Sorbonne Université

Mini-projet

LU3IN003

Alignement de séquences

Marija Lazaroska
Filip Sotiroski

Pierre Fouilhoux
Parham Shams

4 décembre 2019

2 Le problème d'alignement de séquences

Question 1 Montrons 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 $(x \cdot u, y \cdot v)$.
Comme $(\bar{x}, \bar{y})$ et $(\bar{u}, \bar{v})$ sont respectivement des alignements de (x, y) et (u, v) on a:

i. $\pi(\bar{x}) = x$ $\qquad \pi(\bar{u}) = u$

ii. $\pi(\bar{y}) = y$ $\qquad \pi(\bar{v}) = v$

iii. $|\bar{x}| = |\bar{y}|$ $\qquad |\bar{u}| = |\bar{v}|$

iv. $\forall i \in [1..|\bar{x}|], \bar{x}_i \neq -$ ou $\bar{y}_i \neq -$ $\qquad \forall j \in [1..|\bar{u}|], \bar{u}_j \neq -$ ou $\bar{v}_j \neq -$

Donc, par définition 2.2

$$\pi(\bar{x} \cdot \bar{u}) = \pi(\bar{x}) \cdot \pi(\bar{u}) \overset{i.}{=} x \cdot u \text{ et } \pi(\bar{y} \cdot \bar{v}) = \pi(\bar{y}) \cdot \pi(\bar{v}) \overset{ii.}{=} y \cdot v$$

$$|\bar{x} \cdot \bar{u}| = |\bar{x}| + |\bar{u}| \overset{iii.}{=} |\bar{y}| + |\bar{v}| = |\bar{y} \cdot \bar{v}|$$

soit $n = |\bar{x}| \overset{iii.}{=} |\bar{y}|, m = |\bar{u}| \overset{iii.}{=} |\bar{v}|$, le mot $a = \bar{x} \cdot \bar{u} = \bar{x}_1..\bar{x}_i..\bar{x}_n\bar{u}_1..\bar{u}_j..\bar{u}_m$ et le mot $b = \bar{y} \cdot \bar{v} = \bar{y}_1..\bar{y}_i..\bar{y}_n\bar{v}_1..\bar{v}_j..\bar{v}_m$
si $\forall i \in [1..|\bar{x}|], \bar{x}_i \neq -$ ou $\bar{y}_i \neq -$ et $\forall j \in [1..|\bar{u}|], \bar{u}_j \neq -$ ou $\bar{v}_j \neq -$, alors
$\forall k \in [1..|\bar{x} \cdot \bar{u}|], a_k \neq -$ ou $b_k \neq -$

En utilisant la définition 2.2, nous avons prouvé 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 $(x \cdot u, y \cdot v)$.

Question 2 Supposons que la propriété d'alignement pour (x, y) soit respectée et pour chaque couple $(x, y) \in \Sigma^* \times \Sigma^*$, nous avons donc

$$\max(x, y) = |x| + |y| = n + m$$

Exemple: $x = AB$ et $y = AAC$, $n = |x| = 2, m = |y| = 3$

$$max(x, y) = 5 = 2 + 3 = |x| + |y| = n + m$$

si $max(x, y) > 5$, par exemple $max(x, y) = 6$, on peut avoir:

$\bar{x}$: $A\ B - - - -$
$\bar{y}: - - A\ A\ C -$

un alignement qui ne respecte pas la condition iv de la définition 2.2, c'est à dire, $i = 6$, $x_i = -$ et $y_i = -$

3 Algorithmes pour l'alignement de séquences

3.1 Méthode naïve par énumération

Question 3 Pour savoir combien y a-t-il de mots $\bar{x}$ pour un mot $x \in \bar{\Sigma}^*$ avec k gaps ajoutées on utilise la formule binomial $\binom{n+k}{n}$ avec $n = |x|$.

Question 4 Une fois ajoutés k_x gaps à x pour obtenir un mot $\bar{x} \in \bar{\Sigma}^*$, $k_y = n + k_x - m$ gaps seront ajoutés à y, avec $n = |x|, m = |y|$.
Il y a

$$\binom{n+k_x}{n} \cdot \left(\binom{m+k_y}{m} - \sum_{i=1}^{k_x} \binom{m+k_y-i}{m}\right)$$

façons d'insérer k_y gaps dans y.

