Projet LU3IN003: Alignement de séquences

Floria LIM
Myriam MABROUKI

Automne 2022

1 Le problème d'alignement de séquences

1.1 Alignement de deux mots

Question 1

Soient $(\bar{x}, \bar{y})$ un alignement de (x, y) et $(\bar{u}, \bar{v})$ un alignement de (u, v).
Considérons $(\bar{x} \cdot \bar{u}, \bar{y} \cdot \bar{z})$.

1. $\pi(\bar{x}) = x$ et $\pi(\bar{u}) = u$, donc $\pi(\bar{x} \cdot \bar{u}) = \pi(\bar{x}) \cdot \pi(\bar{u}) = x \cdot u$.
2. $\pi(\bar{y}) = y$ et $\pi(\bar{v}) = v$, donc $\pi(\bar{y} \cdot \bar{v}) = \pi(\bar{y}) \cdot \pi(\bar{v}) = y \cdot v$.
3. $|\bar{x}| = |\bar{y}|$ et $|\bar{u}| = |\bar{v}|$ donc $|\bar{x}| \cdot |\bar{u}| = |\bar{y}| \cdot |\bar{v}|$
4. $\forall i \in [\![1, |\bar{x}|]\!]$, $\bar{x_i} \neq -$ ou $\bar{y_i} \neq -$ et $\forall i \in [\![1, |\bar{u}]\!]$, $\bar{u_i} \neq -$ ou $\bar{v_i} \neq - \Rightarrow \forall i \in [\![1, |\bar{x}| + |\bar{u}|]\!]$, $\bar{x_i} \cdot \bar{u_i} \neq -$ ou $\bar{y_i} \cdot \bar{v_i} \neq -$

Ainsi, $(\bar{x} \cdot \bar{u}, \bar{y} \cdot \bar{z})$ est bien un alignement de $(x \cdot u, y \cdot v)$.

Question 2

Soient $x, y \in \Sigma^*$ deux mots de longueurs respectives n et m.
Un alignement de longueur maximale est obtenu lorsque l'on a le maximum de gaps possibles, c'est-à-dire lorsque pour tout $i \in [\![1, |\bar{x}|]\!]$, $\bar{x_i} = -$ ou $\bar{y_i} = -$
Ainsi, la longueur maximale d'un alignement de (x, y) est de $m + n$.

Exemple : Notons $(\bar{x}, \bar{y})$ un alignement maximal de (x, y).
$\bar{x} : - \ldots - x_1 \ldots x_n$
$\bar{y} : y_1 \ldots y_m - \ldots -$

2 Algorithmes pour l'alignement de séquences

2.1 Méthode naïve par énumération

Question 3

Soit $x \in \Sigma^*$ un mot de longueur n.

On cherche à connaître le nombre de mots $\bar{x} \in \bar{\Sigma}^*$ tel que

- $|\bar{x}| = n + k$
- $\pi(\bar{x}) = x$

Cela revient à connaître le nombre de combinaisons possibles de k gaps dans un ensemble de $n + k$ caractères.

Ainsi, on a $\binom{n+k}{k}$ mots $\bar{x}$.

Question 4

Soit $(x, y) \in \Sigma^* \times \Sigma^*$ un couple de mots de longueurs respectives n et m tel que $n \geq m$.

Construisons le mot $\bar{x} \in \bar{\Sigma}^*$ en ajoutant k gaps à x. On a donc $|\bar{x}| = n + k$.

Construisons le mot $\bar{y} \in \bar{\Sigma}^*$ tel que $(\bar{x}, \bar{y})$ soit un alignement de (x, y).

Notons α le nombre de gaps de y. On a alors :

- $|\bar{y}| = |\bar{x}| = n + k$
- $|\bar{y}| = m + \alpha$

On a donc $\alpha = |\bar{y}| - m = n + k - m$ gaps dans $\bar{y}$.

Cherchons maintenant le nombre de mots $\bar{y}$ possibles.

Comme pour tout $i \in [\![1, |\bar{x}|]\!]$ si $\bar{x}_i = -$ alors $\bar{y}_i \neq -$, on a donc $n + k - m$ gaps à insérer dans $\bar{y}$ parmi n positions (les n positions correspondant aux positions des lettres de x dans $\bar{x}$).

Nous avons donc $\binom{n}{n+k-m}$ mots $\bar{y}$ possibles pour un mot $\bar{x}$.

Pour tous les mots $\bar{x}$ composés de k gaps, on a alors $\binom{n+k}{k}\binom{n}{n+k-m}$ possibilités d'alignements.

Enfin, pour connaître le nombre d'alignements total, il suffit de sommer les combinaisons d'alignements pour chaque valeur de k à insérer dans $\bar{x}$.

Comme la longueur maximale d'un alignement est $n + m$, on ne peut insérer qu'au maximum m gaps dans $\bar{x}$.

Ainsi, on obtient $\sum_{k=0}^{m} \binom{n+k}{k}\binom{n}{n+k-m}$ alignements au total.

