5_SELVACHANDRAN _TOURE

December 4, 2019

1 Mini-projet: Alignement de séquences

1.0.1 TOURE Momar Faly

1.0.2 SELVACHANDRAN Nagulan

1.1 2. Le problème d'alignement de séquences

1.1.1 2.2 Alignement de deux mots

2 Question 1

Soit $x = x_1 x_2 \cdots x_n$ $\quad u = u_1 u_2 \cdots u_p$

$y = y_1 y_2 \cdots y_m$ $\quad v = v_1 v_2 \cdots v_q$

- Montrons que $\pi(\bar{x}.\bar{u}) = x.u$ et $\pi(\bar{y}.\bar{v}) = y.v$ (1)

D'après les hypothèses, on a : $\pi(\bar{x}) = x$ et $\pi(\bar{u}) = u$

$\pi(\bar{y}) = y$ et $\pi(\bar{v}) = v$

Montrons que $\forall x, y : \pi(x.y) = \pi(x).\pi(y)$

$$\pi(x.y) = \pi(x_1 x_2 \cdots x_n y_1 y_2 \cdots y_m)$$
$$= x_1 x_2 \cdots x_n y_1 y_2 \cdots y_m$$
$$\pi(x) = \pi(x_1 \cdots x_n) = x_1 \cdots x_n$$
$$\pi(y) = \pi(y_1 \cdots y_m) = y_1 \cdots y_m$$

D'où $\pi(x).\pi(y) = x_1 \cdots x_n y_1 \cdots y_m = \pi(x.y)$

Comme on a : $\forall x, y \quad \pi(x.y) = \pi(x).\pi(y)$,

On a donc $\pi(\bar{x}.\bar{u}) = \pi(\bar{x}).\pi(\bar{u}) = x.u$

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

- Montrons que $|\bar{x}.\bar{u}| = |\bar{y}.\bar{v}|$ ②

  D'après les hypothèses, on a : $|\bar{x}| = |\bar{y}|$ et $|\bar{u}| = |\bar{v}|$

  Montrons que $\forall x, y \quad |x.y| = |x| + |y|$

  $$x.y = x_1 \dots x_n \, y_1 \dots y_m$$
  $$|x.y| = n + m = |x| + |y|$$

  Comme on a $\forall x, y \quad |x.y| = |x| + |y|$,

  On a : $|\bar{x}.\bar{u}| = |\bar{x}| + |\bar{u}| = |\bar{y}| + |\bar{v}| = |\bar{y}.\bar{v}|$

- Montrons que $\forall i \in [1 \dots |\bar{x}.\bar{u}|]$, $\overline{x_i.u_i} \neq -$ ou $\overline{y_i.v_i} \neq -$ ③

  D'après les hypothèses, on a :

  $$\forall i \in [1, \dots |\bar{x}|], \quad \bar{x}_i \neq - \text{ ou } \bar{y}_i \neq -$$
  $$\forall i \in [1, \dots |\bar{v}|], \quad \bar{u}_i \neq - \text{ ou } \bar{v}_i \neq -$$

  Montrons $\forall (x, y) \quad \overline{x.y} = \bar{x}.\bar{y}$

  $$x.y = x_1 \dots x_n \, y_1 \dots y_m$$
  $$\overline{x.y} = x_1 \dots x_n \, y_1 \dots y_m$$
  $$= \bar{x}.\bar{y}$$

  Comme on a : $\forall (x, y) \quad \overline{x.y} = \bar{x}.\bar{y}$, on a :

  $$\bar{x}_i.\bar{u}_i = \overline{x_i.u_i} \quad \text{et} \quad \bar{y}_i.\bar{v}_i = \overline{y_i.v_i}$$

  et comme $\bar{x}_i \neq -$ et $\bar{u}_i \neq -$, alors $\overline{x_i.u_i} \neq -$

  De même, comme $\bar{y}_i \neq -$ et $\bar{v}_i \neq -$, alors $\overline{y_i.v_i} \neq -$

①, ② et ③ $\Rightarrow$ $(\bar{x}.\bar{u}, \bar{y}.\bar{v})$ est un alignement de $(x.u, y.v)$ si $(\bar{x}, \bar{y})$ et $(\bar{u}, \bar{v})$ sont alignements de (x, y) et (u, v)

3 Question 2

Si x est de longueur **n**, et y est de longueur **m**, alors la longueur maximale d'un alignement de (x,y) est **n+m**. **Exemple:** x = ATG et y = AT , $\bar{x}$ = ATG– et $\bar{y}$, ($\bar{x}$,$\bar{y}$) est un alignement de (x,y)

3.1 3 Algorithmes pour l'alignement de séquences

3.1.1 3.1 Méthode naïve par énumération

4 Question 3

Étant donné x un mot de longueur **n**, en ajoutant **k** gaps, on peut obtenir $\binom{n+k}{n}$ mots $\bar{x}$

5 Question 4

Soit (x,y) avec x de longueur **n** et y de longueur **m** en supposant que **m≤n**.
En ajoutant **k** gaps à x, comme la longueur de $\bar{x}$ doit être égale à celle de $\bar{y}$, on a le nombre de gaps de $\bar{y}$ qui doit être égal à **n+k-m**

On aura $\binom{n}{n+k-m}$ façons d'insérer ces gaps dans y.
Sachant qu'il y a $\binom{n+k}{n}$ façons d'avoir $\bar{x}$, et que pour chaque $\bar{x}$, on a $\binom{n}{n+k-m}$ façons d'avoir $\bar{y}$, on aura au total $\binom{n+k}{n}$ x $\binom{n}{n+k-m}$ alignements possibles.
Si le nombre de gaps n'est pas fixé on a au total: $\sum_{k=0}^{m} \binom{n+k}{n} . \binom{n}{n+k-m}$

Pour |x|=15 et |y|=10, le nombre de gaps maximal qu'on peut insérer dans x est 10.
Calculons le nombre d'alignements possibles selon le nombre k de gaps:

Pour k=0 on a: 1 x 3003 = 3003
Pour k=1 on a: 16 x 5005 = 80080
Pour k=2 on a: 136 x 6435 = 875160
Pour k=3 on a: 816 x 6435 = 5250960
Pour k=4 on a: 3876 x 5005 = 19399380
Pour k=5 on a: 15504 x 3003 = 46558512
Pour k=6 on a: 54264 x 1365 = 74070360
Pour k=7 on a: 170544 x 455 = 77597520
Pour k=8 on a: 490314 x 105 = 51482970
Pour k=9 on a: 1307504 x 15 = 19612560
Pour k=10 on a: 3268760 x 1 = 3268760

Au total, on a: **298 199 265 alignements possibles**.

6 Question 5

La complexité serait factorielle O(n!). Pour trouver l'alignement, elle serait doublement exponentielle

7 Question 6

L'algorithme stockerait toutes les valeurs des distances d'éditions pour ensuite les comparer, il y aura donc $\sum_{k=0}^{m} \binom{n+k}{n} . \binom{n}{n+k-m}$.

