Mini-Projet d'Algorithmique
Analyse de séquences en bioinformatique

3I003

Licence d'Informatique L3

Université Pierre et Marie Curie

Année universitaire 2016-2017

1 Introduction

De multiples facteurs sont à l'origine de modifications de la séquence génomique de l'ADN. Ces erreurs et ces mutations sont susceptibles de se propager au sein des populations. Ainsi, la séquence d'un génome d'une espèce évolue dans le temps. Une séquence génomique est l'enchaînement des quatre types d'acides aminés A, C, G, T. On appelle ici *gène* une sous-séquence "courte" d'une séquence génomique que l'on retrouve au sein du génome d'une population et qui a été identifiée comme codant des informations. Le nombre de nucléotides d'une séquence d'ADN est d'environ 3,2 milliards alors qu'un gène a une taille autour d'un millier de nucléotides : cependant certains gènes peuvent atteindre un million de nucléotides.

Un des problèmes importants rencontré en bioinformatique est ainsi d'identifier la fonction des gènes dans une séquence génomique. En particulier, certains gènes correspondent à des protéines qui seront générées par l'organisme. Il est souvent fait l'hypothèse que si deux gènes se ressemblent, les protéines correspondantes assument des fonctions semblables : par exemple si la fonction de l'une est connue, la fonction de la seconde peut s'en déduire. On cherche donc à identifier des similarités entre séquences d'ADN (on parle d'homologie des séquences).

Concevoir une mesure de similarité entre deux séquences de même longueur est une tâche facile. On peut par exemple se limiter à compter le nombre de caractères qui diffèrent comme l'illustre la figure ci-dessous (en rouge dans la figure).

(a)
A G T A T C
A G A T G C

(b)
A G T T T C
A G A T T C

Le problème de la comparaison de séquences est un peu plus compliqué, pour deux raisons. D'une part, les séquences à comparer ont rarement la même longueur ; et lorsque c'est le cas, elles ne doivent pas être forcément comparées sur cette longueur seulement.

Dans la figure ci-dessous, les séquences en (a) sont peu similaires si elles sont comparées sur toutes leurs longueurs mais un simple décalage permet d'augmenter le nombre de caractères concordants (b).

(a)
A C T T G C A T T
A A C T T G C A T

(b)
– A C T T G C A T T
A A C T T G C A T –

©2016-2017 by UPMC/Licence Informatique

Par ailleurs, la séquence d'un génome évoluant dans le temps, des acides aminés ont pu s'insérer ou au contraire disparaître au cours de l'évolution. Pour tenir compte de ces insertions ou disparitions, on introduit un caractère particulier dans l'alphabet utilisé pour représenter les acides aminés, appelé *gap* et noté "-". La mise en correspondance d'un caractère d'une séquence X avec un gap dans une autre séquence Y s'interprète soit comme une insertion du caractère dans la séquence X, soit comme une disparition d'un caractère dans Y.

Le calcul de la similarité de deux séquences requiert donc une opération préalable d'alignement, c'est-à-dire d'insertion de caractères gap dans l'une ou l'autre des séquences et de mise en correspondance des caractères de l'une avec les caractères de l'autre. La figure ci-dessous montre deux alignements possibles de la même paire de séquences.

Pour chaque alignement, il est possible de calculer la similarité des deux séquences alignées en comptant par exemple le nombre de gaps et de caractères non concordants. On cherchera alors à trouver le *meilleur alignement* entre elles pour identifier les concordances (que l'on appellera aussi similarités).

2 Définition du problème et objectifs du projet

Etant données deux séquences $X = (x_1, \cdots, x_m)$ et $Y = (y_1, \cdots, y_n)$ de longueurs respectives m et n et définies sur un alphabet Σ (par exemple $\{A, C, G, T\}$), un *alignement* M est un ensemble de paires (x_i, y_j) tel que chaque caractère n'apparaît qu'au plus une fois et qu'il n'y a pas de croisement entre tout couple de paires, *i.e* si $(x_i, y_j) \in M$ et $(x_{i'}, y_{j'}) \in M$ avec $i < i'$ alors $j < j'$.

