SORBONNE UNIVERSITÉ

Mini-projet LU3IN003:

Alignement de séquences d'ADN

Réalisé par :

BOUSBA Abdellah

TOULA Hider

Encadré par :

Anne-Elisabeth Falq

3ème année Licence Informatique

Mono-disciplinaire

Groupe n°6

SOMMAIRE

DEFINITIONS :

Question 1

$(\bar{x},\bar{y})$ est un alignement de (x,y) veut dire que : $\begin{cases} |\bar{x}|=|\bar{y}| \\ \forall\, i \in [1\,..|\bar{x}|],\ \bar{x}_i \neq - \text{ ou } \bar{y}_i \neq - \end{cases}$

$(\bar{u},\bar{v})$ est un alignement de (u,v) veut dire que : $\begin{cases} |\bar{u}|=|\bar{v}| \\ \forall\, i \in [1\,..|\bar{u}|],\ \bar{u}_i \neq - \text{ ou } \bar{v}_i \neq - \end{cases}$

Donc :

$$\left.\begin{array}{l} |\bar{x}|=|\bar{y}| \\ |\bar{u}|=|\bar{v}| \end{array}\right\} \Rightarrow \quad |\bar{x}.\bar{u}|=|\bar{y}.\bar{v}|$$

Et puisque :

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

Alors:

$\forall\, i \in [1\,..|\bar{x}.\bar{u}|],\ (\bar{x}.\bar{u})_k \neq - \text{ ou } (\bar{y}.\bar{v})_k \neq -$

Conclusion :

$(\bar{x},\bar{y})$ et $(\bar{u},\bar{v})$ alignements de resp. (x,y) et (u,v) $\Rightarrow$ $(\bar{x}.\bar{u},\bar{y}.\bar{v})$ alignement de $(x.u,y.v)$

Question 2

La longueur maximale d'un alignement (x,y) de longueur respectivement n,m est : **n+m**

Exemple :

Soit Σ= {A,G,T,C} avec x =GTA et y=AAAAAA donc n=3 et m=6 , l'alignement de taille maximale est :

$\bar{x}$ = GTA−−−−−−
$\bar{y}$ = −−−AAAAAA

On a bien n+m = 3+6 = 9 la taille de l'alignement ci-dessus.

METHODE NAIF :

Question 3

On sait que la longueur de $\bar{x}$ après positionnement de k gaps dans x est n+k , en sachant que on ne peut pas changer l'ordre de lettre et que l'ordre des gaps et le même on a :

Le nombre de mots $\bar{x}$ avec k gaps est : C^{k}_{n+k}

Question 4

On doit ajouter k' = n+k-m gaps a y

Pour un k donné on a : $C^{k}_{n+k} \times C^{k'}_{n}$ alignements possible

On sait que la taille max d'un alignement est n+m alors k ∈ [0,m] donc le nombre d'alignement possible est :

$$\sum_{k=1}^{m} \left(C^{k}_{n+k} \times C^{k'}_{n} \right) \qquad \text{avec } k' = n+k-m$$

Pour n = 15 et y = 10 on a $\sum_{k=0}^{10} \frac{(k+15)! \times 15!}{15! \times k! \times (10-k)! \times (k+5)!} = 298199265$ alignements possibles

Question 5

La complexité pour trouver une distance d'édition et trouver un alignement de coût minimal est : le nombre d'alignements possible car il sera obliger énumérer tout les alignements.

Donc : nombre alignement possible qui veut dire $\Theta\left(\sum_{k=1}^{m} \left(C^{k}_{n+k} \times C^{k'}_{n} \right) \right)$

Question 6

La complexité spatiale d'un algorithme naïf :

Pour trouver la distance : $\Theta(1)$ pour la variable dist

Pour trouver un alignement minimale : $O(n+m)$ pour le tableau de taille max n+m

En prenant en considération la pile des appel récursif on est en $\Theta\left(\sum_{k=1}^{m} \left(C^{k}_{n+k} \times C^{k'}_{n} \right) \right)$

Tache A

Les tests son indiquer sur le fichier testTacheA.py, les résultats son les suivants :

fichier : Instances_genome/Inst_0000010_7.adn
Distance naif : 8 time used is 5.924769 s
fichier : Instances_genome/Inst_0000010_8.adn
Distance naif : 2 time used is 2.388596 s
fichier : Instances_genome/Inst_0000010_44.adn
Distance naif : 10 time used is 0.032060 s
fichier : Instances_genome/Inst_0000012_32.adn ← **plus grande instance < 1min**
Distance naif : 6 time used is 10.652283 s
fichier : Instances_genome/Inst_0000012_56.adn
Distance naif : 9
time used is 77.781131 s

