Sorbonne Université

Projet de `LU3IN003`

Danaël CARBONNEAU

Alignement de séquences

Semestre 5, Automne 2022

Contents

1

Le problème d'alignement de séquences

Exercice 1.1 Alignement de deux mots

Q.1.1.1

On cherche à montrer que si $(\overline{x}, \overline{y})$ et $(\overline{u}, \overline{v})$ sont des alignements de (x, y) et (u, v), alors $(\overline{x} \cdot \overline{u}, \overline{y} \cdot \overline{v})$ est un alignement de $(x \cdot u, y \cdot v)$.

Il va nous falloir pour cela montrer les quatre propriétés définissant un alignement :

1. $\pi(\overline{x} \cdot \overline{u}) = x \cdot u$
2. $\pi(\overline{y} \cdot \overline{v}) = x \cdot u$
3. $\mid \overline{x} \cdot \overline{u} \mid = \mid \overline{y} \cdot \overline{v} \mid$
4. $\forall i \in [1, ..., \mid \overline{x} \cdot \overline{u} \mid], x_i \neq - \text{ ou } y_i \neq -$

Montrons que $\pi(\overline{x} \cdot \overline{u}) = x \cdot u$

$\pi(\overline{x} \cdot \overline{u})$ est le sous mot obtenu en retirant tous les $-$ de $\overline{x} \cdot \overline{u}$, il contient donc tous les caractères différents de $-$ dans leur ordre d'apparition dans x, soit x car $\pi(\overline{x}) = x$, puis tous les caractères différents de $-$ dans leur ordre d'apparition dans u, soit u car $\pi(\overline{x}) = x$, par définition de la fonction pi, on retrouve alors la chaîne concaténée $x \cdot u$, et on a bien $\pi(\overline{x} \cdot \overline{u}) = x \cdot u$

Montrons que $\pi(\overline{y} \cdot \overline{v}) = y \cdot v$

De même, $\pi(\overline{y} \cdot \overline{v})$ est le sous mot obtenu en retirant tous les $-$ de $\overline{y} \cdot \overline{v}$, il contient donc tous les caractères différents de $-$ dans leur ordre d'apparition dans y, soit y car $\pi(\overline{y}) = y$, puis tous les caractères différents de $-$ dans leur ordre d'apparition dans v, soit v car $\pi(\overline{y}) = y$, par définition de la fonction pi, on retrouve alors la chaîne concaténée $y \cdot v$, et on a bien $\pi(\overline{y} \cdot \overline{v}) = y \cdot v$.

Montrons que $\mid \overline{x} \cdot \overline{u} \mid = \mid \overline{y} \cdot \overline{v} \mid$

$$\begin{aligned} \mid \overline{x} \cdot \overline{u} \mid &= \mid \overline{x} \mid + \mid \overline{u} \mid && \textit{car concaténé} && (1.1) \\ &= \mid \overline{y} \mid + \mid \overline{v} \mid && \textit{car alignements} && (1.2) \\ &= \mid \overline{y} \cdot \overline{v} \mid && \textit{Car concaténé} && (1.3) \end{aligned}$$

Montrons que $\forall i \in [1, ..., \mid \overline{x} \cdot \overline{u} \mid], x_i \neq -$ ***ou*** $y_i \neq -$

Prenons $i \in [1, ..., \mid \overline{x} \cdot \overline{u} \mid]$, soit

- $x_i \in \overline{x}$, et dans ce cas, $y_i \in \overline{y}$ car $\mid \overline{x} \mid = \mid \overline{y} \mid$. Puisque $(\overline{x}, \overline{y})$ est un alignement de (x, y), $x_i \neq -$ ou $y_i \neq -$.

