



MODULE : LU3IN003

Mini-projet : Alignement de séquences

Réalisé par :

AKNOUCHE Anis

HADDADI Hacene

Enseignants :

Olivier Spanjaard

Fanny Pascual

Lionel Tabourier

3ème année Licence Informatique

Mono-disciplinaire

Groupe n°5

2019-2020

## SOMMAIRE :

### 1-Directives :

### 2-TACHE A :

2-1 Code des fonctions

2-2 Teste de validité de l'implémentation

2-3 Evaluer les performances temporelles de DIST\_NAIF

2-4 La consommation mémoire nécessaire au fonctionnement de DIST\_NAIF\_REC

### 3-TACHE B :

3-1 Code des fonctions et testes

3-2 Courbe de consommation de temps CPU pour la fonction PROG\_DYN

3-3 Estimer la quantité mémoire utilisée par PROG\_DYN pour une instance de très grande taille

### 4-TACHE C :

4-1 Code des fonctions et testes

4-2 Courbe de consommation de temps CPU pour la fonction DIST\_2

4-3 Estimer la quantité mémoire utilisée par DIST\_2 pour une instance de très grande taille

### 5-TACHE D :

5-1 Code des fonctions et testes

5-2 Courbe de consommation de temps CPU pour la fonction SOL\_2

5-3 Estimer la consommation mémoire nécessaire au fonctionnement de SOL\_2

### 6-Question 29 de la partie théorique :

## 1-Directives :

Dans toute la suite on définira le coût de la suppression comme étant le macro CDEL de valeur 2, le coût de l'insertion CINS de valeur 2, le coût de la substitution sera calculé à l'aide de la fonction CSUB.

### Les macros :

```
5     #define CDEL 2
6     #define CINS 2
```

### Code de la fonction CSUB :

```
43 int CSUB(char a,char b)
44 {
45     if(a==b)
46     {
47         return 0;
48     }
49     else
50     {
51         if(((a=='A') && (b=='T')) || ((a=='T') && (b=='A')) || ((a=='G') && (b=='C')) || ((a=='C') && (b=='G')))
52         {
53             return 3;
54         }
55         else{
56             return 4;
57         }
58     }
}
```

Remarque : On utilisera des instances de très grande taille, c'est pourquoi on a ajouté une fonction LIRE\_FICHIER, qui accédera directement à l'instance désignée pour récupérer les données nécessaires :

### Code de la fonction LIRE\_FICHIER :

```
161 void LIRE_FICHIER(char* nomFichier,char** x,int* tailleX,char** y,int* tailleY)
162 {
163     FILE* fichier=NULL;
164     fichier=fopen(nomFichier,"r");
165     char* chainel=malloc(10*sizeof(char));
166     char* chaine2=malloc(10*sizeof(char));
167
168     if(fichier!=NULL)
169     {
170         fgets(chainel,9,fichier);
171         *tailleX=atoi(chainel);
172         fgets(chaine2,9,fichier);
173         *tailleY=atoi(chaine2);
174         int i=0;
175         *x=malloc((*tailleX)*sizeof(char));
176         *y=malloc((*tailleY)*sizeof(char));
177         char c;
178
179         while(i<(*tailleX))          /** RECUPERER LE PREMIER MOT X **/
180         {
181             c=fgetc(fichier);
182             if(c!=' ' && c!=EOF)
183             {
184                 (*x)[i]=c;
185                 i++;
186             }
187         }
188     }
189 }
```

```
185         while (fgetc(fichier) != '\n')
186         {
187             i=0;
188
189             while (i<(*tailleY))          /** RECUPERER DEUXIEME MOT Y **/
190             {
191                 c=fgetc(fichier);
192                 if(c!=' ' && c!='EOF')
193                 {
194                     (*y)[i]=c;
195                     i++;
196                 }
197             free(chainel);
198             free(chaine2);
199         }else{
200             printf("Impossible d'ouvrir le fichier\n");
201         }
202     }
```

## 2-TACHE A :

### 2-1 Code des fonctions :

#### Code de la fonction DIST\_NAIF :

```
10 #define INFINI pow(10,100)
11
12 int DIST_NAIF(char* x,int tailleX,char* y,int tailleY)
13 {
14     return DIST_NAIF_REC(x,tailleX,y,tailleY,0,0,0,INFINI);
15 }
```

Remarque : la valeur INFINI=POW(10,100)=  $10^{100}$  est suffisante pour tester les instances fournies.

#### Code de la fonction DIST\_NAIF\_REC :

```
17 int DIST_NAIF_REC(char* x,int t1,char* y,int t2,int i,int j,int c,long int dist)
18 {
19     if((i==t1)&&(j==t2))
20     {
21         if(c<dist){
22             dist=c;
23         }
24     }else{
25         if((i<t1)&&(j<t2))
26         {
27             dist=DIST_NAIF_REC(x,t1,y,t2,i+1,j+1,c+CSUB(x[i],y[j]),dist);
28         }
29         if(i<t1)
30         {
31             dist=DIST_NAIF_REC(x,t1,y,t2,i+1,j,c+CDEL,dist);
32         }
33         if(j<t2)
34         {
35             dist=DIST_NAIF_REC(x,t1,y,t2,i,j+1,c+CINS,dist);
36         }
37     }
38 }
39 return dist;
40 }
41 }
```

### 2-2 Teste de validité de l'implémentation :

On va utiliser ce main :

```
259 /****** test DIST_NAIF***** */
260
261 int tailleX=0;
262 int tailleY=0;
263 char** x=malloc(sizeof(int));
264 char** y=malloc(sizeof(int));
265 *x=NULL; *y=NULL;
266
267 LIRE_FICHIER("Instances_genome/Inst_0000010_7.adn",x,&tailleX,y,&tailleY);
268 printf("** CALCUL DE LA DISTANCE D'EDITION DU COUPLE DE MOTS (X,Y) **\n");
269 afficherCouple(*x,tailleX,*y,tailleY);
270
271 clock_t start,end;
272 double elapsed;
273
274 start=clock();
275
276 int dist=DIST_NAIF(*x,tailleX,*y,tailleY);
277 end=clock(); //** FIN DE L'EXECUTION DE PROG_DYN **
278
279 elapsed = ((double)end - start) / CLOCKS_PER_SEC; //** CALCUL DU TEMPS D'EXECUTION **
280
281 printf("La distance d'édition =%d \n",dist);
282 printf("Temps d'exectution de DIST_NAIF = %2f \n",elapsed);
283
284 //System("ps aux | grep main"); //commande sous linux
285
```

Test 1 : Inst\_0000010\_44.adn

X= "T A T A T G A G T C" de longueur= 10

Y= "T A T T T" de longueur= 5

```
** CALCUL DE LA DISTANCE D'EDITION DU COUPLE DE MOTS (X,Y) **
(X=TATATGAGTC , Y=TATTT)
La distance d'édition =10
Temps d'exécution de DIST_NAIF = 0.000000

Process returned 44 (0x2C)    execution time : 0.048 s
Press any key to continue.
```

Test 2 : Inst\_0000012\_13.adn

X= "C T G G A A A G T G C G " de longueur= 12

Y= "C T G A A C T G G" de longueur=9

```
** CALCUL DE LA DISTANCE D'EDITION DU COUPLE DE MOTS (X,Y) **
(X=CTGGAAAGTGC , Y=CTGAACCTGG)
La distance d'édition =9
Temps d'exécution de DIST_NAIF = 0.207000

Process returned 44 (0x2C)    execution time : 0.252 s
Press any key to continue.
```

Test 3 : Inst\_0000014\_7.adn

X= " T G G G T G C T A T G T G C" de longueur= 14

Y= " T T G G T G T A G T G C" de longueur= 12

```
** CALCUL DE LA DISTANCE D'EDITION DU COUPLE DE MOTS (X,Y) **
(X=TGGGTGCTATGTGC , Y=TTGGTGTAGTGC)
La distance d'édition =8
Temps d'exécution de DIST_NAIF = 18.056000

Process returned 45 (0x2D)    execution time : 18.103 s
Press any key to continue.
```

On remarque que le programme nous renvoi les résultats attendus, donc l'Algorithme est valide pour ces instances.

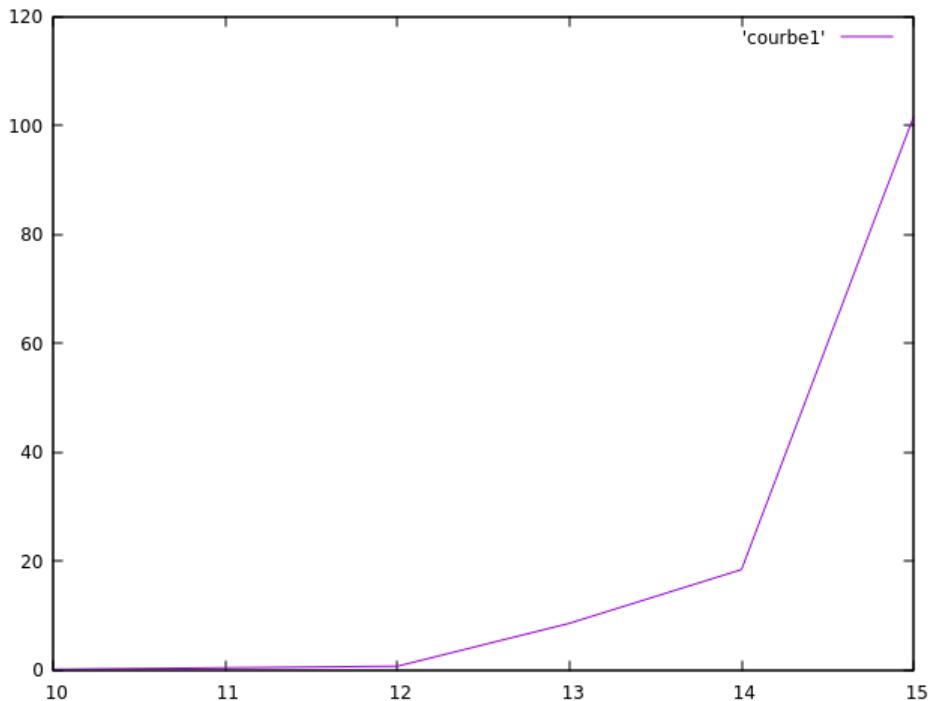
2-3 Evaluer les performances temporelles de DIST\_NAIF :

On va utiliser la fonction CLOCK() de la bibliothèque <time.h> pour calculer le temps d'exécution de la fonction DIST\_NAIF\_REC

Tableau de consommation du temps CPU :

Taille de l'instance	10	12	13	14	15
Temps(s)	0	0.207	8.327	18.056	101.283

Courbe de consommation du temps CPU :



*Graphe de consommation du temps CPU en fonction de la taille de l'instance DIST\_NAIF\_REC*

Remarque : à partir de la taille d'une instance égale à 15 on dépasse largement les 60secondes.

#### 2-4 La consommation mémoire nécessaire au fonctionnement de DIST\_NAIF\_REC :

- Calcul de la taille mémoire des structures de données nécessaires à l'exécution de la fonction DIST\_NAIF\_REC.
- Consommation mémoire du processus exécutant la fonction DIST\_NAIF\_REC sous Linux.

