Sorbonne Université

LU3IN003 (2019/2020)

Mini Projet : Alignement de séquences

CAI Eddy & MOUKOURI Steve

Alignement de deux mots

Question 1

Montrer que si $(\bar{x}, \bar{y})$ et $(\bar{u}, \bar{v})$ sont respectivement des alignements de (x, y) et (u, v), alors $(\bar{x} \cdot \bar{u}\,, \bar{y} \cdot \bar{v})$ est un alignement de $(x \cdot u, y \cdot v)$. Justifiez votre réponse

Supposons que $(\bar{x}, \bar{y})$ et $(\bar{v}, \bar{u})$ sont respectivement des alignements de (x, y) et (u, v), c'est-à-dire :

$$\begin{cases} (i)\ \pi(\bar{x}) = x \\ (ii)\ \pi(\bar{y}) = y \\ (iii)\ |\bar{x}| = |\bar{y}| \\ (iv)\ \forall i \epsilon [1 \dots |\bar{x}|],\ \bar{x}_i \neq - \text{ ou } \bar{y}_i \neq - \end{cases} \text{ Et } \begin{cases} (i')\ \pi(\bar{u}) = u \\ (ii')\ \pi(\bar{v}) = v \\ (iii')\ |\bar{u}| = |\bar{v}| \\ (iv')\ \forall j \epsilon [1 \dots |\bar{u}|],\ \bar{u}_j \neq - \text{ ou } \bar{v}_j \neq - \end{cases}$$

Et montrons que $(\bar{x} \cdot \bar{u}\,, \bar{y} \cdot \bar{v})$ est un alignement de $(x \cdot u, y \cdot v)$, c'est-à-dire :

$$\begin{cases} (a)\ \pi(\bar{x} \cdot \bar{u}) = x \cdot u \\ (b)\ \pi(\bar{y} \cdot \bar{v}) = y \cdot v \\ (c)\ |\bar{x} \cdot \bar{u}| = |\bar{y} \cdot \bar{v}| \\ (d)\ \forall k \epsilon [1 \dots |\bar{x} \cdot \bar{u}|],\ (\bar{x} \cdot \bar{u})_k \neq - \text{ ou } (\bar{y} \cdot \bar{v})_k \neq - \end{cases}$$

On a $\pi(\bar{x} \cdot \bar{u}) = \pi(\bar{x}) \cdot \pi(\bar{u}) = x \cdot u$ (d'après (i) et (i'))

On a $\pi(\bar{y} \cdot \bar{v}) = \pi(\bar{y}) \cdot \pi(\bar{v}) = y \cdot v$ (d'après (ii) et (ii'))

On a $|\bar{x} \cdot \bar{u}| = |\bar{x}| + |\bar{u}| = |\bar{y}| + |\bar{v}| = |\bar{y} \cdot \bar{v}|$ (d'après (iii) et (iii'))

Soit k, $k \epsilon [1 \dots |\bar{x} \cdot \bar{u}|]$ c'est-à-dire $k \epsilon [1 \dots m] \cup [m + 1 \dots n]$ avec $m = |\bar{x}|$ et $n = |\bar{u}|$, montrons que $(x \cdot u)_k \neq -$ ou $(y \cdot v)_k \neq -$.

On a :

Pour $k \epsilon [1 \dots m]$ on a : $\begin{cases} (\bar{x} \cdot \bar{u})_k = \bar{x}_k \\ (\bar{y} \cdot \bar{v})_k = \bar{y}_k \end{cases}$ or d'après (iv) on a : $\bar{x}_k \neq -$ ou $\bar{y}_k \neq -$

Pour $k \epsilon [m+1 \dots n]$ on a : $\begin{cases} (\bar{x} \cdot \bar{u})_k = \bar{u}_k \\ (\bar{y} \cdot \bar{v})_k = \bar{v}_k \end{cases}$ or d'après (iv') on a : $\bar{u}_k \neq -$ ou $\bar{v}_k \neq -$

D'où le résultat attendu $\forall k \epsilon [1 \dots |\bar{x} \cdot \bar{u}|],\ (\bar{x} \cdot \bar{u})_k \neq -$ ou $(\bar{y} \cdot \bar{v})_k \neq -$

Question 2

Si $x \epsilon \sum^*$ est un mot de longueur n et $y \epsilon \sum^*$ est de longueur m, quelle est la longueur maximale d'un alignement de (x, y) ? **NB** : un maximum est un majorant atteint. Vous devez donc justifier par un exemple que la longueur que vous proposez est réalisée par au moins un alignement.