Exemple : Pour $|x| = 15$ et $|y| = 10$, on a $\sum_{k=0}^{10} \binom{15+k}{k}\binom{15}{15+k-10}$

Question 5

Un algorithme naïf énumérant tous les alignements d'un couple (x, y) doit parcourir toutes les combinaisons d'alignements possible.

On a $\sum_{k=0}^{m} \binom{n+k}{k} \binom{n}{n+k-m}$ alignements au total.

Notons A la complexité de cet algorithme.

$A = O(m \times [(C(n+k, k) \times C(n, n+k-m)])$

$A = O(m \times min((n+k)^k, (n+k)^{n+k-k}) \times min(n^{n+k-m}, n^{n-(n+k-m)}))$

$A = O(m \times min((n+k)^k, (n+k)^n) \times min(n^{n+k-m}, n^{m-k}))$

$A = O(m \times (n+k)^k \times n^{n+k-m})$

$A \approx O(m \times (n+m)^m \times n^{n+m-m})$

$A \approx O(m \times (n+m)^m \times n^n)$

Un algorithme naïf permettant de trouver un alignement de coût minimal, et donc la distance d'édition, doit, en plus de parcourir toutes les combinaisons d'alignements, parcourir chaque caractère d'un mot pour calculer son coût dans l'alignement.

La longueur maximale d'un mot étant $n + m$, cette itération se fait en $O(n + m)$.

On obtient donc un algorithme de complexité exponentielle en $O((n+m) \times m \times (n+m)^m \times n^n)$, c'est-à-dire en $O(m \times (n+m)^{m+1} \times n^n)$.

Question 6

Un algorithme naïf énumérant tous les alignements d'un couple (x, y) en vue de trouver la distance d'édition entre ces deux mots nécessite de stocker un alignement à chaque appel récursif.

Au maximum, la longueur d'un alignement est de $n + m$ d'après la Question 2, la complexité spatiale de cet algorithme s'exprime donc en $O(n + m)$.

En vue de trouver un alignement de coût minimal, il est nécessaire de stocker l'alignement de coût minimal en plus, donc l'algorithme aurait une complexité en $O(2 \times (n + m))$ c'est-à-dire également en $O(n + m)$.

Tâche A

- Le calcul de la distance d'édition dépasse une minute dès que x et y atteignent respectivement une longueur de plus de 14 et 10 caractères.
- La consommation mémoire nécessaire de cette méthode est d'environ 10 540 Ko.

2.2 Programmation dynamique

Question 7

Soit $(\bar{u}, \bar{v})$ un alignement de $(x_{[1..i]}, y_{[1..j]})$ de longueur l.

$\forall k \in [\![1, |\bar{u}|]\!]$, $\bar{u_k} \neq -$ ou $\bar{v_k} \neq -$.

— si $\bar{u}_l = -$, alors $\bar{v}_l \neq -$, et $\bar{v}_l$ étant le dernier caractère de $\bar{v}$ on a donc $\bar{v}_l = y_j$

— si $\bar{v}_l = -$, alors $\bar{u}_l = x_i$

— si $\bar{u}_l \neq -$ et $\bar{v}_l \neq -$, $\bar{u}_l$, alors $\bar{u}_l = x_i$ et $\bar{v}_l = y_j$

Question 8

Par définition

$$C(\bar{u}, \bar{v}) = \sum_{k=1}^{l} c(\bar{k}_k, \bar{v}_k),\ \forall (a,b) \in \bar{\Sigma}^2 \setminus \{-,-\},\ c(a,b) = \begin{cases} c_{ins} & \text{si } a = - \\ c_{del} & \text{si } b = - \\ c_{sub}(a,b) & \text{sinon} \end{cases}$$

On a donc

$$C(\bar{u}, \bar{v}) = C(\bar{u}_{[\![1,l-1]\!]}, \bar{v}_{[\![1,l-1]\!]}) + C(\bar{u}_l, \bar{v}_l) = \begin{cases} C(\bar{u}_{[\![1,l-1]\!]}, \bar{v}_{[\![1,l-1]\!]}) + c_{ins} & \text{si } \bar{u}_l = - \\ C(\bar{u}_{[\![1,l-1]\!]}, \bar{v}_{[\![1,l-1]\!]}) + c_{del} & \text{si } \bar{v}_l = - \\ C(\bar{u}_{[\![1,l-1]\!]}, \bar{v}_{[\![1,l-1]\!]}) + c_{sub}(\bar{x}_i, \bar{y}_j) & \text{sinon} \end{cases}$$

Question 9

Soient $i \in [\![1, n]\!]$ et $j \in [\![1, n]\!]$.