Calcul de la taille mémoire des structures de données nécessaires à l'exécution de la fonction DIST\_NAIF\_REC :

On a deux mots X et Y qui sont des tableaux de caractères

Taille du mot X en Octets = longueur de X \* 1 Octet

Taille du mot Y en Octets = longueur de Y \* 1 Octet

Taille du MACRO INFINI qui est de type Long Int : 4 Octets

Donc une Taille de (Longueur(X) + Longueur(Y) + 4) Octets est nécessaire

Exemples :

Instance 1 : Inst\_0000010\_44.adn

X= “T A T A T G A G T C” de longueur= 10

Y= “T A T T T” de longueur= 5

La taille nécessaire est de (10+5+4) Octets=19 Octets

Instance 2 : Inst\_0000014\_23.adn

X= “C C C C A C A G G G C T A G ” de longueur= 14

Y= “A G A C A G G C A G” de longueur=10

La taille nécessaire est de (14+10+4) Octets=28 Octets

Consommation mémoire du processus exécutant la fonction DIST\_NAIF\_REC sous Linux:

On ajoute au fichier source main.c les instructions suivantes :

Au début du fichier : #include<unistd.h>

A la fin du fichier : system(“ ps aux | grep main”);

La commande : PS AUX (sous linux) affiche l'état des processus en cours d'exécution :

```
root@kali:~/Desktop/ProjetAlgo# ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root     1  0.0  0.3 224740  8820 ?        Ss   13:50  0:03 /sbin/init comp
root     2  0.0  0.0     0     0 ?        S     13:50  0:00 [kthreadd]
root     4  0.0  0.0     0     0 ?        I<   13:50  0:00 [kworker/0:0H]
root     6  0.0  0.0     0     0 ?        I<   13:50  0:00 [mm_percpu_wq]
root     7  0.0  0.0     0     0 ?        S    13:50  0:00 [ksoftirqd/0]
root     8  0.0  0.0     0     0 ?        I    13:50  0:01 [rcu_sched]
```

Le champ VSZ renvoi la taille de la mémoire virtuelle allouée au processus :

```
VSZ          VSZ      virtual memory size of the process in KiB
(char*,int,char*,int,int)
Device mappings are currently
excluded; this is subject to change. (alias
r*x,int tailleX,char* vsize)+tailleY)
```

Le champ RSS renvoi la taille de la mémoire physique non-swappée allouée au processus :

```
RSS          RSS      resident set size, the non-swapped physical
(scpu(s): 8.0 us,
c",|KiB Mem : 2506028 memory that a task has used (in kiloBytes).
|KiB Swap: 0 total, 0 used, 0 used. (alias rssize, rsz).
```

### Testes :

Instance 1 : Inst\_0000010\_44.adn

X= "T A T A T G A G T C" de longueur= 10

Y= "T A T T T" de longueur= 5

```
[root] 2396 0.0 0.0 4280 688 pts/0 S+ 17:44 0:00 ./main
```

VSZ=4280 KiloOctets (taille de la mémoire virtuelle allouée)

RSS=688 KiloOctets (taille de la mémoire physique Non-swapée allouée) (RAM)

Instance 2: Inst\_0000014\_23.adn

X= "C C C C A C A G G G C T A G " de longueur= 14

Y= "A G A C A G G C A G" de longueur=10

```
[root] 2435 102 0.0 4280 732 pts/0 S+ 17:50 0:03 ./main
```

VSZ=4280 kiloOctets (taille de la mémoire virtuelle allouée)

RSS=732 kiloOctets (taille de la mémoire physique Non-swapée allouée) (RAM)

### 3-TACHE B :

#### 3-1 Code des fonctions et Testes :

##### Code de la fonction DIST\_1 :

```
22     int DIST_1(char* x,int tailleX,char* y,int tailleY,int *D[])
23 {
24
25     int i,j;
26     for( i=0;i<tailleX+1;i++)
27     {
28         D[i][0]=i*CDEL;
29     }
30
31
32     for( i=0;i<tailleY+1;i++)
33     {
34         D[0][i]=i*CINS;
35     }
36     for( i=1;i<tailleX+1;i++)
37     {
38         for( j=1;j<tailleY+1;j++)
39         {
40             D[i][j]=Min(D[i-1][j]+CDEL,D[i][j-1]+CINS,D[i-1][j-1]+CSUB(x[i-1],y[j-1]));
41         }
42     }
43     return D[tailleX][tailleY];
44 }
```

##### Code de la fonction Min :

```
13     int Min(int x,int y,int z)
14 {
15     int min=y;
16     if (x<y) min=x;
17     if (z<min) min=z;
18
19 }
```

### Tests :

On va utiliser ce MAIN :

```
199 void main(int argc, char *argv[])
200 {
201     /****** test DIST_1***** */
202
203     int tailleX=0;
204     int tailleY=0;
205     char** x=malloc(sizeof(int));
206     char** y=malloc(sizeof(int));
207     *x=NULL; *y=NULL;
208
209     LIRE_FICHIER("Instances_genome/Inst_0000015_4.adn",x,&tailleX,y,&tailleY);
210     printf("** CALCUL DE LA DISTANCE D'EDITION DU COUPLE DE MOTS (X,Y) **\n");
211     afficherCouple(*x,tailleX,*y,tailleY);
212     int** D=(int**)malloc((tailleX+1)*sizeof(int));
213     int i=0;
214     for(i=0;i<tailleX+1;i++)
215     {
216         D[i]=(int*)malloc((tailleY+1)*sizeof(int));
217     }
218     int dist=DIST_1(*x,tailleX,*y,tailleY,D);
219     printf("La distance d'édition =%d \n",dist);
220     AfficherMatrice(D,tailleX,tailleY);
221 }
```

Instance 1: Inst\_0000010\_44.adn

X= "T A T A T G A G T C" de longueur= 10

Y= "T A T T T" de longueur= 5

```
** CALCUL DE LA DISTANCE D'EDITION DU COUPLE DE MOTS (X,Y) **
(X=TATATGAGTC , Y=TATT)
La distance d'édition =10
Afficher la matrice D :
0 2 4 6 8 10
2 0 2 4 6 8
4 2 0 2 4 6
6 4 2 0 2 4
8 6 4 2 3 5
10 8 6 4 2 3
12 10 8 6 4 5
14 12 10 8 6 7
16 14 12 10 8 9
18 16 14 12 10 8
20 18 16 14 12 10

Process returned 11 (0xB)    execution time : 0.041 s
Press any key to continue.
```

Instance 2: Inst\_0000014\_23.adn

X= "C C C C A C A G G G C T A G " de longueur= 14

Y= "A G A C A G G C A G" de longueur=10

```

** CALCUL DE LA DISTANCE D'EDITION DU COUPLE DE MOTS (X,Y) **
(X=CCCCACAGGGCTAG , Y=AGACAGGCAG)
La distance d'édition =15
Afficher la matrice D :
0 2 4 6 8 10 12 14 16 18 20
2 4 5 7 6 8 10 12 14 16 18
4 6 7 9 7 9 11 13 12 14 16
6 8 9 11 9 11 12 14 13 15 17
8 10 11 13 11 13 14 15 14 16 18
10 8 10 11 13 11 13 15 16 14 16
12 10 11 13 11 13 14 16 15 16 17
14 12 13 11 13 11 13 15 17 15 17
16 14 12 13 14 13 11 13 15 17 15
18 16 14 15 16 15 13 11 13 15 17
20 18 16 17 18 17 15 13 14 16 15
22 20 18 19 17 19 17 15 13 15 17
24 22 20 21 19 20 19 17 15 16 18
26 24 22 20 21 19 21 19 17 15 17
28 26 24 22 23 21 19 21 19 17 15

Process returned 15 (0xF)   execution time : 0.051 s
Press any key to continue.

```

Instance 3: Inst\_0000015\_4.adn

X="T G C C G T T G A T A C C A G" de longeur= 15

Y="A C G G G A T C C C C T" de longeur= 12

```

** CALCUL DE LA DISTANCE D'EDITION DU COUPLE DE MOTS (X,Y) **
(X=TGCCGTTGATACCAAG , Y=ACGGGATCCCT)
La distance d'édition =22
Afficher la matrice D :
0 2 4 6 8 10 12 14 16 18 20 22 24
2 3 5 7 9 11 13 12 14 16 18 20 22
4 5 6 5 7 9 11 13 15 17 19 21 23
6 7 5 7 8 10 12 14 13 15 17 19 21
8 9 7 8 10 11 13 15 14 13 15 17 19
10 11 9 7 8 10 12 14 16 15 16 18 20
12 13 11 9 10 12 13 12 14 16 18 20 18
14 15 13 11 12 14 15 13 15 17 19 21 20
16 17 15 13 11 12 14 15 16 18 20 22 22
18 16 17 15 13 14 12 14 16 18 20 22 24
20 18 19 17 15 16 14 12 14 16 18 20 22
22 20 21 19 17 18 16 14 15 17 19 21 23
24 22 20 21 19 20 18 16 14 15 17 19 21
26 24 22 23 21 22 20 18 16 14 15 17 19
28 26 24 25 23 24 22 20 18 16 17 18 20
30 28 26 24 25 23 24 22 20 18 19 20 22

Process returned 16 (0x10)   execution time : 0.057 s
Press any key to continue.

```

### Code de la fonction SOL\_1 :

```
63 void Sol1(char* x, int tailleX, char* y, int tailleY, char* Xb, char* Yb, int **D, int *taille)
64 {
65     int i=tailleX, j=tailleY;
66     int min;
67     *taille=0;
68
69     while(i>0 && j>0)
70     {
71         min=Min(D[i-1][j], D[i][j-1], D[i-1][j-1]);
72         if(min==D[i-1][j])
73         {
74             (Xb)[*taille]=x[i-1];
75             (Yb)[*taille]='-';
76             i--;
77         }
78         else if(min==D[i][j-1])
79         {
80             (Xb)[*taille]='-';
81             (Yb)[*taille]=y[j-1];
82             j--;
83         }
84         else if(min==D[i-1][j-1])
85         {
86             (Xb)[*taille]=x[i-1];
87             (Yb)[*taille]=y[j-1];
88             i--;
89             j--;
90         }
91         /******Inversion******/
92         inversion(Xb, *taille); free(x);
93         inversion(Yb, *taille); free(y);
94     }
95 }
```

### Code de la fonction INVERSION :

```
48 void inversion (char* X, int size) {
49
50     char* xi=malloc(sizeof(char)*size);
51     int i=0;
52
53     for (i=size-1; i>=0; i--) {
54         xi[size-1-i]=X[i];
55     }
56     for (i=size-1; i>=0; i--) {
57         X[i]=xi[i];
58     }
59     free(xi);
60
61 }
```

### Code de la fonction PROG\_DYN :

```
95 void PROG_DYN(char** x, int tailleX, char** y, int tailleY, char* Xb, char* Yb, int **D)
96 {
97
98     int dist=DIST_1(*x,tailleX,*y,tailleY,D);
99     int taille=0; /* taille est la Taille de Xb et Yb */
100    Xb=malloc(sizeof(char)*(tailleX+tailleY));
101    Yb=malloc(sizeof(char)*(tailleX+tailleY));
102
103    Sol1(*x,tailleX,*y,tailleY,Xb,Yb,D,&taille); /* Xb et Yb sont les mots de l'alignement */
104    printf("La distance d'édition = %d\n",dist);
105    printf("L'alignement optimal :\n");
106    afficherCouple(Xb,taille,Yb,taille);
107
108 }
```

### Tests :

On va utiliser ce MAIN :

```
223 //***** Test PROG_DYN *****/
224 int tailleX=0;
225 int tailleY=0;
226 char** x=malloc(sizeof(int));
227 char** y=malloc(sizeof(int));
228 *x=NULL; *y=NULL;
229
230 LIRE_FICHIER("Instances_genome/Inst_0000014_23.adn",x,&tailleX,y,&tailleY);
231 printf("** CALCUL DE LA DISTANCE D'EDITION DU COUPLE DE MOTS (X , Y) **\n");
232 afficherCouple(*x,tailleX,*y,tailleY);
233 printf("De Taille :( %d, %d)\n",tailleX,tailleY);
234
235 int** D=(int**)malloc((tailleX+1)*sizeof(int*));
236 int i=0;
237 for(i=0;i<tailleX+1;i++)
238 {
239     D[i]=(int*)malloc((tailleY+1)*sizeof(int));
240 }
241
242 char* Xb=NULL;
243 char* Yb=NULL;
244
245 PROG_DYN(x,tailleX,y,tailleY,Xb,Yb,D);
246
247 }
```

Instance 1: Inst\_0000014\_23.adn

X= "C C C C A C A G G G C T A G " de longueur= 14

Y= "A G A C A G G C A G" de longueur=10

```
** CALCUL DE LA DISTANCE D'EDITION DU COUPLE DE MOTS (X , Y) **
(X=CCCCCACAGGGCTAG , Y=AGACAGGCAG)
De Taille :(14, 10)
La distance d'édition = 15
L'alignement optimal :
(X=A--CAGGGCTAG , Y=AGACAGG-C-AG)

Process returned 0 (0x0)    execution time : 0.043 s
Press any key to continue.
```

Instance 2: Inst\_0000015\_76.adn

X="T A G C T T G C A A T G C A G" de longueur= 15

Y="A G C G T G C A A A C A G" de longueur=13

```
** CALCUL DE LA DISTANCE D'EDITION DU COUPLE DE MOTS (X , Y) **
(X=TAGCTTGCAATGCAG , Y=AGCGTGAAACAG)
De Taille :(15, 13)
La distance d'édition = 11
L'alignement optimal :
(X=AGC-TTGCAATGCAG , Y=AGCGT-GCAA-ACAG)

Process returned 0 (0x0)    execution time : 0.043 s
Press any key to continue.
```

### Instance 3: Inst\_0000020\_8.adn

X="A A C T G T C T T T G T A G G C A C T T" de longueur=20

Y="A C T G T C T T T G T A G C A C G T" de longueur=18

```
** CALCUL DE LA DISTANCE D'EDITION DU COUPLE DE MOTS (X , Y) **
(X=AACTGTCTTGAGGCACTT , Y=ACTGCTTTGTAGCACGT)
De Taille :(20, 18)
La distance d'édition = 8
L'alignement optimal :
(X=AACTGTCTTGAGGCAC-TT , Y=A-CTGTCTTGAG-CACGT-)

Process returned 0 (0x0)   execution time : 0.045 s
Press any key to continue.
```

### 3-2 Courbe de consommation de temps CPU en fonction de la taille du mot X pour la fonction PROG\_DYN:

On va optimiser le calcul du temps CPU en enlevant les affichages de la fonction PROG\_DYN (qui sont des entrées sorties qui consomment du temps lors de l'exécution du programme).