Considérons par exemple les séquences $X = (C, T, A, C, C, G)$ et $Y = (T, A, C, A, T, G)$. Le tableau ci-dessous illustre un alignement possible de X et Y :

X	x_1	x_2	x_3	x_4	x_5		x_6
	C	T	A	C	C	-	G
	-	T	A	C	A	T	G
Y		y_1	y_2	y_3	y_4	y_5	y_6

L'alignement M de X et Y représenté ci-dessus est :
$M = \{(x_2, y_1), (x_3, y_2), (x_4, y_3), (x_5, y_4), (x_6, y_6)\}$.
On peut noter que certains caractères de X ou de Y peuvent ne pas apparaître dans un alignement lorsqu'ils correspondent à un gap : c'est le cas ici pour x_1 et pour y_5. On dit qu'une paire (x_i, y_j) de M est *concordante* si $x_i = y_j$, c'est le cas dans l'exemple pour la

©2016-2017 by UPMC/Licence Informatique

paire (x_3, y_3) ; la paire (x_5, y_4) est alors dite non-concordante car $x_5 \neq y_4$.

Définissons maintenant le coût d'un alignement M.
On associe une pénalité strictement positive δ_{gap} à chaque gap, c'est-à-dire à toute valeur x_i ou y_j de X ou Y qui n'apparaît pas dans M. On associe également une pénalité de concordance à toute paire de M : pour cela, on utilise la valeur $\delta_{x_i y_j}$ telle que

$$\delta_{x_i y_j} = \begin{cases} 0 & \text{si } x_i = y_j \\ \text{une valeur strictement positive} & \text{sinon} \end{cases}$$

Le coût d'un alignement sera alors défini comme la somme des pénalités, c'est-à-dire

$$f(M) = \sum_{(x_i,y_j)\in M} \delta_{x_i y_j} + \sum_{x_i \notin M} \delta_{gap} + \sum_{y_j \notin M} \delta_{gap}$$

Comme décrit dans la section précédente, plus cette valeur est petite et plus les séquences se "ressemblent".

Etant données deux séquences X et Y, le problème consiste alors à trouver un alignement M^* de coût $f(M^*)$ minimal. Ce projet a pour objet l'analyse et la mise en œuvre de plusieurs algorithmes résolvant ce problème.

Le travail se divise en une **partie théorique** et une **partie expérimentale**. La partie théorique permet d'établir et d'analyser plusieurs algorithmes ainsi que leurs complexités respectives. La partie expérimentale porte sur la mise en œuvre de ces algorithmes et la vérification expérimentale de leurs complexités respectives.

3 Partie théorique

Partie 1

Question 1.1
Quelle serait la complexité d'un algorithme naïf qui consisterait à énumérer tous les alignements possibles entre deux séquences de même longueur d ?

On cherche à résoudre efficacement ce problème en caractérisant la structure d'une solution optimale. On considère pour la suite de l'analyse deux séquences $X = (x_1, \cdots, x_m)$ et $Y = (y_1, \cdots, y_n)$ de longueurs respectives m et n.

Question 1.2
Soit M un alignement de X et Y. Montrer (par l'absurde) que si $(x_m, y_n) \notin M$, alors x_m ou y_n n'apparaît pas dans M.

Question 1.3
En vous basant sur la propriété précédente, en déduire les trois cas de figures qui seront à

©2016-2017 by UPMC/Licence Informatique

considérer pour x_m et y_n dans un alignement. Justifier votre réponse.

Soit $F(i, j)$ le coût minimal pour l'alignement des séquences $(x_1, \cdots, x_i)$ et $(y_1, \cdots, y_j)$.

Question 1.4
Exprimer $F(m, n)$ en fonction de $F(m-1, n-1)$ ou de $F(m, n-1)$ ou de $F(m-1, n)$ pour chaque cas de figures identifié à la question précédente.

Question 1.5
En vous basant sur les mêmes arguments, proposer une formule de récurrence pour calculer $F(i, j)$ pour $i \geq 1$, $j \geq 1$.

Question 1.6
Montrer que $F(i, 0) = i\delta_{gap}$ pour tout $i \in \{1, \cdots, m\}$ et $F(0, j) = j\delta_{gap}$ pour tout $j \in \{1, \cdots, n\}$.

Question 1.7
En déduire un algorithme `COUT1` permettant de trouver la *valeur* d'un alignement de coût minimal pour les séquences X et Y. Préciser sa complexité (en temps et en espace).

Question 1.8
Proposer un algorithme `SOL1` qui permet de construire un *alignement* optimal pour les séquences X et Y en supposant que vous disposez des valeurs des coûts $F(i, j)$. Donner sa complexité en temps de `SOL1` et conclure sur la complexité globale en temps et espace de cette approche.