Soit x et y deux mots quelconques, on définit n = |x| et m = |y|, ce qui nous donne l'alignement maximale suivant :

$$x_1 x_2 \dots x_n - - \cdots -$$

$$- - \ \dots - y_1 y_2 \dots y_n$$

La longueur maximale d'un alignement de (x, y) est donc $n + m$.

Méthode naïve par énumération

Question 3

Etant donné $x \epsilon \sum^*$un mot de longueur n, combien y a-t-il de mots $\bar{x}$ obtenus en ajoutant à x exactement k gaps ? Autrement dit combien y a-t-il de mots $\bar{x} \epsilon \sum^*$ tels que $|\bar{x}| = n + k$ et $\pi(\bar{x}) = x$? On exprimera cette valeur sous forme d'un coefficient binomiale.

Etant donné $x \epsilon \sum^*$un mot de longueur n et en ajoutant à x exactement k gaps, on a :

$\binom{n+k}{k} = \frac{(n+k)!}{k!n!}$ Mots obtenus

Question 4

On chercher maintenant à en déduire le nombre d'alignements possibles d'un couple de mots (x, y) de longueurs respectives n et m, en supposant que $n \geq m$. Une fois ajoutés k gaps à x pour obtenir un mot $\bar{x} \epsilon \sum^*$, combien de gaps seront ajoutés à y ? Combien y a-t-il de façon d'insérer ces gaps dans y sachant qu'un gap du mot $\bar{y} \epsilon \sum^*$ ainsi obtenu ne doit pas être placé à la même position qu'un gap de $\bar{x}$? En déduire le nombre d'alignements possibles de (x, y). On ne demande pas de simplifier l'expression obtenue, mais vous calculerez sur machine, à l'aide de cette expression, le nombre d'alignements possibles pour $|\bar{x}| = 15$ et $|\bar{y}| = 10$.

Soit (x, y) deux mots de longueurs respectives n et m en supposant que $n \geq m$. Une fois ajouté k gaps à x pour obtenir $\bar{x} \epsilon \sum^*$, il faut ajouter $n - m + k$ gaps à y pour obtenir $\bar{y}$.
Il y a donc $\binom{n}{k'}$ façon d'insérer ces gaps dans y avec $k' = n - m + k$.

De ce fait, on trouve :

$\sum_{k=0}^{m} \binom{n}{k'}\binom{n+k}{k} = \sum_{k=0}^{m} \binom{n}{n-m+k}\binom{n+k}{k}$ Nombres d'alignement possibles

Exemple avec $|\bar{x}| = 15$ et $|\bar{y}| = 10$:

$$\sum_{k=0}^{m} \binom{n}{n-m+k}\binom{n+k}{k} = \sum_{k=0}^{m} \frac{n!}{(n-m+k)!\,(m-k)!} * \frac{(n+k)!}{k!\,n!}$$

$$\sum_{k=0}^{10} \binom{15}{15-10+k}\binom{15+k}{k} = \sum_{k=0}^{10} \frac{15!}{(5+k)!\,(10-k)!} * \frac{(10+k)!}{k!\,10!} = \mathbf{298\,199\,265}$$

Question 5

Quel genre de complexité temporelle aurait un algorithme naïf qui consisterait à énumérer tous les alignements de deux mots en vue de trouver la distance d'édition entre ces deux mots ? En vue de trouver un alignement de coût minimal ?

En vue de trouver la distance d'édition entre deux mots et de trouver un alignement de coût minimal, il faut dans un premier temps énumérer tous les alignements de mots, puis dans un second temps calculer leurs coûts respectifs afin de trouver le coût minimal. La complexité dépend donc de l'algorithme d'énumération et ainsi de l'algorithme de tri. La complexité finale sera donc en $O\left((m+1) * \left(\frac{n!}{(n-m+k)!(m-k)!}\right)\right)$.