Plus grande instance execute: Instances_genome/Inst_0000500_8.adn

Consumation mémoire : pour toute instance la consommation est de 0.1% mais la pile des appels récursifs ne peut pas exécuteur une instance plus grande que n = 500 qui est cohérent car la fonction est en **Θ(1)** mais en prenant compte de la pile des appels récursif on est en **Θ(** $\sum_{k=1}^{m}(C^{k}_{n+k} \times C^{k'}_{n})$**)**

Complexité:

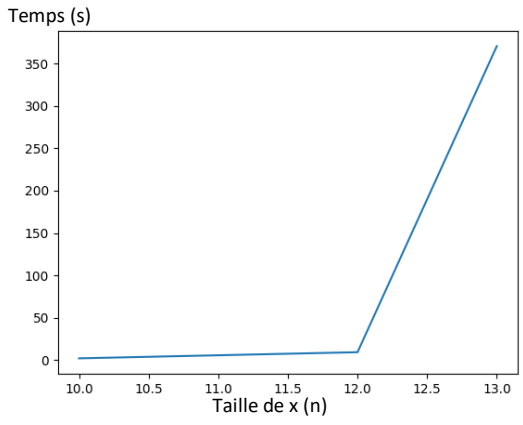

Figure 1 -Performance de la méthode DIST_NAIF

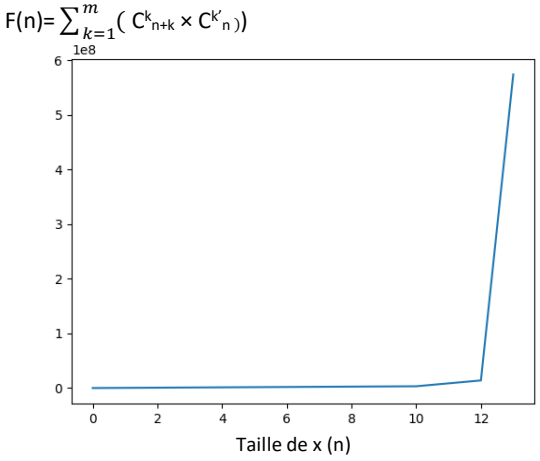

Figure 2 - Attente Théorique

En considérant le graphe de la Figure 2 a partir de n=10 on remarque que c'est exactement le comportement de DIST_NAIF.

PROGRAMMATION DYNAMIQUE

1. CALCUL DE LA DISTANCE

Question7

$(\bar{u},\bar{v})$ est un alignement de $(u_{[1..i]},v_{[1..j]})$ veut dire que : $\forall\, i \in [1\,..|\bar{u}|], \bar{u}_i \neq - \text{ ou } \bar{v}_i \neq -$

donc :

Si on a $\bar{u}_l = -$ alors $\bar{v}_l$ doit être le dernier caractère de v donc v_j car on ne peut pas changer l'ordre des caractères. Et donc si on a $\bar{v}_l = -$ alors $\bar{u}_l = u_i$

Puis si on a $\bar{u}_l \neq -$ ou $\bar{v}_l \neq -$on aura $\bar{u}_l = u_i$ et $\bar{v}_l = v_j$

Question8

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

Question9

On sait que D(i,j) et la distance minimale et que le cout de la case (i,j)est obtenue a partir des case (i,j-1) , (j-1,i-1) ,(i-1,j)

Donc :

$$D(i,j) = \min \begin{cases} D(i-1,j-1) + C_{sub}(x_i,y_j) \\ D(i-1,j) + C_{del} \\ D(i,j-1) + C_{ins} \end{cases}$$

Question10

$D(0,0) = d(x_{[1..0]}, y_{[1..0]}) = d(\emptyset,\emptyset) = 0$ car la distance entre deux mots vide est nulle.

Question11

On sait que D(0,j) est obtenus a partir des case (-1,j)(-1,j-1)(0 ,j-1) comme la position -1 d'une lettre d'un mot n'existe pas , $D(0,j-1) + C_{ins}$ est la seule option donc :

$D(0,j) = D(0,j-1) + C_{ins}$

$D(0,j) = D(0,j-2) + 2 \times C_{ins}$

.

.

.

