Extrait DS de MP2I passé

Exercice 1 : Calcul d'alignements et de distance d'édition

Ce sujet porte sur un problème de génomique : l'alignement de séquences. D'un point de vue biologique, il s'agit de mesurer la similarité entre deux séquences d'ADN, que l'on voit simplement comme des suites de nucléotides. Cela permet, lorsqu'un nouveau génome est séquencé, de comparer ses gènes à ceux de génomes précédemment séquencés, et de repérer ainsi des ressemblances dues au fait que les deux espèces ont un ancêtre commun qui leur a transmis ce gène, même si ce gène a pu subir des mutations (évolutions) au cours du temps.

D'un point de vue informatique, les séquences de nucléotides sont vues comme des mots sur l'alphabet $\{\mathsf{A}, \mathsf{T}, \mathsf{G}, \mathsf{C}\}$ et l'on est ramené à deux problèmes d'algorithmique du texte : le calcul de la distance d'édition entre deux mots quelconques et la production d'un alignement réalisant cette distance. Pour chacun de ces problèmes, on s'intéresse d'abord à un algorithme naïf, puis à un algorithme de programmation dynamique. Enfin, on utilise la méthode diviser pour régner pour améliorer la complexité spatiale de ces algorithmes.

1. Le problème d'alignement de séquences

Dans cette partie on introduit d'abord des définitions générales en algorithmique du texte puis les notions d'alignement et de distance d'édition qui sont centrales pour ce projet.

Rappel. Alphabet, longueur, concaténation, mot vide. Un **alphabet** est un ensemble fini de caractères. Soit Σ un alphabet non vide. Un **mot** sur Σ est une suite finie de caractères de Σ. On notera Σ^* l'ensemble des mots sur Σ. Pour $x \in \Sigma^*$, la **longueur** du mot x, notée $|x|$, est la longueur de la suite. Si $|x| = n$, les caractères de x seront indexés par $[\![1, n]\!]$, le mot x s'écrit alors $x_1 x_2 \ \ldots \ x_n$. Soient x et y deux mots sur Σ de longueurs respectives n et m. La **concaténation** des mots x et y, notée $x \cdot y$ et y est le mot $x_1\, x_2 \ \ldots \ x_n\, y_1\, y_2 \ \ldots \ y_m$. Quel que soit l'alphabet, il existe un seul mot de longueur nulle appelé **le mot vide** et noté ε. Lorsque x est un mot de longueur n et $0 \leqslant i \leqslant n$, on note $x_{[\![1,i]\!]}$ le mot $x_1\ x_2\ \ldots\ x_i$. Aussi $x_{[\![1,0]\!]} = \varepsilon$, $x_{[\![1,1]\!]} = x_1$ et $x_{[\![1,n]\!]} = x$.

Définitions : Σ, $\overline{\Sigma}$, π. Dans le reste de cet exercice, on suppose que Σ ne contient pas le symbole spécial $-$, que l'on appellera un **gap**. On note alors $\overline{\Sigma}$ l'ensemble $\Sigma \cup \{-\}$. On définit de plus la fonction π de $\overline{\Sigma}^*$ dans Σ^* telle que $\pi(x)$ est le mot obtenu en supprimant tous les $-$ de x. Aussi si $\Sigma = \{a, b\}$, on a : $aababb \in \Sigma^*$ et $a-ba--a \in \overline{\Sigma}^*$ et $\pi(a-ba--a) = abaa$.

Alignement de deux mots. Soit $(x, y) \in \Sigma^* \times \Sigma^*$ deux mots, on dit alors que deux mots $(\bar{x}, \bar{y}) \in \overline{\Sigma}^* \times \overline{\Sigma}^*$ sont un **alignement** de x et y si :

$i)$ $\pi(\bar{x}) = x$;
$ii)$ $\pi(\bar{y}) = y$;
$iii)$ $|\bar{x}| = |\bar{y}|$;
$iv)$ $\forall i \in [\![1, |\bar{x}|]\!], \bar{x}_i \neq -$ ou $\bar{y}_i \neq -$.

En français, $(\bar{x}, \bar{y})$ est un alignement de (x, y) si et seulement si on obtient x (respectivement y) en supprimant tous les gaps dans $\bar{x}$ (respectivement dans $\bar{y}$), si $\bar{x}$ et $\bar{y}$ sont de même longueur et s'ils ne présentent pas de gap à la même position.