[1]:
```python
# Variables glolabes:
global x, y, n, m
Cdel=2
Cins=2

import time
```

[2]:
```python
def lireFichier(fichier):
    """
    string-> void
    initialise les valeurs de x,y,n et m
    """
    global x, y, n, m
    x=[]
    y=[]
    f = open("Instances_genome/"+fichier,"r")

    contenu = f.readlines()

    # On lit les longueurs des mots sans le caractère '\n'
    n= int(contenu[0].rstrip('\n'))
    m=int(contenu[1].rstrip('\n'))

    # On stocke lit les mots x et y et on les stocke dans des listes
    for el in contenu[2]:
        if el!=' ' and el!='\n' :
            x.append(el)

    for el in contenu[3]:
        if el!=' ' and el!='\n':
            y.append(el)

    f.close()
```

8 Tâche A

[3]:
```
def Csub(a,b):
    """
    char*char -> int
    renvoie le coût de substitution
    """
    if(a==b):
        return 0
    if((a=='A' and b=='T') or (a=='T' and b=='A') or (a=='G' and b=='C') or␣
 ↪(a=='C' and b=='G')):
        return 3
    return 4


def DIST_NAIF(x,y):
    """
    list[char]*list[char] -> int
    retourne la distance d'édition entre deux mots
    """

    return DIST_NAIF_REC(x,y,0,0,0,float('inf'))


def DIST_NAIF_REC(x,y,i,j,c,dist):
    if(i==len(x) and j==len(y)):
        if(c<dist):
            dist=c

    else:
        if(i<len(x) and j<len(y)):
            dist=DIST_NAIF_REC(x,y,i+1,j+1,c+Csub(x[i],y[j]),dist)


        if (i<len(x)):
                dist=DIST_NAIF_REC(x,y,i+1,j,c+Cdel,dist)

        if(j<len(y)):
                dist=DIST_NAIF_REC(x,y,i,j+1,c+Cins,dist)


    return dist
```

[4]:
```
lireFichier("Inst_0000010_44.adn")
start_time = time.time()
print(DIST_NAIF(x,y)," temps mis:",time.time()-start_time,"secondes")
```

```
lireFichier("Inst_0000010_7.adn")
start_time = time.time()
print(DIST_NAIF(x,y)," temps mis:",time.time()-start_time,"secondes")

lireFichier("Inst_0000010_8.adn")
start_time = time.time()
print(DIST_NAIF(x,y)," temps mis:",time.time()-start_time,"secondes")
```

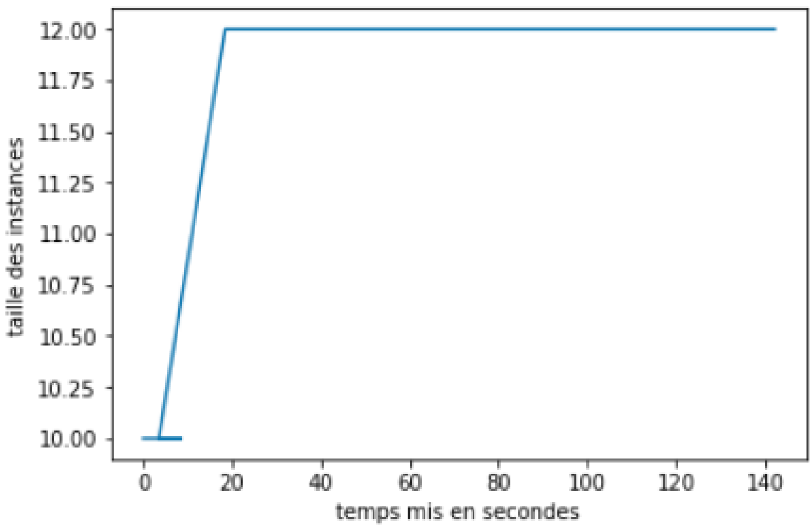
```
(10, ' temps mis:', 0.05928397178649902, 'secondes')
(8, ' temps mis:', 8.212905883789062, 'secondes')
(2, ' temps mis:', 3.2711760997772217, 'secondes')
```

On évalue jusqu'à quelle taille d'instance on peut résoudre les instances fournies en moins d'une minute

[5]:
```
"""

L'exécution pour le tracé de ce graphe prend beaucou trop de temps.
La sortie en image est juste au-dessous:

import matplotlib.pyplot as plt
import numpy as np
import os

end_time=0
X=[]
Y=[]
for root, dirs, files in os.walk("Instances_genome"):
    for filename in sorted(files):
        if(end_time)>60:
            break
        lireFichier(filename)
        start_time = time.time()
        (DIST_NAIF(x,y))
        end_time = time.time()-start_time
        X.append(end_time)
        Y.append(n)


plt.plot (np.array(X),np.array(Y))
plt.ylabel('taille des instances')
plt.xlabel('temps mis en secondes')
plt.show()"""
```

[5]:
```
'"\n\nL\'ex\xc3\xa9cution pour le trac\xc3\xa9 de ce graphe prend beaucou trop
de temps.\nLa sortie en image est juste au-dessous:\n\nimport matplotlib.pyplot
as plt\nimport numpy as np\nimport os\n\nend_time=0\nX=[]\nY=[]\nfor root, dirs,
```

```
files in os.walk("Instances_genome"):\n    for filename in sorted(files):\n
if(end_time)>60:\n            break\n       lireFichier(filename)\n
start_time = time.time()\n        (DIST_NAIF(x,y))\n        end_time =
time.time()-start_time\n        X.append(end_time)\n        Y.append(n)\n
\n\nplt.plot (np.array(X),np.array(Y))\nplt.ylabel(\'taille des
instances\')\nplt.xlabel(\'temps mis en secondes\')\nplt.show()'
```



On peut voir avec la courbe qu'on peut résoudre en moins d'une minute jusqu'à certaines instances de taille 12.

```
12772 momar      20   0 1046960  89628  19968 R 100,0   1,3  57:59.59 python2
```

En exécutant top dans le terminal, on voit qu'elle utilise **89,628 MiB**, soit **1,3%** de la mémoire totale: **6983,8 MiB**

8.0.1 3.2 Programmation dynamique

3.2.1 Calcul de la distance d'édition par programmation dynamique

9 Question 7

Soit ($\bar{u}$,$\bar{v}$) un alignement de (x,y) de longueur **l**. Si $\bar{u}_l$ = - , alors $\bar{v}_l$ est dans {A,T,G,C} Si $\bar{v}_l$ = - , alors $\bar{u}_l$ est dans {A,T,G,C} Si $\bar{u}_l \neq$ - et $\bar{v}_l \neq$ -, alors $\bar{v}_l$ et $\bar{u}_l$ sont dans {A,T,G,C}