D'après les questions 7 et 8,

— $C(\bar{u}, \bar{v}) = C(\bar{u}_{[\![1,l-1]\!]}, \bar{v}_{[\![1,l-1]\!]}) + c_{ins}$ si $\bar{u}_l = -$

— $C(\bar{u}, \bar{v}) = C(\bar{u}_{[\![1,l-1]\!]}, \bar{v}_{[\![1,l-1]\!]}) + c_{del}$ si $\bar{v}_l = -$

— $C(\bar{u}, \bar{v}) = C(\bar{u}_{[\![1,l-1]\!]}, \bar{v}_{[\![1,l-1]\!]}) + c_{sub}(\bar{x}_i, \bar{y}_j)$ sinon

On sait que $d(x,y) = min\{C(\bar{x}, \bar{y}) | (\bar{x}, \bar{y})\ un\ alignement\ de\ (x,y)\}$ et $d(x,y) = D(n,m)$

Ainsi, $D(i,j) = min\{D(i,j-1) + c_{ins},\ D(i-1,j) + c_{del},\ D(i-1,j-1) + c_{sub}\}$

Question 10

L'unique alignement à $(\varepsilon, \varepsilon)$ est $(\varepsilon, \varepsilon)$.

Aucun coût d'insertion, de suppression ou de substitution n'est possible car cela enfreindrait les règles (iii) et/ou (iv). L'alignement vaut donc 0.

On a donc $D(0,0) = d(\varepsilon, \varepsilon) = 0$.

On remarque également que $(\varepsilon, \varepsilon)$ est l'élément neutre de d, la distance d'édition.

Question 11

Pour $j \in [1...m]$, $D(0,j) = j \times c_{ins}$.

En effet, pour que $|\bar{x}| = |\bar{y}|$, nous devons insérer m gaps dans x car $x = 0$ et $y = m$.

Ainsi à $D(0,j)$, on aura ajouté j gaps, et donc le coup est de $j \times c_{ins}$.

De même, pour $i \in [1...n]$, $D(i,0) = i \times c_{sub}$ car pour que $|\bar{x}| = |\bar{y}|$, nous devons supprimer n lettres de x comme $x = n$ et $y = 0$.

À $D(i,0)$, on aura supprimé i gaps, le coup est de $i \times c_{del}$.

Question 12

Algorithme 1 : DIST_1

Entrées : x et y deux mots de longueur n et m
T un tableau indexé par $[0 \dots |x|] \times [0 \dots |y|]$
Sorties : $T[n, m]$, la distance d'édition entre x et y
si *n=0 et m=0* **alors**
  retourner 0
fin
pour *i allant de 1 à n* **faire**
  $T[i][0] \leftarrow i \times c_{ins}$
fin
pour *j allant de 1 à m* **faire**
  $T[0][j] \leftarrow j \times c_{del}$
fin
pour *i allant de 1 à n* **faire**
  pour *j allant de 1 à m* **faire**
    $a \leftarrow T[i][j-1] + c_{ins}$
    $b \leftarrow T[i-1][j] + c_{del}$
    $c \leftarrow T[i-1][j-1] + c_{sub}(x_{i-1}, y_{j-1})$
    $T[i][j] \leftarrow min(a, b, c)$
  fin
fin
retourner $T[n, m]$

Question 13

Pour la fonction $DIST\_1$, nous devons stokcer un en mémoire un tableau à deux dimension de taille $n \times m$.
La complexité spatiale de l'algorithme est alors en $O(n \times m)$

Question 14

La fonction $DIST\_1$ est composée de 4 boucles *pour* dont deux imbriquées allant respectivement de 1 à n et de 1 à m, et deux autres de 1 à n et 1 à m.
On effectue alors $n + m + n \times m$ itérations.
En supposant que les opérations de multiplication, d'addition, d'affectation et de comparaison sont en $O(1)$, la complexité temporelle de $DIST\_1$ est donc en $O(n \times m)$

Question 15

Dans cette question, nous n'allons traiter que le premier cas.
Soit $(i, j) \in [\![0, n]\!] \times [\![0, m]\!]$.
Supposons que $j > 0$ et $D(i, j) = D(i-1, j) + c_{del}$.
Soit $(\bar{s}, \bar{t}) \in Al^*(i, j-1), (\bar{s}, \bar{t})$ est un alignement minimal de $(x_{[1,\dots i]}, y_{[1,\dots j-1]})$.
Le couple $(-, y_j)$ est un alignement minimal de (ε, y_j).
$(\bar{s}, \bar{t}) \cdot (-, y_j) = (\bar{s} \cdot -, \bar{t} \cdot y_j)$
Donc, d'après la question 1, $(\bar{s} \cdot -, \bar{t} \cdot y_j)$ est un alignement de $(x_{[1,\dots i]}, y_{[1,\dots j]})$.
Observons maintenant ce qu'il se passe au niveau de la distance d'édition.
$d(x_{[1,\dots i]}, y_{[1,\dots j]}) = D(i, j) = D(i, j-1) + c_{ins}$
Ainsi, $(\bar{s} \cdot -, \bar{t} * y) \in Al^*(i, j)$.