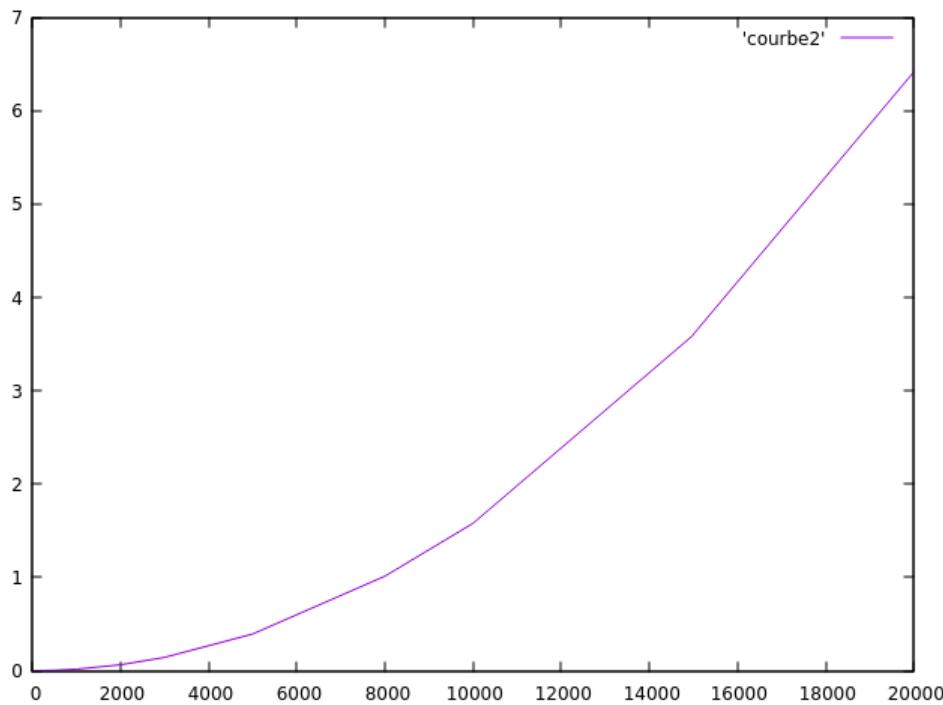
On va aussi utiliser la fonction CLOCK() de la bibliothèque <time.h> pour calculer le temps d'exécution de la fonction PROG\_DYN

```
223 | /****** Test PROG_DYN ***** */
224 |     int tailleX=0;
225 |     int tailleY=0;
226 |     char** x=malloc(sizeof(int));
227 |     char** y=malloc(sizeof(int));
228 |     *x=NULL; *y=NULL;
229 |
230 |     LIRE_FICHIER("Instances_genome/Inst_0000020_8.adn",x,&tailleX,y,&tailleY);
231 |     printf("** CALCUL DE LA DISTANCE D'EDITION DU COUPLE DE MOTS (X , Y) **\n");
232 |     afficherCouple(*x,tailleX,*y,tailleY);
233 |     printf("De Taille :(%d, %d)\n",tailleX,tailleY);
234 |
235 |     int** D=(int**)malloc((tailleX+1)*sizeof(int));
236 |     int i=0;
237 |     for(i=0;i<tailleX+1;i++)
238 |     {
239 |         D[i]=(int*)malloc((tailleY+1)*sizeof(int));
240 |     }
241 |
242 |     char* Xb=NULL;
243 |     char* Yb=NULL;
244 |     clock_t start,end;
245 |     double elapsed;
246 |
247 |     start=clock();           /* DEBUT DE L'EXECUTION DE PROG_DYN */
248 |
249 |     PROG_DYN(x,tailleX,y,tailleY,Xb,Yb,D);
250 |
251 |     end=clock();           /* FIN DE L'EXECUTION DE PROG_DYN */
252 |
253 |     elapsed = ((double)end - start) / CLOCKS_PER_SEC;    /* CALCUL DU TEMPS D'EXECUTION */
254 |     printf ("Temps d'exécution de PROG_DYN = %2f \n",elapsed);
255 }
```

### Tableau de consommation du temps CPU :

Taille des instances	100	130	140	150	200	500	1000	5000	10000	30000	50000	80000	100000	150000	200000
Temps en Secondes	00	00	00	00	00	00	0.004	0.016	0.063	0.141	0.393	1.012	1.579	3.5991	6.411

### Courbe de consommation du temps CPU :



-*Graphe de consommation du temps CPU en fonction de la taille de l'instance PROG\_DYN*

Remarque 1 : Pour les instances de mot de taille inférieure à 500, le temps d'exécution est très faible.

Remarque 2 : Pour les instances de mot de taille 50000 ou supérieure, le programme plante en essayant d'allouer la taille mémoire nécessaire à la création de la matrice D qui est de taille très grande T:

On prend exemple sur une instance de Taille 50000 :

$$T = \text{Longueur X} * \text{Longueur Y} (\text{Octets}) = 50000 * 44523 * (\text{taille INT}) = 2226150000 * 4 \text{ Octets}$$

$$T = 8695898.4375 \text{ kiloOctets}$$

$T = 8492.08 \text{ MégaOctets} = 8,29 \text{ GigaOctets}$  or la configuration du système ne permet pas d'allouer une aussi grande taille en mémoire centrale(Taille de la RAM utilisée est de 8GO), ce qui fait planté le programme.

Conclusion :

On observe que la courbe est de type quadratique ce qui correspond à la complexité temporelle obtenue dans la partie théorique.

### 3-3 Estimer la quantité mémoire utilisée par PROG\_DYN pour une instance de très grande taille :

- Calcul de la taille mémoire des structures de données nécessaires à l'exécution de la fonction PROG\_DYN.
- Consommation mémoire du processus exécutant la fonction PROG\_DYN sous Linux.

On choisit l'instance : Inst\_0015000\_3.adn dont la taille du premier mot est de 15000 caractères.

Calcul de la taille mémoire des structures de données nécessaires à l'exécution de la fonction PROG\_DYN :

- Les deux mots X et Y :

Taille de X=15000\*(Taille d'un Caractère) = 15000 Octets

Taille de Y=13395\*(Taille d'un Caractère )= 13395 Octets

- La matrice D :

Taille de D=Longueur de X \* Longueur de Y=15000\*13395 Octets=200925000 Octets

Taille de D=196215,82 KiloOctets = 191,61 MégaOctets

- Les deux mots de l'alignement obtenu Xb et Yb :

Comme la taille de Xb et Yb n'est pas connu au début, on leur à allouer une taille (taille X+Taille Y)

Taille Xb=Taille X + Taille Y=15000+13395 Octets=28395 Octets

Taille Yb=Taille Xb=28395 Octets

La Taille totale des structures de données :

TailleTotale=15000+13395+200925000+2\*28395 Octets= 201010185 Octets=196299.008 KiloOctets

TailleTotale=191.698 MégaOctets= 0.187 GigaOctets

Consommation mémoire du processus exécutant la fonction PROG\_DYN sous Linux.

On ajoute à la fin du programme l'instruction : system("ps aux || grep main");

Qui est une fonction système de la bibliothèque <unistd.h>

Et la commande PS affiche la consommation mémoire des processus actifs :

```
root@kali:~/Desktop/ProjetAlgo/TACHE_B# ps aux
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
```

### On va exécuter ce MAIN :

```
224 /***** Test PROG_DYN *****/
225     int tailleX=0;
226     int tailleY=0;
227     char** x=malloc(sizeof(int));
228     char** y=malloc(sizeof(int));
229     *x=NULL; *y=NULL;
230
231     LIRE_FICHIER("Instances_genome/Inst_0015000_3.adn",x,&tailleX,y,&tailleY);
232     printf("** CALCUL DE LA DISTANCE D'EDITION DU COUPLE DE MOTS (X , Y) **\n");
233     afficherCouple(*x,tailleX,*y,tailleY);
234     printf("De Taille :(%d, %d)\n",tailleX,tailleY);
235
236     int** D=(int**)malloc((tailleX+1)*sizeof(int*));
237     int i=0;
238     for(i=0;i<tailleX+1;i++)
239     {   D[i]=(int*)malloc((tailleY+1)*sizeof(int)); }
240
241     char* Xb=NULL;
242     char* Yb=NULL;
243     clock_t start,end;
244     double elapsed;
245
246     start=clock();           /* DEBUT DE L'EXECUTION DE PROG_DYN */
247
248     PROG_DYN(x,tailleX,y,tailleY,Xb,Yb,D);
249
250     end=clock();           /* FIN DE L'EXECUTION DE PROG_DYN */
251
252     elapsed = ((double)end - start) / CLOCKS_PER_SEC;    /* CALCUL DU TEMPS D'EXECUTION */
253     printf("Temps d'exécution de PROG_DYN = %2f\n",elapsed);
254     system("ps aux | grep main");
255 }
```

### Le résultat :

```
root@kali:~/Desktop/ProjetAlgo/TACHE_B# ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root      4929  0.2  0.9 295056 24960 ?        Sl   01:28   0:00 leafpad /root/Desktop/ProjetAlgo/TACHE_B/main.c
root      4956 116 31.3 789588 786732 pts/2    S+   01:32   0:04 ./main
root      4957  0.0  0.0  4396   828 pts/2    S+   01:32   0:00 sh -c ps aux | grep main
root      4959  0.0  0.0 12840  1024 pts/2    R+   01:32   0:00 grep main
root@kali:~/Desktop/ProjetAlgo/TACHE_B#
```

VSZ=789588 kiloOctets (taille de la mémoire virtuelle allouée)

VSZ=0,753 GigaOctets

RSS=786732 kiloOctets (taille de la mémoire physique Non-swapée allouée)(RAM)

RSS=0,7502 GigaOctets

## 4-TACHE C :

### 4-1 Code des fonctions et testes :

#### Code de la fonction DIST\_2 :

```

48 int DIST_2 (char* x,int tailleX,char *y, int tailleY)
49 {
50     int D[2][tailleY+1];
51     int cpt=1,j;
52     /****** INITIALISATION *****/
53     for(j=0;j<=tailleY;j++)
54     {
55         D[0][j]=j*CINS;
56     }
57     D[1][0]=CDEL;
58     /****** */
59     while(cpt<=tailleX)
60     {
61         for(j=1;j<=tailleY;j++)
62             D[1][j]=Min(D[0][j-1]+CSUB(x[cpt-1],y[j-1]),D[0][j]+CDEL,D[1][j-1]+CINS);
63     }
64     /******Copie de la ligne 1 dans la ligne 0*****/
65     cpt++;
66     for(j=0;j<=tailleY;j++)
67         D[0][j]=D[1][j];
68     }
69     D[1][0]=D[0][0]+CDEL;
70     /****** */
71 }
72 return D[0][tailleY];
73 }

```

#### Tests :

On va utiliser ce main :

```

285 //***** Test DIST_2 *****/
286 int tailleX=0;
287 int tailleY=0;
288 char** x=malloc(sizeof(int));
289 char** y=malloc(sizeof(int));
290 *x=NULL; *y=NULL;
291
292 clock_t start,end;
293 double elapsed;
294
295 LIRE_FICHIER("Instances_genome/Inst_0000015_4.adn",x,&tailleX,y,&tailleY);
296 printf("** CALCUL DE LA DISTANCE D'EDITION DU COUPLE DE MOTS (X,Y) **\n");
297 afficherCouple(*x,tailleX,*y,tailleY);
298
299 start=clock();
300
301 int dist=DIST_2(*x,tailleX,*y,tailleY);
302
303 end=clock();           /** FIN DE L'EXECUTION DE DIST_2 **/
304
305 elapsed = ((double)end - start) / CLOCKS_PER_SEC;    /** CALCUL DU TEMPS D'EXECUTION **/
306
307 printf("La distance d'édition =%d \n",dist);
308 printf("Temps d'exectution de DYST_2 = %2f \n",elapsed);
309

```

#### Instance 1 : Inst\_0000050\_3.adn

```

** CALCUL DE LA DISTANCE D'EDITION DU COUPLE DE MOTS (X,Y) **
(X=CTAATAAACATGCTAGGGCCCTTACCCACCTGCCACAAGAGTACC , Y=CTAATAAACATGCTAGGGCCTCTCACCCGCCTCAGAGTAC)
La distance d'édition =26
Temps d'exectution de DYST_2 = 0.000000

Process returned 41 (0x29)  execution time : 0.058 s
Press any key to continue.