Question 6

Quelle complexité spatiale (ordre de grandeur de la place requise en mémoire) aurait un algorithme naïf qui consisterait à énumérer tous les alignements de deux mots en vue de trouver la distance d'édition entres ces deux mots ? en vue de trouver un alignement de coût minimal ?

Il faut des tableaux à une dimension pour stocker l'alignements de deux mots et un tableau à une dimension pour stocker les coûts de chaque alignement, la complexité spatiale est donc en $O(n)$

Programmation dynamique

Question 7

Soit $(\bar{u} \cdot \bar{v})$ un alignement de $(x_{[1\ldots i]}, y_{[1\ldots j]})$ de longueur l, Si $\bar{u}_l = -$, que vaut $\bar{v}_l$? Si $\bar{v}_l = -$, que vaut $\bar{u}_l$? Si $\bar{u}_l \neq -$ et $\bar{v}_l \neq -$, que valent $\bar{u}_l$ et $\bar{v}_l$? Justifiez rapidement.

Si $\bar{u}_l = -$, $\bar{v}_l$ est une lettre dans $\sum$

Si $\bar{v}_l = -$, $\bar{u}_l$ est une lettre dans $\sum$

Si $\bar{u}_l \neq -$ et $\bar{v}_l \neq -$, $\bar{u}_l$ et $\bar{v}_l$ sont des lettres dans $\sum$

Question 8

En distinguant les trois cas envisagés à la question 7, exprimer $C(\bar{u} \cdot \bar{v})$ à partir de $C(\bar{u}_{[1\ldots l-1]}, \bar{v}_{[1\ldots l-1]})$. Aucune justification n'est attendue.

Si $\bar{u}_l = -$ et $\bar{v}_l \neq -$, $C(\bar{u}_l, \bar{v}_l) = c_{ins} + C(\bar{u}_{l-1}, \bar{v}_{l-1})$

Si $\bar{u}_l \neq -$ et $\bar{v}_l = -$, $C(\bar{u}_l, \bar{v}_l) = c_{del} + C(\bar{u}_{l-1}, \bar{v}_{l-1})$

Si $\bar{u}_l \neq -$ et $\bar{v}_l \neq -$, $C(\bar{u}_l, \bar{v}_l) = c_{sub} + C(\bar{u}_{l-1}, \bar{v}_{l-1})$

Avec $c_{sub} = \begin{cases} 0 \text{ si } \bar{u}_l = \bar{v}_l \\ 3 \text{ si } \{\bar{u}_l, \bar{v}_l\} = \{A, T\} \text{ ou } \{\bar{u}_l, \bar{v}_l\} = \{G, C\} \\ 4 \text{ sinon} \end{cases}$

Avec $c_{ins} = 2$ et $c_{del} = 2$

Question 9

Pour iϵ[1 ... n] et jϵ[1 ... m], déduire des questions 7 et 8 l'expression de D (i, j) à partir des valeurs de D à des rangs plus petits, c'est-à-dire à partir des termes D (i', j') où i'$\leq$ i, j'$\leq$ j et (i', j') $\neq$ (i, j). Justifiez votre réponse.

D (i, j) $= d\,(x_{[1\ldots i]}, y_{[1\ldots j]})$

$= \min\{c(\bar{x}_{[1\ldots i]}, \bar{y}_{[1\ldots j]} \mid (\bar{x}_{[1\ldots i]}, \bar{y}_{[1\ldots j]}) \text{ est un alignement de } (x_{[1\ldots i]}, y_{[1\ldots j]})\}$

Or, D (i, j) = $d\,(x_{[1\ldots i']}, y_{[1\ldots j']}) + dx_{[(i'+1)\ldots i]}, y_{[j'+1\ldots j]}) = d\left(x_{[1\ldots i']}, y_{[1\ldots j']}\right) +$
$\min\{c(\bar{x}_{[1\ldots i]}, \bar{y}_{[1\ldots j]} \mid (\bar{x}_{[1\ldots i]}, \bar{y}_{[1\ldots j]}) \text{ est un alignement de } (x_{[1\ldots i]}, y_{[1\ldots j]})$

On a donc, si $\begin{cases} i' = i - 1 \\ j' = j - 1 \end{cases}$:

$D(i,j) = D(i-1, j-1) + \min\{c(\bar{x}_l, -), c(-, \bar{y}_l) \text{ est un alignement de } (x_{[1\ldots i]}, y_{[1\ldots j]})\}$

Question 10

Que vaut D (0, 0) ? Justifiez.