Question 16

Algorithme 2 : SOL_1

Entrées : x et y deux mots de longueur n et m
T un tableau indexé par $[0 \dots |x|] \times [0 \dots |y|]$ et contenant les valeurs de D
Sorties : (u, v), un alignement minimal de (x, y)
$i \leftarrow n$
$j \leftarrow m$
tant que $i > 0$ *ou* $j > 0$ **faire**
  si $i > 0$ *et* $j > 0$ *et* $T[i][j] = T[i-1][j-1] + c_{sub}(x_{i-1}, yj-1)$ **alors**
    $u \leftarrow x_i \cdot u$
    $v \leftarrow y_j \cdot v$
    $i \leftarrow i - 1$
    $j \leftarrow j - 1$
  sinon si $i > 0$ *et* $T[i][j] = T[i-1][j] + c_{del}$ **alors**
    $u \leftarrow x_i \cdot u$
    $v \leftarrow - \cdot v$
    $i \leftarrow i - 1$
  sinon
    $u \leftarrow . u$
    $v \leftarrow y_j \cdot v$
    $j \leftarrow j - 1$
fin
retourner (u, v)

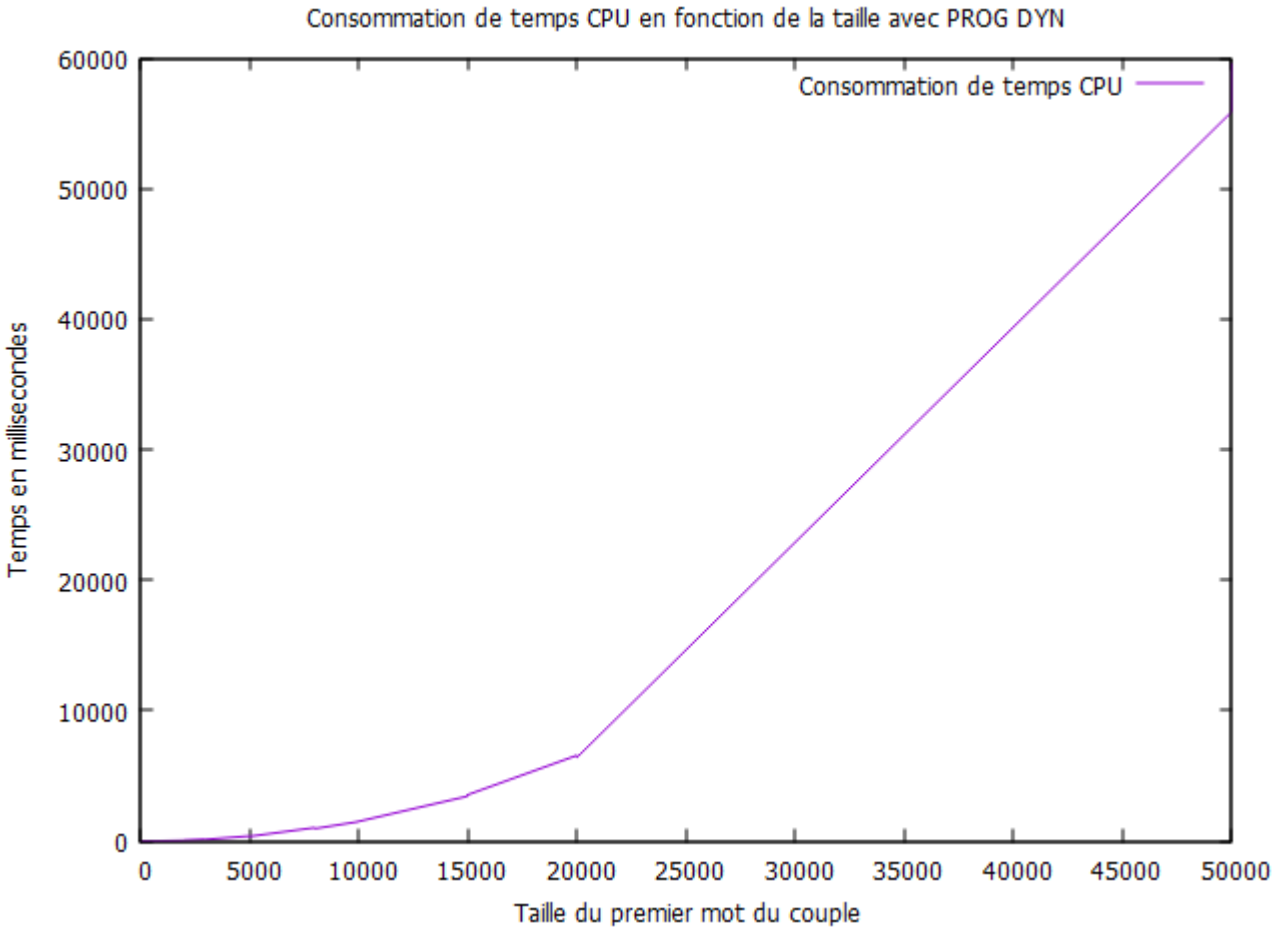
Question 17

La fonction SOL_1 effectue au maximum $n + m$ itérations, sa compléxité est donc en $O(n + m)$.
D'après la question 14, la complexité temporelle de $DIST\_1$ est en $O(n \times m)$.
On peut alors résoudre le problème d'ALI en $O(n \times m)$.