Partie 2

On s'intéresse dans cette partie à une représentation du problème sous forme de graphe. Les notations introduites ici sont nécessaires à la partie 3.

On considère une représentation sous forme de graphe appelé *grille* : dans un tel graphe, les sommets sont numérotés par deux indices correspondant aux lignes et colonnes d'un tableau. Par exemple, dans la représentation de la figure 1, le sommet en haut à gauche est le sommet $(0, 0)$ et celui en bas à droite est le sommet $(4, 4)$.

Dans cette partie, on munit les séquences X et Y d'un préfixe gap noté "-" positionné à $i = 0$ et $j = 0$ respectivement : les séquences ont donc une taille $m + 1$ et $n + 1$ respectivement, avec $x_0 = y_0 =$"-".
Le graphe grille issu de X et de Y, noté G_{XY}, est défini de la façon suivante :

- Les sommets (i, j) représentent tous les couples (x_i, y_j) pour $i = 0, \cdots, m$ et $j = 0, \cdots, n$.

©2016-2017 by UPMC/Licence Informatique

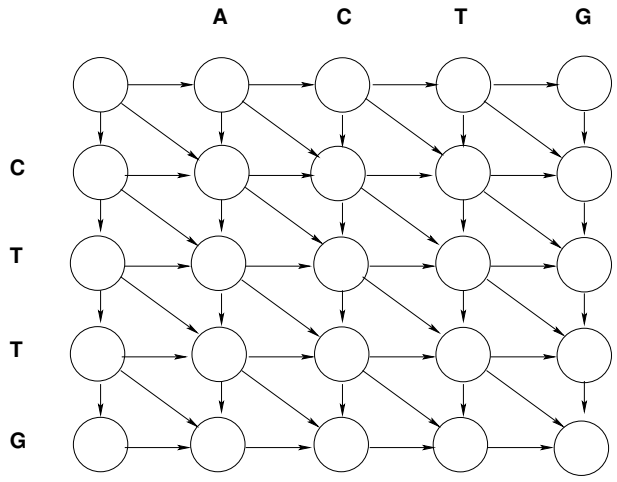


FIGURE 1 – Graphe représentant l'alignement des séquences $X = (C, T, T, G)$ et $Y = (A, C, T, G)$.

- Les arcs "horizontaux" définis entre les sommets (i, j) et $(i, j+1)$ sont de coût δ_{gap} pour $i = 0, \cdots, m$ et $j = 0, \cdots, n-1$.
- Les arcs "verticaux" définis entre les sommets (i, j) et $(i+1, j)$ sont de coût δ_{gap} pour $i = 0, \cdots, m-1$ et $j = 0, \cdots, n$.
- Les arcs "diagonaux" définis entre les sommets $(i-1, j-1)$ et (i, j) sont de coût $\delta_{x_i y_j}$ pour $i = 1, \cdots, m$ et $j = 1, \cdots, n$.

Question 2.1
Donner sur un dessin le coût des arcs pour l'exemple défini dans le graphe représenté à la figure 1 pour les séquences $X = (C, T, T, G)$ et $Y = (A, C, T, G)$ de longueur 4 en supposant que $\delta_{gap} = 2$ et que $\delta_{x_i y_j} = 3$ si x_i et y_j sont une paire non-concordante $(A, T), (T, A), (C, G)$ ou (G, C) et $\delta_{x_i y_j} = 4$ si x_i et y_j sont une autre paire non-concordante.

On note $g(i, j)$, la longueur du plus court chemin entre le sommet $(0, 0)$ et (i, j) dans le graphe G_{XY}.

Question 2.2
[*Cette question est facultative, vous pourrez admettre le résultat*]
Montrer que pour tout couple (i, j), $i \in \{0, \ldots, m\}$ et $j \in \{0, \ldots, n\}$, on a $F(i, j) = g(i, j)$. En déduire que le coût d'un alignement optimal des séquences X et Y est la longueur d'un plus court chemin entre le sommet $(0, 0)$ et le sommet (m, n) dans G_{XY}.
Indication : On raisonnera par récurrence forte sur $i + j$.

Question 2.3
Quel algorithme proposez-vous pour déterminer un plus court chemin de $(0, 0)$ à (m, n) dans G_{XY} : quelle est sa complexité ? Calculer le coût d'un alignement optimal pour l'exemple défini à la question 2.1 pour les séquences $X = (C, T, T, G)$ et $Y = (A, C, T, G)$. Indiquer l'alignement correspondant.