```

#### Instance 2 : Inst\_0000100\_3.adn

```

** CALCUL DE LA DISTANCE D'EDITION DU COUPLE DE MOTS (X,Y) **
(X=CTAATAAAATACATGCTAGTGGCCCTGTACCCACCTGCCACAAGAGTACCGTATATCAGTGGTTAGCCGGCGGATCCTACCAGTTCACTTGATCCATTCTGTAACCGGGA
TAGTGGCCCTGTACCCACCTACACTAGATCCGTATATCAGAGTTTCAGCCGCGGATCATACACTTGCTTGAT
La distance d'édition =52
Temps d'exécution de DYST_2 = 0.000000

Process returned 41 (0x29) execution time : 0.061 s
Press any key to continue.

```

### Instance 3 : Inst\_0000500\_3.adn

```

** CALCUL DE LA DISTANCE D'EDITION DU COUPLE DE MOTS (X,Y) **
(X=CTAATAAAATACATGCTAGTGGCCCTGTACCCACCTGCCACAAGAGTACCGTATATCAGTGGTTAGCCGGCGGATCCTACCAGTTCACTTGATCCATTCTGTAACCGGGA
TGGCATAAGCCCCGAAGCCGTGCGTACGGTGTGCTAAGCAGCACAGCAGAGGGTGGAAAGGGTGGCTGATGGCGACGATCTGGACGGATGGAAAGCCGAAGACCTAAGTG
CTCTTAAAATGATAGCCTCTACGGGCCACATACCGTGTGCTGAGATACCTTGATCGTAGGATTAGATGGCATCGCGATATGACACTTCACCCCTAATCAGTGTAGTGGCTGTT
TCCCGTATACGACTCGAGCTACCTTGACACACTGTGGCTGAGTGGTTATCGAACACCCGAACCGTACGCTGTCAGGAACGGCGTGTGATATCCAACCGTGCCAGTGGCCCTAT
GGGCTGCCAGCGTTATGCCGTA , Y=CTAATAAAATAATTAGTGGCCCTGTACCCCGGCCAAAGTTCCGGATATCATGGTTACGGCGCGGATGCAACCGATCCAGTGGCTTAACGA
TAGCCTTACGCCAACATACACCGTGTAGTAACTGGATCGAAGGTATCTGCATCGCGACTGACAATGCAACCCGATTTGTAGTCTGTTCCCGTAGACGCCCTGAGCTA
TCATTGACACACTGTGGCGTAGTGGTTGACCAACCCGAACCTTAGCAGTCCAGGAACGGGTTGTAACCACCGGCCCTGGCCCTTGCTGCCAGCGTTGCGGTA
La distance d'édition =26
Temps d'exécution de DYST_2 = 0.005000

Process returned 41 (0x29) execution time : 0.106 s
Press any key to continue.

```

### Instance 4 : Inst\_0001000\_2.adn

```

** CALCUL DE LA DISTANCE D'EDITION DU COUPLE DE MOTS (X,Y) **
(X=CGAGTCATTGCGTTTCTGCCATGGCCAGTCCCTAATTTCAGTACCGATGCACTGACTGAGCTTAACACTGCTTCTTATTAGCGATGTGTAATTTCACCCGATTTCCGGCT
TGCTACCTCACATTAAGCGCCTGAGTCATACCTATCGTAGCTGCACTCCGGCGCAGAATAGGTGAGCTGAAACATAGTGTATAGCAGCTGATTAGCATCTTGTAAATTACTG
ATGCCAGGTAGTCGCTAGCTGCACTGGCGTTGGCCACTTTGACTGGATGGAATCTCTTGTAGGACTCGCTGATTCAACATTGTCGTGACGGCGTGTGTTAGCTTGTGG
AGGGTCAACCGCACGGAAAGTACATAGCGCAATAGCGCTTGCACACTCGACTAATGCTTGAGTGCACATTCTAATCCACACCAGTGAAGCTTGTGAGTCCGGATAGCCCG
TCCGAAATAGCGAGACCGTGCAGGGTGAAGTCTTCTACACTATCGGTACTGGCTGCTCAAGTCTCCGGATGTGCTACCAAGGACTGACTACATTAGTGAACCTTGTAAATACC
GGAAGGATACTGAAGCCTAGGTGAGGTAGGCCCTCGTTATTAGCTGCTTAAAGCCTAAGCCCTAGGCTTGTGACCCGGTGGGGCTGCCAACGGCGATTATGGGGCCACTG
TGTGCGTAGTGTATTAGCTGCTCATTAAGTGAAGCGACGTTAGCTCGCATTAATTATAATTAGAATGCGAACTCATCTGGTAACATCCCGGGGGAGTGGTTATTGCTCATCGT
ATGCACTGCAAGCGAACCCGCTAAATGATTGGCTATAAGAATCATCTGGCCATCTCAAATATTGTTAGATTACGACTCATGCGCAGCAAGATAACTATTCCATGAAAC
ACTGTGGATGAAGTAGGCATGTGTTTTGTGTAAGTCTC , Y=CGCGTATTGCGTTTCTGAATGCGCAGTCCCTATTGAACTAGGTGCTGATCCTTATCTCGTCTCTT
AGCGATGTGTAATTGACCGATTACGCCCTGCTTAATTAGCGCTGAGGGCAGACCTATCTTAGATGCACTGGCCCGAAATGGGAGCTGTAACATATCTTCGAGTGTGTTACCAA
CATTTGTAATTACCATCGCAGGTCTCGCTACTCGACGGGCTCGGCCCTTTCTGACTTGATGGATCTTAACTCGCTTGAATATCATGGTCGTGACGGGGATGGAGCTTCT
GGCAGGGTCAACCGCAAGGAACGTAACTCAATAGCGCTTGAATCCCTTATGGCTTGTGACTGAGCTTCAAGGCTCGCTGACCCAGACTGACTACATTGCGTGCACCTGTT
AGAACGCTGCGGTTGAAGTCTTACTCGCGTTTACTGTGCTCAAGTCTCGGCTGACCCAGACTGACTACATTGCGTGCACCTGTTGAATACCGGAAGGTACTGAAGCCTAG
GTGAGGTAGGCCCTCGTTATTAGTGTGTTACTGCTTAAGCGCTTAGTTGACCCGGTGGGGCTGCCACCGGATTATGGGCCCTGTGCTGTTGAGCGGTCAAAGTGA
AAGCGAGTCGGCCGATATTTATAAGAAGCCAATCTGTAATCGTGGTAGTGTGTTAATCGCAGTGTGCACTGGGTTAAACACTGCGTGGTCAAAGAATCTGCTT
GCATCTCAATCTGAGGATTACGGCTGATGAGCAGCAGTTAATTTGTGACACACTGGGATGCACTGGGATGCACTGGGATGCACTGGGATGCACTGGGATGCACTGGGATGCA
La distance d'édition =491
Temps d'exécution de DYST_2 = 0.015000

Process returned 41 (0x29) execution time : 0.171 s
Press any key to continue.

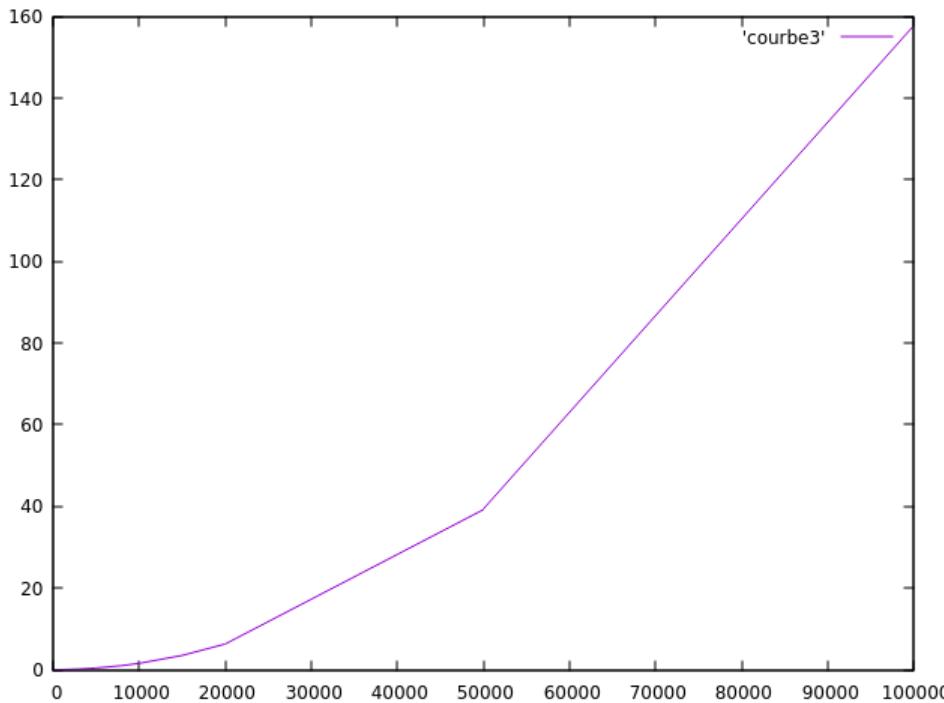
```

### 4-2 Courbe de consommation de temps CPU pour la fonction DIST\_2 :

#### Tableau de consommation du temps CPU :

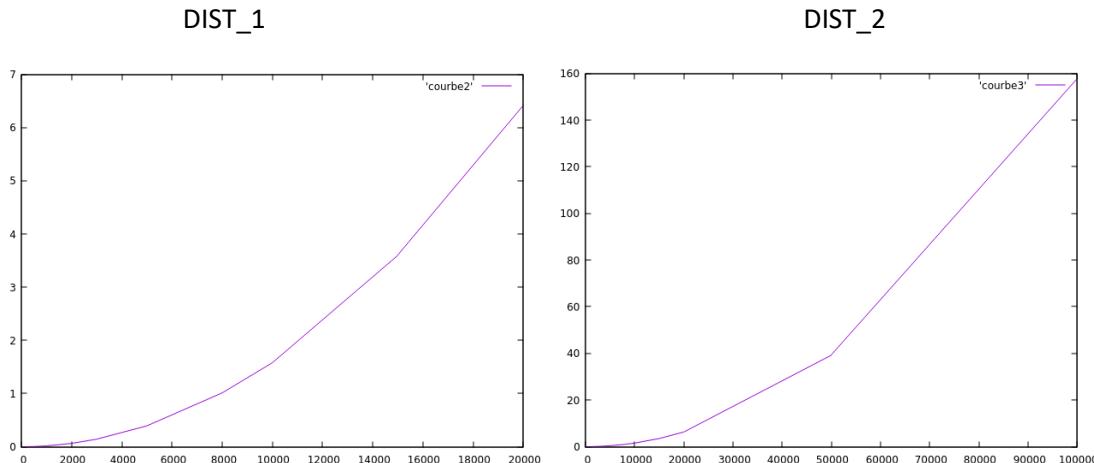
Taille de l'instance	100	500	1000	2000	3000	5000	8000	10000	15000	20000	50000	100000
Temps en seconde	0.05	0.005	0.015	0.072	0.143	0.387	0.992	1.5452	3.482	6.276	39.228	157.662

#### Courbe de consommation du temps CPU :



-Graphe de consommation du temps CPU en fonction de la taille de l'instance pour DIST\_2

Comparaison avec le résultat obtenu avec la fonction DIST\_1 :



La complexité temporelle est la même, les deux graphes de consommation du temps CPU (en fonction de la taille de l'instance) sont presque identique.

