et `11` pour coder `'c'`. Ainsi, pour décoder la chaine, on regarde le premier bit :

- ⋆ si c'est un `0`, on décode le caractère `'a'` et on passe au prochain bit
- ⋆ si c'est un `1`, on regarde le bit suivant :
  - si c'est un `0`, on décode le caractère `'b'` et on passe au prochain bit
  - si c'est un `1`, on décode le caractère `'c'` et on passe au prochain bit

On recommence ainsi jusqu'à la fin de la chaine.

Q2

```
1 def nbCaracteres(c: str, s: str) -> str:
2     cpt = 0
3     for l in s:
4         if l == c:
5             cpt += 1
6     return cpt
```

Q3 `s = 'abaabaca'` renvoie la liste `['a', 'b', 'c']` Cette fonction parcourt les caractères de la chaine `s` et pour chacun d'entre eux vérifie s'il est déjà présent dans la liste `listeCar` : s'il ne l'est pas, elle l'ajoute à cette liste. Ainsi, chaque caractère ne sera présent qu'une seule fois à la fin de la boucle.

Q4 Notons $C(n,k)$ la complexité de cette fonction dans le pire des cas.

Le pire des cas avec k caractères distincts est obtenu dans le cas d'une chaine commençant par k caractères distincts puis $n-k$ caractères identiques au derniers des k premiers caractères.

Dans ce cas, la complexité de la fonction sera :

$$\begin{aligned} C(n,k) &= O(1) + O(1+2+3+\cdots+k+k+\cdots+k) \\ &= O\left(k(k+1)/2 + (n-k)k\right) \\ &= O(nk) \end{aligned}$$

Q5 Cette fonction calcule la liste des caractères distincts d'une chaine `s`, puis, pour chacun d'entre eux, calcule le nombre d'occurrences de ce caractère. Elle ajoute alors à une liste `R` le couple (caractère, nombre d'occurrences) et renvoie cette liste `R`.

La commande `analyseTexte('babaaaabca')` renvoie la liste `[('b', 3), ('a', 6), ('c', 1)]`.

Q6 Notons $C(n,k)$ la complexité de cette fonction dans le pire des cas.

Dans ce cas, la complexité de la fonction sera :

$$\begin{aligned} C(n,k) &= O(nk) + k \times O(n) \\ &= O(nk) \end{aligned}$$

Q7

```python
def analyseTexte(s: str) -> list:
    D = {}
    for c in s:
        if c not in D:
            D[c] = 1
        else:
            D[c] += 1
    return D
```

Q8
```sql
SELECT DISTINCT auteur FROM corpus;
```

Q9
```sql
SELECT
    car.symbole AS Symbole,
    SUM(occ.nombreOccurences) / (SELECT SUM(nombreCaracteres) FROM corpus WHERE
    langue = "Français") AS Fréquence
FROM caractere AS car
JOIN occurrences AS occ ON occ.idCar = car.idCar
JOIN corpus AS cor ON occ.idLivre = cor.idLivre
WHERE cor.langue = "Français"
GROUP BY car.symbole;
```

Q10 Le premier caractère `'b'` nous donne l'intervalle $[0.2, 0.3[$.
Le deuxième caractère `'a'` nous donne l'intervalle $[0.2, 0.22[$.
Le troisième caractère `'c'` nous donne l'intervalle $[0.206, 0.21[$.

Q11
```python
def codage(s: str) -> (float, float):
    (g, d) = (0, 1)
    for car in s:
        (g, d) = codeCar(car, g, d)
    return (g, d)
```