$\binom{n+k_x}{n}$ - nombre total de façon d'insérer k_x gaps dans x

$\binom{m+k_y}{m}$ - nombre total de façon d'insérer k_y gaps dans y

$\sum_{i=1}^{k_x} \binom{m+k_y-i}{m}$ - nombre de façon d'insérer k_y gaps dans y où $\bar{x}_i = -$ et $\bar{y}_i = -$

Pour $|x| = 15$ et $|y| = 10$, $\binom{n+k_x}{n} \cdot \left(\binom{m+k_y}{m} - \sum_{i=1}^{k_x} \binom{m+k_y-i}{m}\right) = 80080$.

Question 5 Pour trouver la complexité temporelle d'un algorithme naïf qui énumère tous les alignements de deux mots x et y on peut utiliser la formule de Question 4. On doit trouver tous les alignements de x, qui sont en $O((n + k_x)n)$ d'après la définition de complexité temporelle d'une formule binomial. Ce qui est $O(n^2)$ car $n > k_x$. Pour chaque $\bar{x}$ on cherche $\bar{y}$. Il existe $O((m + k_y)m) = O(m^2)$ façons pour trouver tous les $\bar{y}$ pour un $\bar{x}$. Donc pour chaque $\bar{x}$ on va trouver $O(n^2m^2)$ alignements, ce qui est la complexité temporelle de l'algorithme.

Question 6 Chaque $\bar{x}$ a une complexité spatiale de $O(n + k_x)$ et est stocke dans un tableau de taille n^2 (nombre des combinassions pour x avec gap k_x). Donc pour ce tableau, la complexité spatiale est en $O(n^3)$. Chaque $\bar{y}$ a une complexité spatiale de $O(m + k_y)$ et stocke dans un tableau de taille m^2. Pour trouver l'alignement optimal, un $\bar{x}$ doit être stocke avec tous les $\bar{y}$. Donc la complexité spatiale est en $O(n^3m^3)$.

Tache A

- Évaluation de temps dépendant de la taille d'instances

Instance	TailleX	TailleY	Seconds
Inst_0000010_7	x:10	y:10	5.566756010055542
Inst_0000010_8	x:10	y:9	2.2577273845672607
Inst_0000010_44	x:10	y:5	0.030281782150268555
Inst_0000012_13	x:12	y:9	10.14145803451538
Inst_0000012_32	x:12	y:9	10.197986602783203
Inst_0000012_56	x:12	y:11	74.36218762397766
Inst_0000013_45	x:13	y:12	409.54640221595764
Inst_0000013_56	x:13	y:12	413.4217278957367
Inst_0000013_89	x:13	y:12	414.7486081123352
Inst_0000014_7	x:14	y:12	911.9344797134399
Inst_0000014_23	x:14	y:10	118.05104804039001
Inst_0000014_83	x:14	y:10	116.8823618888855

À partir de l'instance Inst_0000012_56 , on peut voir que le temps d'exécution de la fonction est supérieur à une minute. Nous avons constaté que le temps d'exécution est plus petit si la différence de longueur des mots x et y est plus grande.

- Estimation de consommation mémoire nécessaire au fonctionnement de méthode DIST_NAIF

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
81	filip	20	0	24260	7176	2864	R	100.0	0.1	0:15.62	python3

On remarque que la fonction pendant son exécution pour l'instance Inst_0000500_3.adn utilise 7176 kb de mémoire. Cette valeur ne change pas pendant le temps d'exécution. Pour l'instance Inst_0008000_32.adn sur notre PC on a obtenu une exception *"RecursionError: maximum recursion depth exceeded in comparison"*. Donc le systeme a arrêté le programme.

