DS2 PCE

Solution. □

Durée de l'épreuve : 2 heures

L'usage de la calculatrice ou de tout autre dispositif électronique est interdit.

Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.

Rappels concernant le langage Python : il n'est pas possible d'utiliser des fonctions internes à Python sur les listes ou les chaînes de caractères telles que `min`, `max`, `count`, `remove` ... Seules les instructions basiques telles que `len(liste)`, `liste.append(e)` sont autorisées. Le tranchage (*slicing*) ainsi que la concaténation sont également permis. Les programmes doivent être commentés lorsque c'est nécessaire pour justifier les choix. Il n'est pas utile de rappeler la signature des fonctions demandées dans les différentes questions.

Erreur d'énoncé Il y avait deux erreurs dans le sujet de DS que j'ai distribué samedi 29 novembre :

- la première, signalée par un camarade, n'est pas de mon fait, elle était déjà présente dans le sujet des Mines initial : un `E[Obs[0][i]]` au lieu d'un `E[Obs[0]][i]`. Cela rendait incompréhensible (et faux) le code de `initialiserGlouton` ;
- la seconde erreur est entièrement de ma faute : j'ai oublié un paragraphe important juste avant la question 16 qui a sans aucun doute empêché beaucoup d'entre vous de répondre correctement aux questions de la fin.

Les deux erreurs observées figurent en rouge dans ce document. Mes plus plates excuses accompagnent la présente mise au point.

Solution proposée Je vous propose de traiter la seconde partie (donc à partir de la question Q16) chez vous ; disons en une heure. J'intégrerai vos réponses à la note finale obtenue au DS2.

De mon côté, je corrigerai les copies ramassées le 29 novembre jusqu'à la question 15 incluse sans consulter les autres.

Vous déposerez vos copies scannées lisiblement avant le jeudi 11 décembre 23h45 sur Cahier de Prépa. Je peux aussi accepter des copies papier (format A4 avec nom et numéo de pages) rendues jeudi 11 novembre à 14h10.

Introduction à deux problèmes en communication numérique

Introduction. On s'intéresse au problème de communication entre deux personnes, nommées **Alice** et **Bob** qui cherchent à s'envoyer un message au travers d'un canal de communication (une bande de fréquences radio par exemple).

Avant d'être lu par Bob, le message original d'Alice passe par plusieurs étapes que nous allons séparer de la manière suivante :

- une **phase de compression**, durant laquelle Alice cherche à trouver la représentation la plus compacte possible du message,
- une **phase d'encodage** durant laquelle le message compressé est transformé en une succession de symboles transmissibles au travers du canal utilisé,
- une **phase de transmission** durant laquelle le message encodé circule sur le canal de communication et est susceptible de subir une altération,
- une **phase de décodage** durant laquelle Bob décode le message qu'il a reçu, le message lui apparaît alors sous la forme compressée,
- une **phase de décompression** durant laquelle Bob applique l'opération réciproque de la compression opérée par Alice.

Ce modèle est décrit par le schéma de la figure 1.

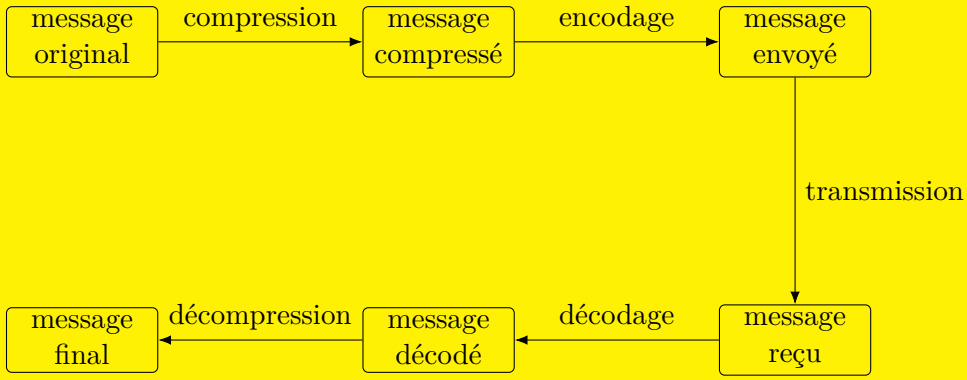


FIGURE 1 – Schématisation du modèle de communication considéré ici.

Dans cette épreuve, nous allons nous intéresser uniquement à deux phases : la **compression** du message d'origine par Alice en un message compact et le **décodage** par Bob d'un message transmis, potentiellement entaché d'erreurs.

1 Compression du message d'Alice : codage arithmétique

Le codage ASCII (American Standard Code for Information Interchange) définit une norme où 128 caractères sont codés sur 7 bits. Ce codage est illustré par les quelques lignes suivantes :

Caractère	Codage binaire	Équivalent numérique
'a'	110 0001	97
'b'	110 0010	98
'z'	111 1010	122

Lorsqu'une chaîne de caractères n'utilise pas l'intégralité des 128 caractères proposés par le codage ASCII, il est possible de convenir d'une représentation différente, plus économique en nombre de symboles. Considérons la chaîne de caractères `s='abaabaca'`. On peut proposer de coder celle-ci à l'aide du tableau suivant :

Caractère	Code
'a'	00
'b'	01
'c'	10

Dans ce cas, la chaîne `s` est codée sur 16 bits par :

00 01 00 00 01 00 10 00

(où des espaces ont été introduits pour faciliter la lecture).

Dans un souci de compression de l'information, il est intéressant de représenter les caractères les plus fréquents par des expressions courtes et de ne plus nécessairement coder avec des codes de longueur constante chaque caractère. Dans l'exemple précédent, il est possible de coder le caractère `'a'` avec 1 bit, et les caractères `'b'` et `'c'` avec 2 bits afin de coder la chaîne `s` sur seulement 11 bits en tout.

Question 1. Codes à longueur variable

Proposer une telle représentation en expliquant pourquoi celle-ci pourra être décodée sans ambiguïté. Vous ferez en sorte que la représentation binaire de `'a'` soit inférieure à celle de `'b'`, elle-même inférieure à celle de `'c'`.