$D(0,j) = D(0,0) + j \times C_{ins}$

et selon la question 10 on a $D(0,0) = 0$ donc $D(0,j) = j \times C_{ins}$

De la même façon pour i $D(i,0) = i \times C_{del}$

Question12

```
DIST_1
    Entrée : x et y deux mots
    Sortie : distance entre x et y
    T = tableau indexé par [o..n][o..m]
    T[o][o] =o ;
    Pour i de 1 à n+1 faire
        T[i][o] = i × Cdel
    Pour j de 1 à m+1 faire
        T[o][j] = j × Cins
    Pour i de 1 à n+1 faire
        Pour j de 1 à m+1 faire
            T[i][j] = min ( T[i-1][j-1]+Csub(xi-1,yi-1),
                            T[i][j-1]+Cins, T[i-1][j] + Cdel)
    retourner T[n][m]
```

Question13

La complexité spatiale de DIST_1 est **Θ(n×m**) car dans tout les cas l'algorithme parcours et sauvegarde toute la matrice qui contient n×m cases.

Question14

La complexité temporelle de DIST_1 est **Θ(n×m)** car on parcours un tableau de deux dimension n et m pour calculer la distance minimale de chaque combinaison.

2. CALCUL D'UN ALIGNEMENT OPTIMAL

Question15

Soit $(i,j) \in [0..n] \times [0..m]$ Mq Si j>0 et D(i,j) = D(i,j-1) + C_{ins} ⇒ $\forall(\bar{u},\bar{v}) \in Al^*(i, j-1)$, $(\bar{u}.-, \bar{v}\cdot y_j) \in Al^*(i, j)$:

$D(i , j) = D(i , j-1) + C_{ins} = d(x_{[1..i]},y_{[1..j-1]}) + C_{ins}$

$= C(\bar{u},\bar{v}) + C_{ins}$ $(\bar{u},\bar{v}) \in Al^*(i, j-1))$

$=C(\bar{u}.-, \bar{v}\cdot y_j)$ *cf Question 8*

On a $D(i,j) = d(x_{[1..i]},y_{[1..j]}) = C(\bar{u}.-, \bar{v}\cdot y_j)$ alors $(\bar{u}.-, \bar{v}\cdot y_j) \in Al^*(i, j)$

Question16

```
SOL_1
    Entrée : x et y deux mots
             T tableau de deux dimension [o..n] × [o..m] remplis par DIST_1
    Sortie : alignement minimale de (x,y)
    x̄=Ø, ȳ=Ø, i=n, j=m
    tantque (j>o) or (i>o) faire
        si (j>o) and T[i][j] = T[i][j-1] + C_ins alors
            x̄= –.x̄
            ȳ=y_{j-1}.ȳ
            j--
        si (i>o) and T[i][j] = T[i-1][j] + C_del alors
            ȳ= –.ȳ
            x̄=x_{i-1}.x̄
            i--
        si (j>o) and (i>o) and T[i][j] = T[i-1][j-1] + C_sub(x_{i-1},y_{j-1}) alors
            x̄= x_{i-1}.x̄
            ȳ=y_{j-1}.ȳ
            j--
            i--
    retourner (x̄,ȳ)
```

Question17

La complexité de PROG_DYN est en **Θ(n×m)** car on doit parcourir toutes les case de la matrice pour la construire dans Dist_1 en **Θ(n×m)** puis appliquer SOL_1 pour trouver l'alignement minimale en **Θ(n×m)**

Question18

La complexité spatiale de SOL_1 est O(n+m) et de DIST_1 est Θ(n×m)

Alors PROG_DYN est en **Θ(n×m)**

Tache B

Les tests son indiquer sur le fichier testTacheB.py, les résultats son les suivants :