10 Question 8

$$C(\bar{u},\bar{v}) = \sum_{k=1}^{\ell} c(\bar{u}_k, \bar{v}_k)$$

$$= \sum_{k=1}^{\ell-1} c(\bar{u}_k, \bar{v}_k) + c(\bar{u}_\ell, \bar{v}_\ell)$$

$$= C(\bar{u}_{[1,\ell-1]}, \bar{v}_{[1,\dots,\ell-1]}) + c(\bar{u}_\ell, \bar{v}_\ell)$$

si $\bar{u}_\ell = -$ alors $C(\bar{u},\bar{v}) = C(\bar{u}_{[1,\ell-1]}, \bar{v}_{[1,\ell-1]}) + C_{ins}$

si $\bar{v}_\ell = -$ alors $C(\bar{u},\bar{v}) = C(\bar{u}_{[1,\dots,\ell-1]}, \bar{v}_{[1,\ell-1]}) + C_{del}$

si $\bar{u}_\ell \neq -$ et $\bar{v}_\ell \neq -$ alors $C(\bar{u},\bar{v}) = C(\bar{u}_{[1,\dots,\ell-1]}, \bar{v}_{[1,\dots,\ell-1]}) + C_{sub}(\bar{u}_\ell, \bar{v}_\ell)$

11 Question 9

D'après 8), on a $D(n,m) = d(x,y) = \min\left(c(\bar{u},\bar{v})\right)$

$$D(n,m) = \min\left(C(\bar{u}_{[1..l-1]}, \bar{v}_{[1..l-1]}) + c(\bar{u}_l, \bar{v}_l)\right)$$

$$= \min C(\bar{u}_{[1..l-1]}, \bar{v}_{[1..l-1]}) + \min(\bar{u}_l, \bar{v}_l)$$

$$D(n,m) = \min \begin{cases} D(n, m-1) + C_{ins} \\ D(n-1, m) + C_{del} \\ D(n-1, m-1) + C_{sub}(\bar{u}_l, \bar{v}_l) \end{cases}$$

On a l'expression générale en remplaçant n par i et m par j

12 Question 10

Si **i=0** et **j=0**, il y a pas d'alignements possibles, donc **D(0,0) n'est pas défini**

13 Question 11

$$D(0,j) = d(x,y) \quad \text{avec} \quad |x| = 0$$

$$= \min \left(c(\bar{x},\bar{y}) \right) \quad \text{avec} \quad |\bar{x}| = |\bar{y}| = j$$

$$= \min \left(\sum_{k=1}^{j} C_{ins} \right) \quad \text{car} \quad \bar{x}_k = -$$

$$= \min \left(j \, C_{ins} \right)$$

Et comme il n'y a qu'un seul aligment pour $(0,j)$,

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

On fait le même raisonnement pour $D(i,0) = i \, C_{del}$

14 Question 12

```
Dist_1:
Entrée : x et y deux mots
sortie : d(x,y)
Pour i=0 jusqu'a len(x) compris :
    Pour j=0 jusqu'a len(y) compris :
        si i=0 et j=0 :
            T[i][j] = 0
        sinon si i=0 :
            T[0][j] = j Cins
        sinon si j=0
            T[i][0] = i Cdel
        sinon :
            T[i][j] = min(T[i][j-1] + Cins,
                          T[i-1][j] + Cdel,
                          T[i-1][j-1] + Csub(x[i-1], y[i-1]))
retourne T[i][j]
```

15 Question 13

L'alogrithme Dist_1 rempli un tableau à deux dimensions de taille **nxm**. Sa complexité spaciale est donc $\theta(nm)$.

16 Question 14

La compltexité temporelle de l'algorithme DIST_1 est $\theta(nm)$ à chaque tour de boucle, il effectue au plus 4 accès à la liste T. Cet accès se fait en temps constant ($\theta(1)$). D'où la complexité $\theta(nm)$ de l'algorithme.

3.2.2 Calcul d'un alignement optimal par programmation dynamique

17 Question 15

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

$$c(\bar{s}, \bar{t}) = D(i, j-1) \quad (1)$$

$$c(\bar{s}.-, \bar{t}.y_j) = c(\bar{s}, \bar{t}) + c(-, y_j)$$
$$= c(\bar{s}, \bar{t}) + CIns$$
$$= D(i, j-1) + CIns \quad \leftarrow (1)$$

Or on sait que $D(i,j) = D(i, j-1) + CIns$

Donc $c(\bar{s}.-, \bar{t}.y_j) = D(i,j)$

D'où $(\bar{s}.-, \bar{t}.y_j) \in Al^*(i,j)$

18 Question 16

```
SOL_1 :

Entrée : x et y deux mots
         Tableau T [0 x ... |x|][0 ... |y|]
Sortie : (newX, newY) alignement minimal de (x,y)
i = len (T) _ 1
j = len (T[0]) - 1
Tant que   T[i] et T[j] différent de 0:
        si T[i][j] = T[i-1][j-1] + Csub(x[i-1], y[i-1])
           alors newX.append(x[i-1])
                 newY.append( Y[i-1])
                 i = i-1
                 j = j-1
        sinon si T[i][j] = T[i-1][j] + Cdel
                 newX.append (x[i-1])
                 newY.append ('_')
                 i = i-1
        sinon si T[i][j] == T[i][j-1] + Cins :
                 newX.append ('_')
                 newY.append( y[j-1])
return (newX, newY)  j = j-1
```

19 Question 17

La complexité de SOL_1 est $\mathcal{O}(n+m)$ puisque c'est dans le pire des cas qu'on parcoure le tableau uniquement sans mouvement diagonale.

20 Question 18

La complexité de SOL_1 est $\mathcal{O}(n+m)$ et celle de DIST_1 est $\mathcal{O}(nm)$.D'où la combinaison est en $\mathcal{O}(nm)$

21 Tâche B

[6]:
```
global T
```

[7]:
```
def DIST_1(x,y):
    """
    List[char]*List[char] - > int
    Retourne la distance d'édition
    """
    # on initialise  le tableau
    global T
    T=[[0 for j in range(len(y)+1)] for i in range (len(x)+1)]

    for i in range(len(x)+1):
        for j in range(len(y)+1):
            # première case du tableau
            if (i==0 and j==0):
                T[i][j]=0
            #première ligne du tableau
            elif i==0:
                T[0][j]=j*Cins

            #première colonne
            elif j==0:
                T[i][0]=i*Cdel
            # Pour toutes les autres cases du tableau
            else:

            #on fait x[i-1] au lieu de x[i] car le tableau T est toujours en␣
 ↪avance de 1
                                ␣
 ↪T[i][j]=min(T[i-1][j-1]+Csub(x[i-1],y[j-1]),T[i-1][j]+Cdel,T[i][j-1]+Cins)

    return T[i][j]
```

[8]:
```
lireFichier("Inst_0000010_44.adn")
start_time = time.time()
print(DIST_1(x,y)," temps mis:",time.time()-start_time,"secondes")


lireFichier("Inst_0000010_7.adn")
start_time = time.time()
print(DIST_1(x,y)," temps mis:",time.time()-start_time,"secondes")

lireFichier("Inst_0000010_8.adn")
start_time = time.time()
print(DIST_1(x,y)," temps mis:",time.time()-start_time,"secondes")
```

```
(10, ' temps mis:', 0.00030612945556640625, 'secondes')
(8, ' temps mis:', 0.0004551410675048828, 'secondes')
(2, ' temps mis:', 0.0002951622009277344, 'secondes')
```

On peut remarquer que là où il nous a fallu **10.63 secondes** pour Inst_0000010_7.adn avec DIST_NAIF, il ne nous faut que **0.0003 secondes** avec DIST_1.