- $x_i \in \overline{u}$, dans ce cas, $y_i \in \overline{v}$ car le dernier caractère de $\overline{y}$ se trouve au même indice que le dernier caractère de $\overline{x}$, puisque leurs normes sont identiques, un caractère de $\overline{u}$ ne peut donc avoir le même indice qu'un caractère de $\overline{y}$, d'où $y_i \in \overline{v}$. Du fait que que $(\overline{u}, \overline{v})$ est un alignement de (u, v), $x_i \neq -$ ou $y_i \neq -$.

Dans les deux cas, la propriété est vérifiée, du fait que les deux chaînes concaténées sont des alignements.

Les quatre propriétés de l'alignement sont donc vérifiées pour $(\overline{x} \cdot \overline{u}, \overline{y} \cdot \overline{v})$ un alignement de $(x \cdot u, y \cdot v)$. Ainsi, on a bien que si $(\overline{x}, \overline{y})$ et $(\overline{u}, \overline{v})$ sont des alignements de (x, y) et (u, v), alors $(\overline{x} \cdot \overline{u}, \overline{y} \cdot \overline{v})$ est un alignement de $(x \cdot u, y \cdot v)$.

Q.1.1.2

La longueur maximale d'un alignement de deux mots x et y de taille respectivement n et m est de $n+m$. Il est construit de tel sorte qu'à chaque caractère de x, le caractère aligné dans $\overline{y}$ soit un $-$, et inversement. Par exemple, pour $\Sigma = \{A, T, G, C\}$, $x = ATGTA$ et y = $GATTACA$, on peut faire l'alignement de longueur $n + m = 5 + 7 = 12$ suivant :

$$\overline{x} = A - T - G - T - A - --$$
$$\overline{y} = -G - A - T - T - ACA$$

Montrons que c'est la longueur maximale par l'absurde. Supposons qu'il existe un alignement de $x, y \in \Sigma*$ de longueur respectivement n et m tels que $\mid \overline{x} \mid = \mid \overline{y} \mid = n + m + 1$.

Il y aura donc, dans $\overline{x}$, $n + m + 1$ éléments, dont n qui seront les caractères de x, et $m + 1$ qui seront des $-$. De même, dans $\overline{y}$, il y aura $n + m + 1$ éléments dont m caractères de y et $n + 1$ qui seront des $-$.

Or, si on construit cet alignement, il nous faudrait $m+1$ caractères dans $\overline{y}$ pour que tous les $-$ de $\overline{x}$ soit alignés avec un $y_i \neq -$. Ne pouvant le faire, il y aura donc au moins un $i \in [1, ..., \mid \overline{x} \mid]$ tel que $x_i = - = y_i$, ce qui ne respecte pas la quatrième propriété des alignements, il y a donc une contradiction. De ce fait, il ne peut y avoir d'alignements d'une longueur supérieure à $n + m$.

2

Algorithmes Pour l'alignement de séquences

Exercice 2.1 Méthode naïve par énumération

Q.2.1.1

Sur un mot de taille n, pour placer les k gaps, on va chercher à combiner les positions de k éléments sur un ensemble de $n+k$ positions. Ce qui correspond au coefficient binomial suivant:

$$\binom{n+k}{k} = C_{n+k}^{k} = \frac{(n+k)!}{k!(n+k-k)!}$$

Q.2.1.2

Pour deux mots x et y de longueur n et m avec $n \geq m$, la longueur maximale de $\overline{x}$ est $n+m$ *(comme vu en 1.1.2)*, avec k gaps qui ont donc une valeur strictement comprise entre 0 et m.