Solution. On code a comme 0, b avec 10 et c avec 11.

De la sorte, le code obtenu est dit *préfixe* : aucun mot n'est le préfixe d'un autre.

Lorsque la lecture de la chaîne codée repère le code d'une lettre x, on sait qu'il faut ajouter x au texte décodé (le code de x n'étant pas le début d'un autre). Le chiffre suivant le code de x est le début du code d'une AUTRE lettre.

□

La représentation précédente emploie la même longueur pour coder les caractères `'b'` et `'c'` alors que le caractère `'b'` est deux fois plus présent que `'c'` dans la chaîne `s`. Il est possible d'aller un cran plus loin et le *codage arithmétique* présenté dans cette étude permet un gain de compression comme s'il parvenait à représenter un caractère avec un nombre non entier de bits au prorata de sa fréquence d'apparition. Ce principe est notamment utilisé par la norme JPEG2000 (ici, nous nous limitons aux chaînes de caractères).

1.1 Analyse du texte source

L'objet de cette partie est d'analyser le contenu d'une chaîne de caractères `s` afin de déterminer :

— les caractères utilisés par la chaîne `s` ;

— le nombre d'occurrences de chacun.

Question 2. Compter les occurrences d'un caractère

Écrire une fonction nommée `nbCaracteres(c:str, s:str) -> int` qui prend comme argument un caractère `c`, une chaîne `s` et qui renvoie le nombre d'occurrences (c'est-à-dire le nombre d'apparitions) de `c` dans `s`. La fonction doit avoir une complexité linéaire en `n` (la longueur de `s`).

Solution. Code

```
1 def nbCaracteres(c:str,s:str)->int:
2     return len([char for char in s if char == c])
```

□

Question 3.

Pour déterminer la liste des caractères utilisés à l'intérieur d'une chaîne `s`, on utilise la fonction :

```
def listeCaracteres(s: str):
    listeCar = []
    n = len(s)
    for i in range(n):
        c = s[i]
        if not (c in listeCar):
            listeCar.append(c)
    return listeCar
```

Que renvoie cette fonction lorsque `s='abaabaca'` ? Expliquer succinctement le principe de fonctionnement de cette fonction.

Solution. Voici

```
s='abaabaca'
listeCaracteres(s)
```

```
['a', 'b', 'c']
```

□

Question 4. Complexité de `listeCaracteres`

En fonction de la longueur `n` de la chaîne et du nombre `k` de caractères distincts dans celle-ci, déterminer la complexité asymptotique (pire cas) de la fonction de la question Q3. Par exemple pour `s='abaabaca'`, on a `n=8` et `k=3`. On négligera la complexité des `append` mais pas celle des tests d'appartenance `i in L` (`append` considéré constant, le test d'appartenance linéaire en la longueur de `listeCar`).

Solution. n passages dans la boucle externe, avec, à chaque tour, au plus k passages dans la boucle interne.

Complexité : $O(nk)$

□

Question 5.

On définit la fonction `analyseTexte(s:str) -> list`

```
def analyseTexte(s: str):
    R = []
    l  = listeCaracteres(s)
    for i in range(len(l)):
        c = l[i]
        R.append((c, nbCaracteres(c, s)))
    return R
```

Expliquer ce que fait cette fonction et donner la valeur renvoyée par la commande `analyseTexte('babaaaabca')`.

Solution. Cette fonction renvoie une liste de tuples (caractère, nombre d'occurrences de ce caractère dans la liste).

Ainsi on obtient `[('a',5),('b',2),('c',1)]`

Notes : Q5a, Q5b □

Question 6. Complexité de `analyseTexte`

En fonction de la longueur `n` de `s` et du nombre `k` de caractères distincts présents dans `s` (autrement dit `k = len(listeCaracteres(s))`), donner une estimation de la complexité asymptotique (pire cas) de la fonction `analyseTexte`.

Solution. La construction de `l` est en $O(nk)$.

Il y a k passages en ligne 4. Pour chacun on fait un appel à `nbCaracteres` en $O(n)$. Cela fait $O(nk)$ pour la boucle en L4.

Au total $O(nk) + O(nk) = O(nk)$

□

Question 7.

Adapter la fonction de la question Q5 pour qu'elle utilise (et renvoie) un **dictionnaire**. Elle devra avoir une complexité :

1. linéaire en la longueur `n` de `s`,
2. indépendante de `k`, le nombre de caractères distincts.

De plus, cette fonction ne devra parcourir **qu'une seule fois** la chaîne `s`. On admet qu'un test d'appartenance d'une clé à un dictionnaire se fait à coût constant. Exemple attendu : `analyseTexte('abracadabra')` renvoie `{'a':5,'b':2,'r':2,'c':1,'d':1}`.

Solution. code

```
1 def analyseTexte(s:str):
2     R = {}
3     for i in range(len(s)):
4         c = s[i]
5         if c in R:
6             R[c]+=1
7         else:
8             R[c]=1
9     return R
```

□

1.2 Exploitation d'analyses existantes

On peut éviter de réaliser une analyse de la chaîne de caractères à coder en exploitant des données statistiques disponibles. La numérisation de larges corpus littéraires a permis la création de bases de données contenant des informations relatives au nombre d'occurrences des différents caractères alphanumériques dans diverses langues. Nous allons supposer disposer d'une base de données constituée de trois tables (les clefs primaires sont soulignées) : :contentReference[oaicite :1]index=1

- `caractere(idCar, symbole, typeCaractere, codeHTML);`
- `corpus(idLivre, titre, auteur, annee, nombreCaracteres, langue);`

— occurrences(idCar, idLivre, nombreOccurrences).
dont voici quelques extraits :

Table caractere :

idCar	symbole	typeCaractere	codeHTML
65	'A'	'lettre'	'A'
48	'0'	'nombre'	'0'

Table corpus :

idLivre	titre	auteur	annee	nombreCaracteres	langue
1	'Germinal'	'Zola'	1885	1152365	'Français'
2	'Les Misérables'	'Hugo'	1862	2245300	'Français'

Table occurrences :

idCar	idLivre	nombreOccurrences
62	31	155
37	21	1550

Question 8.

