2I003 Initiation à l'Algorithmique
Projet: Alignement des séquences ADN

Andrea KOSTAKIS

Novembre 2014

Première partie

Distance minimum de l'alignement de deux séquences

1 Execice 1

Rappelons les différentes définitions utiles pour la suite du projet :

Une séquence d'ADN (acide Deoxyribonucleique) est une suite de nucléotides. Chaque nucléotide est désigné par la première lettre de sa molécule de base.
Une séquence d'ADN est ainsi un mot sur l'alphabet $\mathcal{A} = \{A, C, G, T\}$.
Soient deux séquences ADN composées respectivement de n, m nucléotides : $a = a_0 ... a_{n-1}$ et $b = b_0 ... b_{m-1}$, alors
$\forall i \in \{0, ..., n-1\}, a_i \in \mathcal{A}$ et $\forall j \in \{0, ...m-1\}, b_j \in \mathcal{A}$.

Alignement : On appelle alignement de longueur $L \geq \max(n, m)$ des séquences a et b tout couple (a^*, b^*) où $a^* = a_0^* ... a_L^*$ et $b^* = b_0^* ... b_L^*$ sont deux mots de longueur L constuits en insérant des éléments nuls dans les séquences a et b respectivement. C'est à dire, $\forall i \in \{0, .., L-1\}$, a_i^* et b_i^* sont des éléments de $\mathcal{A} \cup \{-\}$.

Distance de Levenshtein : la distance entre deux éléments de $\mathcal{A} \cup \{-\}$ est définie par :
- $D(\alpha, \alpha) = 0, \forall \alpha \in \mathcal{A} \cup \{-\}$
- $D(\alpha, \beta) = 1, \forall (\alpha, \beta) \in (\mathcal{A} \cup \{-\})^2$ avec $\alpha \neq \beta$

La distance d'un alignement est alors :

$$D(a^*, b^*) = \sum_{k=0}^{L-1} D(a_k^*, b_k^*)$$

1.1 Alignements : Distance et longueur

Montrons que, si pour $j \in$\{0,...,$L-1$\},a$_j^*$=$b_j^* = -$, on peut construire un autre alignement (a'^*, b'^*) de même distance et longueur $L-1$.

Soit L la longueur de l'alignement (a^*, b^*) si pour j $\in\{0,...,L-1\}$, a^*_j=$b^*_j = -$, alors la distance $D(a^*, b^*)$ est :

$$D(a^*, b^*) = \sum_{k=0}^{L-1} D(a^*_k, b^*_k) = \sum_{k=0}^{j-1} D(a^*_k, b^*_k) + D(a^*_j, b^*_j) + \sum_{k=j+1}^{L-1} D(a^*_k, b^*_k)$$

$$\text{par définition, } D(a^*_j, b^*_j) = 0 \text{ donc,}$$

$$D(a^*, b^*) = \sum_{k=0}^{j-1} D(a^*_k, b^*_k) + \sum_{k=j+1}^{L-1} D(a^*_k, b^*_k)$$

Construisons un autre alignement $(a^{'*}, b^{'*})$ de longueur L' en retirant les éléments a^*_j et b^*_j (càd l'éléмént nul - en position j de a^* et b^*) de l'alignement (a^*, b^*) , on obtient donc l'alignement $(a^{'*}, b^{'*})$ où

$$a^{'*} = {a^*}_0...{a^*}_{j-1}{a^*}_{j+1}...{a^*}_{L-1}$$
$$b^{'*} = {b^*}_0...{b^*}_{j-1}{b^*}_{j+1}...{b^*}_{L-1}$$

On remarque d'après la construction que l'on obtient un alignement de longueur $L^{'} = L - 1$. La distance de $(a^{'*}, b^{'*})$ est :

$$D(a^{'*}, b^{'*}) = \sum_{k=0}^{L^{'}-1} D(a^{'*}_k, b^{'*}_k)$$

$$\text{par constuction : } = \sum_{k=0}^{j-1} D(a^*_k, b^*_k) + \sum_{k=j+1}^{L-1} D(a^*_k, b^*_k)$$

$$= D(a^*, b^*)$$

On obtient donc bien un autre alignement $(a^{'*}, b^{'*})$ de même distance et de longueur $L - 1$.