Pour k gaps ajoutés à x dans $\overline{x}$, il nous faut donc $n+k-m$ gaps dans $\overline{y}$ (Longueur du mot diminué nombre de caractères de y)

Pour obtenir le nombre possible d'alignements de (x, y), nous allons procéder en trois étapes :

- Énumérer le nombre de combinaisons de gaps possibles dans $\overline{y}$ pour un mot $\overline{x}$ avec k_x gaps
- Multiplier ce nombre de combinaisons par le nombre de mots $\overline{x}$ faisables avec k_x gaps
- Sommer ces combinaisons selon le nombre $k_x \in [0; m]$ *(plage de valeurs de k possible)*

Prenons un exemple avec $x = ACTG$, $y = AG$ et $k_x = 1$. On a alors

$$\begin{aligned} k_y &= n + k_x - m \\ &= 4 + 2 - 2 \\ &= 4 \end{aligned}$$

Soit le mot $\overline{x} = A - CTG$, pour trouver les insertions de gap dans y possibles, on peut représenter le problème par un tableau de 5 colonnes, où la première ligne représente $\overline{x}$ et les suivantes les différents mots $\overline{y}$ associés :

A	-	C	T	G
-		-	A	G
-		A	-	G
-		A	G	-
A		-	-	G
A		-	G	-
A		G	-	-

Les cases colorées correspondent à la position où un gap ne peut se trouver dans y. Par extension de ce cas de base, on voit que, chaque position occupée par un gap dans $\overline{x}$ ne pouvant être occupée par un gap dans $\overline{y}$, il s'agit d'une combinaison de k_y gaps parmi n positions possibles (celles correspondant à des caractères de x), c'est à dire :

$$\binom{n}{k_y} = \binom{n}{n + k_x - m}$$

On multiplie alors par le nombre de $\overline{x}$ possibles pour un k_x donné, ce qui nous donne le nombre de façons d'insérer des gaps dans y de manière à respecter les contraintes d'alignement pour k_x gaps mis dans $\overline{x}$ suivant :

$$\binom{n+k}{k_x} \times \binom{n}{n + k_x - m}$$

Le nombre d'alignements possibles de (x, y) se trouve alors en sommant le résultat précédent pour $k_x = i$ allant de 0 à m:

$$\sum_{i=0}^{m} \binom{n+i}{i} \times \binom{n}{n + i - m}$$

Pour réaliser l'application numérique sur les valeurs $\mid \overline{x} \mid = 15$ et $\mid \overline{y} \mid = 10$, nous allons utiliser un tableur (présent en fichier annexe) pour calculer les valeurs intermédiaires et leur somme. En résulte le tableau suivant :

i	$\binom{n+i}{i}$	$\binom{n}{n+i-m}$	$\binom{n+i}{i} \times \binom{n}{n+i-m}$
0	1	3003	3003
1	16	5005	80080
2	136	6435	875160
3	816	6435	5250960
4	3876	5005	19399380
5	15504	3003	46558512
6	54264	1365	74070360
7	170544	455	77597520
8	490314	105	51482970
9	1307504	15	19612560
10	3268760	1	3268760

On somme les éléments de la troisième colonne pour obtenir

$$\sum_{i=0}^{10} \binom{15+i}{i} \times \binom{15}{5+i} = 298199265$$

Q.2.1.3

La complexité temporelle d'une méthode naïve qui consisterait à énumérer toutes les combinaisons possibles puis calculer la distance d'édition en vue de trouver un alignement de coût minimal serait alors :

- La complexité temporelle de création d'un alignement qu'on nommera c_a
- Qu'on répéterait $\sum_{i=0}^{m} \binom{n+i}{i} \times \binom{n}{n+i-m}$ fois pour obtenir tous les alignements possibles.
- Puis on parcourrait tous les alignements pour trouver leur distance d'édition tout en stockant, au fur et à mesure, le coût minimal[1].

Le parcours seul des éléments énumérés se fait en $O(m \times (n+m)!)$, puisque le calcul du produit dans la somme faite sur m est un produit de coefficient binomiaux qui s'expriment à l'aide de factorielles[2]. De même pour la répétition de c_a pour obtenir tous les alignements. La complexité totale est alors en $O(c_a \times (n+m)! + (n+m)!)$. Ce qui est une complexité supérieure à du $O(e^n)$, c'est à dire pire qu'une complexité exponentielle. Il n'est donc pas réaliste de le penser utilisable.