Question 18

L'algorithme SOL_1 n'alloue que deux chaînes de caractères de taille maximum $n + m$, donc sa complexité spatiale est en $O(n + m)$.
D'après la question 13, celle de $DIST\_1$ est en $O(n \times m)$, donc la complexité spatiale de ALI est en $O(n \times m + n + m)$ c'est-à-dire en $O(n \times m)$.

Tâche B



— La courbe obtenue correspond bien à la complexité théorique qui est de $O(n \times m)$: en effet, elle a une forme hyperbolique ce qui correspond bien à une fonction polynomiale.

— Nous avons comme consommation mémoire utilisée par $PROG\_DYN$:

  — pour $|x| = 15\ 000$, elle est de 816 800 Ko.

  — pour $|x| = 20\ 000$, elle est de 1 469 088 Ko.

  — pour $|x| = 50\ 000$, elle est de 9 000 128 Ko à 9 993 700 Ko.

2.3 Amélioration de la complexité spatiale du calcul de la distance

Question 19

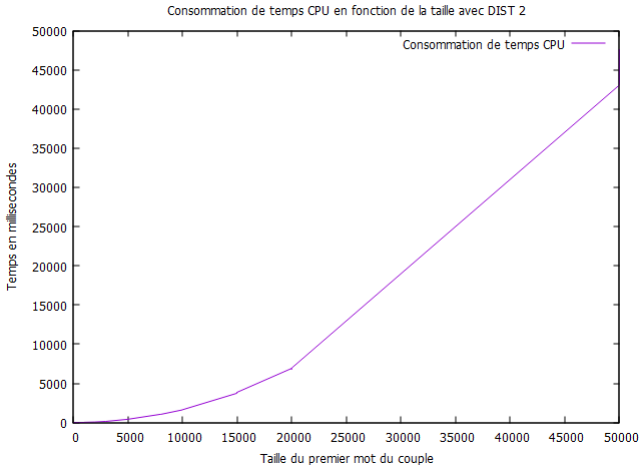
Lors du remplissage d'une ligne $i > 0$ dans $DIST\_1$, seules les lignes i et $i-1$ du tableau T sont utilisées.
En effet, pour calculer $A = T[i-1][j] + c_{ins}$ et $C = T[i-1][j-1] + c_{sub}(x[i], y[j])$, on a besoin seulement de la ligne $i-1$ de T, et pour calculer $B = T[i][j-1] + c_{del}$, on a besoin seulement de la ligne ligne i.

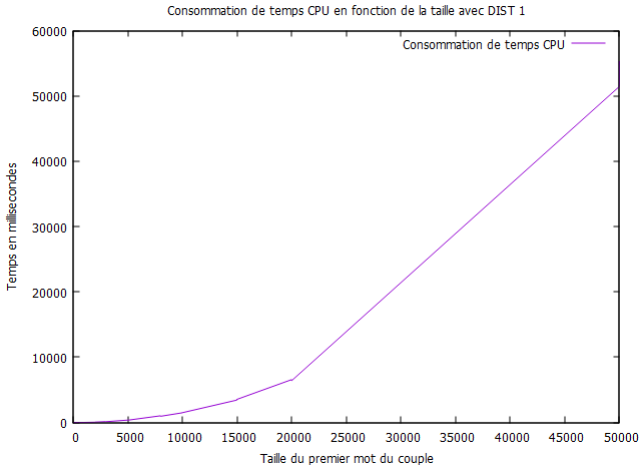
Question 20

Algorithme 3 : DIST_2

Entrées : x et y deux mots de longueur n et m
Sorties : $T[n, m]$, la distance d'édition entre x et y
pour j *allant de 1 à* m **faire**
  $lig\_1 = j \times c_{ins}$
fin
$lig\_2[0] = 0$
pour i *allant de 1 à* n **faire**
  $lig\_2[0] = i \times c_{del}$
  pour j *allant de 1 à* m **faire**
    $a \leftarrow lig\_1[j] + c_{ins}$
    $b \leftarrow lig\_2[j-1] + c_{del}$
    $c \leftarrow lig\_1[j-1] + c_{sub}(x_{i-1}, y_{j-1})$
    $lig\_2 \leftarrow min\{a, b, c\}$
  fin
fin
pour k *allant de 0 à* m **faire**
  $lig\_1[k] \leftarrow lig\_2[k]$
fin
retourner $T[n, m]$

Tâche C





— La courbe obtenue de $DIST\_2$ située à gauche correspond bien à la complexité théorique qui est de $O(n \times m)$: en effet, elle a une forme hyperbolique ce qui correspond bien à une fonction polynomiale.
— La courbe obtenue a bien une forme similéaire à $DIST\_1$, dont la courbe se situe ci-dessus à droite. Ces deux algorithmes ayant la même complexité, cela correspond bien au résultat attendu.
— Nous avons comme consommation mémoire utilisée par $DIST\_2$:
  — pour $|x| = 15\ 000$, elle est de 10 720 Ko.
  — pour $|x| = 20\ 000$, elle est de 10 764 Ko.
  — pour $|x| = 50\ 000$, elle est de 10 940 Ko à 11 032 Ko.