D (0, 0) = 0 car le seul alignement possible pour 2 mots (x, y) avec x = ε et y = ε est $(\varepsilon, \varepsilon)$. Vu qu'ils sont identique, $C(\varepsilon, \varepsilon) = 0$ ($c_{sub}(a, b) = 0$ avec $a = \varepsilon$ et $b = \varepsilon$)

Question 11

Que vaut D (0, j) pour jϵ[1 ... m] ? Que vaut D (i, 0) pour iϵ[1 ... n] ? Justifiez.

D (0, j) = 2× j car les seuls alignements possibles pour 2 mots (x, y) avec x = ε est une insertion de gap pour chaque lettre de y. On a donc un coût de $|\bar{y}| \times c_{ins} = |\bar{y}| \times 2$.

D (i, 0) = 2× 0 car les seuls alignements possibles pour 2 mots (x, y) avec y = ε est une suppression de gap pour chaque lettre de x. On a donc un coût de $|\bar{x}| \times c_{del} = |\bar{x}| \times 2$.

Question 12

En s'appuyant sur les réponses aux questions 9, 10 et 11, donner le pseudo-code d'un algorithme itératif nommé DIST_1, qui prends en entrée deux mots, qui remplit un tableau à deux dimensions T avec toutes les valeurs de D pour finalement renvoyer la distance d'édition entre ces deux mots.

```
DIST_1

    Entrée : deux mots x et y
    Sortie : d(x,y) distance d'édition

    n <- |x|
    m <- |y|
    tab [n+1][m+1]

    Pour i de 0 à n :
        Pour j de 0 à m :
            tab[i][j] <- 0

    Pour i de 0 à n :
        tab[i][0] <- i * c_del

    Pour i de 0 à m :
        tab [0][i] <- i * c_ins

    Pour i allant de 1 à n:
        Pour j allant de 1 à m :
            case_haut <- tableau[i-1][j]
            case_gauche  <-tableau[i][j-1]
            case_haut_gauche <- tableau[i-1][j-1]
            tableau[i][j] <- min(case_haut + c_sub, case_gauche + c_ins, case_haut_gauche + c_sub(x[i-1],y[j-1]))

    retourne tab[n][m]
```

Question 13

Quelle est la complexité spatiale de l'algorithme DIST_1 ?

La complexité spatiale de l'algorithme DIST_1 est $O(n \times m)$

Question 14

Quelle est la complexité temporelle de l'algorithme DIST_1 ? Justifiez rapidement.

La complexité temporelle de l'algorithme DIST_1 est $O(n \times m)$. L'algorithme est constitué de 2 boucles qui font respectivement $|x|$itérations et $|y|$itérations, il est donc bien en $O(n \times m)$.

Question 15

Soit $(i, j) \in [0 \dots n] \times [0 \dots m]$. Montrer que :

Si $j > 0$ et $D(i,j) = D(i,j-1) + c_{ins}$, alors $\forall (\bar{s}, \bar{t}) \in Al^*(i,j-1), (\bar{s} \cdot -, \bar{t} \cdot y_j) \in Al^*(i,j)$

Si $i > 0$ et $D(i,j) = D(i-1,j) + c_{del}$, alors $\forall (\bar{s}, \bar{t}) \in Al^*(i-1,j), (\bar{s} \cdot x_i, \bar{t} \cdot -) \in Al^*(i,j)$

Si $D(i,j) = D(i-1,j-1) + c_{sub}(x_i, y_j)$, alors $\forall (\bar{s}, \bar{t}) \in Al^*(i-1,j-1), (\bar{s} \cdot x_i, \bar{t} \cdot y_i) \in Al^*(i,j)$

Vous pouvez ne développer que l'un des trois cas au choix.