Écrire en langage SQL une requête permettant d'obtenir la **liste sans doublon des auteurs** présents dans la table corpus.

Solution. Code

```
1    SELECT DISTINCT auteur FROM corpus
```

□

Question 9.

Écrire une requête SQL permettant d'obtenir la **fréquence d'occurrences** (donc comprise entre 0 et 1) de chaque caractère en 'Français'. Plus précisément, la requête devra renvoyer une table à deux attributs : *une colonne contenant le symbole de chaque caractère*, et *une autre colonne contenant le rapport entre le nombre total d'occurrences du caractère et le nombre total de caractères des textes de tout le corpus.*

Solution.

On calcule à part la somme des nombres de caractères des livres en Français et on la met en second membre d'un produit cartésien (B.total).

Pour le premier membre du produit cartésien, on regroupe les caractères trouvés dans les livres en Français par identifiant de caractère (GROUP BY). l'opération nécessite plusieurs jointures.

La fréquence cherchée pour un caractère donné est la somme des nombres d'occurrences pour ce caractère (nboc) dans les livres Français où il apparait divisée par la somme totale (B.total) des nombres de caractères.

Une coercion de type un peu technique (on ne veut pas une division euclidienne mais une division par un flottant) est nécessaire

```
SELECT A.symb, A.nboc / (B.total + 0.0) FROM
    (SELECT car.symbole AS symb, SUM(oc.nombreOccurrences) AS nboc
    FROM caractere AS car
    INNER JOIN occurrences AS oc
    ON oc.idCar = car.idCar
    INNER JOIN corpus AS cor
    ON cor.idLivre=oc.idLivre
    WHERE cor.langue = 'Français'
    GROUP BY car.idCar) AS A,
    (SELECT SUM(nombreCaracteres) AS total FROM corpus
    WHERE langue = 'Français') AS B
```

□

1.3 Compression

La compression par codage arithmétique consiste à représenter une chaîne de caractères `s` par un nombre réel déterminé à l'intérieur de l'intervalle `[0; 1[`.

Initialement, on attribue à chaque caractère utile une portion de l'intervalle `[0; 1[` proportionnelle à sa fréquence d'occurrences. Par exemple, pour un alphabet à 5 lettres `'abcde'`, on pourrait avoir un tableau comme ci-dessous :

Caractère	'a'	'b'	'c'	'd'	'e'
Fréquence	0.2	0.1	0.2	0.4	0.1
Intervalle	[0 ;0.2[	[0.2 ;0.3[	[0.3 ;0.5[	[0.5 ;0.9[	[0.9 ;1[

La chaîne de caractères `s` est codée en partant de l'intervalle `[0; 1[`. À chaque caractère successif de celle-ci, on affine cet intervalle en ne considérant que la portion correspondant au caractère lu.

Si par exemple la chaîne à coder est `s='dac'` :

— on obtient d'abord l'intervalle `[0.5; 0.9[` correspondant au caractère `'d'` ;

— le caractère `'a'` détermine alors le sous-intervalle `[0.50; 0.58[` de `[0.5; 0.9[` correspondant à la portion associée au caractère `'a'` ;

— le caractère `'c'` détermine enfin l'intervalle `[0.524; 0.540[`.

La figure qui suit illustre ce processus (Figure 2 : Encodage de `s='dac'`).

Question 10.

En considérant la table des fréquences précédente, proposer l'intervalle correspondant à la chaîne `s='bac'`.

Solution. Séquentiellement

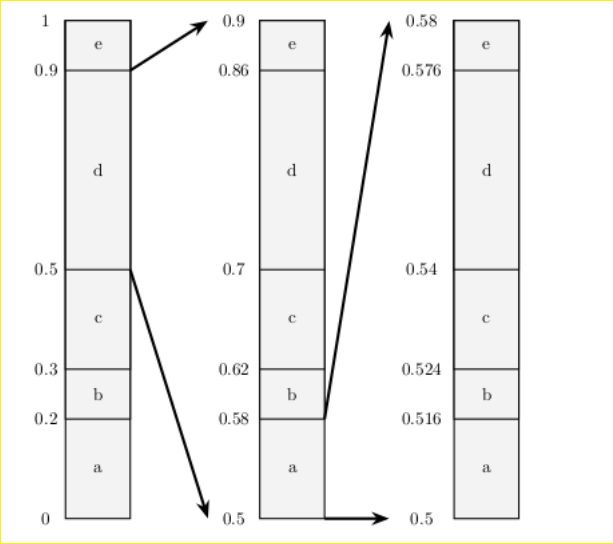


FIGURE 2 – Encodage de `'dac'`

— le premier intervalle est celui donné par `b` : $[0.2; 0.3[$, de largeur 0.1.
— On a $0.1 \times 0.2 = 0.02$, donc `ba` est encodé par $[0.2 + 0; 0.2 + 0.02[= [0.2, 0.22[$ de largeur 0.02.
— Ensuite, on lit `c` : $0.3 \times 0.02 = 0.006$ et $0.5 \times 0.02 = 0.01$. Donc `bac` est encodé par $[0.2 + 0.006; 0.2 + 0.1[= [0.206; 0.21[$

□

On dispose d'une fonction `codeCar(car:str, g:float, d:float) -> (float, float)` qui prend en argument un caractère `car` et les deux extrémités d'un intervalle `[g, d[` et qui renvoie un tuple composé des extrémités du sous-intervalle de `[g, d[` déterminé par le caractère `car`. En reprenant l'illustration précédente, `codeCar('d', 0, 1)` produit `(0.5, 0.9)` et `codeCar('a', 0.5, 0.9)` produit `(0.5, 0.58)`. On ne demande pas d'écrire cette fonction.

Question 11.