Pour $\Sigma = \{\mathsf{A}, \mathsf{T}, \mathsf{G}, \mathsf{C}\}$, $x = \mathsf{ATTGTA}$ et $y = \mathsf{ATCTTA}$, voici plusieurs alignements de (x, y) :

$\bar{x} : \mathsf{ATTGTA}$	$\bar{x} : \mathsf{AT-TGTA}$	$\bar{x} : \mathsf{ATTGT-A}$	$\bar{x} : \mathsf{------ATTGTA}$
$\bar{y} : \mathsf{ATCTTA}$	$\bar{y} : \mathsf{ATCT-TA}$	$\bar{y} : \mathsf{AT-CTTA}$	$\bar{y} : \mathsf{ATCTTA------}$

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

Q. 1 Si $x \in \Sigma^*$ est un mot de longueur n et $y \in \Sigma^*$ est un mot de longueur m, quelle est la longueur maximale d'un alignement de (x, y) ? **On n'attend pas une preuve du résultat, seulement une borne.**

Q. 2 Étant donné $\bar{x} \in \overline{\Sigma}^*$ de longueur $n+k$ ayant exactement k gaps et un mot $y \in \Sigma^*$ de longueur $m \in [\![k, n+k]\!]$, combien y a-t-il de mots $\bar{y}$ obtenus en ajoutant à y des gaps jusqu'à atteindre la longueur de $\bar{x}$ et en prenant soin que ceux-ci ne soient pas aux mêmes positions que les gaps dans $\bar{x}$? Autrement dit combien y a-t-il de mots $\bar{y} \in \overline{\Sigma}^*$ tels que $|\bar{y}| = |\bar{x}|$, $\pi(\bar{y}) = y$ et $\forall i \in [\![1, |\bar{x}|]\!], \bar{y}_i \neq -$ ou $\bar{x}_i \neq -$?

Q. 3 En déduire le nombre d'alignements possibles d'un couple de mots (x, y) de longueurs respectives n et m, en supposant que $n \geqslant m$. On ne demande pas de simplifier l'expression obtenue.

2. Mise en équation récursive du/des problème(s)

On s'efforce ici de briser récursivement le problème en sous-problèmes. Étant donné deux mots x et y on définit donc :

$$\mathsf{Align}(x, y) = \Big\{(\bar{x}, \bar{y}) \mid (\bar{x}, \bar{y}) \text{ est un alignement de } (x, y)\Big\}$$

En raisonnant sur les dernières lettres des alignements on obtient la relation suivante.

$$\begin{aligned}\mathsf{Align}(x, y) = &\{(\bar{x}, \bar{y}) \mid (\bar{x}, \bar{y}) \text{ est un alignement de } (x, y) \text{ et } \bar{x}_{|\bar{x}|} = -\} \\ &\cup \{(\bar{x}, \bar{y}) \mid (\bar{x}, \bar{y}) \text{ est un alignement de } (x, y) \text{ et } \bar{y}_{|\bar{y}|} = -\} \\ &\cup \{(\bar{x}, \bar{y}) \mid (\bar{x}, \bar{y}) \text{ est un alignement de } (x, y) \text{ et } \bar{x}_{|\bar{x}|} \neq - \text{ et } \bar{y}_{|\bar{y}|} \neq -\}\end{aligned}$$

Lorsque x est un mot non vide de Σ^*, on note $\tilde{x}$ le sous mot $x_{[\![1,|x|-1]\!]}$.

Q. 4 Démontrer que pour tout couple de mots $(x, y) \in \Sigma^* \times \Sigma^*$ on a :

$$\{(\bar{x}, \bar{y}) \mid (\bar{x}, \bar{y}) \in \mathsf{Align}(x, y) \text{ et } \bar{x}_{|\bar{x}|} = -\} = \{(\bar{u} \cdot -, \bar{v} \cdot y_{|y|}) \mid (\bar{u}, \bar{v}) \in \mathsf{Align}(x, \tilde{y})\}$$

On admettra de même que :