fichier : Instances_genome/Inst_0000010_7.adn
Distance1 + Align 1 : (8, 'TGGG-TG-CTAT', '-GGGGT-TCTAT') time used is 0.000082 s
fichier : Instances_genome/Inst_0000010_8.adn
Distance1 + Align 1 : (2, 'AACTGTCTTT', 'AACTGT-TTT') time used is 0.000076 s
fichier : Instances_genome/Inst_0000010_44.adn
Distance1 + Align 1 : (10, 'TATATGAGTC', 'TAT-T---T-') time used is 0.000113 s
fichier : Instances_genome/Inst_0000012_32.adn
Distance1 + Align 1 : (6, 'CCATTGATTTTC', 'C-ATTGATT-T-') time used is 0.000089 s
fichier : Instances_genome/Inst_0000012_56.adn
Distance1 + Align 1 : (9, 'GCTTAACTAACG', 'GCTAAACTA-CT') time used is 0.000097 s
fichier : Instances_genome/Inst_0000100_7.adn
Distance1 + Align 1 : (62, 'TGGGTGCTA….-CCTCAGT', 'TGGGTGCA….-TCCTCA-T') time used is 0.004208 s
fichier : Instances_genome/Inst_0000500_88.adn
Distance1 + Align 1 : (296, 'TGT-…GAATTTCCA-G', '-GTTCTCG-…AATTT--AAG') time used is 0.101040 s
fichier : Instances_genome/Inst_0002000_44.adn
Distance1 + Align 1 : (1078, 'TATATGA…-TGTCAGT', '-ATATGAGT…GTGTCAGA') time used is 1.669113s

Plus grande instance execute : Instances_genome/Inst_0020000_5.adn
Pour l'instance **"Instances_genome/Inst_0020000_5.adn" :**

Consommation mémoire pour DIST1 : 69% soit 11,2 GB.
Consommation mémoire pour PROG_DYN : 85% soit 13,9 GB.

Complexité:

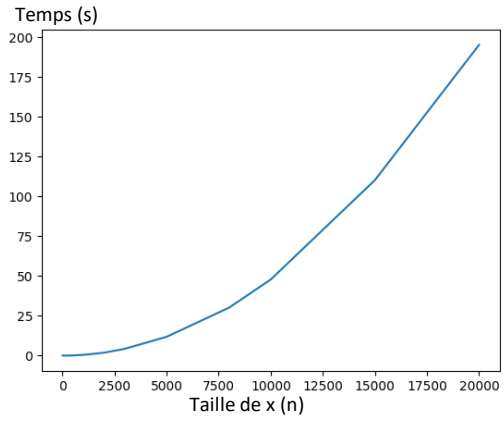

Figure 3 - Performance PROG_DYN

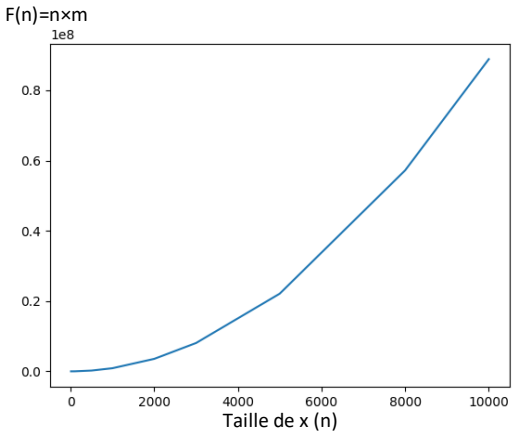

Figure 4 - Attente Théorique

On remarque que la complexité calculé expérimentalement (Figure 3) est exactement le comportement qu'on attendait de PROG_DYN théoriquement (Figure 4).

AMELIORATION DE LA COMPLIXITE SPATIALE DU CALCUL DE LA DISTANCE

Question 19

Parce que la valeur de D(i,j) est obtenu a partir de 3 valeurs D(i-1,j-1),D(i,j-1),D(i-1,j) qui se trouve a la ligne i et la ligne i-1 alors on peut utilisé seulement une matrice de deux lignes pour améliorer la complexité spatiale.

Question 20

```
DIST_2
    Entrée : x et y deux mots
    Sortie : distance entre x et y
    T = tableau indexé par [0..1][0..m]
    T[0][0] =0 ;
    Pour j de 1 à m+1 faire
        T[0][j] = j × Cins
    Pour i de 1 à n+1 faire
        T[1][0] = i × Cdel
        Pour j de 1 à m+1 faire
            T[1][j] = min ( T[0][j-1]+Csub(xi-1,yi-1),
                            T[1][j-1]+Cins, T[0][j] + Cdel)
        Pour j de 0 à m faire
            T[0][j] = T[1][j]
    retourner T[1][m]
```

