Rapport du Projet LU3IN003

ZHOU runlin 28717281 **MA peiran 28717249**

Introduction :

Le but de ee projet porte est 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.

D'un point de vue informatique, les séquences de nucléotides sont vues comme des mots sur **l'alphabet {A,T,G,C}**. On s'intéresse d'abord à un *algorithme naïf*, puis à un algorithme de *programmation dynamique*. On utilise la méthode diviser pour régner pour améliorer la complexité spatiale de ces algorithmes.

Nous présenterons notre réalisation de l'algorithme en plusieurs parties.

Le problème d'alignement de séquences (Ex 2) :

Question 1:

Soit $(\overline{xu}, \overline{yv})$ est un alignement de (xu, yv) ssi

> (i) $\pi(\overline{xu}) = xu$
> (ii) $\pi(\overline{yv}) = yv$
> (iii) $|\overline{xu}| = |\overline{yv}|$
> (iv) $\forall i \in [1, ..., |\overline{xu}|], \overline{xu}_i \neq - \ \ ou \ \ \overline{yv}_i \neq -$

Nous pouvons facilement déduire que :

- $\pi(\overline{xu}) = \pi(\overline{x}) * \pi(\overline{u}) = xu$
- $\pi(\overline{yv}) = \pi(\overline{y}) * \pi(\overline{v}) = yv$
- $|\overline{xu}| = |\overline{x}| + |\overline{u}|$, car $(\overline{x}, \overline{y})$ et $(\overline{u}, \overline{v})$ sont respectivement des alignements de (x, y) et (u, v), donc on a $|\overline{x}| + |\overline{u}| = |\overline{y}| + |\overline{v}| = |\overline{yv}|$
- Car $(\overline{x}, \overline{y})$ et $(\overline{u}, \overline{v})$ sont respectivement des alignements de (x, y) et (u, v), donc on a $\forall i \in [1, ..., |\overline{x}|], \overline{x}_i \neq - \ \ ou \ \ \overline{y}_i \neq -$ et $\forall i \in [1, ..., |\overline{u}|], \overline{u}_i \neq - \ \ ou \ \ \overline{v}_i \neq -$, on ajoute les deux conditions emsemble. Et on a $\forall i \in [1, ..., |\overline{xu}|], \overline{xu}_i \neq - \ \ ou \ \ \overline{yv}_i \neq -$

Donc, $(\overline{xu}, \overline{yv})$ est bien que un alignement de (xu, yv)

Question 2 :

Si $(\overline{x}, \overline{y})$ est un alignment de (x, y), on a $\forall i \in [1, ..., |\overline{x}|], \overline{x}_i \neq - \ \ ou \ \ \overline{y}_1 \neq -$

donc la longueur maximale est $|x| + |y|$

Algorithmes pour l'alignement de séquences (Ex 3) :

Programmation naive :

Question 3 :

Nous pouvons convertir le problème en sélectionnant k dans n + k positions comme '-', donc le nombre de mot est C_{n+k}^{n}

Question 4 :

Soit $k \in [0, m]$ (l'intervalle étant choisi de manière à ne pas dépasser le nombre maximum de gaps, déterminé à la question (2)).

On suppose que l'on ajoute k gaps à x pour former une chaîne $\overline{x}$ de longueur $n + k$. Alors les alignements $(\overline{x}, \overline{y})$ de (x, y) seront exactement les alignements construits en ajoutant $n + k - m$ gaps à y à des indices i tels que $\overline{x}_i \neq -$.
Les alignements sont donc en bijection avec les parties à $n + k - m$ éléments de $[1, n]$: il y en a C_{n+k-m}^{n} , en notant qu'on a bien $0 \leq n + k - m \leq n$.

Soit A_k l'ensemble des alignements produits en ajoutant k gaps à x, et A l'ensemble des alignements de (x, y). On a :

$$|A| = \sum_{i=0}^{m} |A_i| = \sum_{i=0}^{m} C_{n+i}^{i} * C_{n+i}^{n+i-m}$$

Pour $|x| = 15$ et $|y| = 10$, on calcule $298, 199, 265$ alignements.

Question 5 :

Soit la compelxité de énumérer un alignement de (x, y) est $O(1)$.

Le nombre de alignment de (x, y) est $\sum_{i=0}^{m} C_{n+i}^{i} * C_{n+i-m}^{n}$, avec $|x| = n$ et $|y| = m$.

$$|A| = \sum_{i=0}^{m} C_{n+i}^{i} * C_{n}^{n+i-m} = \sum_{i=0}^{m} \frac{(n+i)!}{n!i!} * \frac{(n+i)!}{m!(n+i-m)!}$$

Donc, la complexité est en $O(m * n!)$

Et pour déduire un alignement de coût minimal, on a besion de rechercher le nombre minimal dans la liste de coût des alignments, qui a la même complexité d'énumeration.

Question 6 :

Estimation de complexité spatiale :

Selon l'algorithme poser dans le sujet, on peut déduire que le nombre de programme à traiter au maxiume (dans le pire cas) est n+m. En même temp, il y a 6 variables stockées dans chaque inération de boucle. On suppose que la complexité en mémoire pour les 6 variables est en $O(k)$, donc en totale, la complexité spatiale est $O(k(n + m))$.