©2016-2017 by UPMC/Licence Informatique

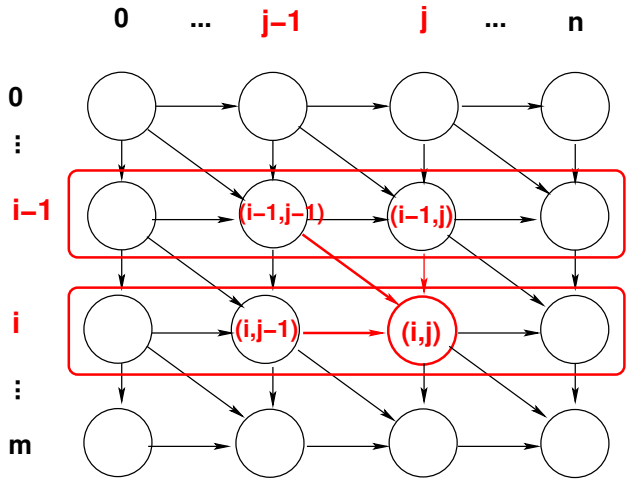
Question 2.4
Est-ce que la méthode proposée dans la partie 2 est de meilleure complexité que celle de la partie 1 ?

Partie 3

Dans cette partie, on s'intéresse à améliorer la complexité spatiale des algorithmes de la partie 1.

Question 3.1
Compte-tenu de la mémoire vive (RAM) disponible dans un ordinateur, quelle longueur maximale (en nombre de nucléotides) peut-on traiter par les méthodes des parties précédentes ? On peut par exemple, considérer ici que les gènes ont la même taille, qu'une mémoire vive d'ordinateur varie de 8 à 32GO et que le codage d'un caractère demande 1 octet.

On remarque que le calcul de $F(i,j)$ nécessite seulement l'évaluation préalable des valeurs de F pour les lignes $i-1$ et i comme l'illustre le graphe ci-dessous :



Question 3.2
Proposer une nouvelle variante que l'on appellera `COUT2` de l'algorithme `COUT1` qui améliore sa complexité spatiale : cette fonction doit calculer la valeur $F(i,j)$ pour des valeurs $i \in \{0,\ldots,m\}$ et $j \in \{0,\ldots,n\}$ quelconques. Donner sa complexité temps et espace.
Indication : on utilise pour ce faire deux tableaux de taille n pour stocker les valeurs de F : un pour la ligne $k-1$ et un pour la ligne k avec k de 1 à i.

Malheureusement, en ne conservant pas en mémoire les valeurs de F pour toutes les paires (i,j), l'algorithme `SOL1` n'est plus utilisable. On s'intéresse à présent à un autre algorithme `SOL2` basé sur le principe Diviser Pour Régner.

Les questions 3.3 à 3.5 et 4.7 à 4.9 seront notées sur 4 points.

On rappelle qu'à la question 2.2, on a montré que $F(i,j) = g(i,j)$ qui est la longueur du plus court chemin du sommet $(0,0)$ au sommet (i,j) dans le graphe G_{XY}. On note à

©2016-2017 by UPMC/Licence Informatique

présent $h(i,j)$ la longueur du plus court chemin du sommet (i,j) au sommet (m,n) dans le graphe G_{XY}.

Question 3.3
De façon analogue au calcul de $F(i,j)$ par `COUT2`, proposer un algorithme `COUT2BIS` pour calculer la valeur $h(i,j)$ de même complexité spatiale et temporelle que `COUT2`.

Question 3.4
Montrer que la longueur d'un plus court chemin du sommet $(0,0)$ au sommet (m,n) passant par le sommet (i,j) dans le graphe G_{XY} est égale à $g(i,j)+h(i,j)$. En déduire une méthode pour calculer la valeur de ce plus court chemin passant par (i,j).

Le principe de l'algorithme qui calcule un alignement optimal repose sur la propriété suivante (on ne vous demande pas de la prouver).

Théorème 3.1. *Etant donné un indice $j \in \{1,\ldots,n\}$, considérons l'indice i^* qui minimise la valeur $g(i,j)+h(i,j)$ pour $i = 0,\cdots,m$.*
Il existe alors un plus court chemin du sommet $(0,0)$ au sommet (m,n) passant par le sommet (i^*,j) dans le graphe G_{XY}.