[9]:
```
# bibliothèque pour implémenter des listes doublement chaînées
#On pourra utiliser appendleft qui a une complexité de O(1)
from pyllist import dllist
```

[10]:
```
def SOL_1(x,y,T):
    """
    List[char]*List[char]*List[List[int]] -> (List[char],List[char])
    Retourne un alignement minimal
    """

    # on initialise i et j aux indices du dernier élément du tableau
    i=len(T)-1
    j=len(T[0])-1

    #l'alignement minimal
    newX=dllist()
    newY=dllist()

    while(i!=0 and j!=0):

        #on remonte diagonalement
        if (T[i][j]==T[i-1][j-1]+Csub(x[i-1],y[j-1])):  #on fait x[i-1] au lieu␣
 ↪de x[i] car le tableau T est toujours en avance de 1

            newX.appendleft(x[i-1])
            newY.appendleft(y[j-1])
            i=i-1
            j=j-1

        #on remonte verticalement
        elif (T[i][j]==T[i-1][j]+Cdel):
            newX.appendleft(x[i-1])
            newY.appendleft('-')
            i=i-1

        #on remonte horizontalement
        elif (T[i][j]==T[i][j-1]+Cins):
            newX.appendleft('-')
            newY.appendleft(y[i-1])
            j=j-1
```

```
    return (newX,newY)
```

[11]:
```
global T
lireFichier("Inst_0000010_44.adn")
DIST_1(x,y)
print(SOL_1(x,y,T))


lireFichier("Inst_0000010_7.adn")
DIST_1(x,y)
print(SOL_1(x,y,T))

lireFichier("Inst_0000010_8.adn")
DIST_1(x,y)
print(SOL_1(x,y,T))
```

```
(dllist(['T', 'A', 'T', 'A', 'T', 'G', 'A', 'G', 'T', 'C']), dllist(['T', 'A',
'T', '-', 'T', '-', '-', '-', 'T', '-']))
(dllist(['T', 'G', 'G', 'G', 'T', 'G', 'C', 'T', 'A', 'T']), dllist(['G', 'G',
'G', 'G', 'T', 'T', 'C', 'T', 'A', 'T']))
(dllist(['A', 'A', 'C', 'T', 'G', 'T', 'C', 'T', 'T', 'T']), dllist(['A', 'A',
'C', 'T', 'G', 'T', '-', 'T', 'T', 'T']))
```

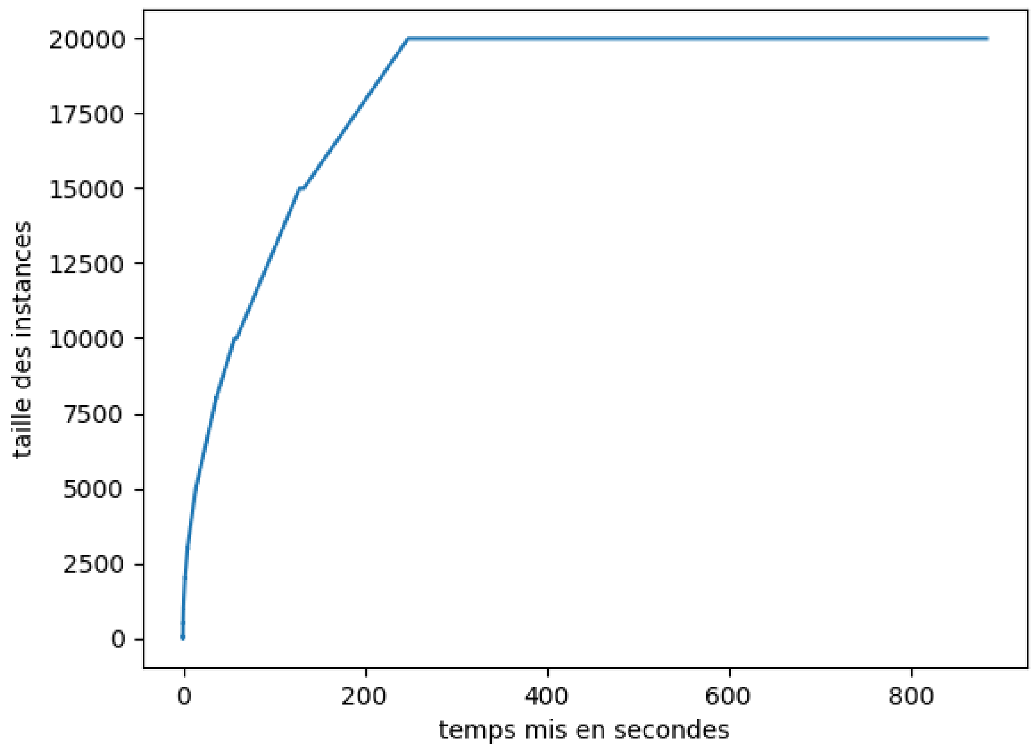
[12]:
```
def PROG_DYN(x,y):
    """
    List[char]*List[char] -> (int,List[char]*List[char])
    retourne d(x,y) et un alignement correspondant
    """
    return (DIST_1(x,y),SOL_1(x,y,T))
```

[ ]:
```
lireFichier("Inst_0000010_44.adn")
start_time = time.time()
print(PROG_DYN(x,y))
print("temps mis:",time.time()-start_time,"secondes")


lireFichier("Inst_0000010_7.adn")
start_time = time.time()
print(PROG_DYN(x,y))
print("temps mis:",time.time()-start_time,"secondes")


lireFichier("Inst_0100000_8.adn")
start_time = time.time()
print(PROG_DYN(x,y))
print("temps mis:",time.time()-start_time,"secondes")
```