Écrire une fonction `codage(s:str) -> (float, float)` prenant en argument la chaîne `s` et fournissant en réponse le tuple `(g, d)` constitué des deux extrémités de l'intervalle `[g, d[` produit par l'algorithme de codage précédent.

Solution. Code

```
def codage(s: str) -> (float, float):
    g, d = 0.0, 1.0
    for car in s:
        g, d = codeCar(car, g, d)
    return (g, d)
```

□

Le codage arithmétique consiste alors à coder la chaîne `s` par un flottant `x` choisi arbitrairement à l'intérieur de l'intervalle `[g, d[`.

1.4 Décodage

Pour effectuer le décodage d'un flottant `x`, il suffit de repérer dans quelle succession d'intervalles celui-ci se trouve. À titre d'exemple, reprenons le tableau précédent et considérons le nombre `x=0.123` :

Caractère	'a'	'b'	'c'	'd'	'e'
Fréquence	0.2	0.1	0.2	0.4	0.1
Intervalle	[0 ;0.2[	[0.2 ;0.3[	[0.3 ;0.5[	[0.5 ;0.9[	[0.9 ;1[

Puisque x appartient à l'intervalle [0 ; 0.2[, le premier caractère est un `'a'`. Puisque x appartient au sous-intervalle [0.10 ; 0.18[, le caractère suivant est un `'d'`, etc.

Question 12.

Déterminer le caractère qui suit `'ad'` dans la chaîne codée par `x=0.123` en spécifiant le sous-intervalle qui a permis de décoder ce caractère.

Solution. Au départ g et d valent 0 et 1

- d'après le tableau des fréquences cumulées `[0,0.2,0.3,0.5,0.9,1]`, x est dans le 1er intervalle, lequel correspond à un `'a'`. La largeur de l'intervalle est 0.2
- Le tableau des fréquences cumulées devient `0+[0. , 0.04, 0.06, 0.1 , 0.18, 0.2 ]`. Alors x est dans le 4ème intervalle $[0.1; 0.18[$, donc on a un `'d'`. La largeur est 0.08.
- Le tableau des fréquences cumulées devient `0.1+[0. , 0.016, 0.024, 0.04 , 0.072, 0.08 ]` soit `[0.1 , 0.116, 0.124, 0.14 , 0.172, 0.18 ]`. Alors x est dans l'intervalle $[0.116; 0.124[$ qui correspond à un `'b'`.
  Notes Q12a,Q12b

□

Question 13.

Dans le cadre de l'exemple de cette partie, indiquer deux chaînes qui peuvent correspondre au flottant `0.2`. Expliquer par une phrase ce qui est à l'origine de cette ambiguïté.

Solution. `"ba"` et `"b"` aboutissent au même codage 0.2.

Si s s'encode dans l'intervalle $[x, y[$, alors $s{+}a$ aussi (si a est le premier symbole de l'alphabet).

Le problème vient donc du premier caractère de l'alphabet (puisqu'on ajoute 0 à la borne inf de l'intervalle courant). La borne inf du nouvel intervalle est égale à celle de l'intervalle précédent.

Il faut un caractère pour indiquer la fin d'une chaine.

Notes Q13a, Q13b □

Une solution possible pour résoudre le problème précédent consiste à introduire un caractère nouveau signifiant la fin de la chaîne de caractères. Nous conviendrons de désigner ce caractère par `'#'`. Ce caractère se voit attribuer une plage non vide au voisinage de 0. Dans la suite, on suppose que la table des fréquences est adaptée de sorte à prendre en compte la présence de ce nouveau caractère.

On suppose disposer, en plus de la fonction `codeCar(car,g,d)` précédente, d'une fonction `decodeCar(x:float, g:float, d:float) -> str` qui détermine le caractère correspondant à la valeur `x` quand celle-ci est comprise dans la plage `[g, d[`. Par exemple `decodeCar(0.123, 0, 1)` donne `'a'` tandis que `decodeCar(0.123, 0, 0.2)` donne le caractère `'d'`.

Question 14.

Écrire une fonction `decodage(x:float) -> str` produisant la chaîne de caractères `s` déterminée par la valeur de `x` (avec le caractère `'#'` compris).

Solution. Code

```
def decodage(x: float) -> str:
    s = ""
    g, d = 0.0, 1.0
    while True:
        c = decodeCar(x, g, d)   # renvoie un caractère selon [g, d[
        s += c
        if c == '#':
            break
        g, d = codeCar(c, g, d)  # restreint l'intervalle pour la suite
    return s
```

□

Remarque. Le codage précédent ne fonctionne que sous l'hypothèse illusoire d'exactitude des calculs jusqu'à une très grande précision. Cette précision est limitée par la taille de la mantisse représentant un réel ; il ne peut être mis en œuvre sous la forme précédente qu'avec des chaînes relativement courtes. Pour résoudre cette difficulté, on adopte un codage en arithmétique entière : au lieu de considérer un intervalle flottant $[0\,;1[$, c'est un intervalle entier $[\![000, 999]\!]$ qui est utilisé. Cette technique ne sera pas abordée dans le cadre de cette épreuve.

1 Décodage du message reçu par Bob à l'aide de l'algorithme de Viterbi

1.1 Modélisation du canal de communication par un graphe

Dans cette partie, nous allons désormais considérer que le message compressé par Alice a été envoyé au travers d'un canal de communication. À cette fin, et indépendamment de la phase de compression étudiée dans la première partie, le message a subi une deuxième phase de transformation, dite d'encodage (cf 1). Le message envoyé sur le canal est une suite de symboles à valeurs dans un alphabet, noté Σ, comportant K symboles. Le choix d'un alphabet efficace n'est pas l'objet de notre étude et constitue un sujet à part entière. Suite au passage dans le canal de communication, le message envoyé subit une altération de sorte que Bob reçoit une séquence de symboles de Σ qui ne correspond pas nécessairement à celle qui a été émise.

Dans cette partie, nous allons voir une approche permettant à Bob de décoder le message reçu et de potentiellement corriger quelques erreurs liées à la transmission du message et à la connaissance a priori de la propension du canal de communication à altérer les symboles du message lors de la transmission.