1.2 Alignement optimal de sous séquence

Soit le couple (a^*, b^*) de longueur L des séquences $a = a_0...a_{n-1}$ et $b = b_0...b_{m-1}$ de distance minimum avec $n > 0$ et $m > 0$. c(n-1,n) $\geq$ $1 + c(n-2, n-1) + c(n-2, n) + c(n-1, n-1)$

1.2.1 Demonstration

Montrons par l'absurde que si $a^*_{L-1} = -$ et $b^*_{L-1} = b_{m-1}$ alors le couple $({a^*}_0...{a^*}_{L-2}, {b^*}_0...{b^*}_{L-2})$ est un alignement optimal de a avec la sous séquence et $b_0...b_{m-2}$.
Raisonons donc par l'absurde :
Supposons que le couple $({a^*}_0...{a^*}_{L-2}, {b^*}_0...{b^*}_{L-2})$ n'est pas un alignement optimal.
Donc, il existe un autre alignement $(a^{'*}, b^{'*})$ provenant des mêmes séquences tel que

$$D(a^{'*}, b^{'*}) < D({a^*}_0...{a^*}_{L-2}, {b^*}_0...{b^*}_{L-2})$$

Par définition de la distance,

$$D(a^{'*}, b^{'*}) < \sum_{k=0}^{L-2} D(a^*_k, b^*_k)$$

(a^*, b^*) est un alignement optimal et sa distance peut s'écrire de la manière suivante :

$$D(a^*, b^*) = \sum_{k=0}^{L-2} D(a^*_k, b^*_k) + D(a^*_{L-1}, b^*_{L-1})$$

on remarque que

$$D(a^*_{L-1}, b^*_{L-1}) = 1 \text{ car } (a^*_{L-1} \neq b^*_{L-1})$$

enfin, on a également

$$\sum_{k=0}^{L-2} D(a^*_k, b^*_k) = D({a^*}_0...{a^*}_{L-2}, {b^*}_0...{b^*}_{L-2})$$

on peut donc écrire

$$D({a^*}_0...{a^*}_{L-2}, {b^*}_0...{b^*}_{L-2}) = D(a^*, b^*) - 1$$

A fortiori,

$$D(a^{'*}, b^{'*}) < D(a^*, b^*) - 1$$

Cette inégalité nous ramène à une contradiction. (en retirant une unité à une distance optimale on obtient forcement un résultat optimal). Il n'existe donc pas d'autre alignement optimal. On peut en conclure que le couple $({a^*}_0...{a^*}_{L-2}, {b^*}_0...{b^*}_{L-2})$ est un alignement optimal de a avec la sous séquence et $b_0...b_{m-2}$.

1.2.2 Autres valeurs possibles

Examinons le cas des autres valeurs possible de a^*_{L-1} et b^*_{L-1} :

- si $a^*_{L-1} = -$ et $b^*_{L-1} = -$ alors, $D({a^*}_0...{a^*}_{L-2}, {b^*}_0...{b^*}_{L-2}) = D(a^*, b^*)$ (question 1.1) et $({a^*}_0...{a^*}_{L-2}, {b^*}_0...{b^*}_{L-2})$ est un alignement optimal
- respectivement si $a^*_{L-1} = b^*_{L-1}$
- si $a^*_{L-1} = a_{n-1}$ et $b^*_{L-1} = -$ alors $({a^*}_0...{a^*}_{L-2}, {b^*}_0...{b^*}_{L-2})$ est un alignement optimal avec le même raisonemant que la question précédente