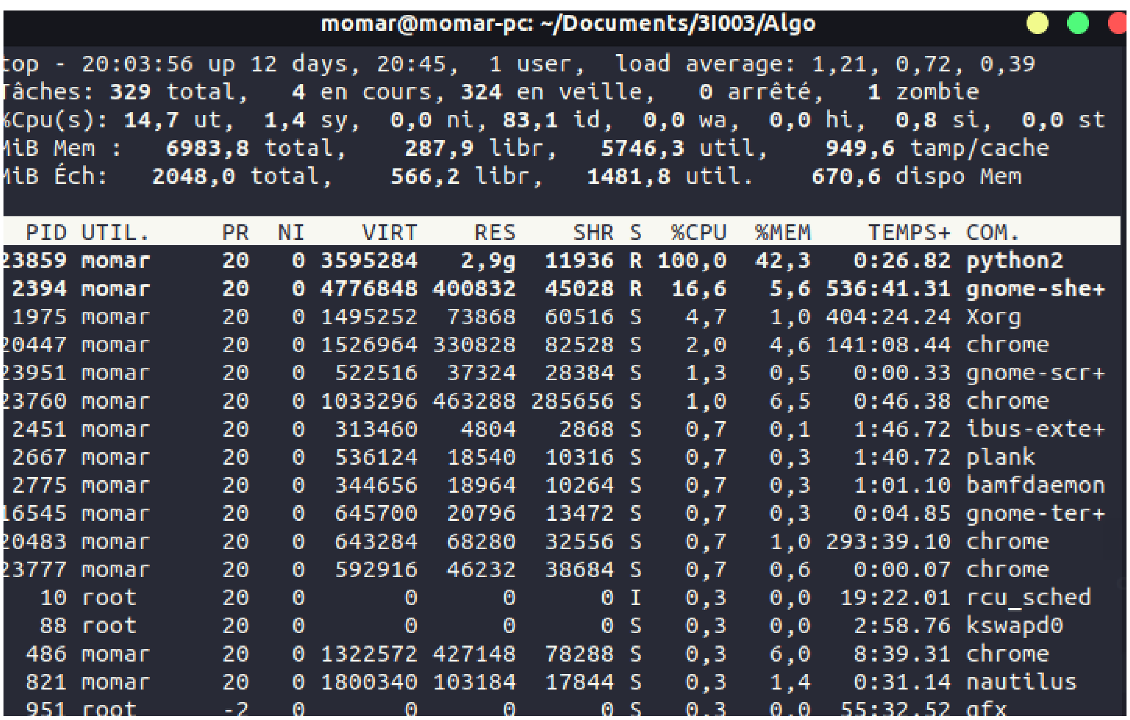
```
(10, (dllist(['T', 'A', 'T', 'A', 'T', 'G', 'A', 'G', 'T', 'C']), dllist(['T',
'A', 'T', '-', 'T', '-', '-', '-', 'T', '-'])))
('temps mis:', 0.00047707557678222656, 'secondes')
(8, (dllist(['T', 'G', 'G', 'G', 'T', 'G', 'C', 'T', 'A', 'T']), dllist(['G',
'G', 'G', 'G', 'T', 'T', 'C', 'T', 'A', 'T'])))
('temps mis:', 0.00032210350036621094, 'secondes')
```

21.0.1 Tracé de la courbe de PROG_DYN(x,y) en fonction du temps

[ ]:
```
"""import matplotlib.pyplot as plt
import numpy as np
import os

end_time=0
X=[]
Y=[]
for root, dirs, files in os.walk("Instances_genome"):
    for filename in sorted(files):
        if(end_time)>60:
            break
        lireFichier(filename)
        start_time = time.time()
        (PROG_DYN(x,y))
        end_time = time.time()-start_time
        X.append(end_time)
        Y.append(n)

print(X)
print(Y)
plt.plot (np.array(X),np.array(Y))
plt.ylabel('taille des instances')
plt.xlabel('temps mis en secondes')
plt.show()"""
```



Les résultats correspondes bien vu que: Pour n =20000 on a m = 17799 et donc n*m= 355980000 microS= 356 secondes



```
momar@momar-pc: ~/Documents/3I003/Algo
top - 20:03:56 up 12 days, 20:45,  1 user,  load average: 1,21, 0,72, 0,39
Tâches: 329 total,   4 en cours, 324 en veille,   0 arrêté,   1 zombie
%Cpu(s): 14,7 ut,  1,4 sy,  0,0 ni, 83,1 id,  0,0 wa,  0,0 hi,  0,8 si,  0,0 st
MiB Mem :   6983,8 total,    287,9 libr,   5746,3 util,    949,6 tamp/cache
MiB Éch:   2048,0 total,    566,2 libr,   1481,8 util.    670,6 dispo Mem

  PID UTIL.      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TEMPS+ COM.
23859 momar      20   0 3595284   2,9g  11936 R 100,0  42,3    0:26.82 python2
 2394 momar      20   0 4776848 400832  45028 R  16,6   5,6  536:41.31 gnome-she+
 1975 momar      20   0 1495252  73868  60516 S   4,7   1,0  404:24.24 Xorg
20447 momar      20   0 1526964 330828  82528 S   2,0   4,6  141:08.44 chrome
23951 momar      20   0  522516  37324  28384 S   1,3   0,5    0:00.33 gnome-scr+
23760 momar      20   0 1033296 463288 285656 S   1,0   6,5    0:46.38 chrome
 2451 momar      20   0  313460   4804   2868 S   0,7   0,1    1:46.72 ibus-exte+
 2667 momar      20   0  536124  18540  10316 S   0,7   0,3    1:40.72 plank
 2775 momar      20   0  344656  18964  10264 S   0,7   0,3    1:01.10 bamfdaemon
16545 momar      20   0  645700  20796  13472 S   0,7   0,3    0:04.85 gnome-ter+
20483 momar      20   0  643284  68280  32556 S   0,7   1,0  293:39.10 chrome
23777 momar      20   0  592916  46232  38684 S   0,7   0,6    0:00.07 chrome
   10 root       20   0       0      0      0 I   0,3   0,0   19:22.01 rcu_sched
   88 root       20   0       0      0      0 S   0,3   0,0    2:58.76 kswapd0
  486 momar      20   0 1322572 427148  78288 S   0,3   6,0    8:39.31 chrome
  821 momar      20   0 1800340 103184  17844 S   0,3   1,4    0:31.14 nautilus
  951 root       -2   0       0      0      0 S   0,3   0,0   55:32.52 gfx
```

22 Question 19

Dans le calcul des distances d'édition dans la question 15, on remarque qu'on a seulement besoin des valeurs de la ligne précédente et des valeurs sur la ligne sur laquelle on se trouve.