— $\{(\bar{x},\bar{y}) \mid (\bar{x},\bar{y}) \in \mathsf{Align}(x,y) \text{ et } \bar{y}_{|\bar{y}|} = -\} = \{(\bar{u}\cdot x_{|x|}, \bar{v}\cdot -) \mid (\bar{u},\bar{v}) \in \mathsf{Align}(\widetilde{x},y)\}$
— $\{(\bar{x},\bar{y}) \mid (\bar{x},\bar{y}) \in \mathsf{Align}(x,y) \text{ et } \bar{x}_{|\bar{x}|} \neq - \text{ et } \bar{y}_{|\bar{y}|} \neq -\} = \{(\bar{u}\cdot x_{|x|}, \bar{v}\cdot y_{|y|}) \mid (\bar{u},\bar{v}) \in \mathsf{Align}(\widetilde{x},\widetilde{y})\}$

Q. 5 En déduire **rigoureusement** que pour tout couple de mots $(x,y) \in \Sigma^* \times \Sigma^*$ on a :

$$d(x,y) = \begin{cases} 0 & \text{si } x = \varepsilon \text{ et } y = \varepsilon \quad \star \\ c_{\mathsf{i}} + d(x,\widetilde{y}) & \text{si } x = \varepsilon \text{ et } y \neq \varepsilon \\ c_{\mathsf{d}} + d(\widetilde{x},y) & \text{si } x \neq \varepsilon \text{ et } y = \varepsilon \\ \min(c_{\mathsf{i}} + d(x,\widetilde{y}), c_{\mathsf{d}} + d(\widetilde{x},y), c_{\mathsf{s}}(x_{|x|},y_{|y|}) + d(\widetilde{x},\widetilde{y})) & \text{sinon} \quad \star \end{cases}$$

On se contentera de démontrer les cas $\star$ et on donnera l'intuition des autres cas.

Q. 6 Supposons avoir implanté récursivement la fonction d en utilisant les équations de la question précédente. On note alors $c_{n,m}$ le nombre d'appels de fonction induit par de telles équations sur des mots d'entrées x et y de taille n et m (on admet que le nombre d'appels dépend uniquement de la taille des mots et non pas de leurs lettres).

a) Donner une définition récursive de $(c_{n,m})_{(n,m)\in\mathbb{N}^2}$.

b) Montrer que $\forall n \in \mathbb{N}, c_{n,n} \geqslant 3^n$.

Q. 7 Mettre en avant, informellement, les éléments clés justifiant de la pertinence d'un algorithme de programmation dynamique pour le calcul de $d(x,y)$.

3. Programmation dynamique en C

Pour les questions qui suivent, on considère fixé $(x,y) \in \Sigma^* \times \Sigma^*$ un couple de mots de longueurs respectives n et m, et $(\bar{x},\bar{y})$ un alignement de (x,y) de longueur l. De plus, pour $i \in [\![0,n]\!]$ et $j \in [\![0,m]\!]$, on introduit la notation suivante :

$$\mathsf{D}(i,j) = d(x_{[\![1,i]\!]}, y_{[\![1,j]\!]})$$

Q. 8 Donner la définition récursive de $\mathsf{D}(i,j)$ induite par les résultats des questions précédentes.

Structure matricielle en C. Afin d'implanter en C et par programmation dynamique le calcul de $\mathsf{D}(i,j)$, nous devons nous munir d'une structure de matrice. On définit donc le type suivant :

```
23 struct matrix_c {
24   int  largeur;
25   int  hauteur;
26   int* contenu;
27 };
28 typedef struct matrix_c* matrix;
```

Dans une telle structure, le champ `largeur` représente le nombre de colonnes, le champ `hauteur` représente le nombre de lignes et le champ `contenu`, qui contient les lignes de la matrice mises bout à bout, enregistre les coefficients. Aussi si l'on représente une matrice $(M_{i,j})_{i\in[\![0,h-1]\!],j\in[\![0,l-1]\!]}$ au moyen d'une telle structure, `hauteur` vaut h, `largeur` vaut l, le tableau d'entiers `contenu` est de longueur `hauteur`$\times$`largeur` et ses l premières cases contiennent (dans cet ordre) $M_{0,0}$, $M_{0,1}$, …, $M_{0,l-1}$, les l cases suivantes contiennent (dans cet ordre) $M_{1,0}$, $M_{1,1}$, …, $M_{1,l-1}$, ….