3.2 Programmation dynamique

3.2.1 Calcul de la distance d'édition par programmation dynamique

Question 7 Soit $(\bar{u}, \bar{v})$ un alignement de $(x_{[1..i]}, y_{[1..j]})$ de longueur l.
Si $\bar{u}_l = -$ alors $\bar{v}_l = y_j$ parce que si $\bar{v}_l \neq y_j$:

1 cas: $\bar{v}_l = -$, la condition iv de la définition 2.2 ne sera pas respectée

2 cas: $\bar{v}_l = y_k$ où $1 \leq k < j$ le mot $\bar{v}$ aura une longueur $> l$

Si $\bar{v}_l = -$ alors $\bar{u}_l = x_i$ parce que si $\bar{u}_l \neq x_i$:

1 cas: $\bar{u}_l = -$, la condition iv de la définition 2.2 ne sera pas respectée

2 cas: $\bar{u}_l = x_k$ où $1 \leq k < i$ le mot $\bar{u}$ aura une longueur $> l$

Si $\bar{u}_l \neq -$ et $\bar{v}_l \neq -$, $\bar{u}_l = x_i$ et $\bar{v}_l = y_j$

Question 8

$$C(\bar{u}, \bar{v}) = C(\bar{u}_{[1...l-1]}, \bar{v}_{[1...l-1]}) + \begin{cases} C_{ins}(\bar{u}_l, \bar{v}_l), & \text{si } \bar{u}_l = - \\ C_{del}(\bar{u}_l, \bar{v}_l), & \text{si } \bar{v}_l = - \\ C_{sub}(\bar{u}_l, \bar{v}_l), & \text{si } \bar{u}_l \neq - \text{ et } \bar{v}_l \neq - \end{cases}$$

Question 9 Pour $i = [1..n]$, $j = [1..m]$, $i' \leq i$ et $j' \leq j$ où $(i', j') \neq (i, j)$, la distance d'édition de x_i à y_j est:

$D(i, j) = min(D(i', j) + c_{del}(x_i, y_j), D(i, j') + c_{ins}(x_i, y_j), D(i', j') + c_{sub}(x_i, y_j))$

Pour trouver chaque $D(i, j)$ d'alignement (x_i, y_j), on cherche le minimum des trois cas à partir desquels on peut obtenir $D(i, j)$. Soit $x_{i'}$ avec $i' \leq i$ et $y_{j'}$ avec $j' \leq j$ et $D(i', j')$ la distance d'alignement $(x_{i'}, y_{j'})$

1$^{\text{er}}$ cas: On prend les mots $x_{i'}$ et y_j. Pour obtenir le mot x_i on doit ajouter une lettre à $x_{i'}$.
Supposons que $D(i', j)$ est la plus petite distance pour l'alignement $(x_{i'}, y_j)$. En ajoutant le coût de suppression $c_{del}(x_i, y_j)$,car $x_i \neq -$ et $y_j = -$, on obtient la distance $D(i, j)$ qui est le candidat potentiel pour devenir la distance minimale.

2$^{\text{me}}$ cas: On prend les mots x_i et $y_{j'}$. Pour obtenir le mot y_j on doit ajouter une lettre à $y_{j'}$.
Supposons que $D(i, j')$ est la plus petite distance pour l'alignement $(x_i, y_{j'})$. En ajoutant le coût de insertion $c_{ins}(x_i, y_j)$,car $x_i = -$ et $y_j \neq -$, on obtient la distance $D(i, j)$ qui est le candidat potentiel pour devenir la distance minimale.

3$^{\text{me}}$ cas: On prend les mots $x_{i'}$ et $y_{j'}$. Pour obtenir les mots x_i et y_j on doit ajouter une lettre à $x_{i'}$ et à $y_{j'}$.
Supposons que $D(i', j')$ est la plus petite distance pour l'alignement $(x_{i'}, y_{j'})$. En ajoutant le coût de substitution $c_{sub}(x_i, y_j)$,car $x_i \neq -$ et $y_j \neq -$, on obtient la distance $D(i, j)$ qui est le candidat potentiel pour devenir la distance minimale.

Question 10 $D(0, 0)$ est la distance d'édition de mot vide ε à mot vide ε, donc il vaut 0.

Question 11
Pour $j \in [1..m]$, $D(0, j) = \sum_{k=1}^{l} c_{ins}(-, v_k)$
Pour $i \in [1..n]$, $D(i, 0) = \sum_{k=1}^{l} c_{del}(u_k, -)$