1.3 Décompositon de la distance minimale

Pour tout couple $(i, j) \in \{-1, ..., n-1\} \times \{-1, ..., m-1\}$ on désigne par $\bar{D}_{i,j}$ la distance minimale d'un alignement des séquences ${a^*}_0...{a^*}_i$ et ${b^*}_0...{b^*}_j$. Le cas $i = -1$ (resp.$j = -1$) correspond à une séquence vide notée ϵ. Par convention, on pose

$$D_{-1,-1} = 0$$

Montrons que,

$$\forall i \in \{0, ..., n-1\}, \bar{D}_{i,-1} = \bar{D}_{i-1,-1} + D(a_i, -)$$

et que,

$$\forall j \in \{0, ..., m-1\}, \bar{D}_{-1,j} = \bar{D}_{-1,j-1} + D(-, b_i)$$

$\forall i \in \{0, ..., n-1\}, \bar{D}_{i,-1}$ correspond à la distance optimale entre $a = a_0...a_i$ et ϵ. D'après 1.2, si $D_{i,-1}$ est optimale alors $D_{i-1,-1}$ est optimale.
on peut donc écrire,

$$\bar{D}_{i,-1} = \bar{D}_{i-1,-1} + D(a_i, -)$$

car $\forall i \in \{0, ..., n-1\} D(a_i, -)$ est la distance du dernier couple.

Respectivemevent,

$$\forall j \in \{0, ..., m-1\}, \bar{D}_{-1,j} = \bar{D}_{-1,j-1} + D(-, b_i)$$

1.4 Alignement optimal : calcul

Démontrons que pour tout couple $(i, j) \in \{0, ..., n-1\} \times \{0, ...m-1\}$,

$$\bar{D}_{i,j} = min(\bar{D}_{i-1,j} + D(a_i, -), \bar{D}_{i-1,j-1} + D(a_i, b_j), \bar{D}_{i,j-1} + D(b_j, -))$$

Examinons le cas d'un alignement optimal pour tout couple (i, j), on observe trois cas particuliers que l'on peut décomposer d'après les questions précédentes :

– si $n > m$ alors, $\forall (i,j) \in \{0,...,n-1\} \times \{0,...m-1\}$

$$\bar{D}_{i,j} = \bar{D}(a_{i-1}, b_j) + D(a_i, -) = \bar{D}_{i-1,j} + D(a_i, -) \tag{1}$$

– si $n = m$, alors $\forall (i,j) \in \{0,...,n-1\} \times \{0,...m-1\}$

$$\bar{D}_{i,j} = \bar{D}_{i-1,j-1} + D(a_i, b_j) \tag{2}$$

– si $n < m$ alors,$\forall (i,j) \in \{0,...,n-1\} \times \{0,...m-1\}$

$$\bar{D}_{i,j} = \bar{D}_{i,j-1} + D(b_j, -) \tag{3}$$

Par conséquant si l'on veut obtenir l'alignement optimal, on doit regrouper les trois cas, càd minimiser (1)(2)(3).
D'où,

$$\bar{D}_{i,j} = min(\bar{D}_{i-1,j} + D(a_i, -), \bar{D}_{i-1,j-1} + D(a_i, b_j), \bar{D}_{i,j-1} + D(b_j, -))$$

C'est une définition récursive de la distance d'un l'alignement optimal.

1.5 DistanceMinRec(a,b)

D'après 1.4 on peut definir la fonction récursive suivante :

Supposons definie la fonction Distance(a[i],b[i]) qui rend la distance entre deux éléments. (implémenté Exercice 2).

```
def DistanceMinRec(a,b):
n=len(a)
m=len(b)
d=0
if(n==0 and m==0):
  return 0
if(n=0):
  return m
if (m=0):
  return n
return min(DistanceMinRec(a[n-1],b)+1,
  DistanceMinRec(a[n-1],b[m-1])+ Distance(a[n],b[m]),\subsection{Fonctions}
  DistanceMinRec(a,b[m-1])+1)
```

1.6 Terminaison et validité

Par récurrence (forte) sur $k = n + m$ Prouvous que DistanceMinRec se termine et renvoie la distance minimale dede l'alignement des séquences a,b.