Q. 9 Définir les trois fonctions :

- `matrix matrix_alloc(int h, int l, int init)`, permettant l'allocation d'une matrice de hauteur `h`, de largeur `l` et dont tout les coefficients sont initialisés à `init` ;
- `int matrix_get(matrix m, int i, int j)`, renvoyant le coefficient d'indice `i`, `j` de `m` ;
- `void matrix_set(matrix m, int i, int j, int val)`, permettant de modifier le coefficient d'indices `i`, `j` de la matrice `m` à la valeur `val`.

Q. 10 Proposer une fonction C, `int* dist(char* x, char* y)` prenant en arguments deux chaînes de caractères `x` et `y`, et retournant la matrice des $(\mathsf{D}(i,j))_{(i,j)\in[\![0,n]\!]\times[\![0,m]\!]}$ où n est la longueur de `x` et m est la longueur de `y`. On pourra supposer définies :

- deux constantes `int c_i` et `int c_d` représentant respectivement c_{i} et c_{d} ;
- une fonction `int c_s (char a, char b)` permettant le calcul de $c_{\mathsf{s}}(\mathsf{a},\mathsf{b})$;
- une fonction `int min3(int a, int b, int c);` calculant le minimum de 3 valeurs entières.

On rappelle que la fonction `int strlen(char* s)` permet de calculer la longueur de la chaîne de caractères `s`.

Q. 11 Donner un ordre de grandeur, au moyen d'un $\mathcal{O}$ le plus précis possible, du coût en espace et en temps de la fonction `dist`.

Dans le reste de cet exercice les implantations ne sont plus demandées en C, on vous demandera uniquement de fournir des algorithmes en pseudo-code de bonne qualité.

Q. 12 Proposer un algorithme en pseudo-code, prenant en arguments deux mots x et y ainsi que la matrice $(\mathsf{D}(i,j))_{(i,j)\in[\![0,n]\!]\times[\![0,m]\!]}$ (celle calculée par `dist`) qui calcule un alignement optimal $(\bar{x},\bar{y})$ de (x,y).

4. Amélioration spatiale

Q. 13 Proposer un algorithme prenant en arguments deux chaînes de caractères `x` et `y`, et calculant la distance d'édition de `x` à `y`, votre algorithme devra avoir une complexité temporelle de l'ordre de celle `dist`, mais ayant une complexité spatiale de l'ordre de $\Theta(m)$ où m est la taille de la chaîne de caractères `y`.

Nous avons donc amélioré la complexité spatiale du calcul de la distance d'édition, toutefois cette amélioration a un coût : il n'est plus possible de reconstruire l'alignement optimal, seulement d'obtenir sa valeur. Dans les sections suivantes on essaie de retrouver le meilleur des deux mondes : un calcul de l'alignement optimal avec une complexité mémoire en $\Theta(m)$ et une complexité temporelle en $\Theta(nm)$.

5. Une approche "diviser-pour-régner"

On va utiliser ici la méthode diviser-pour-régner. Grosso modo, on va décomposer respectivement x en $x^1\cdot x^2$ et y en $y^1\cdot y^2$, puis calculer récursivement $(\bar{s},\bar{t})$ un alignement optimal de (x^1,y^1) et $(\bar{u},\bar{v})$ un alignement optimal de (x^2,y^2), et enfin les concaténer, c'est-à-dire qu'on retournera $(\bar{s}\cdot\bar{u},\bar{t}\cdot\bar{v})$, On ne procédera à ce découpage de x et y que lorsque $|x|\geqslant 2$ et $|y|\geqslant 1$, les autres cas seront à traiter directement, sans appels récursifs.

Q. 14 Montrer qu'il existe $(\overline{s},\overline{t})$ un alignement optimal de (x^1,y^1) et $(\overline{u},\overline{v})$ un alignement optimal de (x^2,y^2) tels que $(\overline{s}\cdot\overline{u},\overline{t}\cdot\overline{v})$ n'est pas un alignement optimal de $(x^1\cdot x^2, y^1\cdot y^2)$.

La réponse précédente justifie qu'on ne peut se contenter d'une approche "diviser-pour-régner" classique, où l'on couperait les deux chaînes en leur milieu.