Question 12

---

procedure DIST_1(X,Y)
  $n \leftarrow$ taille de mot x
  $m \leftarrow$ taille de mot y
  T variable global, tableau à deux dimensions (n+1)*(m+1)
  c_{ins}, c_{del}, c_{sub} variables globales
  Initialisation de T avec des zéros
  for $j \leftarrow 1$ `to` m **do**
    $T_{0,j} \leftarrow c_{ins} * j$
  for $i \leftarrow 1$ `to` n **do**
    $T_{i,0} \leftarrow c_{del} * i$
  for $i \leftarrow 1$ `to` n **do**
    for $j \leftarrow 1$ `to` m **do**
      $T_{i,j} \leftarrow min(T_{i-1,j} + c_{del}, T_{i,j-1} + c_{ins}, T_{i-1,j-1} + c_{sub}(x_{i-1}, y_{j-1})$
  return $T_{n,m}$

Question 13 La complexité spatiale de cette fonction est $O(nm)$ car nous utilisons une liste de listes (tableau à deux dimensions) afin de stocker les distances d'édition.

Question 14 La fonction crée une liste des listes pour le coût de $O(nm)$. Ensuite, nous parcourons une ligne n et une colonne m avec $O(n)$ et $O(m)$. Enfin, nous bouclons la liste des listes afin d'ajouter la distance minimale pour chaque i et j supérieur à zéro. On peut en conclure que la complexité temporaire est en $O(nm + n + m + nm) = O(nm)$

3.2.2 Calcul d'un alignement optimal par programmation dynamique

Question 15 A l'aide des trois cas on peut arriver a un alignement de distance minimal. On montre:

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

Supposons que $D(i-1,j-1)$ c'est la distance minimale de l'alignement $Al^*(i-1,j-1)$. Si la somme de la distance $D(i-1,j-1)$ et le coût de substitution de (x_i, y_j) est égale a $D(i,j)$ on peut déduire que en ajoutant a l'alignement $Al^*(i-1,j-1)$ les lettres x_i et y_j on obtient l'alignement pour la distance minimale $D(i,j)$.

Question 16

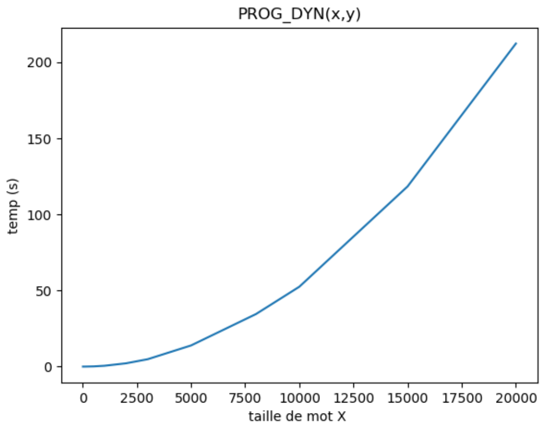
procedure SOL_1(T,X,Y)
    $\bar{x} \leftarrow$ tableau vide
    $\bar{y} \leftarrow$ tableau vide
    $i \leftarrow$ taille de mot x
    $j \leftarrow$ taille de mot y
    c_{ins}, c_{del}, c_{sub} variables globales
    while $i > 0$ or $j > 0$ **do**
        if $j > 0$ and $T_{i,j} = T_{i,j-1} + c_{ins}$ **then**
            ajout de $-$ au début du tableau $\bar{x}$
            ajout d'élément y_{j-1} au début du tableau $\bar{y}$
            $j \leftarrow j-1$
        else if $i > 0$ and $T_{i,j} = T_{i-1,j} + c_{del}$ **then**
            ajout de x_{i-1} au début du tableau $\bar{x}$
            ajout d'élément $-$ au début du tableau $\bar{y}$
            $i \leftarrow i-1$
        else if $i > 0$ and $j > 0$ and $T_{i,j} = T_{i-1,j-1} + c_{sub}(x_{i-1}, y_{j-1})$ **then**
            ajout d'élément x_{i-1} au début du tableau $\bar{x}$
            ajout d'élément y_{j-1} au début du tableau $\bar{y}$
            $i \leftarrow i-1$
            $j \leftarrow j-1$
    return $\bar{x}$,$\bar{y}$

Question 17 La complexité temporelle de DIST_1 est $O(nm)$. Dans la fonction SOL_1 il y a une boucle qui peut avoir au plus $O(n+m)$ itérations. A chaque itération on ajout une lettre dans le mot avec un coût de $O(n)$ pour x et $O(m)$ pour y. Donc, la complexité temporelle de SOL_1 est $O((n+m)^2)$
Si on combine les deux fonction, on obtient une complexité de $O(nm + (n+m)^2)$ Donc, si la taille de x est plus grand que la taille de y on a la complexité de $O(n^2)$. Dans le cas contraire $O(m^2)$

Question 18 La complexité spatiale DIST_1 est $O(nm)$. Dans SOL_1 on crée deux listes qui peut avoir une taille maximale de $O(n+m)$ Si on combine les deux fonctions, on obtient une complexité de $O(nm + 2(n+m))$. Ce qui est donc $O(nm)$.