La modélisation proposée est la suivante :

— Bob observe une suite de N symboles $obs_0, \ldots, obs_{N-1}$, que nous allons représenter par une liste Python `Obs`.

— pour simplifier, on supposera que l'alphabet Σ est un ensemble de K entiers consécutifs commençant à 0, de sorte que $\Sigma = [\![0, K-1]\!]$. Par exemple si $K = 3$ et $N = 8$, un message valide reçu par Bob pourrait être `[2,0,0,2,1,1,0,0]`.

— chacun des symboles observés obs_t correspond à l'altération d'un symbole s_t envoyé par Alice. On note $[s_0, \dots, s_{N-1}]$ le message original ; pour reprendre l'exemple précédent, Alice pourrait avoir envoyé `[2,0,0,2,1,1,2,0]`.

— on connaît, pour chaque paire $(i, j) \in \Sigma^2$, la probabilité $E_{i,j}$ que le canal altère le symbole j en un symbole i. On stocke ces probabilités dans une liste de listes `E` ; autrement dit, `E[i][j]` est la probabilité conditionnelle d'observer le symbole i sachant que le symbole j a été émis.
Ici (par exemple) on pourrait considérer

$$E = \begin{pmatrix} 0.7 & 0.2 & 0.3 \\ 0.2 & 0.7 & 0.1 \\ 0.1 & 0.1 & 0.6 \end{pmatrix}$$

représentée par la liste de listes : E = `[[0.7,0.2,0.3],[0.2,0.7,0.1],[0.1,0.1,0.6]]`.
Le fait que `E[2][0]` vaille 0.1 signifie donc que la probabilité que le symbole observé par Bob soit un 2 sachant qu'Alice a émis un 0 est de 0.1.

— on suppose également que le symbole courant s_t envoyé par Alice a une incidence sur le symbole suivant s_{t+1} qu'elle peut envoyer, au même titre que dans une langue comme le français, la probabilité d'observer un 't' dans un mot, n'est pas la même suivant que le caractère précédent est un 'e' ou un 'z'.
Ainsi pour chaque couple de symboles $(i, j) \in \Sigma^2$, on suppose que l'on connaît la probabilité d'émettre le symbole j à l'instant $t+1$ sachant que symbole i a été émis à l'instant t. On suppose également que cette probabilité ne dépend pas de t.
L'information concernant ces probabilités de transition d'un symbole à l'autre peut se stocker dans une matrice P de taille $K \times K$, que l'on représente informatiquement par une liste de listes `P`. Chaque entrée `P[i][j]` donne la probabilité qu'Alice émette le symbole j à l'instant $t+1$ sachant qu'elle a émis le symbole i à l'instant t. En d'autres termes, `P[i][j]` désigne $\mathbb{P}(s_{t+1} = j \mid s_t = i)$.
On prendra ici à titre d'exemple :

$$P = \begin{pmatrix} 0.3 & 0.2 & 0.5 \\ 0.4 & 0.4 & 0.2 \\ 0.2 & 0.3 & 0.5 \end{pmatrix}$$

représentée par la liste 0.2 0.3 0.5 de listes :
`P=[[0.3,0.2,0.5],[0.4,0.4,0.2],[0.2,0.3,0.5]]`.
Le fait que `P[2][0]` vaille 0.2 signifie donc que la probabilité que le symbole envoyé par Alice à l'instant $t+1$ soit un 0 sachant que celui envoyé à l'instant t est un 2 vaut 0.2.

Nous allons désormais nous intéresser au problème du décodage : étant donné la liste $Obs = [obs_0, \dots, obs_{N-1}]$ des symboles observés par Bob, quelle séquence $s'_0, \dots, s'_{N-1}$ est la plus probable ? En d'autres termes, $s'_0, \dots, s'_{N-1}$ est l'estimation la plus probable faite par Bob du message d'origine $s_0, \dots, s_{N-1}$, étant donné les observations $obs_0, \dots, obs_{N-1}$.

La modélisation précédente peut se représenter à l'aide d'un graphe défini comme suit (voir 3 pour un exemple) :

— on crée un sommet $S_{i,j}$ pour chaque symbole possible $0 \leq i \leq K-1$ et chaque indice d'observation $0 \leq j \leq N-1$. Chaque couche verticale dans le graphe correspond à un caractère dans le message. Chaque strate horizontale correspond à un symbole.

— au niveau de la j-ème couche verticale, les sommets $S_{i,j}$ pour $j < N-1$ ont pour successeurs les états $S_{k,j+1}$ pour tous les symboles k possibles,

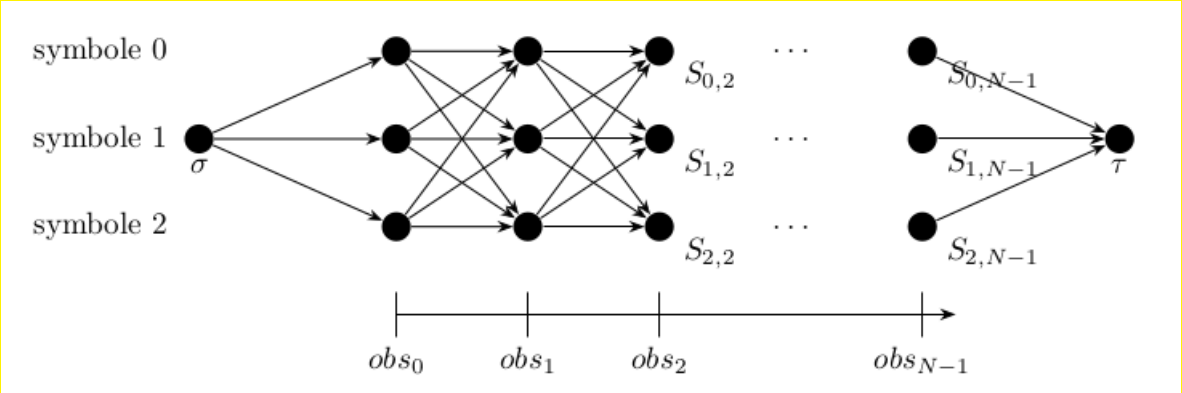


FIGURE 3 – Illustration du modèle de décodage considéré ici.