Base : Pour n=m=0 (k=0) la fonction se termine et renvoie 0, qui est bien la distance minimale des séquences nulles.

Induction : Soit DistanceMinRec ce termine à un certain rang k et renvoie la distance minimale des k derniers elements de la séquence a,b.
A chaque appel récursif de DistanceMinRec on renvoie la distance des sous-séquences k-1 à n+m.
Au rang $k_0 + 1 (k_0 > k)$ la fonction renvoie la distance minimale de la sous-séquence k-1 à n+m.
A ce rang on obtiendra m=0 où n=0 donc la fonction ce termminera.
De plus, comme pour chaque sous-séquence la distance minimale est calculé pour les $k_0 - 1$ éléménents.

Conclusion : Par récurrence au rang k_0 DistanceMinRec(a,b) se termine et est valide.

1.7 Complexité de DistanceMinRec

Soit $c(n, m)$ la compléxité pour deux séquences de taille réspective n et m. On note $u_n = c(n, m)$ pour deux séquences de même taille.

1.7.1 Calcul de $c(n, m)$

Pour $n > 0$ et $m > 0$,
d'après la fonction on appelle récursivement DistanceMinRec(a[n-1],b), DistanceMinRec(a[n-1],b[m-1]) et DistanceMinRec(a,b[m-1]) puis Distance(a[n],b[m]) qui s'éxecute en temps constant (= 1).

La compléxité $c(n, m)$ s'écrit :

$$c(n, m) \geq 1 + c(n-1, m) + c(n-1, m-1) + c(n, m-1)$$

En remplacant dans l'inégalité obtenue, on obtient :

$$\begin{aligned} c(n-1, n) &\geq 1 + c(n-2, n-1) + c(n-2, n) + c(n-1, n-1) \\ \text{et } c(n, n-1) &\geq 1 + c(n-1, n-2) + c(n-1, n-1) + c(n-1, n-2) \end{aligned}$$

On en déduit que pour $n > 1$:

$$c(n-1, n) \geq c(n-1, n-1) \text{ et } c(n, n-1) \geq c(n-1, n-1)$$

1.7.2 Compléxité u_n

$$
\begin{aligned}
u_n = c(n,n) \geq 1 + c(n-1,n-1) + c(n-1,n) + c(n,n-1) \\
\text{d'après 1.7.1 :} \\
c(n-1,n) \geq c(n-1,n-1) \\
c(n,n-1) \geq c(n-1,n-1) \\
\text{donc,} \\
u_n \geq 1 + 3c(n-1,n-1) \\
u_n \geq 1 + 3u_{n-1} \text{ et } u_0 = 1
\end{aligned}
$$

Donc, comme u_{n-1} est borné par $(-1/3)^{n-1}$ alors $u_n \geq (-1/3)^n$ qui est bien une fonction exponentielle.

1.8 DistanceMinIter(a,b)

Voici le code proposé :

```
DistanceMonIter(a,b):
n=len(a)
m=len(b)
result=0
for (i=0;i<=n;i++):
  M[i][0]=i
for (j=0;j<=m;j++):
  M[0][j]=j
for (i=0;i<=n;i++):
   for (j=0;j<=m;j++):
      flag=Distance(a[n],b[n])
     M[i]][j]=min(M[i-1][j]+1,M[i-1][j-1]+val,M[i][j-1]
result=M[n][m]
return result
```

1.9 Complexité de DistanceMinIter(a,b)

La fonction itérative ci-dessus est composé de plusieurs boucles de types for. On parcourt $n \times m$ cases acr M est un tableau à deux dimensions.

D'où une complexité en $\Theta(n \times m)$

Deuxième partie

Mesure expérimentale de la complexité

2 Exercice 2

Le langage choisit pour la suite du projet est le C. Rappelons que la complexité expérimentale des fonctions est la mesure du temps d'execution.