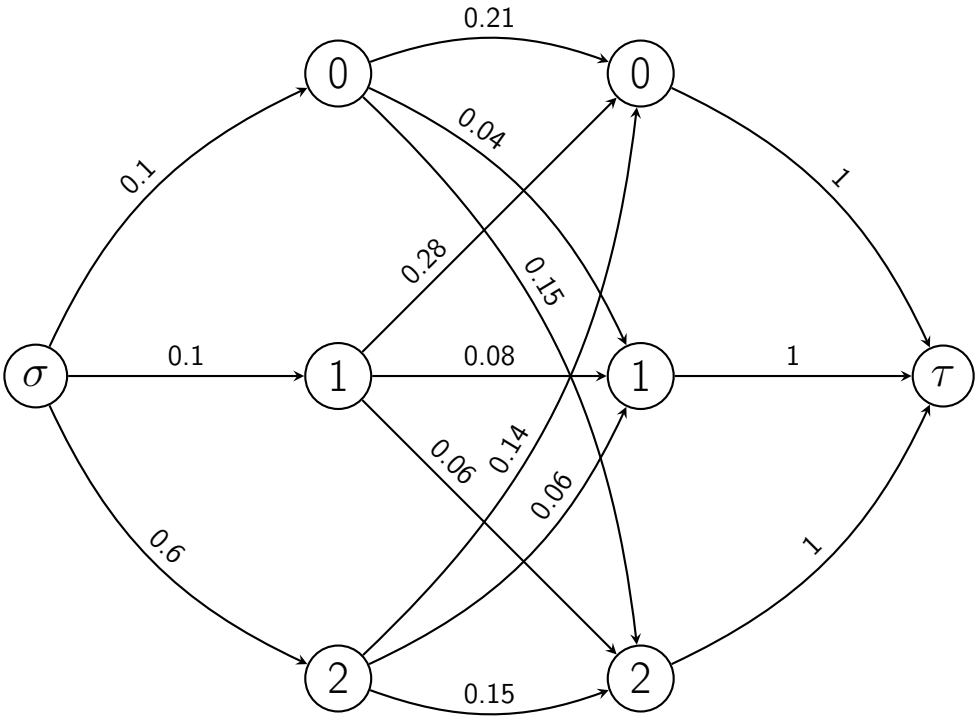
Q12 Dans l'intervalle $[0.10, 0.18[$, le sous-intervalle qui correspond à `'a'` est $[0.10, 0.116[$ mais il ne contient pas x. Le sous-intervalle qui correspond à `'b'` est $[0.116, 0.124[$ qui contient x, donc le caractère suivant est un `'b'`.

Q13 Les chaines `'ba'` et `'baa'` peuvent toutes deux correspondre au flottant 0.2.
Le problème est que toutes les chaines du type `'baaa...'` seront codées par un intervalle commençant par 0.2, mais on ne sait pas quand il faut s'arrêter dans le décodage car on ne connaît pas le nombre de caractères du message à décoder.

Q14
```python
def decodage(x: float) -> str:
    (g, d) = (0, 1)
    car = ''
    s = ""
    while car != "#":
        s += car
        car = decodeCar(x, g, d)
        (g, d) = codeCar(car, g, d)
    return s
```

Q15 Il y a N couches verticales et K sommets par couche. Il y a donc NK sommets dans le graphe.
Chaque sommet (sauf ceux de la dernière couche) est relié à K sommets de la couche suivante. Il y a donc $(N-1)K \cdot K = (N-1)K^2$ arcs dans le graphe.

Q16



Q17 Pour choisir un chemin de σ à τ, on choisit à chaque couche un arc parmi K possibles et il y a N couches. Il y a donc K^N chemins de σ à τ.

Un algorithme d'exploitation exhaustive ne peut être envisagé que pour de petites valeurs de N.

Q18

```
def maximumListe(liste: [float]) -> (float, int):
    i = 0
    for k in range(1, len(liste)):
        if liste[k] > liste[i]:
            i = k
    return liste[i], i
```

Q19

```
def glouton(Obs: [int], P: [[float]], E: [[float]], K: int, N: int) -> [int]:
    i = initialiserGlouton(Obs, E, K)
    chemin = [i]
    for j in range(N - 1):
        probasTransitions = [E[Obs[j+1]][k] * P[i][k] for k in range(K)]
        s, i = maximumListe(probasTransitions)
        chemin.append(i)
    return chemin
```

Q20 La fonction `initialiserGlouton` est de complexité $O(K)$. La fonction `glouton` est donc de complexité $O(NK)$.

Q21 Le chemin renvoyé par l'algorithme sera le chemin $[0, 0]$.

Cette approche n'est pas optimale car le chemin $[0, 0]$ a pour probabilité 0.3, alors que le chemin $[1, 0]$ a pour probabilité 0.36.

Q22 On cherche à maximiser le produit des probabilités des poids des arcs du chemin. Cela revient à maximiser la somme des logarithmes des poids de ces arcs et donc à minimiser l'opposé de cette somme.

En considérant le graphe pondéré obtenu en remplaçant chaque poids par l'opposé de son logarithme (les poids étant entre 0 et 1, l'opposé de leur logarithme est positif), le plus court chemin pour aller de σ à τ sera bien le chemin de probabilité maximale.

On peut donc utiliser l'algorithme de Dijkstra ou l'algorithme A* pour résoudre ce probléme.

Q23

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

Q24 En partant de la dernière colonne de `T`, on voit que le chemin de probabilité maximale se termine au sommet `0`. Puis en regardant dans la dernière colonne de `argT`, on voit que le dernier arc parcouru par le chemin optimal provient du sommet `0`. Ensuite, il suffit de remonter la matrice `argT` en suivant les sommets : `1`, `1`, `2`, `0`, `0`, `2`.

On inverse alors le chemin ce qui donne `[2, 0, 0, 2, 1, 1, 0, 0]`.

Q25 Pour la complexité temporelle, elle est de $O(NK^2)$ et la complexité spatiale de $O(NK)$.