— par commodité, on ajoute un état source σ correspondant au début du message décodé et un état cible τ correspondant à la fin du message, ces états étant respectivement reliés à la première et la dernière couche,
— le décodage du message envoyé par Alice correspond à un chemin entre σ et τ dans ce graphe. A chaque sommet du chemin correspond une lettre décodée. Par exemple, le chemin passant par $S_{0,0}, S_{2,1}, S_{0,2}, S_{1,3}$, correspond au décodage de `[0, 2, 0, 1]`.

Question 15.

En fonction de N et de K, donner le nombre de sommets et d'arcs du graphe illustré par la figure 3. On ne comptera pas les sommets source σ et cible τ, ni les arcs partant du sommet source σ ni ceux arrivant à la cible τ.

Solution. Il y a NK sommets si on compte pas les sommets source σ et cible τ.

Pour chaque sommet sauf pour les K de la dernière colonne (sans voisin), il y a K arcs issus de ce sommet. Cela nous donne donc $(N-1)K^2$ arcs si on ne compte les arcs partant du sommet source ni ceux arrivant à la cible.

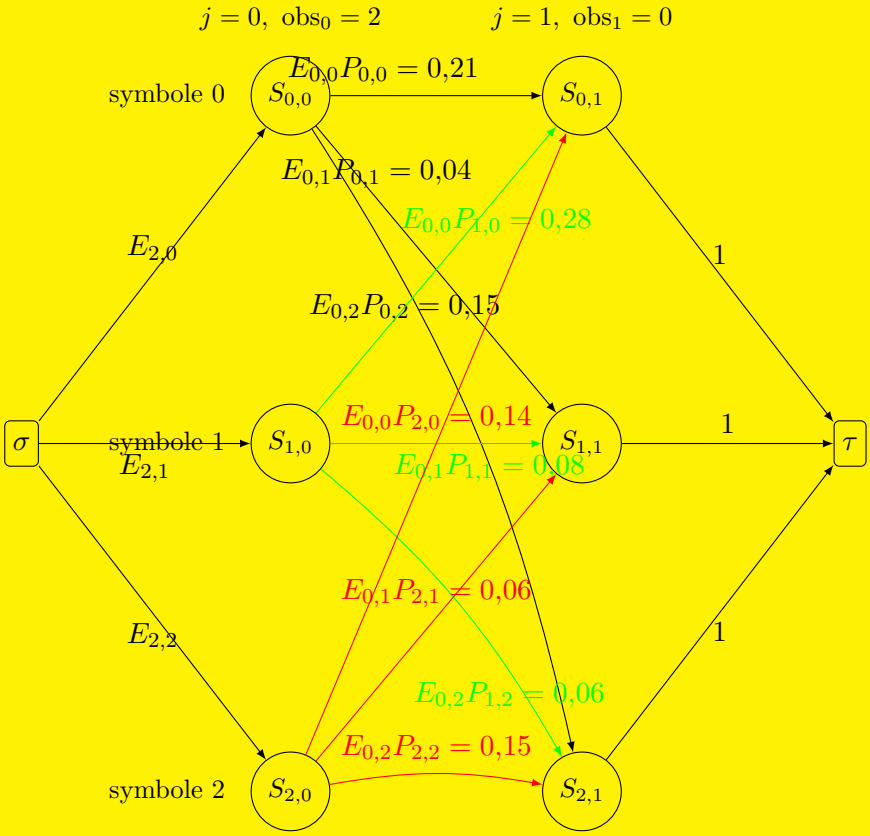
Notes Q15a, Q15b □

Paragraphe oublié par M. Noyer On choisit désormais de pondérer chaque arc par la probabilité de transiter par cet arc. Autrement dit :
— les arcs issus de la source σ vers $S_{i,0}$ sont pondérés par $E_{obs_0,i}$ la probabilité d'observer le symbole obs_0 sachant que le symbole i a été émis par Alice ;
— les arcs arrivant à la cible τ sont pondérés par 1 (en fin de message, on transite forcément vers l'état final) ;
— les arcs internes entre $S_{i,j}$ et $S_{k,j+1}$ sont pondérés par la probabilité $E_{obs_{j+1},k}P_{i,k}$. La probabilité d'un chemin $\sigma S_{i_0,0}S_{i_1,1}\dots S_{i_{N-1},N-1}\tau$ entre σ et τ est le produit des probabilités des arcs qui le composent.
L'objectif va être de trouver le chemin de probabilité maximale dans ce graphe entre le sommet source σ et le sommet cible τ.
FIN DU PARAGRAPHE OUBLIÉ PAR M.NOYER.

Question 16. Construction de graphe

On suppose que Bob a observé la séquence `[2,0]`. En utilisant les matrices E et P données dans l'énoncé (avec $K = 3$), construire le graphe pondéré associé à ce message de longueur $N = 2$. Les arcs entre les sommets devront être pondérés par les probabilités correspondantes.

Solution. Voici



□

Question 17.

On revient dans le cas général, N et K sont désormais quelconques. Indiquer combien il existe de chemins entre σ et τ (un ordre de grandeur utilisant la notation O ou Θ est accepté). Préciser si un algorithme d'exploration exhaustive est envisageable dans ce cas.

Solution. Un chemin est un N-uplet, chaque item pouvant prendre K valeurs.
Il y a K^N chemins possibles. Exponentiel. Pas gérable pour de longs messages. □

1.2 Stratégie gloutonne

Pour pouvoir implémenter correctement la recherche du chemin de probabilité maximale, il est utile de disposer d'une fonction auxiliaire qui sera utilisée dès que nécessaire.

Question 18.

Pour une liste `liste`, on appelle *argument du maximum* et on note `argMax` tout indice i tel que `liste[i]` soit maximal.