2.1 Fonctions

Nous créons les fonctions Distance(a,b) et minimum(a,b,c) suplémentaires afin d'alleger le code.

Les listings de codes ce trouvent à la fin du rapport. Voici un exemple d'execution pour des séquences de longueur L = 10 pour la fonction itératibe et récursive respectivement le retour est en premienre ligne le temps d'execution puis le résultat :

```
La distance minimale des sequences
 CCTCTGTCCC
et
 TGCTCTATGT
= 6
 Rec= 0.970000sec
La distance  minimale des sequences
 CCTCTGTCCC
et
 TGCTCTATGT
= 6
```

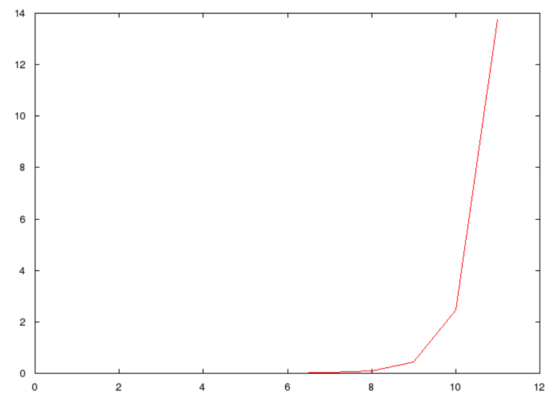
Nous créons égalements les fichier resultatsiter.txt et resultasrec.txt avec le temps d'éxecution (1ere colone) et longueur (2eme colone) :

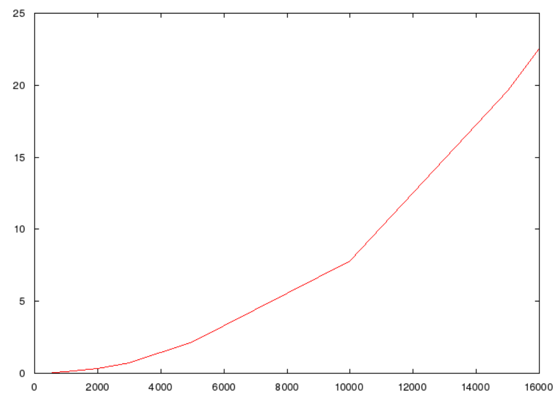
```
 ==> resultatsite.txt <==
0.000183 10
0.000268 15
0.000090 20
0.000572 40
0.000566 50
0.001793 100
0.004398 200
0.009176 300
0.015309 400
0.023304 500
```

```
0.032963 600
0.084551 1000
0.321449 2000
0.714503 3000
2.187023 5000
7.810801 10000
19.658818 15000
22.574136 16000
==> resultatsrec.txt <==
0.000008 1
0.000019 2
0.000053 3
0.000200 4
0.000589 5
0.003614 6
0.016190 7
0.080566 8
0.438319 9
2.449000 10
13.736783 11
```

2.2 Temps d'éxecution

En fonction des performance de l'ordinateur on observe que pour des longeurs supérieurs à 12 la pile de récurtion ne marche pas, le processus est tué par le système. Pour la fonction itérative cette limite de mémoire est un peu près à L=20000.
D'après les fichiers de résultat et la fonction gnuplot du shell on obtient les graphes suivants en fonction de la taille.





——— DistanceMinRec ——————— DistanceMinIter—

D'après les graphes on voit bien que DistanceMinRec à une complexité expérimentale exponentielle. DistanceMinIter à une compléxité expérimentale linéaire si l'on neglige les "bruits du processeur".