Supposons $D(i,j) = D(i,j-1) + c_{ins}$, avec $j > 0$

Montrons que $\forall (\bar{s}, \bar{t}) \in Al^*(i,j-1), (\bar{s} \cdot -, \bar{t} \cdot y_j) \in Al^*(i,j)$:

Soit $\forall (\bar{s}, \bar{t}) \in Al^*(i,j-1)$ on a :

$C(\bar{s}, \bar{t}) = d(x_{[1\dots i]}, y_{[1\dots j-1]})$

Or,

$C(\bar{s} \cdot -, \bar{t} \cdot y_j) = C(\bar{s}, \bar{t}) + c_{ins} = D(i,j-1) + c_{ins} = D(i,j)$ par hypothèse

Question 16

Donner le pseudo-code d'un algorithme itératif nommé SOL_1, qui à partir d'un couple de mots (x, y) et d'un tableau T indexé par $[0 \ldots |x|] \times [0 \ldots |y|]$ contenant les valeurs de D, renvoie un alignement minimal de (x, y).

```
SOL_1
    Entrée : un couple de mots (x,y), une matrice a deux dimension T[n][m] contenant les
    valeur de D avec n et m les tailles respectives de x et y
    Sortie : un alignement minimal de (x,y)

    i <- n
    j <- m
    x_return <- ""
    y_return <- ""

    Tant que i > 0 ou j > 0 faire :
        si T[i][j]-T[i-1][j] = c_del faire :
            x_return = x_return + x[i-1]
            y_return = y_return + "-"
            i = i - 1
        si T[i][j]-T[i][j-1] = c_ins faire :
            x_return = x_return + "-"
            y_return = y_return + y[j-1]
            j = j - 1
        si T[i][j]-T[i-1][j-1] = c_sub(x[i-1],y[i-1]) faire :
            x_return = x_return + x[i-1]
            y_return = y_return + y[j-1]
            i = i - 1
            j = j - 1

    ali_x = reversed(x_return) #Renverse les lettres de x_return
    ali_y = reversed(y_return) #Renverse les lettres de y_return
    retourne (ali_x, ali_y)
```

Question 17

En combinant les algorithmes DIST_1 et SOL_1 avec quelle complexité temporelle résout-on le problème ALI ?

En combinant les algorithmes DIST_1 et SOL_1, on obtient donc $O((n \times m) + n + m)$ soit $O(n \times m)$, car DIST_1 est en $O(n \times m)$ et SOL_1 est composé d'une seule boucle qui fait au plus $n + m$ itérations.

Question 18

En combinant les algorithmes DIST_1 et SOL_1 avec quelle complexité spatiale résout-on le problème ALI ?

En combinant les algorithmes DIST_1 et SOL_1, on obtient une complexité mémoire $O((n \times m) + n + m)$ soit $O(n \times m)$, car on a besoin d'une matrice $n \times m$ pour stocker les coûts de chaque alignement et de deux chaines de caractères de taille maximale $n + m$ pour renvoyer l'alignement optimal.

Amélioration de la complexité spatiale du calcul de la distance

Question 19

Expliquer pourquoi lors du remplissage de la ligne i > 0 du Tableau T dans l'algorithme DIST_1, il suffirait d'avoir accès aux lignes $i-1$ et i du tableau (partiellement rempli pour cette dernière).

Pour compléter la case (i,j) du tableau dans DIST_1, il est nécessaire d'avoir accès aux 3 cases suivantes : $(i,j-1)$, $(i-1,j)$ et $(i-1,j-1)$. Ces cases correspondent aux 3 cas indiqué dans la question 15, ainsi il suffit donc d'avoir accès seulement aux lignes $i-1$ et i du tableau pour le remplir.

Question 20

En utilisant la remarque de la question précédente, donner le pseudo-code d'un algorithme itératif DIST_2, qui a la même spécification que DIST_1, mais qui a une complexité linéaire (en $O(m)$).

```
DIST_2:

    Entrée : deux mots x et y
    Sortie : d(x,y) distance d'édition

    n <- |x|
    m <- |y|
    tableau[2][m+1]
    tableau_vide[m+1]

    Pour i de 0 à 2 :
        Pour j de 0 à m+1 :
            tableau[i][j] <- 0

    Pour j de 0 à m+1 :
        tableau_vide[j] <- 0

    Pour j allant de 1 à m+1: #la ligne 0
        tableau[0][j] <- j*c_ins

    Pour i allant de 1 à n :
        tableau[1][0] <- c_del * i
        Pour  j in range(1,len(y)+1):
            case_haut <- tableau[0][j]
            case_gauche <- tableau[1][j-1]
            case_haut_gauche <- tableau[0][j-1]
            tableau[1][j] <- min(case_haut + c_del, case_gauche+count_ins, case_haut_gauche + c_sub(x[i-1], y[j-1]))

        Pour  j allant de 0 à m+1 :
            tableau[0][j] = tableau[1][j]

        tableau[1] = tableau_vide

    t <- tableau[0]
    return t[len(t)-1]
```