  On peut remarquer que la consommation mémoire est bien plus inférieure à celle de SOL_1, qui pour $|x|$ valant 50 000, atteignait 9 993 700.

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

Question 21

Algorithme 4 : MOTS_GAPS

Entrées : k, un entier naturel
Sorties : mot_gaps, un mot composé de k gaps
$mot\_gaps \leftarrow \varepsilon$
pour *i allant de 1 à k* **faire**
  $mot\_gaps \leftarrow mot\_gaps \cdot -$
fin
retourner mot_gaps

Question 22

Algorithme 5 : ALIGN_LETTRE_MOT

Entrées : x et y deux mots de longueur 1 et m
Sorties : (u, v), un alignement minimal de (x, y)
$cout\_sub\_min \leftarrow c_{sub}(x_0, y_0)$
$indice\_sub\_min \leftarrow 0$
pour *i allant de 1 à m* **faire**
  si $cout\_sub\_min > c_{sub}(x_0, y_i)$ **alors**
    $cout\_sub\_min = c_{sub}(x_0, y_i)$ $indice\_sub\_min = i$
  fin
fin
si $cout\_sub\_min > c_{ins} + c_{del}$ **alors**
  $(u, v) \leftarrow (x_0 \cdot MOTS\_GAPS(m),\ - \cdot y)$
sinon
  $(u, v) \leftarrow (MOTS\_GAPS(indice\_sub\_min) \cdot x_0 \cdot MOTS\_GAPS(m - indice\_sub\_min),\ y)$
fin
retourner (u, v)

Question 23

Soient $x = BALLON$ et $y = ROND$, avec $x_1 = BAL$, $x_2 = LON$, $y_1 = RO$ et $y_2 = ND$

Notons $(\bar{s}, \bar{t})$ et $(\bar{u}, \bar{v})$ deux alignements optimaux respectivement de (x_1, y_1) et (x_2, y_2).

$\bar{s} = B\ A\ L$ $\qquad$ $u = L\ O\ N\ -$

$\bar{t} = R\ O\ -$ $\qquad$ $v = -\ -\ N\ D$

Ces alignements donnent les coûts suivants : $c(\bar{s}, \bar{t}) = 5 + 5 + 3 = 13$ et $c(\bar{u}, \bar{v}) = 3 + 3 + 3 = 9$

On obtient ainsi :

$\bar{s} \cdot \bar{u} = B\ A\ L\ L\ O\ N\ -$

$\bar{t} \cdot \bar{v} = R\ O\ -\ -\ -\ N\ D$ $\qquad$ $c(\bar{s} \cdot \bar{u}, \bar{t} \cdot \bar{v}) = 13 + 9 = 22$

Cependant, on peut obtenir un meilleur alignement.

$a = B\ A\ L\ L\ O\ N\ -$

$b = -\ -\ -R\ O\ N\ D$ $\qquad$ $c(a, b) = 4 \times 3 + 5 = 17 < 22$

Ainsi, $c(\bar{s} \cdot \bar{u}, \bar{t} \cdot \bar{v})$ n'est pas un alignement optimal de (x, y).

Question 24

Algorithme 6 : SOL_2

Entrées : x et y deux mots de longueur n et m
T un tableau indexé par $[0 \dots |x|] \times [0 \dots |y|]$ et contenant les valeurs de D
Sorties : (u, v), un alignement minimal de (x, y)
$i \leftarrow \lfloor \frac{n}{2} \rfloor$
$j \leftarrow COUPURE(x, y)$
si $m = 0$ *et* $j = 0$ **alors**
  $(u, v) \leftarrow (\varepsilon, \varepsilon)$
sinon si $n = 0$ **alors**
  $(u, v) \leftarrow (MOTS\_GAPS(m), y)$
sinon si $m = 0$ **alors**
  $(u, v) \leftarrow (x, MOTS\_GAPS(n))$
sinon si $n = 1$ **alors**
  $(u, v) \leftarrow ALIGN\_LETTRE\_MOT(x, y)$
sinon si $m = 1$ **alors**
  $(v, u) \leftarrow ALIGN\_LETTRE\_MOT(y, x)$
sinon
  $(a, b) \leftarrow SOL\_2(x_{[\![0,i]\!]}, y_{[\![0,j]\!]}$
  $(c, d) \leftarrow SOL\_2(x_{[\![i+1,n]\!]}, y_{[\![j+1,m]\!]}$
  $(u, v) \leftarrow (a \cdot c, b \cdot d)$
retourner (u, v)