On peut choisir $j = \lceil \frac{n}{2} \rceil$, déterminer alors l'indice i^* correspondant à cet indice j peut être fait en calculant $g(i, \lceil \frac{n}{2} \rceil) + h(i, \lceil \frac{n}{2} \rceil)$ pour chaque valeur de $i \in \{1,\ldots m\}$. Par le théorème précédent, il existe un plus court chemin du sommet $(0,0)$ au sommet (m,n) passant par le sommet $(i^*, \lceil \frac{n}{2} \rceil)$ dans le graphe G_{XY}.

La remarque précédente permet d'établir un algorithme Diviser Pour Régner qui peut être décrit comme suit :

Étape 1 : Déterminer l'indice i^* pour la valeur $j = \lceil \frac{n}{2} \rceil$.

Étape 2 : Diviser le problème en deux sous problèmes :
a) pour les séquences $(x_1,\cdots,x_{i^*})$ et $(y_1,\cdots,y_{\lceil \frac{n}{2} \rceil})$
b) pour les séquences $(x_{i^*},\cdots,x_m)$ et $(y_{\lceil \frac{n}{2} \rceil},\cdots,y_n)$

Étape 3 : Résoudre récursivement ces deux sous-problèmes.

En pratique, cela correspond à la fonction récursive `SOL2`(k_1,l_1,k_2,l_2) fournie en annexe du document.

Question 3.5
Expliquez pourquoi la complexité spatiale de l'appel à `SOL2`$(0,0,m,n)$ est en $O(n+m)$. Vous préciserez en particulier l'algorithme utilisé pour le calcul de l'étape 1.

©2016-2017 by UPMC/Licence Informatique

4 Mise en œuvre

Dans cette section, nous allons détailler la mise en œuvre des algorithmes proposés dans les parties 1 et 3.

Instances

Nous proposons sur le site une série d'instances pour le problème. Le format texte est très simple, on précise les tailles des deux séquences puis la liste des nucléotides représentées par leurs lettres et séparées par un espace.

```
m                Taille de la séquence X
n                Taille de la séquence Y
A C T G C ...
C G A T T ...
```

Vous pouvez noter que les instances ont des tailles n et m de valeurs assez proches. De plus, elles sont rangées en tailles m croissantes.

Question 4.1
Implémenter la lecture de ces instances. Pour les tailles raisonnables, implémenter un affichage des instances.

Implémentation et Analyse de complexité expérimentale

Question 4.2
Implémenter les algorithmes `COUT1` et `SOL1` de la partie 1. En sortie de `SOL1`, la fonction doit fournir, outre l'alignement, une liste correspondant à l'intégralité du parcours inverse de la matrice F.

Question 4.3
A partir de cette liste, donner une fonction d'affichage de l'alignement optimal obtenu.

On peut noter que comme n est une valeur proche de m pour les instances proposées, on peut se limiter à étudier les complexités vis-à-vis de m. Pour mesurer le temps d'exécution d'un algorithme, il faudra pour chaque valeur de m, si les temps obtenus sont significatifs, tracer des courbes de temps d'exécution en fonction de m. Il s'agira de tracer des fichiers de points qui seront utilisés pour générer des courbes à l'aide d'un tableur ou d'outil graphique tel que `xgraphic` ou `gnuplot` (une documentation est en ligne sur le site du module). Vous pourrez ainsi analyser de façon critique la valeur expérimentale obtenue en fonction des complexités théoriques et aussi comparer les performances des algorithmes.

Question 4.4
Vérifier si les complexités temporelles de `COUT1` et `SOL1` correspondent à la valeur théorique attendue.

Question 4.5
Tester jusqu'à quelle taille d'instance les méthodes de la partie 1 peut être utilisée sur

©2016-2017 by UPMC/Licence Informatique

votre ordinateur (précisez les caractéristiques mémoire de votre ordinateur).

Question 4.6
Implémenter l'algorithme `COUT2` et comparer ses solutions "valeurs" et son temps d'exécution par rapport à `COUT1`. Tester jusqu'à quelle taille d'instance l'algorithme `COUT2` peut résoudre une instance.

Question 4.7
Implémenter les algorithmes `COUT2BIS` et tester ses solutions "valeurs" par rapport à `COUT1` et `COUT2`.

Question 4.8
Implémenter l'algorithme `SOL2` de la partie 3. Comparer ses solutions avec celles de l'algorithme `SOL1`.