2.3 Listings

```
 ==> Distance.c <==
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

int Distance(char a,char b)
{
  if(a==b)
    return 0;
  return 1;
}


==> DistanceMinIter.c <==
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include "Distance.h"
#include "Minimum.h"

int DistanceMinIter(char* a,char* b)
{

  int i,j;
  int n=strlen(a);
  int m=strlen(b);

  int* M=(int *)malloc((n+1)*(m+1)*sizeof(int));
  int tmp;
  int res=0;

  for(i=0;i <=n ; i++){
    *(M + i*(m+1))=i;
  }
  for(j=0;j <=m ; j++){
    *(M+j)=j;
  }


  for(i=1;i <=n ; i++){
    for(j=1 ; j<=m ;j++){
      tmp=Distance(*(a+i-1),*(b+j-1));
      *(M + i*(m+1) + j) = minimum(*(M +(i-1)*(m+1) + j) +1,
                                   *(M +(i-1)*(m+1) + (j-1))+tmp,
```

```
                              *(M +i*(m+1) + (j-1)) +1);
    }
    printf("\n");
  }
  res= *(M + n*(m+1) +m);

  free(M);
  return res;

}

==> DistanceMinRec.c <==
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include "Distance.h"
#include "Minimum.h"


int DistanceMinRec(char* a,char* b)
{

  int n=strlen(a);
  int m=strlen(b);

  if(n==0 && m==0)
    return 0;
  if(n==0)
    return m;
  if(m==0)
    return n;

  char* a_bis=strndup(a,n-1); //strndup: duplicate a string,
                                rend un pointeur sur le nouveau
  char* b_bis=strndup(b,m-1);

  return minimum(DistanceMinRec(a_bis,b) +1,
              DistanceMinRec(a_bis,b_bis)+
              Distance(*(a+n-1),*(b+m-1)),
              DistanceMinRec(a,b_bis) +1);

}


==> main.c <==

#include <stdio.h>
```

```c
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include "Distance.h"
#include "Minimum.h"
#include "SequenceAleatoire.h"
#include "DistanceMinRec.h"
#include"DistanceMinIter.h"

#define L 10

int main(){
    srand(time(NULL));
    char* a=SeqAleatoire(L);
    char* b=SeqAleatoire(L);

    int dm;
    int dmin;

    clock_t start,end;


    FILE *fit, *frec;
    fit=fopen("resultatsite.txt","a+");
    frec=fopen("resultatsrec.txt","a+");
    start = clock();
    {
      dm=DistanceMinIter(a,b);
    }
    end = clock();
    printf("Iter= %fsec\n",(end - start)/(double)CLOCKS_PER_SEC);
    fprintf(fit,"%f %d\n",(end - start)/(double)CLOCKS_PER_SEC,L);
    printf("La distance minimale des s[U+FFFD]quences \n %s\net\n %s \n= %d \n",
            a,b,dm);

    start = clock();
    {
      dmin=DistanceMinRec(a,b);
    }
    end = clock();

    printf(" Rec= %fsec\n",(end -start)/(double)CLOCKS_PER_SEC);
    fprintf(frec,"%f %d\n",(end - start)/(double)CLOCKS_PER_SEC,L);
    printf("La distance  minimale des s[U+FFFD]quences\n %s\net\n %s\n= %d \n",
            a,b,dmin);


    free(a);
```

```
    free(b);

    return 0;
    }

==> Minimum.c <==
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>


int minimum(int a,int b,int c)
{
  if(a==b && a==c)
    return a;
  if(a<=b && a<=c)
    return a;
  if(b<=a && b<=c)
    return b;

  return c;
}


==> SequenceAleatoire.c <==
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>



char* SeqAleatoire(int n)
{
  int flag;
  int i;
  char* chaine=malloc((n+1)*sizeof(char));


  for(i=0;i<n;i++)
    {
    flag=rand()%4;
    if(flag==0)
      *(chaine+i)='A';
    if(flag==1)
      *(chaine+i)='C';
    if(flag==2)
      *(chaine+i)='G';
```

```
      if(flag==3)
        *(chaine+i)='T';
      }

    *(chaine+i)='\0';
    return chaine;}
```