4-3 Estimer la quantité mémoire utilisée par DIST\_2 pour une instance de très grande taille :

- Calcul de la taille mémoire des structures de données nécessaires à l'exécution de la fonction DIST\_2.
- Consommation mémoire du processus exécutant la fonction DIST\_2 sous Linux.

On choisit l'instance : Inst\_0020000\_5.adn dont la taille du premier mot est de 20000 caractères.

Calcul de la taille mémoire des structures de données nécessaires à l'exécution de la fonction DIST\_2:

Taille du mot X= Longueur de X\*(Taille d'un Char) = $20000 * 4$  Octets=80000 Octets=78.125 KiloOctets

Taille du mot Y= Longueur de Y\*(Taille d'un Char) = $17779 * 4$  Octets=71116 Octets=69.449 KiloOctets

Taille de la matrice D= Nombre de Lignes \* Nombre de Colonnes =  $2 * \text{Longueur de X}$

Taille de la matrice D= $2 * 20000 * 4$  Octets=160000 Octets= 156.25 KiloOctets

Taille totale des structure de données :

Taille Totale= 80000 + 71116 + 160000 Octets= 303.824 KiloOctets

Consommation mémoire du processus exécutant la fonction DIST\_2 sous Linux :

Avec la commande system("ps aux | grep main");

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	5703	0.2	0.9	295052	24872	?	Sl	15:37	0:02	leafpad /root/Desktop/ProjetAlgo/Programme/main.c
root	5861	90.3	0.0	4300	732	pts/1	S+	15:55	0:08	./main
root	5862	0.0	0.0	4396	804	pts/1	S+	15:55	0:00	sh -c ps aux
										grep main   grep main" ) //commande sous linux
root	5864	0.0	0.0	12840	976	pts/1	S+	15:55	0:00	grep main

VSZ=4300 kiloOctets (taille de la mémoire virtuelle allouée)

RSS=732 kiloOctets (taille de la mémoire physique Non-swapée allouée) (RAM)

## 5-TACHE D :

### 5-1 Code des fonctions et testes :

#### Code de la fonction SOL\_2 :

```

325  void SOL_2 (char* x, int tailleX, char* y, int tailleY, char** Xb, char** Yb) {
326
327      if (tailleY==0) {
328          concatener(Xb, x);
329          (*Yb)=mot_gaps(tailleX);
330      }
331      else if (tailleX==1) {
332          concatener(Xb, x);
333          concatener(Yb, y);
334          aligne_lettre_mot(Xb, Yb);
335      }
336      else{
337          int j=coupure(x,y,tailleX/2);
338
339          char *x1=malloc(sizeof(char)*(tailleX/2+2));
340
341          char** X1b=malloc(sizeof(int));
342          (*X1b)=malloc(sizeof(char)*(tailleX+tailleY));
343          (*X1b)[0]='\0';
344
345          char *y1=malloc(sizeof(char)*(j+1));
346
347          char **Y1b=malloc(sizeof(int));
348          (*Y1b)=malloc(sizeof(char)*(tailleX+tailleY));
349          (*Y1b)[0]='\0';
350
351          char *x2=malloc(sizeof(char)*(tailleX/2+2));
352
353          char **X2b=malloc(sizeof(int));
354          (*X2b)=malloc(sizeof(char)*(tailleX+tailleY));
355          (*X2b)[0]='\0';
356
357          char *y2=malloc(sizeof(char)*(tailleY-j+1));
358
359          char **Y2b=malloc(sizeof(int));
360          (*Y2b)=malloc(sizeof(char)*(tailleX+tailleY));
361          (*Y2b)[0]='\0';
362          diviser(x,&x1,&x2,(tailleX/2));
363          diviser(y,&y1,&y2,j);
364
365          SOL_2(x1,strlen(x1),y1,strlen(y1),X1b,Y1b);
366          SOL_2(x2,strlen(x2),y2,strlen(y2),X2b,Y2b);
367
368          (*Xb)=concat_2(*X1b,*X2b);
369          (*Yb)=concat_2(*Y1b,*Y2b);
370
371          free(x1);
372          free(y1);
373          free(x2);
374          free(y2);
375          free(*X1b);
376          free(*Y1b);
377          free(*X2b);
378          free(*Y2b);
379      }
380  }
381
382 }
```

On aura besoin d'autre fonctions :

#### Code de la fonction : mot\_gaps

```

217  char* mot_gaps (int k) {
218      char *x=malloc(sizeof(char)*(k+1));
219      int j;
220      for(j=0;j<k;j++)
221      {
222          x[j]='-';
223      }
224      x[k]='\0';
225      return x;
226  }
```

### Code de la fonction : contientLettre

```
228 int contientLettre (char *y,char* x){  
229     /** retourne sa position si la lettre de x est dans y,0 sinon**/  
230     int i;  
231     for (i=0;i<strlen(y);i++){  
232         if (y[i]==x[0]) return i;  
233     }  
234     return -1;  
235 }
```

### Code de la fonction : lettreMemeFamille

```
239 int lettreMemefamille (char* y,char* x) {  
240     int i;  
241     if (x[0]=='T'){  
242         for (i=0;i<strlen(y);i++){  
243             if (y[i]=='A') return i;  
244         }  
245     }  
246     else if (x[0]=='A'){  
247         for (i=0;i<strlen(y);i++){  
248             if (y[i]=='T') return i;  
249         }  
250     }  
251     else if (x[0]=='C'){  
252         for (i=0;i<strlen(y);i++){  
253             if (y[i]=='G') return i;  
254         }  
255     }  
256     else if (x[0]=='G'){  
257         for (i=0;i<strlen(y);i++){  
258             if (y[i]=='C') return i;  
259         }  
260     }  
261     return -1;  
262 }
```

### Code de la fonction : Coupure

```
264 int coupure (char* x,char *y,int i){  
265     int tailleX=strlen(x);  
266     int tailleY=strlen(y);  
267     int D[2][tailleY+1];  
268     int I[2][tailleY+1];  
269     int cpt=1,j;  
270     /***** INITIALISATION *****/  
271     for(j=0;j<=tailleY;j++)  
272     {  
273         D[0][j]=j*CINS;  
274         I[0][j]=j;  
275     }  
276     D[1][0]=CDEL;  
277     I[1][0]=I[0][0];  
278     /*****  
279     while(cpt<=tailleX)  
280     {  
281         for(j=1;j<=tailleY;j++)  
282             D[1][j]=Min(D[0][j-1]+CSUB(x[cpt-1],y[j-1]),D[0][j]+CDEL,D[1][j-1]+CINS);  
283             if (cpt>i){  
284                 if (D[1][j]==D[0][j-1]+CSUB(x[cpt-1],y[j-1])) {  
285                     I[1][j]=I[0][j-1];  
286                 }  
287                 else if(D[1][j]==D[1][j-1]+CINS){  
288                     I[1][j]=I[1][j-1];  
289                 }  
290                 else if(D[1][j]==D[0][j]+CDEL){  
291                     I[1][j]=I[0][j];  
292                 }  
293             }  
294     }
```

```

295     /******Copie de la ligne 1 dans la ligne 0*****/
296     for(j=0;j<=tailleY;j++) {
297         D[0][j]=D[1][j];
298     }
299     if (cpt>i) {
300         for(j=0;j<=tailleY;j++) {
301             I[0][j]=I[1][j];
302         }
303         cpt++;
304         I[1][0]=I[0][0];
305         D[1][0]=D[0][0]+CDEL;
306         /******/
307     }
308 }
309 return I[1][tailleY];
310 }
```

#### Code de la fonction : diviser

```

312 void diviser(char *x,char ** x1,char ** x2,int pos) {
313     int j;
314     for (j=0;j<pos;j++) {
315         (*x1)[j]=x[j];
316     }
317     for (j=pos;j<strlen(x);j++) {
318         (*x2)[j-pos]=x[j];
319     }
320     (*x1)[pos]='\0';
321     (*x2)[strlen(x)-pos]='\0';
322 }
323 }
```

#### Code de la fonction : aligne\_lettre\_mot

```

384 void aligne_lettre_mot (char** x,char** y) {
385     char * xi=malloc(sizeof(char)*(strlen((*y)+1)));
386     int j;
387     for (j=0;j<strlen(*y);j++) {
388         xi[j]='\0';
389     }
390     int m=strlen(*y);
391     int i=contientLettre(*y,*x);
392     if (i!=-1){
393         concatener(&xi,mot_gaps(i));
394         xi[i]=(*x)[0];
395         concatener(&xi,mot_gaps(m-i-1));
396         xi[m]='\0';
397         (*x)=xi;
398     }
399     else{
400         i=lettreMemefamille(*y,*x);
401         if (i!=-1){
402             concatener(&xi,mot_gaps(i));
403             xi[i]=(*x)[0];
404             concatener(&xi,mot_gaps(m-i-1));
405             xi[m]='\0';
406             (*x)=xi;
407         }
408         else{
409             xi[0]=(*x)[0];
410             concatener(&xi,mot_gaps(m-1));
411             xi[m]='\0';
412             (*x)=xi;
413         }
414     }
415 }
416 }
```

### Code de la fonction : concatener

```
418 void concatener (char **x,char *y) {  
419     int m=strlen(*x),j;  
420     for (j=m;j<m+strlen(y);j++) {  
421         (*x)[j]=y[j-m];  
422     }  
423     (*x)[m+strlen(y)]='\\0';  
424 }
```

### Code de la fonction : concat\_2

```
427 char* concat_2 (char *x,char *y) {  
428     char* z=malloc(sizeof(char)*(strlen(x)+strlen(y)+1));  
429     int m=strlen(x),j;  
430     for (j=0;j<m;j++) {  
431         z[j]=x[j];  
432     }  
433     for (j=m;j<m+strlen(y);j++) {  
434         z[j]=y[j-m];  
435     }  
436     z[m+strlen(y)]='\\0';  
437     return z;  
438 }
```

### Tests :

On va utiliser ce main :

```
618 /***** Test SOL_2 *****/  
619 int tailleX=0;  
620 int tailleY=0;  
621 char** x=malloc(sizeof(int));  
622 char** y=malloc(sizeof(int));  
623 *x=NULL; *y=NULL;  
624 int i;  
625  
626 clock_t start,end;  
627 double elapsed;  
628  
629 LIRE_FICHIER("Instances_genome/Inst_0050000_6.adn",x,&tailleX,y,&tailleY);  
630  
631 char * xb=malloc(sizeof(char)*(strlen(*y)+strlen(*x)));  
632 char * yb=malloc(sizeof(char)*(strlen(*y)+strlen(*x)));  
633  
634 for (i=0;i<strlen(*x);i++) {  
635     xb[i]='\\0';  
636 }  
637 for (i=0;i<strlen(*y);i++) {  
638     yb[i]='\\0';  
639 }  
640  
641 printf("** ALIGNEMENT OPTIMAL DU COUPLE DE MOTS (X,Y) **\\n");  
642 afficherCouple(*x,tailleX,*y,tailleY);  
643 (*x)[tailleX]='\\0';  
644 (*y)[tailleY]='\\0';  
645 printf("LA COUPURE EST DE : %d\\n",coupure(*x,*y,strlen(*x)/2));  
646  
647 start=clock();  
648  
649 SOL_2(*x,strlen(*x),*y,strlen(*y),&xb,&yb);  
650  
651 end=clock();  
652 //** FIN DE L'EXECUTION DE DIST_2 **  
653 elapsed = ((double)end - start) / CLOCKS_PER_SEC; //** CALCUL DU TEMPS D'EXECUTION **  
654  
655 printf("L'ALIGNEMENT OPTIMAL EST : (%s , %s )\\n",xb,yb);  
656 printf("Temps d'exécution de SOL_2 = %2f \\n",elapsed);  
657  
658 //System("pa aux || grep main"); //commande sous linux  
659
```