Question 25

Algorithme 7 : COUPURE

Entrées : x et y deux mots de longueur n et m
Sorties : $lig\_4[m]$, un indice qui est la coupure associée à (x, y)
lig_1, tableau associée à la ligne $i-1$ de D
lig_2, tableau associée à la ligne i de D
lig_3, tableau associée à la ligne $i-1$ de I
lig_4, tableau associée à la ligne i de I
$moitie \leftarrow \lfloor \frac{n}{2} \rfloor$
pour j *allant de 1 à* m **faire**
  $lig\_1 = j \times c_{ins}$
  $lig\_1 = j$
fin
$lig\_2[0] = 0$ $lig\_3[0] = 0$ **pour** i *allant de 1 à* n **faire**
  $lig\_2[0] = i \times c_{del}$
  pour j *allant de 1 à* m **faire**
    $a \leftarrow lig\_1[j] + c_{ins}$
    $b \leftarrow lig\_2[j-1] + c_{del}$
    $c \leftarrow lig\_1[j-1] + c_{sub}(x_{i-1}, y_{j-1})$
    $lig\_2 \leftarrow min\{a, b, c\}$
    si $lig\_2[j] = a$ **alors**
      $lig\_4[j] \leftarrow lig\_3[j]$
    sinon si $lig\_2[j] = a$ **alors**
      $lig\_4[j] \leftarrow lig\_4[j-1]$
    sinon
      $lig\_4[j] \leftarrow lig\_3[j-1]$
  fin
fin
si $i > moitie$ **alors**
  pour k *allant de 0 à* m **faire**
    $lig\_3[k] \leftarrow lig\_4[k]$
  fin
fin
pour k *allant de 0 à* m **faire**
  $lig\_1[k] \leftarrow lig\_2[k]$
fin
retourner $lig\_4[m]$

Question 26

La fonction $COUPURE$ utilise 4 tableaux de dimension $(m+1)$.
Sa complexité spatiale est donc en $O(4 \times m)$, soit $O(m)$.

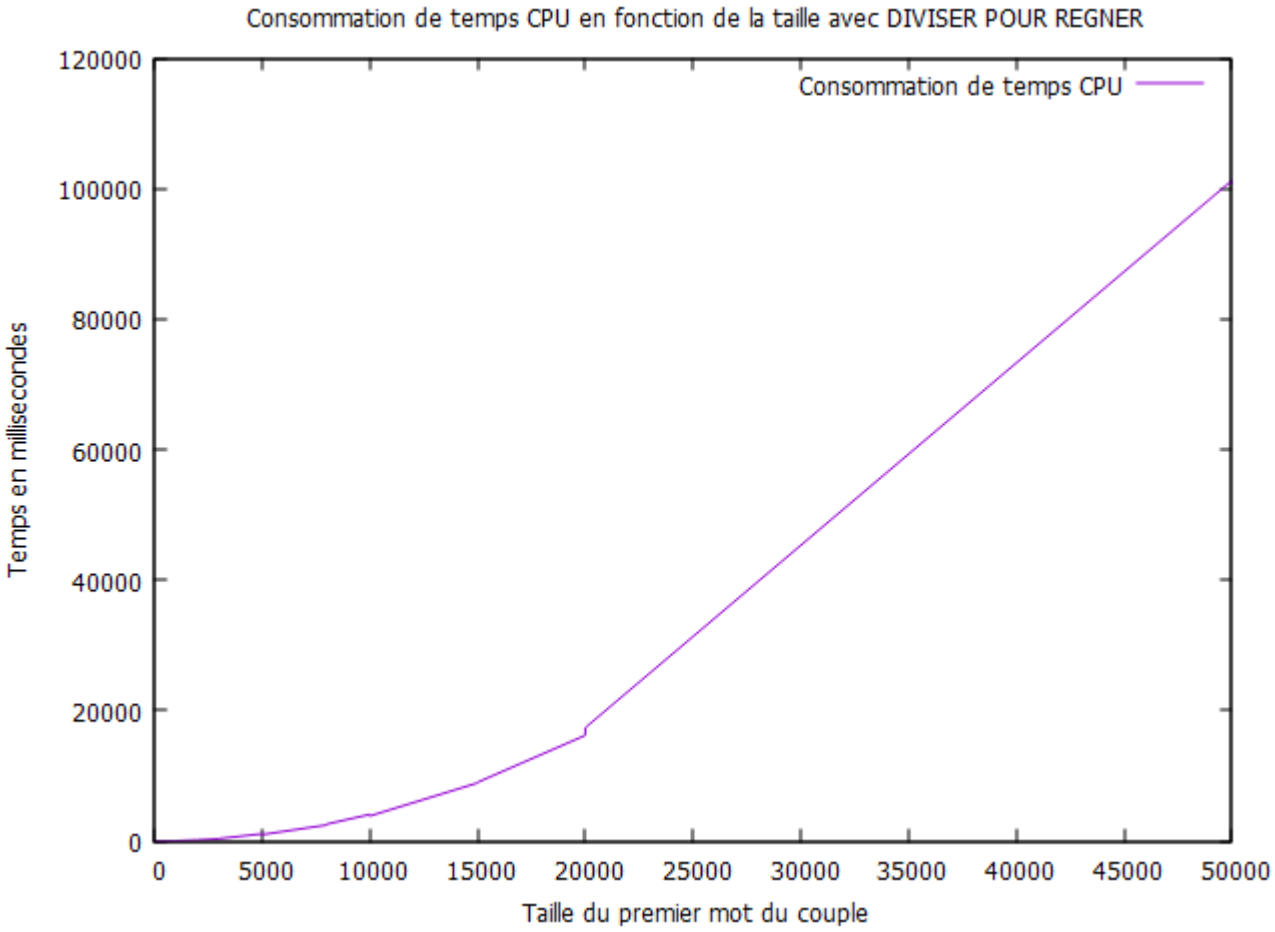
Question 27

SOL_2 renvoie 2 chaînes de caractères de longueur maximale $n+m$ dans le pire des cas, et utilise la fonction $COUPURE$ qui, d'après la Question 26, a une complexité spatiale en $O(m)$.
Ainsi, la complexité spatiale de SOL_2 est en $O(2 \times (n+m) + m)$, c'est-à-dire en $O(n+m)$.