Coupure. Soient x et y deux mots sur Σ de longueurs respectives $n \geqslant 2$ et $m \geqslant 1$. Soit $i \in [\![1,n]\!]$ et $j \in [\![1,m]\!]$ on dit que j est une **coupure** de (x,y) associée à i si et seulement s'il existe $(\overline{s},\overline{t})$ un alignement de $(x_{[\![1,i]\!]}, y_{[\![1,j]\!]})$ et $(\overline{u},\overline{v})$ un alignement de $(x_{[\![i+1,n]\!]}, y_{[\![j+1,m]\!]})$ tels que $(\overline{s}\cdot\overline{u},\overline{t}\cdot\overline{v})$ est un alignement minimal de (x,y).

On suppose fournie, et utilisable dans vos algorithmes une fonction coupure qui à partir d'un couple de mots (x,y) de longueurs respectives $n \geqslant 2$ et $m \geqslant 1$, renvoie un indice j^* qui est une coupure de (x,y) associée à $i^* = \left\lfloor \frac{|x|}{2} \right\rfloor$. On suppose de plus que cette fonction a une complexité algorithme pire cas majorée par $K_0\,nm$ (pour une certaine constante $K_0 \in \mathbb{R}^{+,*}$) sur deux entrées x et y de tailles respectives n et m.

On se concentre dans cette partie sur le cœur récursif de l'algorithme d'alignement par diviser-pour-régner, on suppose donc connue et utilisable une fonction base telle que base(x,y) calcule un alignement optimal de deux mots x et y lorsque $|x| \leqslant 1$ en une complexité majorée par $K_1\,|y|$.

Q. 15 Définir alors un algorithme alignementDpr calculant un alignement optimal de (x,y) par une approche "diviser-pour-régner".

Q. 16 Soit $C_{n,m}$ la suite donnant la complexité pire cas de l'algorithme alignementDpr sur des entrées de taille n et m.

- Montrer qu'il existe une constante K telle que $\forall p \in \mathbb{N}, \forall m \in NN, C_{2^p,m} \leqslant K 2^{p+1} m$.
- En admettant la croissance de la suite $(C_{n,m})_{(n,m)\in\mathbb{N}^2}$, donner une majoration de $(C_{n,m})_{(n,m)\in\mathbb{N}^2}$ au moyen d'un $\mathcal{O}$.

On remarque que la complexité spatiale de alignementDpr est essentiellement due à celle de coupure. Dans la fin de cet exercice on s'intéresse donc à une définition de la fonction coupure de complexité spatiale plus faible que $\Theta(nm)$, et de complexité temporelle en $\mathcal{O}(nm)$.

6. Le problème de la coupure

Pour $i \in [\![\frac{|x|}{2}, n]\!]$ et pour $j \in [\![0,m]\!]$, on définit $\mathcal{C}(i,j)$ comme suit.

$$\mathcal{C}(i,j) = \left\{ k \in [\![0,m]\!] \;\middle|\; k \text{ est une coupure de } (x_{[\![1,i]\!]}, y_{[\![1,j]\!]}) \text{ associée à } \left\lfloor \frac{|x|}{2} \right\rfloor \right\}$$

On cherche donc à calculer un élément de $\mathcal{C}(n,m)$. On remarque que $\forall j \in [\![1,m]\!],\ j \in \mathcal{C}(\lfloor \frac{|x|}{2} \rfloor, j)$.

Q. 17 Soit $i \in [\![\frac{|x|}{2}+1, n]\!]$. Soit $j \in [\![0,m]\!]$. Monter que si $j > 0$ et $\mathsf{D}(i,j) = \mathsf{D}(i,j-1) + c_{\mathsf{i}}$, alors $\mathcal{C}(i,j-1) \subseteq \mathcal{C}(i,j)$

On a en fait de même les deux affirmations suivantes.

- Si $\mathsf{D}(i,j) = \mathsf{D}(i-1,j) + c_{\mathsf{d}}$, alors $\mathcal{C}(i-1,j) \subseteq \mathcal{C}(i,j)$

- Si $\mathsf{D}(i,j) = \mathsf{D}(i-1,j-1) + c_{\mathsf{s}}(x_i, y_j)$ alors $\mathcal{C}(i-1,j-1) \subseteq \mathcal{C}(i,j)$

Q. 18 Conclure, en expliquant comment calculer $\mathsf{coupure}(x,y)$ avec une complexité temporelle pire cas en $\Theta(nm)$ et une complexité mémoire de l'ordre de $\Theta(m)$.