Tache C

Les tests son indiquer sur le fichier testTacheC.py, les résultats son les suivants :

fichier : Instances_genome/Inst_0000010_7.adn
Distance methode 2 : 8 time used is 0.000074 s
fichier : Instances_genome/Inst_0000010_8.adn
Distance methode 2 : 2 time used is 0.000070 s
fichier : Instances_genome/Inst_0000010_44.adn
Distance methode 2 : 10 time used is 0.000047 s
fichier : Instances_genome/Inst_0000012_32.adn
Distance methode 2 : 6 time used is 0.000074 s
fichier : Instances_genome/Inst_0000012_56.adn
Distance methode 2 : 9 time used is 0.000088 s
fichier : Instances_genome/Inst_0000100_7.adn
Distance methode 2 : 62 time used is 0.004237 s
fichier : Instances_genome/Inst_0000500_88.adn
Distance methode 2 : 296 time used is 0.102112 s
fichier : Instances_genome/Inst_0002000_44.adn
Distance methode 2 : 1078 time used is 1.608261 s

En comparant la consommation mémoire pour la même instance **"Instances_genome/Inst_0020000_5.adn" :**

Consommation mémoire pour DIST1 : 69% soit 11,2 GB.
Consommation mémoire pour DIST2 : 17,5% soit 2,8 GB.
On confirme que DIST_2 est meilleur que DIST_1 en complexité spatiale.

Complexité:

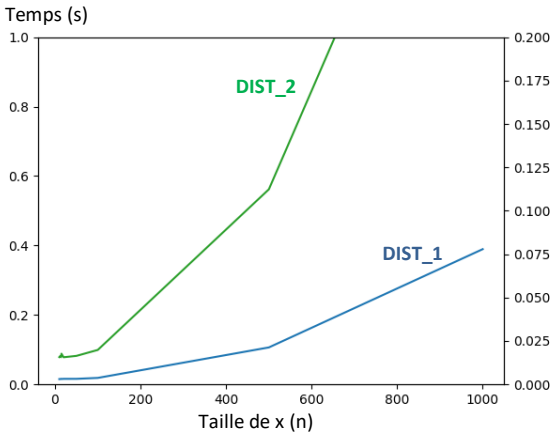

Figure 5 - Comparaison entre dist1 et dist2

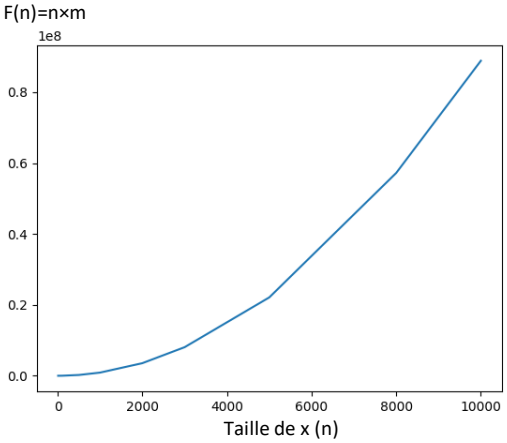

Figure 6 - Attente Theorique

L'attente théorique **Θ(n×m)**représenté dans la figure 6 est bien la même pour la complexité calcule expérimentalement DIST_1 (voir entre 0,1000) par contre pour DIST_2 est supérieur a **n×m** mais c'est logique car elle prend en compte le temps d'exécution des opération de décalage.

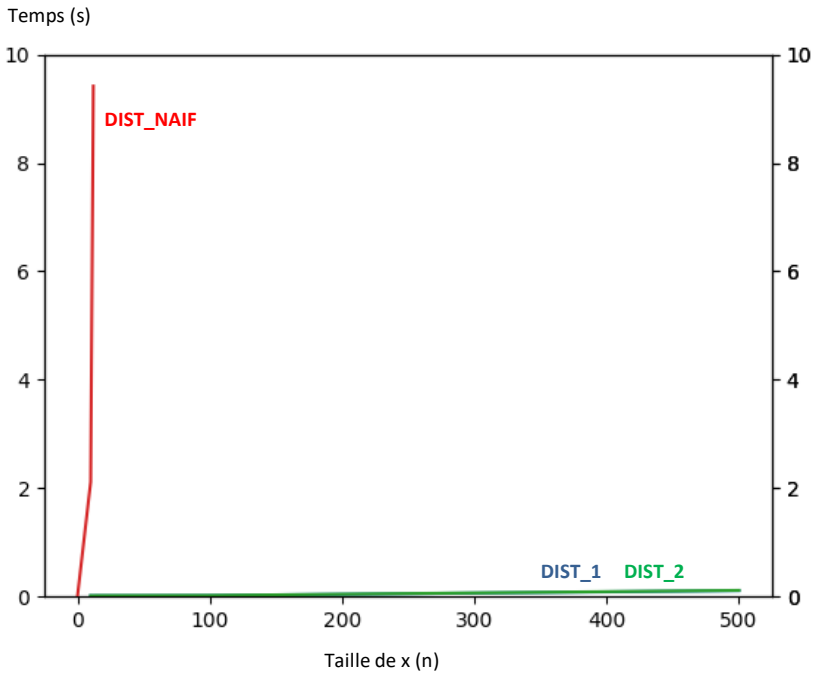

Figure 7 - Comparaison DIST_NAIF, DIST_1, DIST_2

En comparant DIST_NAIF avec DIST_1 et DIST_2 on voit clairement la différance dans la performance des fonctions.