23 Question 20

```
DIST_2 :
Entrée : x et y deux mots
Sortie : Distance d'édition entre x et y

Global T
T un tableau de deux ligne pour j allant de 0 à |y|+1 et i allant de 0 à 2
Pour i allant de 0 à |x| + 1
    Pour j allant de 0 à |y| + 1
        Si (i == 0 et j == 0)
            Alors T[i %2][j] ← 0
        Sinon si (i == 0)
            Alors T[0][j] ← j * c_ins
        Sinon si (j == 0)
            Alors T[i%2][0] ← i × c_del
        Sinon
            T[i %2][j] ← min( T[(i-1)%2][j-1] + c_sub(x_{i-1}, y_{j-1})
            , T[(i-1)%2][j] + c_del , T[i%2][j-1] + c_ins )
Retourner T[i%2][j]
```

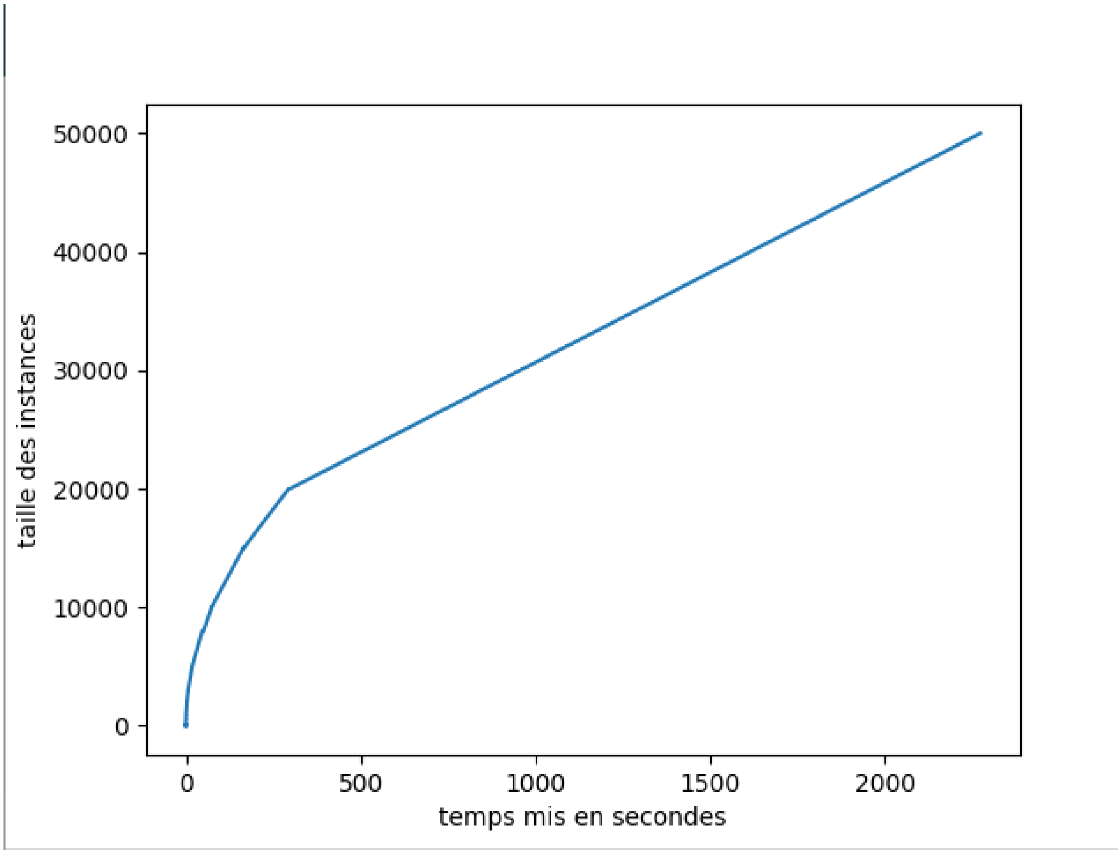
[4]:

```python
def DIST_2(x,y):
    """
```

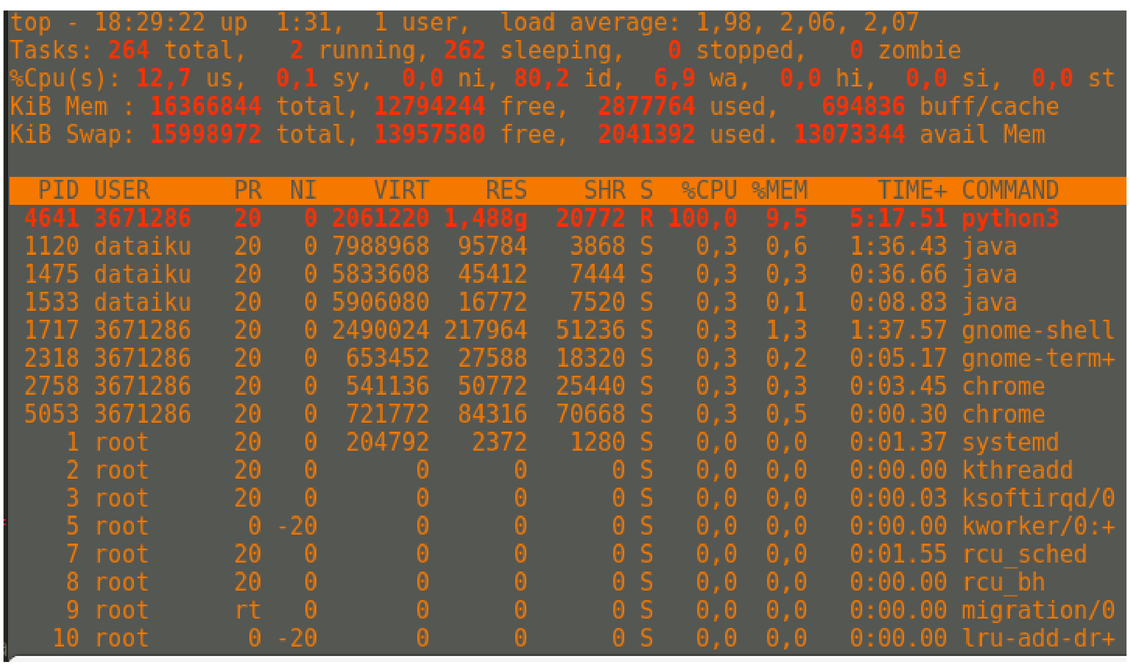
```
    List[char]*List[char] - > int
    Retourne la distance d'édition
    """
    # on initialise  le tableau
    global T
    T=[[0 for j in range(len(y)+1)] for i in range (2)]

    for i in range(len(x)+1):
        for j in range(len(y)+1):
            # première case du tableau
            if (i==0 and j==0):
                T[i%2][j]=0
            #première ligne du tableau
            elif i==0:
                T[0][j]=j*Cins

            #première colonne
            elif j==0:
                T[i%2][0]=i*Cdel
            # Pour toutes les autres cases du tableau
            else:

            #on fait x[i-1] au lieu de x[i] car le tableau T est toujours en␣
↪avance de 1
                ␣
↪T[i%2][j]=min(T[(i-1)%2][j-1]+Csub(x[i-1],y[j-1]),T[(i-1)%2][j]+Cdel,T[i%2][j-1]+Cins)

    return T[i%2][j]
```

[413]:
```
lireFichier("Inst_0000010_44.adn")
start_time = time.time()
print(DIST_2(x,y)," temps mis:",time.time()-start_time,"secondes")


lireFichier("Inst_0000010_7.adn")
start_time = time.time()
print(DIST_2(x,y)," temps mis:",time.time()-start_time,"secondes")

lireFichier("Inst_0000010_8.adn")
start_time = time.time()
print(DIST_2(x,y)," temps mis:",time.time()-start_time,"secondes")
```

```
(10, ' temps mis:', 0.0002579689025878906, 'secondes')
(8, ' temps mis:', 0.00038504600524902344, 'secondes')
(2, ' temps mis:', 0.0002701282501220703, 'secondes')
```