TACHE B

- Courbe de consommation de temps CPU en fonction de taille $|x|$



La complexité théorique est polynomial. La courbe que on a obtenu est aussi une courbe de fonction polynomial.

- Estimation de quantité mémoire utilise par PROG_DYN

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
20371 3874261   20   0 28,338g 0,028t   4888 R 100,0 90,6   4:35.95 python3
```

Pour l'instance Inst_0050000_63.adn, la fonction a utilisé beaucoup de mémoire et elle a bloqué l'ordinateur dans les salles de PPTI quand elle a dépasse 90.6% de mémoire disponible après plus de 4 minutes. On remarque aussi que elle a utilise plus de 28 GB de mémoire. Et donc, avec cette fonction on pourrait pas calculer le meilleur alignement pour une instance de longueur de 50000 lettres.

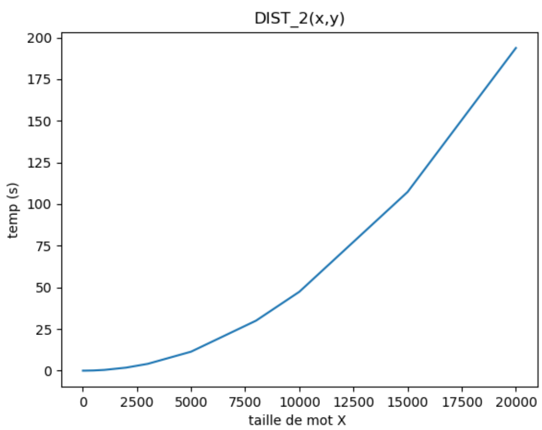
Question 19 On remarque que dans la fonction DIST_1 pour obtenir la distance $D(i, j)$ on utilise soit $D(i-1, j)$, soit $D(i-1, j-1)$, soit $D(i, j-1)$, donc on fait un accès aux lignes i et $i-1$ a chaque tour de boucle. On peut optimiser notre fonction en utilisant deux lignes avec m colonnes $(2*m)$ au lieux d'une matrice de taille $n*m$.

Question 20

procedure DIST_2(X,Y)
  $n \leftarrow$ taille de mot x
  $m \leftarrow$ taille de mot y
  D tableau à deux dimensions 2*(m+1)
  Initialisation de D avec des zéros
  for $j \leftarrow 1$ `to` m **do**
    $D_{0,j} \leftarrow c_{ins} * j$
  for $i \leftarrow 1$ `to` n **do**
    $D_{1,0} \leftarrow c_{del} * i$
    for $j \leftarrow 1$ `to` m **do**
      $D_{1,j} \leftarrow min(D_{1,j-1} + c_{del}, D_{0,j} + c_{ins}, D_{0,j-1} + c_{sub}(x_{i-1}, y_{j-1})$
    if $i! = n$ **then**
      $tmp \leftarrow D_0$
      $D_0 \leftarrow D_1$
      $D_1 \leftarrow tmp$
  return $D_{1,m}$

TACHE C

- Courbe de consommation de temps CPU en fonction de taille $|x|$



- Comparaison des résultats obtenu avec DIST_1 et DIST_2
  On remarque que les deux courbes sont presque les mémés pour les mémés tailles. On peut donc déduire que la complexité temporelle est la mémé. Par contre, l'utilisation de mémoire est beaucoup mieux pour la fonction DIST_2.

- Estimation de quantité mémoire utilise par DIST_2

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
21399	3874261	20	0	29428	13512	4892	R	100,0	0,0	4:58.23	python3

Pour calculer l'instance Inst_0050000_63.adn qui consiste de longueur de 50000 lettres, on utilise beaucoup moins de mémoire que la fonction dans TACHE B. Après 5 minutes d'exécution de cette instance, on utilise 13512 kb de mémoire. Et la mémoire ne pose pas de problèmes pendant l'exécution.

Question 21