Question 4.9
Tester jusqu'à quelle taille d'instance, l'algorithme `SOL2` peut être utilisé.

5 Cahier des charges

- Pour la mise en œuvre algorithmique, vous êtes libres de choisir le langage de programmation que vous utiliserez. Seuls les aspects purement algorithmiques seront abordés avec vos enseignants.
- Un rapport devra être remis au plus tard **22 Novembre 2016 à 23h59** à votre chargé de TD. Ce rapport doit contenir l'analyse théorique (dans laquelle vous prendrez soin de justifier toutes vos réponses), ainsi que l'analyse expérimentale de la complexité des algorithmes. Vous veillerez à commenter les courbes que vous aurez obtenues. Le plan du rapport suivra le plan du sujet.
- Vous devez créer un répertoire ayant pour nom `nomBinome1_nomBinome2`, où `nomBinome1` et `nomBinome2` correspondent à vos noms de famille. Si vous êtes seul, donnez votre nom au répertoire. Ce répertoire contiendra votre rapport ainsi que les fichiers sources *commentés* de vos programmes. Compressez ce répertoire sous forme d'un fichier `tgz` (par `tar cvzf nomRepertoire.tgz nomRepertoire`). Envoyez ce fichier en pièce attachée d'un courrier électronique ayant pour sujet `3i003 MiniProjet`, et adressé à votre chargé de TD.
- Le projet se fait par binôme du même groupe.

6 Annexe : Détail de la fonction récursive SOL2

La fonction utilise deux lots de trois tableaux $X2a, Y2a, F2a$ et $X2b, Y2b, F2b$ lors des appels. Il est utile d'effectuer l'allocation de manière globale (variable globale en C, variable d'une classe objet en C++ ou Java) de ces tableaux une seule fois avant l'appel à `SOL2`. Cette allocation est comme suit :
$X2a$: `tableau de caractères de taille` 3
$Y2a$: `tableau de caractères de taille` $n+1$
$F2a$: `matrice de caractères de taille` $3 \times (n+1)$

©2016-2017 by UPMC/Licence Informatique

```
X2b: tableau de caractères de taille m+1
Y2b: tableau de caractères de taille 3
F2b: matrice de caractères de taille (m+1) × 3

Fonction SOL2(k1, l1, k2, l2, L)
    Si k2 − k1 > 0 ou l2 − l1 > 0 Alors
        Si k2 − k1 <= 2 Alors
            Pour i de k1 à k2 Faire X2a[i − k1] ← X[i]
            Pour j de l1 à l2 Faire Y2a[j − l1] ← Y[j]
            Utiliser le code de la partie 1 pour X2a et Y2a
              en utilisant la matrice F2a
            Concaténer le chemin obtenu dans L.
            Retourner F2b[k2 − k1, l2 − l1]
        Sinon
            Si l2 − l1 <= 2 Alors
                Pour i de k1 à k2 Faire X2b[i − k1] ← X[i]
                Pour j de l1 à l2 Faire Y2b[j − l1] ← Y[j]
                Utiliser le code de la partie 1 pour X2b et Y2b
                  en utilisant la matrice F2b
                Concaténer le chemin obtenu dans L.
                Retourner F2b[k2 − k1, l2 − l1]
            Sinon
                j ← l1 + ⌈(l2 − l1)/2⌉
                i* ← k1
                valmin ← COUT2(k1, j)+COUT2BIS(k1, j)
                Pour i de k1 + 1 à k2 Faire
                    val=COUT2(i, j)+COUT2BIS(i, j)
                    Si valmin > val Alors
                        valmin ← val
                        i* ← i
                    FinSi
                FinPour
                Retourner SOL2(k1, l1, i*, j, L) + SOL2(i*, j, k2, l2, L)
```

La fonction prend en paramètre une liste chaînée L de couples d'indices : cette liste va permettre la mémorisation des sommets (i^*, j) trouvés à chaque récursion : ces valeurs correspondent alors au plus court chemin du sommet $(0, 0)$ au sommet (m, n), à partir duquel on peut déterminer l'alignement optimal. On peut noter qu'il y a au plus $n + m$ sommets dans un tel plus court chemin. Elle peut ne pas être dans l'ordre du chemin parcouru suivant la façon dont vous opérez les appels récursifs : ce n'est pas un problème car elle peut être triée a posteriori sur un tri utilisant hiérarchiquement le premier indice puis le second indice.

©2016-2017 by UPMC/Licence Informatique