### Instance 1 : Inst\_0000050\_3.adn

```
** ALIGNEMENT OPTIMAL DU COUPLE DE MOTS (X,Y) **
(X=CTAATAAATACATGCTAGTGGCCCTGTTACCCACCTGCCACAAGAGTACC , Y=CTAATAAATCATGCTAGGGCCTTCAACCGCCTCAGAGTAC)
LA COUPURE EST DE : 22
L'ALIGNEMENT OPTIMAL EST : ( CTAATAAATACATGCTAGTGGCCCTGTTACCCACCTGCCACAAGAGTACC , CTAATAAAT-CATGCTAG-GG-CCTCTT---CACCGC
CTC-AGAGTA-C )
Temps d'exectution de SOL_2 = 0.000000

Process returned 40 (0x28)    execution time : 0.054 s
Press any key to continue.
```

### Instance 2 : Inst\_0000100\_3.adn

```
** ALIGNEMENT OPTIMAL DU COUPLE DE MOTS (X,Y) **
(X=CTAATAAATACATGCTAGTGGCCCTGTTACCCACCTGCCACAAGAGTACCGTATATATCAGTGGTTAGCCGGCGGGATCCTACCAGTTCACTTGAT , Y=TATAAATAAATG
TAGTGGCCCTGTTACCCACCTACACTAGATCCGTATATATCAGAGTTAGCCGGCGGGATCATACTTGCTTGAT)
LA COUPURE EST DE : 45
L'ALIGNEMENT OPTIMAL EST : ( CTAATAAATACATGCTAGTGGCCCTGTTACCCACCTGCCACAAGAGTACCGTATATATCAGTGGTTAGCCGGCGGGATCCTACCAGT
TCCACTTGAT , -T-ATAAATAAATG-TAGTGGCCCTGTTACCCACCT-ACTAGA-T-CCGGTATATATCAGA-GTTTCAAGGC-GCGCGGATCATA-CACTT--GCTTGAT )
Temps d'exectution de SOL_2 = 0.000000

Process returned 40 (0x28)    execution time : 0.063 s
Press any key to continue.
```

### Instance 3 : Inst\_0000500\_8.adn

```
** ALIGNEMENT OPTIMAL DU COUPLE DE MOTS (X,Y) **
(X=AACGTCTTTAGGCACCTTCAGCTGGTAAGTGTCAAGTGCATATCGCAGCCTTCTTGTCCCTGCAACGGCAGGCCCAAGTTGCCATACCGACGCTCATCAGACT
CCTTGGCCCTACCTAAAACAGACGCTAGTTGGATGGGTTACCGCGCGGTGCTCCCTGGCGCCATTGGCATTCAATGGCGGTATGTTGAGCGTCACTGATATAATTAAAGGCCACGGA
ACAGAGACACAGCTAGTAACATTGGGCGCGCTGCCGACCATACTAGCACCTGCTGTGATTGAAATAAGCTGCCAGAGAATTGATGTCTGCGTCTCGTTCACTATTACGGA
GAGTGACCTATGCCCTGGGGTAGCCAATGAAAATGAGTCACTCACATTGTAATACGACAGCTGCCGCGTCTATAAGCAGCTCTCGCCCTATAGGATTCTCGCTTTGTGCCAA
CTCAAAATAAAGATAGCGCAAGC , Y=AACGTCTTAGGAACTTACGCTGGAAAGTGTGCAATTGCGATATCGCAGCCTGTTGCCCCGAGAGCCTTGTGCGTACTGATCTATTAAAGGCACTG
AGGCGCCTCCTCCCTGGCCGACCTAAACAGAGTATTGGCTGGTTACTCGCGTCTACCCCTGTGCGCGAGTGGGTTCTAGCGCGTATTTGCTACTGATCTATTAAAGGCACTG
AACAGAGAGAGCTAGGAACATGGGCGCGGCCACCATACGCGCAACTGCACTGCAATTGAAATAAGCTAGCCAGAAATTCACTATGCGTCTCGGTGCCATTACGACCTTGAACCT
TGCCCTCGGTTAGCCATTGAGAGTCACTACATTGTTAGCAGCTAGCGCGTCTTAAGAGACGTCGCGCATAGGATTCTCGCTGGCACTAAAATAAGATACCAAGC
LA COUPURE EST DE : 224
L'ALIGNEMENT OPTIMAL EST : ( AACGTCTTTAGGCACCTTCAGCTGGTAAGTGTCAAGTGCATATCGCAGCCTTCTTGTCCCTGCAACGGCAGGCCCAAGT
TTGCCATACCGCCTCAGACTCCTGGCCCTACCTAAAACAGACGCTAGTTGGATGGGTTACCGCGCGGTGCTCCCTGGCGCCATTGGCATTCAATGGCGGTATGTTGAG
CGTCACTGATATAATTAAAGGCCAGGAACAGAGACACAGCTA-GTAACATTGGGCGCGCTGCCGACCATACTAGCACCTGCTGTGATTGAAATAAGCTGCCAGAGAATTGCG
TCTTGTGCTCGGTTCACTATTACGGAGAGTACCTATGCCCTGGGGTAGCCAATGAAAATGAGTCACTCACATTGTAATACGACAGCT-GCCCGCGTCTATAAGGACGCTTGTG
CCCTATAGGATTCTGCTTGTGCCACTCAAATAAGATAGCGCAAGC , AACGT-CTTAGTGGAACTTA-TCAAGCTGG-AAGTGTG-CAATTGCGATATCGCAGCC-TTGTG
GTCCCTGC-A--G-AAGCCCCAGTTGG-CATACCGCGC-C-TC---CTCCTGGCCGACCTAAACAGA-GTA-TTT-GGCTGTTACTCGGC-GTTCACCCGTGCGAG
TGG-GTTTAT-GCGCGTATGTT-TGC-TCACTGATCT-ATTAAGG-CACTGAACAGAG--AGAGCTAGC-AACA-TGGGGCGCG-CGCC-ACCATACGCGCAACTGCA
GCTGATCTTCAAAGAGACGCGCTTGTGCGTCTCGGTTAGCCATTG-TAG-GAGTCACT-ACATTGTTATA-G-CAGCTAG-CGCGTCTCTAAAGAGACGCGCTTGTGCGT
ATAAGCTA-GCCGAGA-AATT-CA-GT-ATGCGTCTCGGTGC-CTATTACGGACCTTGAACCTGCTGCTTGTGCGTCTCGGTTAGCCATTG-TAG-GAGTCACT-ACATTGTTATA-G-CAGCTAG
-CGCGTCTCTAAAGAGACG---TCGTGCC-ATAGGGATTCTCGC---TGTGCC-ACT-AAAAT-AAGATA-C-CAAGC )
Temps d'exectution de SOL_2 = 0.011000

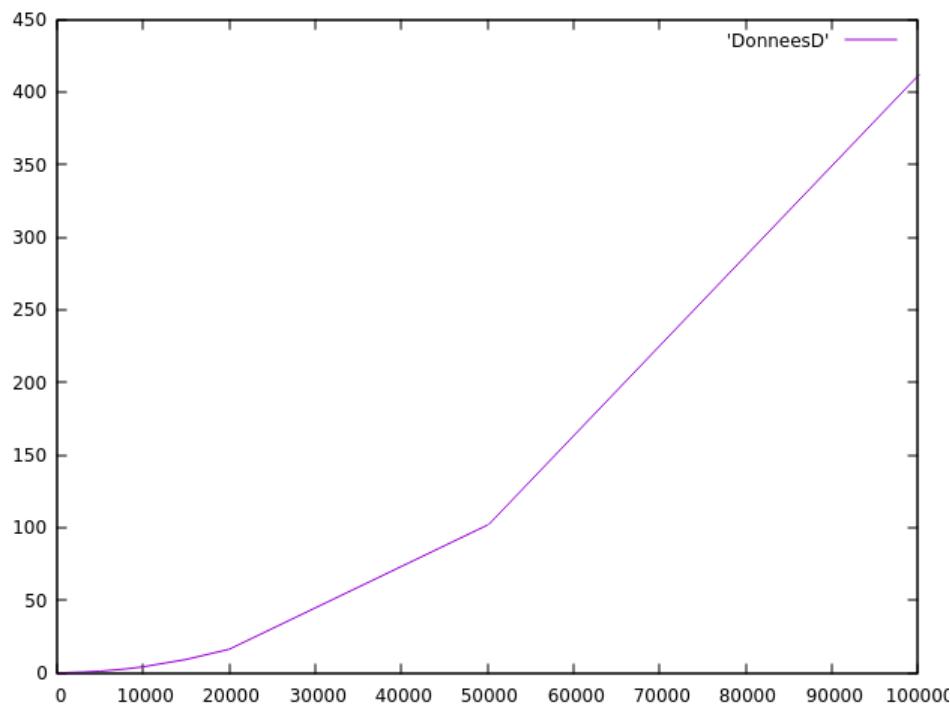
Process returned 40 (0x28)    execution time : 0.091 s
Press any key to continue.
```

## 5-2 Courbe de consommation de temps CPU pour la fonction SOL\_2 :

Tableau de consommation du temps CPU :

Taille instance	100	500	1000	2000	3000	5000	8000	10000	1500	2000	5000	10000
Temps de second	0.001	0.011	0.045	0.171	0.391	1.021	2.633	4.072	9.108	16.108	101.678	411.708

### Courbe de consommation du temps CPU :



-Graphe de consommation en du temps CPU en fonction de la taille des instances pour SOL\_2.

### 5-3 Estimer la consommation mémoire nécessaire au fonctionnement de SOL\_2 :

- Calcul de la taille mémoire des structures de données nécessaires à l'exécution de la fonction SOL\_2.
- Consommation mémoire du processus exécutant la fonction SOL\_2 sous Linux.

On choisit l'instance : Inst\_0020000\_5.adn dont la taille du premier mot est de 20000 caractères.

#### Calcul de la taille mémoire des structures de données nécessaires à l'exécution de la fonction SOL\_2:

- Les deux mots X et Y :

Taille du mot X= Longueur de X\*(Taille d'un Char) = $20000 * 4$  Octets=80000 Octets=78.125 KiloOctets

Taille du mot Y= Longueur de Y\*(Taille d'un Char) = $17779 * 4$  Octets=71116 Octets=69.449 KiloOctets

- Les deux mots de l'alignement obtenu Xb et Yb :

Comme la taille de Xb et Yb n'est pas connu au début, on leur à allouer une taille (taille X+Taille Y)

Taille Xb=Taille X + Taille Y= $80000 + 71116$  Octets= 151116 Octets

Taille Yb=Taille Xb=151116 Octets

#### La Taille totale des structures de données :

TailleTotale= $80000 + 71116 + 151116 + 151116$  Octets=453348 Octets= 442KiloOctets

### Consommation mémoire du processus exécutant la fonction SOL\_2 sous Linux :

```
root      6632  0.3  0.9 295052 24808 ?        Sl  ('16:07)  0:00 leafpad /root/Desktop/ProjetAlgo/Programme/main.c
root      6675  100  0.3 11608  8452 pts/1   S+   16:12  0:30 ./main
root    66775  0.0  0.0  4396  840 pts/1   S+   16:13  0:00 sh -c ps aux | grep main
root      6679  0.0  0.0 12840 1032 pts/1   S+   16:13  0:00 grep main
root@kali:~/Desktop/ProjetAlgo/Programme#
```

VSZ=11608 kiloOctets (taille de la mémoire virtuelle allouée)

RSS=8452 kiloOctets (taille de la mémoire physique Non-swapée allouée) (RAM)

### Remarque :

Sous linux on nous affiche un taille mémoire très grande comparé à celle calculé, c'est dû au fait des allocation mémoire intermédiaire qui n'ont peut-être pas été libérer correctement.

### 6-Question 29 :

Oui, on a perdu en complexité temporelle mais on a gagné en complexité spatiale.