```
procedure MOT_GAPS(K)
    mot_gaps tableau vide de taille k
    for i from 1 to k do
        mot_gaps_i ← '−'
    return mot_gaps
```

Question 22

```
procedure ALIGN_LETTRE_MOT(X,Y)
    m ← taille de y
    mot_x ← MOT_GAPS(m)
    coût ← inf
    indice ← 0
    for i from 1 to m do
        if x = y_i then
            indice ← i
            break
        else
            if c_sub(x, y_i) < coût then
                coût ← c_sub(x, y_i)
                indice ← i
    mot_x_indice ← x
    return mot_x, y
```

Question 23

$x = BALLON$, $x^1 = BAL$, $x^2 = LON$

$y = ROND$, $y^1 = RO$, $y^2 = ND$

$(\bar{s}, \bar{t})$ alignement optimal de (x^1, y^1), (BAL, RO):

$$\begin{matrix} B & A & L \\ R & O & - \end{matrix}$$

$(\bar{u}, \bar{v})$ alignement optimal de (x^2, y^2), (LON, ND):

$$\begin{matrix} L & O & N \\ N & - & D \end{matrix}$$

On obtient $(\bar{s} \cdot \bar{u}, \bar{t} \cdot \bar{v})$:

$$\begin{matrix} B & A & L & L & O & N \\ R & O & - & N & - & D \end{matrix}$$

$d(x, y) = D(\bar{s}, \bar{t}) + D(\bar{u}, \bar{v}) =$
$c_{sub}(B, R) + c_{sub}(A, O) + c_{del}(L, -) + c_{sub}(L, N) + c_{del}(O, -) + c_{sub}(N, D) = 5+5+3+5+3+5 = 26$

Mais l'alignement $(\bar{s} \cdot \bar{u}, \bar{t} \cdot \bar{v})$ n'est pas un alignement optimal, car il existe un alignement plus optimal:

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

$$- - - R\ O\ N\ D$$

$d(x,y) = c_{del}(B,-)+c_{del}(A,-)+c_{del}(L,-)+c_{sub}(L,R)+c_{sub}(O,O)+c_{sub}(N,N)+c_{ins}(-,D) = 3+3+3+5+0+0+3 = 17.$

Question 24