On voit que dist2 est plus performant.



```
top - 18:29:22 up  1:31,  1 user,  load average: 1,98, 2,06, 2,07
Tasks: 264 total,   2 running, 262 sleeping,   0 stopped,   0 zombie
%Cpu(s): 12,7 us,  0,1 sy,  0,0 ni, 80,2 id,  6,9 wa,  0,0 hi,  0,0 si,  0,0 st
KiB Mem : 16366844 total, 12794244 free,  2877764 used,   694836 buff/cache
KiB Swap: 15998972 total, 13957580 free,  2041392 used. 13073344 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 4641 3671286   20   0 2061220 1,488g  20772 R 100,0  9,5   5:17.51 python3
 1120 dataiku   20   0 7988968  95784   3868 S   0,3  0,6   1:36.43 java
 1475 dataiku   20   0 5833608  45412   7444 S   0,3  0,3   0:36.66 java
 1533 dataiku   20   0 5906080  16772   7520 S   0,3  0,1   0:08.83 java
 1717 3671286   20   0 2490024 217964  51236 S   0,3  1,3   1:37.57 gnome-shell
 2318 3671286   20   0  653452  27588  18320 S   0,3  0,2   0:05.17 gnome-term+
 2758 3671286   20   0  541136  50772  25440 S   0,3  0,3   0:03.45 chrome
 5053 3671286   20   0  721772  84316  70668 S   0,3  0,5   0:00.30 chrome
    1 root      20   0  204792   2372   1280 S   0,0  0,0   0:01.37 systemd
    2 root      20   0       0      0      0 S   0,0  0,0   0:00.00 kthreadd
    3 root      20   0       0      0      0 S   0,0  0,0   0:00.03 ksoftirqd/0
    5 root       0 -20       0      0      0 S   0,0  0,0   0:00.00 kworker/0:+
    7 root      20   0       0      0      0 S   0,0  0,0   0:01.55 rcu_sched
    8 root      20   0       0      0      0 S   0,0  0,0   0:00.00 rcu_bh
    9 root      rt   0       0      0      0 S   0,0  0,0   0:00.00 migration/0
   10 root       0 -20       0      0      0 S   0,0  0,0   0:00.00 lru-add-dr+
```

On voit qu'au début du programme, elle utilise déjà 1,48 Gigas

24 Question 21

Mots - gaps
Entrée : k nombre de gaps
Sortie : Liste contenant k gaps

Retourner ['_'] x k

25 Question 22

aligne_lettre_mot :
Entrée : x et y deux mots
Sortie : Le meilleur alignement de (x,y)

Pour i allant de 0 à |y|
Si ($x_0 == y_i$)
Alors on sort de la boucle

Retourner (mots_gaps(i) + X_0 + mots_gaps(len(y) - i - 1), y)

26 Question 23

x1=BAL y1=RO x2=LON y2=ND Cins=Cdel=3 Csub(a,b)=0 si a=b 5 si a et b deux voyelles 7 si a et b deux consonnes (s,t) alignement de (x1,y1): 13 (u,v) alignement de (x2,y2): 8 Soit B A L L O N R O D N Soit 21 Cependant pour un alignement optimale de (x,y) on obtient: Soit B A L L O N R D O N Soit 16 Donc on constate que (s*u,t*v) n'est pas un alignement optimal de (x,y), de ce fait couper le milieu du mot n'est pas la bonne façon pour obtenir un alignement optimale.

27 Question 24

```
SOL_2
 Entrée : x et y deux mots
 Sortie : Alignement minimal de (x,y)

 Si (|y|==0)
   Alors retourner (x, mots-gaps(|x|))
 Sinon si (|x|==0)
   Alors retourner (mots-gaps(|y|), y)
 Sinon si (|x|==1 et |y|==1)
   Alors retourne (liste(x), liste(y))
 Sinon si (|x|==1)
   Alors retourne align-lettre-mot(x,y)
 Sinon
   ALX1 ← mot x de 0 à |x|/2
   ALY1 ← mot y de 0 à coupure(x,y)
   ALX2 ← mot x de |x|/2 à |x|
   ALY2 ← mot y de coupure(x,y) à |y|
   a = SOL_2 (ALX1, ALY1)
   b = SOL_2 (ALX2, ALY2)
   retourner (liste(a[0]) + liste(b[0]), liste(a[1]), liste(b[1]))
```

28 Question 25

Coupure

Entrée : x et y deux mots

Sortie : renvoi la valeur du tableau I à l'indice j qui est une coupure de (x, y)

Global T

T un tableau de deux lignes pour j allant de 0 à |y|+1 et i allant de 0 à 2

I une liste initialisé à 0 de taille |y|+1

```
Pour i allant de |x|/2 à |x|+1
    Pour j allant de 0 à |y|+1
        p ← i
        q ← j
        Tant que p != |x|/2
            DIST_2(x allant de 0 à p-1, y)
            Si (T[p%2][q] == T[(p-1)%2][q] + C_del)
                Alors p ← p - 1
            Sinon si (T[p%2][q] == T[p%2][(q-1)] + C_ins)
                Alors q ← q - 1
            Sinon si (T[p%2][q] == T[(p-1)%2][(q-1)] + C_sub(x[p-1], y[q-1]))
                Alors
                    p ← p - 1
                    q ← q - 1
        I[j] = q
Retourner I[j]
```

[165]:
```
def aligne_lettre_mot(x,y):
    i=0
    for i in range(len(y)):
        if (x[0] == y[i]):
            break;
    return (mots_gaps(i)+[x[0]]+mots_gaps(len(y)-i-1),y)
```

[166]:
```
def mots_gaps(k):
    return ['-']*k
```

[167]:
```
x=['T']
y=['A','T','C']
print(aligne_lettre_mot(x,y))
```

```
(['-', 'T', '-'], ['A', 'T', 'C'])
```

[427]:
```
def SOL_2(x,y):

    # On introduit les listes doublement chaînées pour avoir une complexité de␣
 ↪concaténation = O(1)


    if(len(y)==0):
        return (x,mots_gaps(len(x)))
    elif(len(x)==0):
        return (mots_gaps(len(y)),y)



    elif(len(x)==1 and len(y)==1):

        return (dllist(x),dllist(y))

    elif(len(x)==1):
        return aligne_lettre_mot(x,y)



    else:
        AlX1= x[0:len(x)//2]

        AlX2= x[len(x)//2:]
        AlY1 = y[0:coupure(x,y)]
        AlY2= y[coupure(x,y):]


        a=SOL_2(AlX1,AlY1)
```

```
        b=SOL_2(AlX2,AlY2)
        return (dllist(a[0])+dllist(b[0]),dllist(a[1])+dllist(b[1]))
```