Proposer une fonction `maximumListe(liste:[float])->(float,int)` qui prend en entrée une liste de nombres et qui renvoie la valeur du maximum de la liste ainsi que le plus petit argument du maximum, i.e. le premier indice auquel cette valeur maximale apparaît.

Solution. Code

```
1  def maximumListe(liste):
2      i, M = 0, liste[0]
3      for j in range(len(liste)):
```

```
        if liste[j] > M:
            M,i = liste[j], j
    return M,i
```

□

On souhaite appliquer un algorithme glouton pour trouver le chemin de probabilité maximale entre le sommet source σ et le sommet cible τ. On rappelle qu'un algorithme glouton cherche, à chaque étape, à faire le choix localement optimal. Ici, si à une étape on se retrouve au sommet $S_{i,j}$, il s'agit de choisir l'arc de plus forte probabilité partant de ce sommet.

Dans un premier temps on se donne une fonction `initialiserGlouton(Obs:[int], E:[[float]], K:int)->int` qui permet d'initialiser l'algorithme glouton en trouvant le sommet le plus probable parmi $S_{i,0}$ pour i variant entre 0 et $K-1$. Pour cela il faut regarder la colonne `Obs[0]` de E et relever l'indice de la plus grande valeur.

```
def initialiserGlouton(Obs, E, K):
  probasInitiales = [E[Obs[0]][i] for i in range(K)]
  s, symbole = maximumListe(probasInitiales)
  return symbole
```

NOYER : il y avait une coquille dans le sujet original des Mines que j'avais reproduite à l'identique sans y faire attention. C'est bien `[E[Obs[0]][i] for i in range(K)]` et non `[E[Obs[0][i]] for i in range(K)]`

Question 19.

Proposer une fonction `glouton(Obs:[int],P:[[float]],E:[[float]],K:int,N:int)-> list[tuple[int, int]]` qui renvoie la liste d'états (un tuple d'entiers) obtenue par l'approche gloutonne. Même si cela n'est pas nécessaire, K, N seront des arguments de cette fonction.

Solution. On renvoie une liste de tuples plutôt qu'une liste d'entiers.

Certes l'énoncé demande une liste d'entiers, mais un état est un tuple (numéro de colonne, numéro de ligne) d'après les schémas donnés.

```
def glouton(Obs:[int],P:[[float]],E:[[float]],K:int,N:int)->[int]:
    symb = initialiserGlouton(Obs, E, K)
    j=0
    res = [(symb,j)]
    while j < N-1: # N-1 passages
        probas = [E[Obs[j+1]][k] * P[symb][k] for k in range(K)] # O(K)
        _, symb = maximumListe(probas) # O(K)
        j = j+1
        res.append((symb,j))
    return res
```

□

Question 20.

En fonction de K et de N , quelle est, en ordre de grandeur, la complexité temporelle asymptotique de l'approche gloutonne ?

Solution. Il y a une initialisation en $O(K) + O(1)$.

Puis $N-1$ passages dans la boucle, chacun en $O(K)$.

On a une cpx en $O(NK)$. Quadratique vs exponentielle en force brute. □

Question 21.

Indiquer le chemin renvoyé par l'algorithme glouton appliqué à la figure 4. Conclure quant à l'optimalité de l'approche.

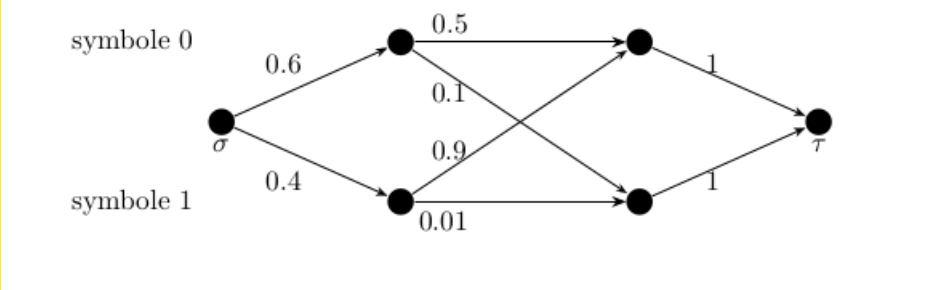


FIGURE 4 – Que donne l'algorithme glouton ici ?

Solution. L'algorithme glouton renvoie $S_{0,0}, S_{0,1}$ pour un poids de $0.6 \times 0.5 = 0.3$ alors qu'un meilleur chemin est $S_{0,0}, S_{0,1}$ pour un poids de $0.4 \times 0.9 = 0.36$.

Donc l'algorithme glouton n'est pas optimal. □

1.3 Stratégie de programmation dynamique

Question 22.

Expliquer en quoi rechercher un chemin de probabilité maximale pourrait se transformer en un problème de recherche de plus court chemin dans un graphe pondéré à poids positifs. Préciser alors quel algorithme pourrait être utilisé.

Solution. Q22a et Q22b

Ramener la recherche d'un produit maximum (problème actuel) à celle d'une somme minimale de poids des arcs (recherche de PCC) demande de passer par les logarithmes des inverses de probas. Analysons :

Un chemin dont le produit $p = \Pi_{i=0}^{N-1} p_i$ de la valeur des arcs est maximum est aussi un chemin dont l'inverse du produit de la valeur des arcs est minimum.

Il s'avère (par croissance du logarithme) que c'est aussi un chemin dont le logarithme de l'inverse du produit de la valeur des arcs est minimum. Or,

$$\log(\frac{1}{p}) = -\sum_{i=0}^{N-1} \log(p_i) = \sum_{i=0}^{N-1} (-\log(p_i))$$

et chaque $-\log(p_i)$ est positif puisque les probabilités sont comprises entre 0 et 1.

Donc il suffit de rechercher par Dijkstra le PCC dans un graphe dont les sommets et les arcs sont les mêmes mais dont les valuations des arcs sont les opposés des logarithmes des probabilités/valuation du graphe initial.