- Pour la complexité Temporelle :

#### Avec SOL\_1 :

Taille des instances	10	13	14	15	20	50	100	500	1000	2000	3000	5000	8000	10000	15000	20000
Temps en Secondes	00	00	00	00	00	00	0.04	0.016	0.063	0.141	0.393	1.012	1.579	3.599	6.411	

Tableau de la consommation du temps CPU de SOL\_1 en fonction de la taille des instances.

#### Avec SOL\_2 :

Taille instance	1000	5000	10000	20000	30000	50000	80000	100000	150000	200000	500000	1000000			
Temps de seconde	0.001	0.011	0.045	0.171	0.391	1.021	2.633	4.072	9.108	16.108	101.678	411.708			

Tableau de la consommation du temps CPU de SOL\_2 en fonction de la taille des instances.

-On remarque que SOL\_1 est plus rapide que SOL\_2, donc cela confirme qu'on a bien perdu en complexité temporelle.

- Pour la complexité Spatiale :

Avec SOL\_1 : (Cette partie est bien expliquée dans la partie TACHE C )

La taille totale des structures de données nécessaires au fonctionnement de SOL\_1 = 0.187 GigaOctets

VSZ=789588 kiloOctets = 0,753 GigaOctets (taille de la mémoire virtuelle allouée)

RSS=786732 kiloOctets = 0,7502 GigaOctets (taille de la mémoire physique Non-swapée allouée)(RAM)

Avec SOL\_2 : (Cette partie est bien expliquée dans la partie TACHE D )

La taille totale des structures de données nécessaires au fonctionnement de SOL\_2 = 442KiloOctets

VSZ=11608 kiloOctets (taille de la mémoire virtuelle allouée)

RSS=8452 kiloOctets (taille de la mémoire physique Non-swapée allouée) (RAM)

-On remarque que SOL\_1 consomme plus d'espace mémoire que SOL\_2, donc cela confirme qu'on a bien gagné en complexité spatiale.

### Question 1:

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

On a:  $(\bar{n} \cdot \bar{u}, \bar{y} \cdot \bar{v})$  est un alignement de  $(n \cdot u, y \cdot v)$  ssi:

$$1) \Pi(\bar{n} \cdot \bar{u}) = n \cdot u$$

$$2) \Pi(\bar{y} \cdot \bar{v}) = y \cdot v$$

$$3) |\bar{n} \cdot \bar{u}| = |\bar{y} \cdot \bar{v}|$$

$$4) \forall i \in [1,..|\bar{n} \cdot \bar{u}|], (\bar{x}_i, \bar{u}_i)_i \neq - \text{ ou } (\bar{y}_i, \bar{v}_i)_i \neq -$$

1) Montrer que:  $\Pi(\bar{n} \cdot \bar{u}) = n \cdot u$

$$\underline{\text{On a:}} \quad \Pi(\bar{n} \cdot \bar{u}) = \Pi(\bar{n}) \cdot \Pi(\bar{u})$$

on a aussi:  $(\bar{n}, \bar{y})$  est un alignement de  $(n, y)$  Donc  $\Pi(\bar{n}) = n$   
 $(\bar{u}, \bar{v})$  est un alignement de  $(u, v)$  Donc  $\Pi(\bar{u}) = u$

$$\underline{\text{Donc:}} \quad \Pi(\bar{n} \cdot \bar{u}) = \Pi(\bar{n}) \cdot \Pi(\bar{u}) = n \cdot u$$

$$\underline{\text{d'où:}} \quad \boxed{\Pi(\bar{n} \cdot \bar{u}) = n \cdot u}$$

2) Montrer que:  $\Pi(\bar{y} \cdot \bar{v}) = y \cdot v$

$$\underline{\text{On a:}} \quad \Pi(\bar{y} \cdot \bar{v}) = \Pi(\bar{y}) \cdot \Pi(\bar{v})$$

on a aussi:  $(\bar{n}, \bar{y})$  est un alignement de  $(n, y)$  Donc  $\Pi(\bar{y}) = y$   
 $(\bar{u}, \bar{v})$  est un alignement de  $(u, v)$  Donc  $\Pi(\bar{v}) = v$

$$\underline{\text{Donc:}} \quad \Pi(\bar{y} \cdot \bar{v}) = \Pi(\bar{y}) \cdot \Pi(\bar{v}) = y \cdot v$$

$$\underline{\text{d'où:}} \quad \boxed{\Pi(\bar{y} \cdot \bar{v}) = y \cdot v}$$

3) Montrer que:  $|\bar{n} \cdot \bar{u}| = |\bar{y} \cdot \bar{v}|$

on a:  $|\bar{n} \cdot \bar{u}| = |\bar{n}| + |\bar{u}|$   
et  $|\bar{y} \cdot \bar{v}| = |\bar{y}| + |\bar{v}|$

on a ou  $(\bar{n}, \bar{y})$  est un alignement de  $(n, y)$   
Dans  $|\bar{n}| = |\bar{y}|$   
et  $(\bar{u}, \bar{v})$  est un alignement de  $(u, v)$   
Dans  $|\bar{u}| = |\bar{v}|$

Dans:  $|\bar{y} \cdot \bar{v}| = |\bar{y}| + |\bar{v}| = |\bar{n}| + |\bar{u}|$

Dans on a:  $|\bar{y} \cdot \bar{v}| = |\bar{n}| + |\bar{u}|$   
et  $|\bar{n} \cdot \bar{u}| = |\bar{n}| + |\bar{u}|$

Dans  $|\bar{x} \cdot \bar{u}| = |\bar{y} \cdot \bar{v}|$

4) Montrer que:  $\forall i \in [1, \dots, |\bar{n} \cdot \bar{u}|], (\bar{n} \cdot \bar{u})_i \neq -$  ou  $(\bar{y} \cdot \bar{v})_i \neq -$

Si:  $i \in [1, \dots, |\bar{n}|]$  Dans  $i \in [1, \dots, |\bar{y}|]$

on a:  $\forall i \in [1, \dots, |\bar{n}|], (\bar{n})_i \neq -$  ou  $(\bar{y})_i \neq -$

Dans:  $\forall i \in [1, \dots, |\bar{n}|]$  on a  $(\bar{x} \cdot \bar{u})_i \neq -$  ou  $(\bar{y} \cdot \bar{v})_i \neq -$  ... ①

Si:  $i \in [|\bar{x}|+1, \dots, |\bar{n} \cdot \bar{u}|]$  Dans  $i \in [|\bar{y}|+1, \dots, |\bar{y} \cdot \bar{v}|]$

on pose:  $j = i - |\bar{n}|$  Dans  $j \in [1, \dots, |\bar{u}|]$  et  $j \in [1, \dots, |\bar{v}|]$

on a:  $\forall j \in [1, \dots, |\bar{u}|], (\bar{u})_j \neq -$  ou  $(\bar{v})_j \neq -$

Dans:  $\forall i \in [|\bar{n}|+1, \dots, |\bar{n} \cdot \bar{u}|], (\bar{n} \cdot \bar{u})_i \neq -$  ou  $(\bar{y} \cdot \bar{v})_i \neq -$  ... ②

De ① et ② o  $\forall i \in [1, \dots, |\bar{n} \cdot \bar{u}|], (\bar{n} \cdot \bar{u})_i \neq -$  ou  $(\bar{y} \cdot \bar{v})_i \neq -$

Conclusion:  $(\bar{n} \cdot \bar{u}, \bar{y} \cdot \bar{v})$  est un alignement  
de  $(n \cdot u, y \cdot v)$

Question 2: la longueur Max d'un alignement  $(x, y)$

Soient  $x \in \Sigma^*$ ,  $y \in \Sigma^*$  avec  $|x|=n \in \mathbb{N}$  et  $|y|=m \in \mathbb{N}$

\* Si  $|x|=|y|$ :

$$|\bar{x}|_{\text{max}} = |\bar{y}|_{\text{max}} = 2|x| = 2|y| = 2n = 2m$$

Exemple: Si  $x = \text{ATTGT}$ ,  $y = \text{ATCGT}$

- L'alignement de longueur Max est

$$\begin{cases} \bar{x} = \text{----- ATTGT} \\ \bar{y} = \text{ATCGT -----} \end{cases}$$

$$|\bar{x}| = 2|x| = 2 \times 5 = 10 = |\bar{y}|$$

$$|\bar{x}| = |\bar{y}| = |x| + |y| = n + m \quad \text{et comme } |x|=|y| \text{ alors } n=m$$

d'où  $|\bar{x}| = |\bar{y}| = 2n = 2m$

\* Si  $|x| \neq |y|$

$$|\bar{x}|_{\text{max}} = |\bar{y}|_{\text{max}} = n+m$$

Exemple: Si  $x = \text{ATTGT}$ ,  $y = \text{AGC}$

$$\begin{cases} \bar{x} = \text{ATTGT---} \\ \bar{y} = ---\text{AGC} \end{cases}$$

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

$$|\bar{x}| = |\bar{y}| = 5 + 3 = 8$$

Question 3: Nombre de Mot  $\bar{n} \in \bar{\Sigma}$  tq  $|\bar{n}| = |n| + K = n + K$   
avec  $|n|=n$  et  $K$  représente le Nombre de gaps.

$$C_K^{n+K} = \frac{(n+K)!}{K!(n+K-K)!} = \frac{(n+K)!}{K! n!}$$

Question 4: Soit un couple de mots  $(x, y)$  de longueurs respectives  $n$  et  $m$

\* une fois  $k$  gaps ajoutés à  $x$  pour obtenir  $\bar{x}$ , on ajoute :

-  $k + (n - m)$  gaps à  $y$  pour obtenir  $\bar{y}$ .

\* Nombre de façons d'insérer ces gaps dans  $y$ :

$$+ C_{n+k}^k \cdot C_n^{n+k-m}$$

donc: il y'a  $\sum_{k=0}^m C_{n+k}^k \cdot C_{n+k}^{n+k-m}$  alignements possibles de  $(x, y)$

- l'nombre d'alignement pour  $|x|=15$  et  $|y|=10$  est :

Question 5:

Si on pose  $\alpha$  le temps nécessaire à un algorithme naïf pour trouver un alignement, le temps nécessaire pour énumérer tous les alignements sera de:  $\alpha \times \sum_{k=0}^m C_{n+k}^k \cdot C_{n+k}^{n+k-m}$

donc: c'est une complexité de type factorielle.

\* pour trouver un alignement de coût minimal :

pour chaque alignement, on calcule son coût, donc si  $c$  est le temps nécessaire pour calculer le coût d'un alignement; le temps nécessaire pour l'algorithme est:  $(\alpha + c) \sum_{k=0}^m C_{n+k}^k \cdot C_{n+k}^{n+k-m}$ .

donc: ce n'est pas une complexité de type factorielle.

Question 6:

Si on pose  $\alpha$  la place mémoire requise pour un alignement maximal ( $|x| + |y| = \alpha$ ), on aura au pire besoin de:  $\alpha \times \text{nbr d'alignements en place mémoire}$ .

donc: c'est une complexité de type factorielle.

pour trouver un alignement minimal, il ne faut pas d'espace mémoire en plus

donc: c'est une complexité de type factorielle.

### 3.2) Programmation dynamique:

#### Question 7:

Comme  $(\bar{u}, \bar{v})$  est un alignement de  $(x, y)$ , donc  $\forall i \in [1..l]$ , avec  $l$  la longueur de  $\bar{u}$  et  $\bar{v}$ ,  $\bar{u}_i$  et  $\bar{v}_i$  ne peuvent pas être tous les deux égaux à "-", et ce par définition.

donc si  $\begin{cases} \bar{u}_i = - & \text{alors } \bar{v}_i = \{A, G, C, T\} \end{cases}$

$\begin{cases} \bar{v}_i = - & \text{alors } \bar{u}_i = \{A, G, C, T\} \end{cases}$

$\begin{cases} \bar{u}_i \neq - \text{ et } \bar{v}_i \neq - & \text{alors } \bar{v}_i = \{A, G, C, T\} \text{ et } \bar{u}_i = \{A, G, C, T\} \end{cases}$

#### Question 8:

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

#### Question 9:

on distingue 3 cas :

- 1) pour passer de  $D(i-1, j-1)$  à  $D(i, j)$ , il faut faire une substitution de  $(x_i, y_j)$ .
- 2) pour passer de  $D(i-1, j)$  à  $D(i, j)$ , il faut faire une suppression dans  $x$  de  $x_i$
- 3) pour passer de  $D(i, j-1)$  à  $D(i, j)$ , il faut faire une insertion dans  $x$  de  $y_j$

Donc : comme on cherche la distance d'édition, on prend le minimum des 3 cas

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

### Question 10 :

$D(0,0)$  vaut 0, car pour passer du mot vide au mot vide on ne fait rien.