Q.2.1.4

Pour ce qui est de la complexité spatiale, nous nous baserons sur une structure de type tableau de tableaux de n+m cases où chaque case correspond à un caractère de $(\overline{x}, \overline{y})$ et chaque ligne un alignement possible de $(\overline{x}, \overline{y})$.[3]

- Chaque case est composée de deux caractères (un octet chacun)
- Il y a au total $\sum_{i=0}^{m} \binom{n+i}{i} \times \binom{n}{n+i-m}$ lignes dans notre matrice de dimension 2

La taille en mémoire de notre tableau d'alignements est donc de

$$c_{s_{tableau}} = 2 * \sum_{i=0}^{m} \binom{n+i}{i} \times \binom{n}{n+i-m}$$

Il nous faut en plus en mémoire un couple de valeurs max où un champ correspond au coût d'alignement minimal, et l'autre à un pointeur vers la première case de la ligne correspondant à l'alignement de coût minimal[4]. Cette structure de couple prend alors l'espace en mémoire d'un `int` et d'une adresse.

On a alors une complexité spatiale totale[5] de

$$c_s = 8 + 2 * \sum_{i=0}^{m} \binom{n+i}{i} \times \binom{n}{n+i-m} octets$$

Donc $c_s \in O((n+m)!)$, ce qui est une complexité, là encore, très importante.

[1] Le fait de chercher un coût minimal ne change rien à la complexité temporelle car il suffit de mettre à jour la valeur max après une comparaison en temps constant lors du calcul des coûts

[2] $(n+m)!$ majore le produit de nos coefficients binomiaux

[3] La liste chaînée pour représenter un alignement n'est pas spécialement avantageuse malgré le fait que sa taille soit inférieure ou égale à $n+m$ car l'adresse de la case suivante augmente considérablement la taille de chaque cellule par rapport à un tableau.

[4] Considérant la taille du tableau, le coût d'accès à un élément à partir de son indice ne se fera pas forcément en $\Theta(1)$ car il ne l'est pas sur des tableaux de taille trop importante, garder un pointeur directement sur la case n'est pas non plus un "gâchis" d'espace car l'indice pourra vite ne plus être stocké sur un `int`, ni même sur un `long` pour des valeurs de n et m peu élevées

[5] On fait la supposition d'une architecture en 32bits

Exercice 2.2 Programmation dynamique

Calcul de la distance d'édition par programmation dynamique

Q.2.2.1

Si $\overline{u_l} = -$***, que vaut*** $\overline{v_l}$ ***?***

Puisqu'on ne peut avoir $\overline{u_l} = -$ et $\overline{v_l} = -$ (définition de l'alignement), et que $\overline{v_l}$ contient tous les éléments de $y_{[1...j]}$) sont dans $\overline{v_l}$ dans leur ordre d'apparition dans $y_{[1...j]}$), $\overline{v_l} = y_j$ la dernière lettre de $y_{[1...j]}$.

Si $\overline{v_l} = -$***, que vaut*** $\overline{u_l}$ ***?***

De même, puisqu'on ne peut avoir $\overline{u_l} = -$ et $\overline{v_l} = -$ (définition de l'alignement), et que $\overline{u_l}$ contient tous les éléments de $x_{[1...j]}$ sont dans $\overline{u_l}$ dans leur ordre d'apparition dans $x_{[1...i]}$, $\overline{u_l} = u_i$ la dernière lettre de $u_{[1...i]}$.

Si $\overline{u_l} \neq -$ ***et*** $\overline{v_l} \neq -$***, que valent*** $\overline{u_l}$ ***et*** $\overline{v_l}$ ***?***

$\overline{u_l}$ et $\overline{v_l}$ étant les deux derniers éléments de $(\overline{u}, \overline{v})$, et puisqu'un alignement contient toutes les lettres de $(x_{[1...i]}, y_{[1...j]})$, alors, nécessairement, $\overline{u_l} = x_i$ et $\overline{v_l} = y_j$.