DIVISER POUR REGNER

Question 21

```
mot_gaps
    Entrée : entier k
    Sortie : mot constituer de k gaps
    out= Ø
    Pour i de 1 à k faire
        out =out.–
    retourner out
```

Question 22

```
Align_lettre_mot
    Entrée : x et y deux mots avec x de taille 1
    Sortie : alignement minimale de (x,y)
    ȳ=y,x̄=x, min = +∞ , pos = -1
    Pour j de 1 à m faire
        si C_sub(x_o,y_j) < min alors
            min = C_sub(x_o,y_j)
            pos = i
    si (min + (m-1) × C_ins) > (C_del + m × C_ins)) alors
            x̄=x̄.mot_gaps(m)
            ȳ=ȳ.–
    sinon
        x̄=mot_gaps(pos).x̄.mot_gaps(m-pos)
    retourner (x̄,ȳ)
```

Question 23

$(\bar{x}_1,\bar{y}_1)$ est un alignement optimale de (x^1,y^1) est :

$\bar{x}_1$ = BAL
$\bar{y}_1$ = RO−

$(\bar{x}_2,\bar{y}_2)$ est un alignement optimale de (x^2,y^2) est :

$\bar{x}_2$ = LON−
$\bar{y}_2$ = −−ND

Supposant que $(\bar{x}_1.\bar{x}_2,\bar{y}_1.\bar{y}_2)$ est optimale . on a $C(\bar{x}_1.\bar{x}_2,\bar{y}_1.\bar{y}_2) = 22$

$\bar{x}_1.\bar{x}_2$= BALLON−
$\bar{y}_1.\bar{y}_2$= RO−−−ND

or il existe un alignement du même taille $(\bar{x}_3.\bar{x}_3)$ tq $C(\bar{x}_3.\bar{x}_3) = 17 < 22 < C(\bar{x}_1.\bar{x}_2,\bar{y}_1.\bar{y}_2)$ **ABS.**

$\bar{x}_3$= BALLON−
$\bar{y}_3$= R−−−OND

Donc par l'absurde $(\bar{x}_1.\bar{x}_2,\bar{y}_1.\bar{y}_2)$ n'est pas optimale.

Question 24

```
SOL_2_REC
    Entrée : x et y deux mots
    Sortie : alignement minimale de (x,y)
    si n = 0 alors
        retourner (x,mots_gaps(m))
    si m = 0 alors
        retourner (mots_gaps(n),y)

    si n = 1 alors
        retourner align_lettre_mot(x,y)
    i = [ n/2 ]
    j = coupure(x,y,i)
    (x1,y1) = SOL_2_REC(x[0..i],y[0..j])
    (x2,y2) = SOL_2_REC(x[i+1..n],y[j+1..m])
    Retourner(x1+x2,y1+y2)
```

Question 25

```
coupure
    Entrée : x et y deux mots
    Sortie : entier coupure de y
    n= |x| ,m=|y|
    D = tableau indexé par [0..1][0..m]
    I = tableau indexé par [0..1][0..m]
    pour k de 1 à m+1 faire
        I[0][k]=k
        D[0][k]=k × Cdel
    pour i de 1 à n+1 faire
        I[1][0]=0
        D[1][0]=i × Cins
        pour j de 1 a m+1 faire
            dell=D[1][j-1] + Cdel
            ins =D[0][j] + Cins
            sub=D[0][j-1] + Csub(x[i-1],y[j-1])
            D[1][j]=min(dell,ins,sub)
            si (i > n/2) alors
                Si (D[1][j]=dell) alors
                    I[1][j]=I[1][j-1]
                Sinon Si (D[1][j]=ins) alors
                    I[1][j]=I[0][j]
                Sinon
                    I[1][j]=I[0][j-1]
        D[0]=D[1]
        si (i > n/2) alors
            I[0]=I[1]
    retourner I[0][m]
```

Question 26

La complexité spatiale de coupure est en **Θ(m)** cars seule deux tableau de deux dimension de taille m sont stockée.