### Question 11 :

$D(0,j) = j \times C_{ins}$ , car comme on a vu dans la question ⑨, pour passer de  $D(i,j-1)$  à  $D(i,j)$ , il faut insérer.

$D(i,0) = i \times C_{del}$ ,  $\forall i \in [1..n]$ , car comme on a vu dans la question ⑨, pour passer de  $D(i-1,j)$  à  $D(i,j)$  il faut supprimer.

### Question 12 :

$Dist_1(x,y)$

- on déclare D une matrice de taille  $(n+1) \times (m+1)$
- on l'initialise :

$$D(0,j) = j \times C_{ins}, \text{ pour } \forall j \in [1..m]$$

$$D(i,0) = i \times C_{del}, \text{ pour } \forall i \in [1..n]$$

$$D(0,0) = 0$$

- on parcourt toutes les cases de D, et pour chaque case, on lui affecte le :

$$\min ( D[i-1][j-1] + C_{sub}(x_i, y_j), D[i][j-1] + C_{ins}, D[i-1][j] + C_{del} )$$

- à la fin on retourne  $D[n][m]$  qui est la distance d'édition de  $(x,y)$

### Question 13 : Complexité spatiale

dans Dist-1, on utilise uniquement une matrice de taille  $n \times m$ , donc Dist-1 est en  $\Theta(n \times m)$

### Question 14 : Complexité temporelle.

Dist-1 est un algorithme itératif, composé de deux boucles imbriquées, avec à l'intérieur des opérations élémentaires.

Dist-1 fait  $(n \times m)$  itération.

donc Dist-1 est en  $\Theta(n \times m)$

### 3.2.2) Calcul d'un alignement optimal par programmation dynamique:

#### Question 15:

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

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

Soit  $i > 0$  et  $D(i, j) = D(i, j-1) + C_{ins}$

comme on a  $D(i, j-1)$ , on a donc des alignements qu'on note  $(\bar{x}, \bar{y})$  de  $(x_{[1 \dots i]}, y_{[1 \dots j-1]})$  optimaux, donc  $(\bar{x}, \bar{y}) \in Al^*(i, j-1)$

ici on a :  $D(i, j) = D(i, j-1) + C_{ins}$ , donc pour passer de

$x_{[1 \dots i]}$  à  $y_{[1 \dots j]}$ , il suffit d'insérer dans  $x_{[1 \dots i]}$  : "  $y_j$ "

donc  $\forall (\bar{x}, \bar{y}) \in Al^*(i, j-1), (\bar{x} \cdot -, \bar{y} \cdot y_j)$  (insertion de  $y_j$  dans  $x_{[1 \dots i]}$ ) est un alignement optimal de  $(x_{[1 \dots i]}, y_{[1 \dots j]})$

donc :  $(\bar{x} \cdot -, \bar{y} \cdot y_j) \in Al^*(i, j)$ .

### Question 16

Sol-1( $x, y, D$ )

Soit  $(x, y)$  un couple de  $\Sigma^* \times \Sigma^*$  et  $(\bar{x}, \bar{y}) \in \tilde{\Sigma}^* \times \tilde{\Sigma}^*$

D'une matrice de taille  $(n+1) \times (m+1)$  contenant les distances d'éditions.

$j = n$  et  $i = m$

Tant que  $i > 0$  et  $j > 0$ .

choisir le min entre  $D[i-1][j]$ ,  $D[i][j-1]$  et  $D[i-1][j-1]$

Si c'est  $D[i-1][j]$  alors:  $\bar{x} = \bar{x}_i \cdot x_i$  et  $\bar{y} = \bar{y}_{i-1}$

si c'est  $D[i][j-1]$  alors:  $\bar{x} = \bar{x}_{i-1} \dots$  et  $\bar{y} = \bar{y}_j \cdot y_j$

si c'est  $D[i-1][j-1]$  alors:  $\bar{x} = \bar{x}_{i-1} \cdot x_i$  et  $\bar{y} = \bar{y}_{j-1} \cdot y_j$

$i = i+1$  et  $j = j+1$

On inverse  $\bar{x}$  et  $\bar{y}$  (car on les a remplis à l'envers)

on renvoie  $\bar{x}$  et  $\bar{y}$

### Question 17: Complexité temporelle.

Dist-1 est en  $\Theta(n \cdot m)$ , Sol-1 fait au pire  $(n+m)$  itération et pour chaque itération, elle exécute des opérations élémentaires.

dans Sol-1 est en  $\Theta(n+m)$

donc le problème ALI est résolu en  $\Theta(n \cdot m)$

### Question 18: Complexité spatiale

Sol-1 n'utilise que la matrice  $D$  générée par DIST-1, donc SOL-1 est en  $\Theta(1)$ , Dist-1 est en  $\Theta(n \cdot m)$  comme vu précédemment, donc le problème ALI est résolu en  $\Theta(n \cdot m)$

### 3.3) Amélioration de la complexité spatiale du calcul de la distance

#### Question 19:

Dans DIST-1, pour remplir le tableau D pour une ligne  $i \gg$ , il suffit d'avoir accès aux lignes  $i-1$  et  $i$  car on utilise le tableau pour comparer entre  $D[i-1][j-1]$ ,  $D[i][j-1]$ ,  $D[i-1][j]$ , donc on accède qu'aux lignes  $i$  et  $i-1$ .

#### Question 20:

##### Dist-2

Soit  $D[2][m+1]$  une matrice de taille  $2 \times (m+1)$

on initialise D tel que :

$$D[0][j] = j \times \text{cins}, \forall j \in [0..m]$$

et  $D[1][0] = \text{Cdel}$

$$\text{cpt} = 1$$

Tant que  $\text{cpt} \leq n$

pour  $j$  allant de 1 à  $m$  faire :

on place en  $D[1][j]$  le min entre  $D[1][j-1] + C_{\text{ins}}$

$$D[0][j] + \text{Cdel}$$
 et  $D[0][j-1] + \text{Csub}(x_{\text{cpt}}, y_j)$

$$\text{cpt} = \text{cpt} + 1$$

on écrase la 1<sup>ère</sup> ligne du tableau avec la seconde.

$$D[1][0] = D[0][0] + \text{Cdel}$$

on retourne  $D[1][m]$

Ici Dist-2 est ~~On~~  $\Theta(m)$  en complexité spatiale car elle utilise une matrice de taille 2  $(m+1)$

### Question 21:

Mot-gaps( $K$ )

$x = \emptyset$  (mot vide)

pour  $i$  allant de 1 à  $K$  faire

$x = x_{i-}$

on retourne  $x$

### Question 22:

Soit  $x$  un mot de longueur  $l$  et  $y$  un mot de longueur quelconque qu'on note  $m$

- on vérifie si  $y$  contient la lettre du mot  $x$ 
  - 1) si oui: on note sa position  $i$ 
    - on ajoute  $i-1$  gaps à gauche de  $x$  avec mot-gaps( $i-1$ )
    - on ajoute  $m-i$  gaps à droite de  $x$  avec mot-gaps( $m-i$ )
  - 2) sinon:

- on cherche une lettre dans  $y$  pour former une paire concordante avec la lettre de  $x$

- si on la trouve on note son indice  $i$ :
  - on ajoute  $i-1$  gaps à gauche de  $x$
  - on ajoute  $m-i$  gaps à droite de  $x$

- Sinon on ajoute  $m-1$  gaps à droite de  $x$  on renvoie  $(x, y)$ .

### Question 23

\*  $\bar{S} = \text{BAL}$  ;  $\bar{E} = \text{RD-}$

\*  $\bar{u} = \text{LON-}$  ;  $\bar{v} = \text{--ND}$

Montre que  $(\bar{S}, \bar{u}, \bar{E}, \bar{v})$  n'est pas un alignement optimal de  $(x, y)$

- on calcule le coût de  $(\bar{S}, \bar{u}, \bar{E}, \bar{v})$  :

$$\text{Coût} = C_{\text{sub}}(B, R) + C_{\text{sub}}(A, \emptyset) + 3 \times C_{\text{del}} + C_{\text{sub}}(N, N) + C_{\text{ins}}$$

$$\Rightarrow \text{Coût} = 5 + 5 + 3 \times 3 + 0 + 3 = 21$$

or : pour  $(\bar{u}, \bar{v}) = (\text{BALLOON-}, \text{R---OND})$

le coût = 19

donc  $(\bar{S}, \bar{u}, \bar{E}, \bar{v})$  n'est pas un alignement optimal de  $(x, y)$

### Question 24: $\text{Sol}_2(x, y)$

- si  $x$  ne contient qu'une lettre (longueur = 1) :

on renvoie Align-Lettre-mot  $(x, y)$

- Si  $y$  est le mot vide :

on renvoie  $(x, \text{mot-gaps}(\text{longueur de } x))$

• Sinon :  $(x_1, y_1) = \text{Sol}_2(x[1..[\frac{n}{2}]], y[1..j])$

$(x_2, y_2) = \text{Sol}_2(x[[\frac{n}{2}]..n], y[j+1..m])$

on retourne  $(x_1 \cdot x_2, y_1 \cdot y_2)$

avec  $j$  la coupure  $(x, y)$  et  $n$  la longueur de  $x$

### Question 25:

Coupage ( $x, y$ )

Soit  $D$  et  $I$  deux matrices de taille  $2 \times (m+1)$

on initialise  $D$  tel que :

$$D[0][j] = c_{ins} \times j, \forall j \in [0..m]$$

$$\text{et } D[0][0] = C_{del}$$

on initialise  $I$  tel que :

$$I[0][j] = j \quad \forall j \in [0..m]$$

$$\text{et } I[1][0] = I[0][0]$$

$$cpt = 1, m = \text{longueur}(y)$$

Tant que  $cpt \leq n$

+ pour  $j$  allant 1 à  $m$  faire :

- on place en  $D[1][j]$  le min entre

$$D[0][j] + C_{del}, D[1][j-1] + c_{ins} \text{ et } D[0][j-1] + c_{sub}(x_{cpt}, y_j)$$

- si  $cpt > i^* (i^* = \text{longueur}(x) / 2)$

\* Si  $D[1][j] = D[0][j] + C_{del}$  alors.

$$I[1][j] = I[0][j]$$

\* si  $D[1][j] = D[1][j-1] + c_{ins}$  alors

$$I[1][j] = I[1][j-1]$$

\* si  $D[1][j] = D[0][j-1] + c_{sub}(x_{cpt}, y_j)$

$$I[1][j] = I[0][j-1]$$

+ on écrase dans les deux matrices la première ligne avec la deuxième.

$$\text{et } D[1][0] = D[0][0] + C_{del}$$

$$I[1][0] = I[0][0]$$

on retourne  $I[1][m]$

### Question 26

Coupure n'utilise que 2 matrices de taille  $2 \times (m+1)$

donc Coupure est en  $\Theta(m)$

### Question 27 :

Complexité spatiale de SOL-2 :

ici on décompose notre problème de taille  $n$  en deux sous-problème de taille  $\frac{n}{2}$ .

donc dans l'arbre des appels récursifs, on aura  $\log(n)$  niveaux. Donc au total on fait  $\alpha \cdot \log(n)$  appels récursifs, avec  $\alpha$  un nombre  $> 0$ .

pour chaque appel, on fait appel à Coupure qui est ici en  $\Theta(m)$  (pire cas taille =  $m$ ).

Donc : SOL-2 est en  $\Theta(m \cdot \log(n))$ .

### Question 28 :

Complexité temporelle de Coupure :

Coupure, comme DIST-2, est composé principalement de deux boucles imbriquées, qui fait donc  $n \times m$  itération, et à chaque itération, on fait des opérations élémentaires

Donc Coupure est en  $\Theta(n \cdot m)$

### Question 29 :

Oui, on a perdu en complexité temporelle en améliorant la complexité spatiale. La comparaison expérimentale se trouve dans le rapport consacré à la partie programmation.