Q.2.2.2

$$c(\overline{u}, \overline{v}) = c(\overline{u}_{[1...l-1]}, \overline{v}_{[1...l-1]}) + c(\overline{u_l}, \overline{v_l})$$

Avec

$$c(\overline{u_l}, \overline{v_l}) = \begin{cases} c_{ins} & \text{si } \overline{u}_l = - \\ c_{del} & \text{si } \overline{v}_l = - \\ c_{sub}(x_i, y_j) & \text{sinon} \end{cases}$$

Q.2.2.3

La distance d'édition sera la distance d'édition jusqu'à (i,j) se calcule récursivement à partir de la distance d'édition jusqu'aux indices $i-1$ et $j-1$, à laquelle on va ajouter le minimum entre les trois opérations réalisables sur les lettres de chaque mots courantes (x_i, y_j), c'est à dire soit une insertion, soit une suppression[6], soit une substitution entre x_i et y_j[7].

On obtient alors la formule suivante :

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

Q.2.2.4

D(0,0) signifie la distance d'édition entre deux sous mots vides (on n'a ni avancé avec les deux indices dans $x_{[1...i]}$ ou $y_{[1...j]}$). La distance d'édition est donc nulle :

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

[6]On sépare le coût d'insertion et de suppression pour rester généraux malgré le fait que les valeurs numériques données par l'énoncé soient identiques.

[7]Puisque i n'est pas nécessairement égal à j, on prend bien en compte le fait qu'on peut potentiellement décaler des lettres avec les insertions et suppressions précédentes

Q.2.2.5

Que vaut $D(0, j)$ ***pour*** $j \in [1...m]$ ***?***

Puisque $i = 0$, on a visité aucune lettre de x, d'autres termes, pour avoir un alignement partiel de longueur j, on n'a fait que des insertions dans y, représentées par un $-$ dans $\overline{x}_{[1...j]}$. Ce qui signifie que le coût d'édition est unique (aucun choix possible) et ainsi $D(0, j) = j \times c_{ins}$.

Que vaut $D(i, 0)$ ***pour*** $i \in [1...n]$ ***?***

Puisque $j = 0$, on a visité aucune lettre de y, d'autres termes, pour avoir un alignement partiel de longueur j, on n'a fait que des suppressions dans y, représentées par un $-$ dans $\overline{y}_{[1...i]}$. Ce qui signifie que le coût d'édition est unique (aucun choix possible) et ainsi $D(i, 0) = i \times c_{del}$.

Q.2.2.6

Algorithm 1 DIST_1

Require: x,y deux mots de taille n et m

```
1: Distances[n+1][m+1]
2: i, j
3: for i in range [0;n] do
4:    for j in range [0;m] do
5:       if i = 0 then
6:          if j = 0 then
7:             Distances[i][j] = 0
8:          else
9:             Distances[i][j] = j × c_ins
10:         end if
11:      else
12:         if j = 0 then
13:            Distances[i][j] = i × c_del
14:         else
15:            Distances[i][j] = Distances[i − 1, j − 1] + min(c_ins, c_del, c_sub(x_i, y_j))
16:         end if
17:      end if
18:   end for
19: end for
20: return Distances[n][m]
```

Q.2.2.7

Il faut stocker un tableau de taille $n \times m$[8] en mémoire, contenant des entiers, d'où une complexité spatiale en $\Theta(n \times m)$

Q.2.2.8

On itère sur deux boucles imbriquées, la première allant de 0 à n et la seconde allant de 0 à m (parcourt du tableau pour renseigner la valeur de chaque case). Les opérations à l'intérieur de la seconde boucle sont des comparaisons et des affectations, qui se font en temps constant. En résulte une complexité temporelle en $\Theta(n \times m)$.

[8] n la taille de x et m la taille de y