Question 28

SOL_2 possède une première boucle allant de 0 à $m+1$, ainsi que deux autres boucles imbriquées allant respectivement de 1 à $n+1$ et de 1 à $m+1$. La complexité temporelle est donc en $O(m+1+n \times m)$, c'est-à-dire en $O(n \times m)$.

Tâche D



— Nous avons comme consommation mémoire utilisée par $DIST\_2$:
  — pour $|x| = 15\ 000$, elle est de 11 708 Ko.
  — pour $|x| = 20\ 000$, elle est de 12 052 Ko.
  — pour $|x| = 50\ 000$, elle est de 12 840 Ko à 14 248 Ko.

Question 29

On a perdu en complexité temporelle en améliorant la complexité spatiale.

En effet, la consommation de temps CPU en fonction de la taille avec SOL_2 atteint les 100s pour une taille de premier mot du couple égal à 50 000, alors qu'elle était d'environ 55s pour SOL_1 : elle a quasiment doublée.

3 Une extension : l'alignement local de séquences (Bonus)

Question 30

Soit l'alphabet A, T, G, C et $c_{del} = 1$.

- En prenant $x = ATCGTAGCTGATCGT$ et $y = A$, on obtient l'alignement minimal suivant :
  $\bar{x} : A\ T\ C\ G\ T\ A\ G\ C\ T\ G\ A\ T\ C\ G\ T$
  $\bar{y} : A - - - - - - - - - - - - - - -$
  avec un coût global minimal valant 14, et $(|x| - |y|)c_{del} = (15 - 1) \times c_{del} = 14$.
- En prenant $x = TACGTCCTAGGTCTAAGCA$ et $y = GCTAC$, on obtient l'alignement minimal suivant :
  $\bar{x} : T\ A\ C\ G\ T\ C\ C\ T\ A\ G\ G\ T\ C\ T\ A\ A\ G\ C\ A$
  $\bar{y} : - - - G\ - C\ - T\ A\ - - - C\ - - - - - - -$
  avec un coût global minimal valant 14, et $(|x| - |y|)c_{del} = (19 - 5) \times c_{del} = 14$.
- En prenant $x = CGATCGATAGTCTAGCACGACTAGCTAGCA$ et $y = GCATCGATC$, on obtient l'alignement minimal suivant :
  $\bar{x} : C\ G\ A\ T\ C\ G\ A\ T\ A\ G\ T\ C\ T\ A\ G\ C\ A\ \ C\ G\ A\ C\ T\ A\ G\ C\ T\ A\ G\ C\ A$
  $\bar{y} : - G - - C - \ A\ T - - - \ C - - G\ - A\ - - - - - T\ - - C\ - - - - - -$
  avec un coût global minimal valant 21, et $(|x| - |y|)c_{del} = (30 - 9) \times c_{del} = 21$.

Le coût d'un alignement global de (x, y) quand $|y| << |x|$ est bien $(|x| - |y|)c_{del}$ car l'alphabet étant petit, la lettre y_i sera forcément compris dans x et sera positionné après la lettre x_i de x, et donc pour aligner y_i il suffit de rajouter des gaps.

Question 31

Cette implémentation n'est pas une bonne idée.
En effet, prenons par exemple $x = ACTCGATAAA$ et $y = CGT$ avec $c_{del} = 1$ et $c_{sub} = 2$.
En sachant que les gaps en début et en fin de mot de $\bar{x}$ et de $\bar{y}$ ont un coût nul, on peut obtenir, avec cet algorithme, l'alignement minimal suivant :
$\bar{x} : A\ C\ T\ C\ G\ A\ T\ A\ A\ A\ - - -$
$\bar{y} : - - - - - - - - - - - - \ C\ G\ T$
Cet alignement de coût minimal valant 0 disloque complètement y, alors que l'alignement
$\bar{x} : A\ C\ T\ C\ G\ A\ T\ A\ A\ A$
$\bar{y} : - - - C\ G\ T - - - -$
qui a un coût supérieur, valant 2, est plus pertinent.

Question 32

On pourrait résoudre les problèmes $BEST\_SCORE$ en ajoutant dans $DIST\_1$ un tableau en plus représentant le score. A chaque lettre visitée, on vérifie si elle forme un facteur avec les lettres précédents. Si oui, on remplit la case de du tableau par le score du facteur incrémenté de 1, sinon on le décrémente.
ALI_LOCAL sera alors identique à SOL_1, où on aura ajouté en plus une vérification prenant en compte le tableau de score.