Observons que si certaines probabilités sont nulles, les valuations correspondantes dans le nouveau graphe sont infinies : cela correspond à une absence d'arc avec les conventions du cours. □

Les algorithmes évoqués à la question précédente ne sont cependant pas optimaux dans ce cas. L'algorithme optimal est dû à Andrew Viterbi et date de 1967. Il repose sur le paradigme de la programmation dynamique.

On appelle $T_{i,j}$ la valeur de probabilité maximale entre la source et l'état $S_{i,j}$. On peut alors établir l'équation de programmation dynamique suivante :

$$\begin{cases} T_{i,j} &= \max_{k\in[\![0,K-1]\!]} \left\{T_{k,j-1} \times P_{k,i} \times E_{obs_j,i}\right\} \text{ si } N-1 \geq j > 0 \\ T_{i,0} &= E_{obs_0,i} \end{cases}$$

Comme on cherche également à obtenir la valeur des états correspondant au chemin optimal, on maintient également le tableau des prédécesseurs suivant :

$$\begin{cases} argT_{i,j} &= \texttt{Argmax}_{k\in[\![0,K-1]\!]} \left\{T_{k,j-1} \times P_{k,i} \times E_{obs_j,i}\right\} \text{ si } N-1 \geq j > 0 \\ argT_{i,0} &= -1 \end{cases}$$

La valeur -1 du deuxième tableau est purement conventionnelle et ne sert qu'à représenter l'état source qui ne correspond pas à une observation.

On se donne une fonction `initialiserViterbi(E:[[float]], Obs0:int, K:int, N:int)->([[float]],[[int]])` qui prend en entrée la matrice `E` d'émission, la valeur de la première observation `Obs0`, le nombre d'éléments K de Σ, le nombre d'observations N et qui renvoie deux tableaux (liste de listes) `T` et `argT` vérifiant les caractéristiques suivantes :

- `T` et `argT` sont de dimensions K lignes par N colonnes,
- `T[i][0]` contient la valeur de $E_{obs_0,i}$,
- `argT[i][0]` contient la valeur de -1,
- les autres valeurs de `T` et `argT` sont à 0.

```
def initialiserViterbi(E, Obs0, K, N):
    probasInitiales = [E[Obs0][i] for i in range(K)]
    T = [[0 for j in range(N)] for i in range(K)]
    argT = [[0 for j in range(N)] for i in range(K)]
    for i in range(K):
        T[i][0] = probasInitiales[i]
        argT[i][0] = -1
    return T, argT
```

Question 23.

Proposer une fonction (méthode de bas en haut de programmation dynamique)

```
construireTableauViterbi(Obs:[int], P:[[float]], E:[[float]],
  K:int, N:int) ->([[float]],[[int]])
```

qui prend comme arguments la liste des observations `Obs`, la matrice des probabilités de transition `P` et la matrice des probabilités `E` et renvoie les deux listes de listes `T` et `argT` de taille $K \times N$.

Solution. Code

```
    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(0,K):
            probas = [T[k][j-1] * E[Obs[j]][i] * P[k][i] for k in range(K)]
            #on renseigne T_{i,j} car on connait les T_{_,j-1}
```

```
            M,symb = maximumListe(probas)
            T[i][j]=M
            argT[i][j]=symb
    return T,argT
```

□

Question 24.

L'algorithme de Viterbi codé en Python et appliqué à un message en entrée donne les tableaux `T` et `argT` suivants. Indiquer la séquence d'états la plus probable.

$$T = \begin{bmatrix} 0.1 & 0.084 & 0.01764 & 0.0005292 & 0.00021168 & 8.89056 \cdot 10^{-5} & 8.7127488 \cdot 10^{-5} & 1.829677248 \cdot 10^{-5} \\ 0.1 & 0.036 & 0.0054 & 0.000405 & 0.00111132 & 0.0003111696 & 2.4893568 \cdot 10^{-5} & 3.48509952 \cdot 10^{-6} \\ 0.6 & 0.09 & 0.0135 & 0.005292 & 0.0002646 & 2.22264 \cdot 10^{-5} & 1.8670176 \cdot 10^{-5} & 1.30691232 \cdot 10^{-5} \end{bmatrix}$$

$$argT = \begin{bmatrix} -1 & 2 & 0 & 0 & 2 & 1 & 1 & 0 \\ -1 & 2 & 2 & 2 & 2 & 1 & 1 & 0 \\ -1 & 2 & 2 & 0 & 2 & 1 & 1 & 0 \end{bmatrix}$$

Solution. On se place sur le maximum de la dernière colonne de T. Il est sur la ligne 0 : donc la dernière lettre lue est un 0.

A partir de là, on se déplace dans le tableau $argT$ qui indique pour chaque état de quel état on vient.

— $artgT_{0,7}$ vaut 0.
— $artgT_{0,6}$ vaut 1.
— $artgT_{1,5}$ vaut 1.
— $artgT_{1,4}$ vaut 2.
— $artgT_{2,3}$ vaut 0.
— $artgT_{0,2}$ vaut 0.
— $artgT_{0,1}$ vaut 2. FIN

donc le message lu par Bob est $2, 0, 0, 2, 1, 1, 0, 0$ ce qui correspond aux états

$$S_{2,0}, S_{0,1}, S_{0,2}, S_{2,3}, S_{1,4}, S_{1,5}, S_{0,6}, S_{0,7}$$

□

Question 25.

En fonction de K et N, donner l'ordre de grandeur de la complexité temporelle de l'approche de programmation dynamique, ainsi que la complexité spatiale.

Solution. La complexité spatiale est d el'ordre de la taille des matrices donc $O(KN)$.

Complexité temporelle :

— Il y a la création des matrices avec `initialiserViterbi` en $O(KN)$,
— pour chacun des $K(N-1)$ coefficients $T_{i,j}$ des colonnes 1 à $N-1$, il faut faire K produits de 3 coefficients de matrices et choisir le plus grand. La complexité est donc un $O(K(N-1)K)$ soit $O(K^2N)$.
— Idem pour les coefficients de $argT$: $O(K^2N)$.

Donc complexité temporelle en $O(K^2N)$: beaucoup mieux que la force brute.

□