```
XA ← liste vide globale
YA ← liste vide globale

procedure SOL_2(x,y)
    n ← taille de x
    m ← taille de y
    if n > 1 and m >= 1 then
        i ← ⌊|x|/2⌋
        j ← coupure(x, y)
        SOL_2(x[0..i],y[0..j])
        SOL_2(x[i..n],y[j..m])
    else if n = 1 and m > 1 then
        a ← align_lettre_mot(x, y)[0]
        for each lettreX from a do
            XA ← ajouter en fin de liste(lettreX)
        for each lettreY from y do
            YA ← ajouter en fin de liste(lettreY)
    else
        if n = 0 then
            for each lettreY from y do
                XA ← ajouter en fin de liste('−')
        else
            for each lettreX from x do
                XA ← ajouter en fin de liste(lettreX)
        if m = 0 then
            for each lettreX from x do
                YA ← ajouter en fin de liste('−')
        else
            for each lettreY from y do
                YA ← ajouter en fin de liste(lettreY)
```

Question 25

1: **procedure** COUPURE(X,Y)
2: $n \leftarrow$ taille de x
3: $m \leftarrow$ taille de y
4: $iStar \leftarrow \left\lfloor \frac{|x|}{2} \right\rfloor$
5: D tableau à deux dimensions 2*(m+1)
6: I tableau à deux dimensions 2*(m+1)
7: Initialisation de D avec des zéros
8: Initialisation de I avec des zéros
9: **for** $j \leftarrow 1$ `to` $m+1$ **do**
10: $D_{0,j} \leftarrow c_{ins} * j$
11: $I_{0,j} \leftarrow j$
12: **for** $i \leftarrow 1$ `to` $n+1$ **do**
13: $D_{1,0} \leftarrow c_{del} * i$
14: $I_{1,0} \leftarrow 0$
15: **for** $j \leftarrow 1$ `to` $m+1$ **do**
16: $D_{1,j} \leftarrow min(D_{1,j-1} + c_{del}, D_{0,j} + c_{ins}, D_{0,j-1} + c_{sub}(x_{i-1}, y_{j-1})$
17: **if** $i > iStar$ **then**
18: **if** $D_{1,j} = D_{1,j-1} + c_{del}$ **then**
19: $I_{1,j} \leftarrow I_{1,j-1}$
20: **else if** $D_{1,j} = D_{0,j} + c_{ins}$ **then**
21: $I_{1,j} \leftarrow I_{0,j}$
22: **else if** $D_{1,j} = D_{0,j-1} + c_{sub}(x_{i-1}, y_{j-1})$ **then**
23: $I_{1,j} \leftarrow I_{0,j-1}$
24: **if** $i > iStar$ and $i! = n$ **then**
25: $tmp \leftarrow I_0$
26: $I_0 \leftarrow I_1$
27: $I_1 \leftarrow tmp$
28: **if** $i! = n$ **then**
29: $tmp \leftarrow D_0$
30: $D_0 \leftarrow D_1$
31: $D_1 \leftarrow tmp$
32: **return** $I_{1,m}$

Question 26 La fonction coupure (x, y) utilise deux tableaux a deux dimensions D et I qui ont deux lignes et $m+1$ colonnes. Donc, la complexité spatiale de la fonction est linéaire $O(m)$.

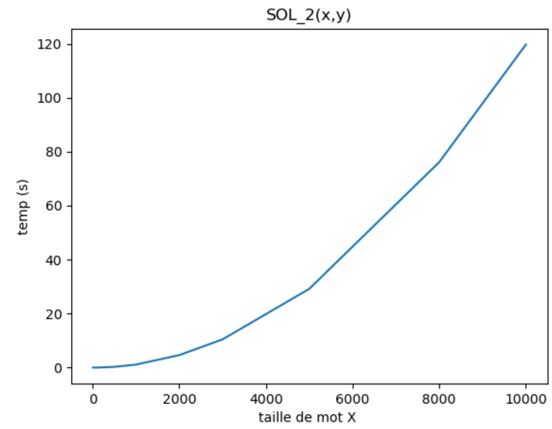
Question 27 On utilise deux listes globales XA et YA de taille maximale $m+n$. La fonction *align_lettre_mot*(x, y) retourne une liste de taille m. Donc, la complexité spatiale de la fonction est linéaire $O(m+n)$.

Question 28 La création et initialisation de deux tableaux a deux dimensions coûte $O(2m)$. A la ligne 9 on a un boucle for qui coûte $O(m)$. A la ligne 12 et 15 on a deux boucles imbriques qui ont des instructions élémentaires. Donc ça complexité est en $O(mn)$.

A la fin, on peut conclure que la complexité temporelle de coupure(x,y) est en $O(nm)$.

TACHE D

- Courbe de consommation de temps CPU en fonction de taille $|x|$



- Estimation de quantité mémoire utilisé par SOL_2

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
22403	3874261	20	0	26876	10148	4924	R	100,0	0,0	7:17.14	python3

La fonction SOL_2 pendant l'exécution d'instances de taille 10 a 100000 lettres, après 7 minutes d'exécution on remarque qu'elle a utilisé 10148 kb. Donc, la fonction ne pose pas des problèmes pour l'exécution des instances des grandes tailles.

Question 29

$\|x\|$	SOL_1 temps (s)	SOL_2 temps (s)
10	0.0	0.0001
12	0.0001	0.0002
13	0.0001	0.0002
14	0.0001	0.0002
15	0.0002	0.0003
20	0.0002	0.0006
50	0.0011	0.003
100	0.0049	0.0107
500	0.1284	0.2643
1000	0.534	1.109
2000	2.1504	4.6042
3000	4.8437	10.4477
5000	13.8837	29.1532
8000	34.534	76.1142
10000	52.483	119.6985

La fonction SOL_2 prend plus de temps pour trouver l'alignement de mot x que la fonction SOL_1. On peut constater que on a perdu en complexité temporelle en améliorant la complexité spatiale.

```
  PID USER       PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
22782 3874261    20   0 2707556 2,549g   4836 R  99,7  8,2   1:56.23 python3
```

On remarque que si on fait la mémé exécution comme dans l'exemple de la mémoire de TACHE D, nous aurons besoin plus de 2 GB de mémoire après 2 minutes d'exécution.