Évaluer la performance :

Selon la résultat qu'on obtenu, la taille d'instance au maximum est 12. (avec GCTAACTAACG et GCTAAACTACT utilise 80s)

```
(base) zhourunlin@zhoudemacbook-pro codes % python3 projet.py
Time taken for execution : 0.030066251754760742
distance de TATATGAGTC et TATTT : 10
Time taken for execution : 5.88909387588501
distance de TGGGTGCTAT et GGGGTTCTAT : 8
Time taken for execution : 2.28288197517395
distance de AACTGTCTTT et AACTGTTTT 2
Time taken for execution : 10.330514192581177
distance de CTGGAAAGTGCG et CTGAACTGG : 9
Time taken for execution : 80.32119083404541
distance de GCTTAACTAACG et GCTAAACTACT : 9
```

Ensuit, on exécute la commande *./python3 porjet.py &* (hangup), et la commande *top* pour voir la consommation mémoire (99.5% in the first line)

```
PID    COMMAND      %CPU  TIME     #TH   #WQ  #PORT MEM    PURG   CMPRS PGRP  PPID  STATE    BOOSTS          %CPU_ME %CPU_OTHRS UID
63042  python3.9    99.5  00:41.78 2/1   0    27    6785K  0B     0B    63042 63019 running  *0[1]           0.00000 0.00000    501
370    WindowServer 41.6  05:07:07 23    4    2775+ 1253M+ 12M+   305M  370   1     sleeping *0[1]           6.66574 1.03168    88
673    avconference 27.8  66:50.83 26    6    399   142M+  0B     14M   673   1     stuck    *0[1]           3.57124 15.55610   501
```

Programmation dynamique

Question 7 :

Comme $\pi(\overline{u}) = x$ et $\pi(\overline{v}) = y$, on a $\overline{u}_l \in \{-, x_i\}$ et $\overline{v}_l \in \{-, v_i\}$. En particulier,

- si $\overline{u}_l = -$, alors $\overline{v}_l = y_j$;
- si $\overline{v}_l = -$, alors $\overline{u}_l = x_i$ et
- si u_l et v_l sont différents de $-$ alors ils valent respectivement x_i et y_j .

Question 8 :

On note $u = (\overline{u}_i)_{i=1}^{l-1}$ et $v = (\overline{v}_i)_{i=1}^{l-1}$

- Si $\overline{u}_l = -$, alors $C(\overline{u}, \overline{v}) = c_{ins} + C(u, v)$.
- Si $\overline{v}_l = -$, alors $C(\overline{u}, \overline{v}) = c_{dél} + C(u, v)$.
- Sinon, $C(\overline{u}, \overline{v}) = c_{sub}(\overline{u}_l, \overline{v}_l) + C(u, v)$.

Question 9 :

Comme les distances sont positives, par croissance de la somme il suffit d'optimiser la distance de manière gloutonne en appliquant une transformation de coût minimal au dernier caractère. Plus précisément :

- Si $i = 0$ ou $j = 0$ alors...
- Sinon
  - Si $x_i = y_j$ alors $D(i, j) = D(i-1, j-1)$ (comme $c_{sub}(x_i, y_j) = 0$, le meilleur choix est fait en laissant les deux caractères).

- Sinon, $D(i,j) = min\{c_{dél} + D(i-1,j), c_{ins} + D(i,j-1), c_{sub}(x_i, y_j) + D(i-1,j-1)\}$.
  Comme on ne peut pas mettre de gap au même endroit, on a le choix
  entre consommer x_i (dans quel cas on place un gap à y, induisant le coût $c_{dél}$)
  et consommer y_j (dans quel cas on place un gap à x, induisant le coût c_ins) ou
  consommer x_i et y_j (cas traité à l'étape précédente).

Plus généralement, on a la relation de récurrence $C(i+1, j+1) = min\{c_{dél} + D(i-1,j), c_{ins} + D(i,j-1), c_{sub}(xi, yj) + D(i-1,j-1)\}$.

Question 10 :

On a $D(0,0) = d(\varepsilon, \varepsilon)$. Par la borne supérieure sur la longueur des alignements trouvée plus haut, le seul alignement de $(\varepsilon, \varepsilon)$ est vide, donc de coût nul : $D(0,0) = 0$.

Question 11 :

Soit $u \in \Sigma*$ de longueur j. Par un argument sur les longueurs, le seul alignement possible de (ε, u) (respectivement (u, ε)) est $(-, u)$ (respectivement $(u, -)$). On a ainsi $C(0,j) = jc_{ins}$ et $C(i,0) = ic_{dél}$.

Question 12 :

```
def DIST_1(X, Y):
    n ← |X|
    p ← |Y|
    if n=0 && p=0, then
        return 0
    end if

    if n = 0, then
        return p * C_ins
    end if

    if p = 0, then
        return n * C_dél
    end if
    D ← 0
```

[Partie 3.2]

Une extension : l'alignement local de séquences (Ex 4) :

Conclusion :