Amélioration de la complexité spatiale du calcul d'un alignement optimal par la méthode « diviser pour régner »

Question 21

Donner le pseudo code d'une fonction mot_gaps qui, étant donné un entier naturel k, renvoie le mot constitué de k gaps.

```
mots_gaps
    Entrée : k un entier
    Sortie : un mot consitué de k gaps

    mot <- ""
    Tant que k > 0 faire :
        mot <- mot + "-"
        k <- k - 1

    retourne mot
```

Question 22

Donner le pseudo code d'une fonction align_lettre_mot qui, étant donné x un mot de longueur 1 et y un mot non vide de longueur quelconque, renvoie un meilleur alignement de (x, y).

```
align_lettre_mot
    Entrée : x un mot de longueur 1 et y un mot non vide
    Sortie : meilleur alignement de (x,y)

    indice <- 0
    i <- 0
    m <- |y|
    L <- ""
    mini <- max(c_sub)
    Tant que i > m faire :
        si mini > c_sub(x,y[i]) faire :
        mini = c_sub(x,y[i])
        indice = i

    L <- L + mots_gaps(indice)
    L <- L + x
    L <- L + mots_gaps(m-indice-1)
    retourne L
```

Question 23

On considère un exemple où $\sum$ est l'alphabet latin, constitué de 26 lettres majuscules, x = BALLON et y = ROND. On coupe x et y en leur milieux : x^1 = BAL, x^2 = LON, y^1 = RO et y^2 = ND.

On fixe $c_{ins} = c_{del} = 3$, et $c_{sub}(a, b) = \begin{cases} 0 \text{ si } a = b \\ 5 \text{ si a et b sont deux voyelles distinctes} \\ 5 \text{ si a et b sont deux consonnes distinctes} \\ 7 \text{ sinon} \end{cases}$

Donner $(\bar{s}, \bar{t})$ un alignement optimal de (x^1, y^1) et $(\bar{u}, \bar{v})$ un alignement optimal de (x^2, y^2). Montrer que $(\bar{s} \cdot \bar{u}, \bar{t} \cdot \bar{v})$ n'est pas un alignement optimal de (x, y). Pas besoin de justifier l'optimalité de $(\bar{s}, \bar{t})$ et $(\bar{u}, \bar{v})$.

$(\bar{s}, \bar{t})$ Est un alignement optimal de (x^1, y^1) avec :

$\begin{cases} B\ A\ L \\ R\ O\ - \end{cases}$: le coût est 13

$(\bar{u}, \bar{v})$ est un alignement optimal de (x^2, y^2) avec :

$\begin{cases} L\ O\ N\ - \\ -\ -\ N\ D \end{cases}$: le coût est 9

$(\bar{s} \cdot \bar{u}, \bar{t} \cdot \bar{v})$ n'est pas un alignement optimal de (x, y) car il existe un meilleur alignement :

$\begin{cases} B\ A\ L\ L\ O\ N\ - \\ -\ -\ -\ R\ O\ N\ D \end{cases}$: le coût est 17

Etant donné que le coût de $(\bar{s} \cdot \bar{u}, \bar{t} \cdot \bar{v})$ vaut la somme des coûts de $(\bar{s}, \bar{t})$ et $(\bar{u}, \bar{v})$ qui est égale à 13 + 9 = 21 et qu'on a 21 > 17, on peut donc conclure que $(\bar{s} \cdot \bar{u}, \bar{t} \cdot \bar{v})$ n'est pas un alignement optimal de (x, y).

Question 24

En supposant disposer de la fonction coupure, donner le pseudo code de l'algorithme récursif de type diviser pour régner, nommé SOL_2, qui à partir d'un couple de mots (x, y) calcule un alignement minimal de (x, y).