Question 27

On sait que la complexité de coupure est en **Θ(m)** qui est en **O(n)** car n>m . Donc pour chaque appel récursive de SOL_2 on a deux sous problème de taille n/2

Et pour chaque appel récursive on a une complexité spatiale en O(n)

Donc : $T(n)=2\times T(n/2)+O(n)$, a=2, b=2, d=1 On a $\log_2(2) = 1$ donc d'après le théorème de maitre :

$T(n)=O(n\times\log(n))$

Question 28

La complexité temporelle de coupure est en **Θ(n×m)** car on doit parcourir toute la matrice de taille n×m pour trouver la coupure.

Tache D

Les tests son indiquer sur le fichier testTacheC.py, les résultats son les suivants :

fichier : Instances_genome/Inst_0000010_7.adn
Alignement methode 2 : ('TGGGTG--CTAT', '-GGG-GTTCTAT') time used is 0.000267 s
fichier : Instances_genome/Inst_0000010_8.adn
Alignement methode 2 : ('AACTGTCTTT', 'AACTGT-TTT') time used is 0.000200 s
fichier : Instances_genome/Inst_0000010_44.adn
Alignement methode 2 : ('TATATGAGTC', 'TAT-T---T-') time used is 0.000148 s
fichier : Instances_genome/Inst_0000012_32.adn
Alignement methode 2 : ('CCATTGATTTTC', 'C-ATTGATTT--') time used is 0.000228 s
fichier : Instances_genome/Inst_0000012_56.adn
Alignement methode 2 : ('GCTTAACTAACG', 'GCTAAACTA-CT') time used is 0.000262 s
fichier : Instances_genome/Inst_0000100_7.adn
Alignement methode 2 : ('TGGGTGCGTG…CTCAGT', 'TGGGTTA…-CCTCA-T') time used is 0.010963 s
fichier : Instances_genome/Inst_0000500_88.adn
Alignement methode 2 : ('TGT-CTCGG-…CAATTTCCA-G', '-GTTCT…TTT--AAG') time used is 0.247998 s
fichier : Instances_genome/Inst_0002000_44.adn
Alignement methode 2 : ('TATATGA….-TGTCAGT', '-ATATGA….GTCAGA') time used is 3.917269 s

Consommation mémoire de SOL_2 : "Instances_genome/Inst_0020000_5.adn" 17.5% soit 2,8 GB.

Complexité:

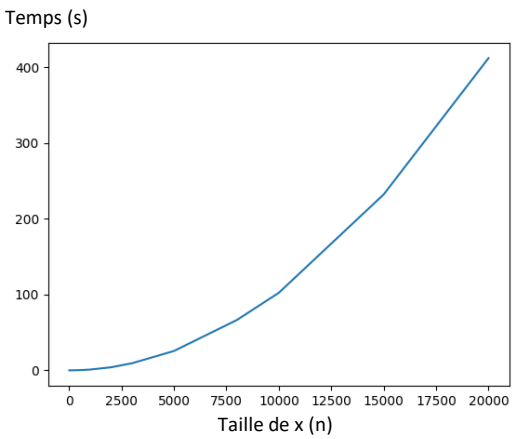

Figure 8 - Performance de SOL_2

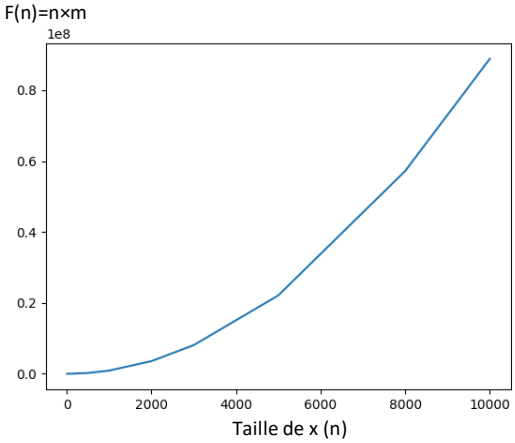

Figure 9 - Courbe théorique de Θ(n×m)

SOL_2 est supérieure a **Θ(n×m)** voir (entre 0 et 10000) mais inferieure a **Θ(n²)** mais largement en **O(n²)**