[428]:
```
def coupure(x,y):
    """
    List[char]*List[char]*List[List[int]] -> int)
    Retourne la coupure j*
    """
    # on initialise  le tableau
    global T
    T=[[0 for j in range(len(y)+1)] for i in range (2)]
    I=[0]*(len(y)+1)

    for i in range(len(x)//2,len(x)+1):
        for j in range(len(y)+1):



            # on utilise ces variables temporaires pour pouvoir retrouver les␣
↪valeurs initiales de i et j
            p=i
            q=j

            while( p!=len(x)//2):

                # On calcule la distance D pour les lignes p et p-1
                DIST_2(x[:p],y)

                #on remonte verticalement
                if (T[p%2][q]==T[(p-1)%2][q]+Cdel):

                    p=p-1


                #on remonte horizontalement
                elif (T[p%2][q]==T[p%2][(q-1)]+Cins):
                    q=q-1




                #on remonte diagonalement
                elif (T[p%2][q]==T[(p-1)%2][(q-1)]+Csub(x[p-1],y[q-1])):

                    p=p-1
```

```
                q=q-1



        # on ajoute à la liste I la valeur de j (càd q) trouvé
        I[j] = q
    return I[j]
```

[429]:
```
lireFichier("Inst_0000010_8.adn")
print(SOL_2(x,y))
```

```
(dllist(['A', 'A', 'C', 'T', 'G', 'T', 'C', 'T', 'T', 'T']), dllist(['A', 'A',
'C', 'T', 'G', 'T', '-', 'T', 'T', 'T']))
```

[430]:
```
lireFichier("Inst_0000010_44.adn")
start_time = time.time()
print(SOL_2(x,y)," temps mis:",time.time()-start_time,"secondes")



lireFichier("Inst_0000010_7.adn")
start_time = time.time()
print(SOL_2(x,y)," temps mis:",time.time()-start_time,"secondes")

lireFichier("Inst_0000010_8.adn")
start_time = time.time()
print(SOL_2(x,y)," temps mis:",time.time()-start_time,"secondes")
```

```
((dllist(['T', 'A', 'T', 'A', 'T', 'G', 'A', 'G', 'T', 'C']), dllist(['T', 'A',
'T', '-', 'T', '-', '-', '-', 'T', '-'])), ' temps mis:', 0.010593891143798828,
'secondes')
((dllist(['T', 'G', 'G', 'G', '-', 'T', '-', 'G', 'C', 'T', 'A', 'T']),
dllist(['-', 'G', 'G', 'G', 'G', 'T', 'T', '-', 'C', 'T', 'A', 'T'])), ' temps
mis:', 0.03449702262878418, 'secondes')
((dllist(['A', 'A', 'C', 'T', 'G', 'T', 'C', 'T', 'T', 'T']), dllist(['A', 'A',
'C', 'T', 'G', 'T', '-', 'T', 'T', 'T'])), ' temps mis:', 0.02656698226928711,
'secondes')
```

29 Question 26

La fonction coupure utilise un tableau T à deux lignes et à m (longueur de y) colonnes et un tableau I à une ligne et m colonnes. Donc au total **3 x m** . D'où la complexité de $\theta(m)$

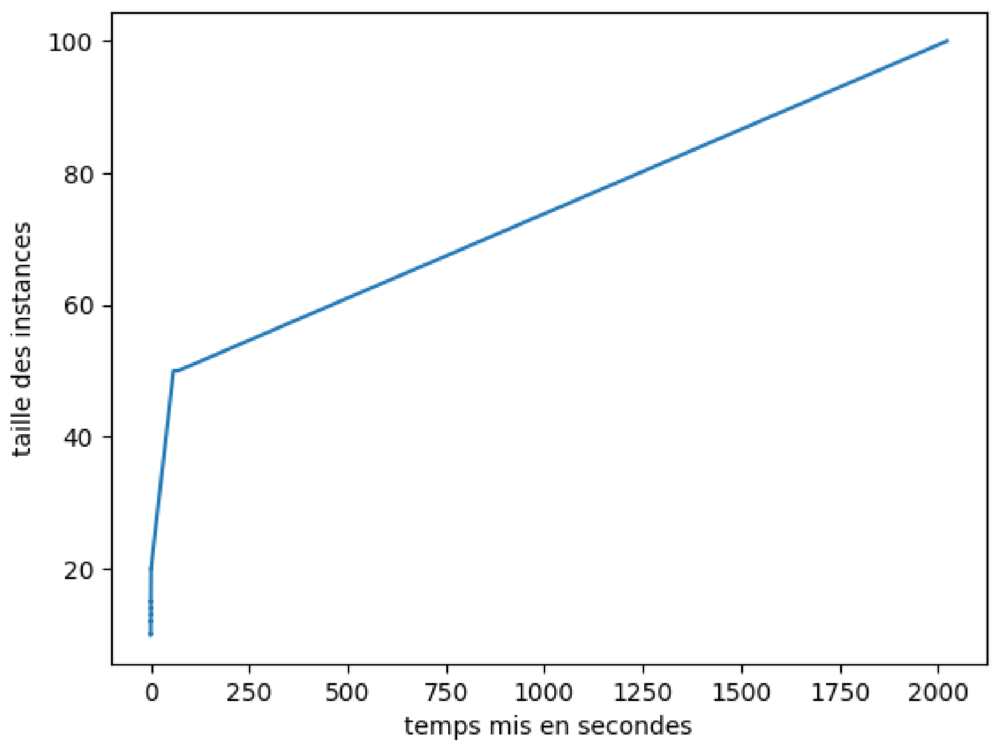
30 Question 27

La fonction SOL_2 démarre avec deux listes de longueur n et m. Et pour chaque tableau, on effectue un appel récursif sur les deux moitiés du tableau, et ainsi de suite. On remarque que pour un tableau de longueur n, on a occuple les espaces mémoires suivants: n + 2 * n/2 + 2 * n/4 +....+ 1= n + 2 x n/2 x $\frac{1-(\frac{1}{2})^{(}log(n))}{\frac{1}{2}}$ = n (3 - $(\frac{1}{2})^{log(n)}$) . Le nombre d'opérations pour les deux mots est: n (3 - $(\frac{1}{2})^{log(n)}$) + m (3 - $(\frac{1}{2})^{log(m)}$)

31 Question 28

Pour la boucle i, elle la fait au total n/2; pour la boucle j, elle la fait m fois et dans la boucle while, elle fait au pire (j+i-len(x)/2) tours de boucles dans lesquelles elle fait appelle à DIST_2 qui a O(nm) en complexité. Donc la compléxité est:$O(m^2n^2)$

Tâche D



---

32 Question 29

On remarque qu'effectivement en améliorant la complexité mémoire, on a perdu en complexité en temps, puisque dans le temps imparti, il ne va que jusqu'à n=100