Notre fonction Sol_2 nous renvoie une liste contenant les lettres du meilleur alignement de x aux indices pairs et les lettres du meilleur alignement de y aux indices impairs. La fonction Sol_2_2 nous permet donc de recomposer et de renvoyer $(\overline{x}, \overline{y})$, le meilleur alignement du couple (x, y).

```
Sol_2
    Entrée : x et y deux mots
    Sortie : retourne un alignement optimal de (x,y)

    Si |x|=0 et |y|>0 faire :
        retourne [mots_gap(|y|), y]

    Si |x|>0 et |y|=0 faire :
        retourne [x, mots_gap(|x|)]

    Si |x|=0 et |y|=0 faire :
        retourne []

    Si |x|=1 et |y|>0 faire :
        retourne [align_lettre_mot(x, y), y]

    Si |x|>1 et |y|>=1 faire :
        h <- partie_entiere(|x|/2)
        k <- coupure(x, y)
        a <- [Sol_2(x[0...h-1], y[0...k-1])]
        b <- [Sol_2(x[h...|x|], y[k...|k|])]
        retourne a[0] + b[0]

Sol_2_2
    Entrée : x et y deux mots
    Sortie : retourne un alignement optimal de (x,y)

    x_ali <- ""
    y_ali <- ""
    resultat <- Sol_2(x,y)
    cpt <- len(resultat)

    Pour i dans resultat faire :
        Si cpt%2 = 0 faire :
            x_ali <- x_ali + i
        Sinon :
            y_ali <- y_ali + i
        cpt <- cpt - 1
    retourne (x_ali, y_ali)
```

Question 25

Donner le pseudo-code d'une fonction coupure telle que décrite ci-dessus.

```
coupure
    Entrée : x et y deux mots
    Sortie : renvoi l'indice de la coupure de y

    x_gauche <- ""
    x_droit <- ""
    y_gauche <- ""
    y_droit <- ""

    Pour i allant de 0 a |y|+1 faire :
        x_gauche <- x_gauche + x[0...partie_entiere(|x|/2)]
        x_droit <- x_droit + x[partie_entiere(|x|/2)...|x|]
        y_gauche <- y_gauche + y[0...i]
        y_droit <- y_droit + y[i...l]

        Si Dist_2(x_gauche, y_gauche) + Dist_2(x_droit, y_droit) = Dist_2(x, y) faire :
            Retourne i

        x_gauche <- ""
        x_droit <- ""
        y_gauche <- ""
        y_droit <- ""
```

Question 26

Quelle est la complexité spatiale de coupure ? Justifiez rapidement.

La complexité spatiale de coupure est en $O(m)$ car on prend 4 sous-séquences à une dimension dans chaque boucle, et la complexité spatiale de DIST_2 est $O(m)$.

Question 27

Quelle est la complexité spatiale de SOL_2 ? Justifiez rapidement.

La complexité spatiale de SOL_2 est en $O(n + m)$ car chaque appel récursif prend un espace de $O(m)$ pour manipuler 2 séquences (gauche et droite) et seulement $O(1)$ espace pour stocker le retour d'appel. Le nombre d'appel est inférieur à $O(n)$.

Question 28

Quelle est la complexité temporelle de coupure ? Justifiez rapidement.

La complexité temporelle de coupure est en $O(m^2)$ car on aura au maximum une boucle avec 3m appels de DIST_2. Sachant que la complexité de DIST_2 est en $O(m)$, on a donc comme complexité : $m * O(m) = O(m^2)$

Question 29

A-t-on perdu en complexité temporelle en améliorant la complexité spatiale ? Comparez expérimentalement la complexité temporelle de SOL_2 à celle de SOL_1. On ne demande pas de preuve théorique.

$$\text{SOL\_2}\begin{cases}\text{Complexité temporelle} : O(m^2) \\ \text{Complexité Spatiale} : O(n+m)\end{cases} \text{ et } \text{SOL\_1}\begin{cases}\text{Complexité temporelle} : O(n+m) \\ \text{Complexité Spatiale} : O(n \times m)\end{cases}$$

Oui, on a amélioré la complexité spatiale en perdant en complexité temporelle.