Question 29

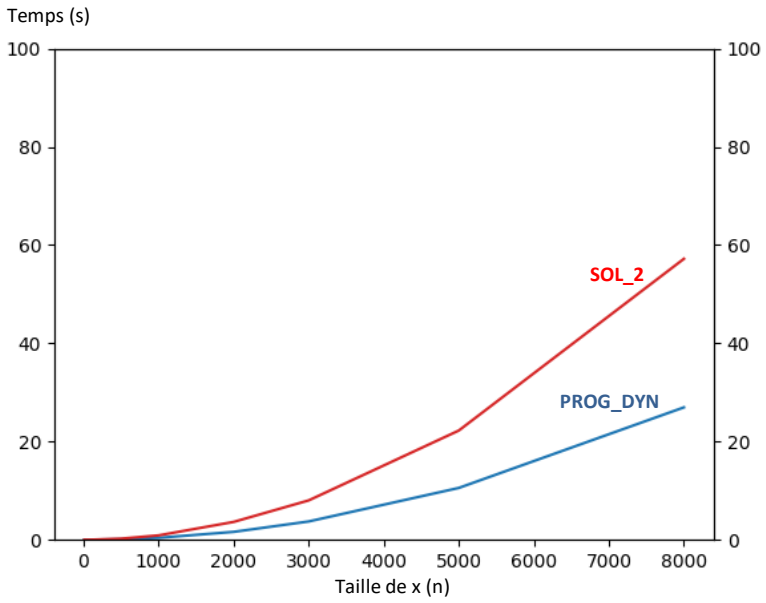

Figure 10 - Comparaison entre PROG_DYN et SOL_2

En comparant la complexité de PROG_DYN et SOL_2 on remarque qu'on a perdu en complexité temporelle en améliorant la complexité spatiale

UNE EXTENSION : L'ALIGNEMENT LOCAL DE SEQUENCES (BONUS)

Question 30

En calculant le cout minimal des exemples suivants (inclus dans le répertoire BONUS) on obtient :

Pour **Inst_1** : n=20 , m=2 , DIST_1 retourne 36 qui est bien $2 \times (20-2) = C_{del} \times (n-m)$

Pour **Inst_2** : n=25 , m=4 , DIST_1 retourne qui est bien $2 \times (25-4) = C_{del} \times (n-m)$

Pour **Inst_3** : n=18 , m=3 , DIST_1 retourne qui est bien $2 \times (18-3) = C_{del} \times (n-m)$

Pour **Inst_4** : n=40 , m=9 , DIST_1 retourne qui est bien $2 \times (40-9) = C_{del} \times (n-m)$

Question 31

C'est une bonne idée pour un alignement petit mais pour un alignement de taille max qui est de n+m on aura une complexité de $(n+m)^2$ car on enlevant les gaps du début et fin on parcours tout de même $(n+m) \times (n+m) \times$ (le cout d'aligner T[i][|y|]et T[|x|][j])

Question 32

COMPARAISON ENTRE LES PERFORMANCE DES ALGORITHMES

Algorithme	Complexité temporelle	Complexité spatiale	Consommation mémoire
DIST_NAIF	$\Theta(\sum_{k=1}^{m}(C^{k}_{n+k} \times C^{k'}_{n}))$	$\Theta(1)$	0.1%
DIST_1	$\Theta(n \times m)$	$\Theta(n \times m)$	69%
DIST_2	$\Theta(n \times m)$	$\Theta(m)$	17.5%
SOL_1	$O(n+m)$	$O(n+m)$	/
PROG_DYN	$\Theta(n \times m)$	$\Theta(n \times m)$	85%
Align_lettre_mot	$\Theta(m)$	$\Theta(m)$	/
Coupure	$\Theta(n \times m)$	$\Theta(m)$	/
SOL_2	$O(n^2)$	$O(n \times \log(n))$	17.5%

Les consommations mémoire sont calculer pour une taille n = 20000

Conclusions :

- DIST_NAIF est la plus mauvaise méthode pour résoudre le problème malgré ca complexité spéciale en $\Theta(1)$ mais le nombre d'appels récursifs est trop important et la capacité de la pile ne suffit pas pour une taille supérieure à 500.
- DIST_2 est mieux que DIST_1 en termes de complexité spatiale avec une augmentation négligeable de complexité temporelle à cause des décalages comme on voit sur la Figure 5.
- SOL_2 est mieux que PROG_DYN en termes de complexité spatiale pour un prix considérable de complexité temporelle.
- La consommation mémoire de SOL_2 est similaire a celle de DIST_2 car coupure utilise le même principe que DIST_2.