Mini-projet : Alignement de séquences
LU3IN003 - Sorbonne Université

PEREIRA GAMA Gustavo
EL BEBLAWY Rami

Question 1

Si $(\bar{x}, \bar{y})$ et $(\bar{u}, \bar{v})$ sont respectivement des alignements de (x, y) et (u, v) alors $(\bar{x}.\bar{u}, \bar{y}.\bar{v})$ est un alignement de $(x.u, y.v)$ car :

- d'après (i), $\pi(\bar{y}) = y$ et $\pi(\bar{v}) = v$ donc $\pi(\bar{y} \cdot \bar{v}) = y \cdot v$.

- et d'après (ii), $\pi(\bar{y}) = y$ et $\pi(\bar{v}) = v$ donc $\pi(\bar{y} \cdot \bar{v}) = y \cdot v$.

- si $(\bar{x}, \bar{y})$ et $(\bar{u}, \bar{v})$ sont des alignements, alors $|\bar{x}| = |\bar{y}|$ et $|\bar{u}| = |\bar{v}|$, donc $(\bar{x}.\bar{u}) = |\bar{x}| + |\bar{u}|$ et $(\bar{y}.\bar{v}) = |\bar{y}| + |\bar{v}|$

- il faut que (iv) soit respectée c'est-à-dire(définition) : * $\forall i \in [1..|\bar{x}|]$, $\bar{x_i} \neq$ — ou $\bar{y_i} \neq$ — et $\forall i \in [1..|\bar{u}|], \bar{u_i} \neq -$ ou $\bar{v_i} \neq$ —

Donc $\forall i \in [1..(|\bar{x}| + |\bar{u}|)]$, $((\overline{x \cdot u})_i \neq$ — ou $(\overline{y \cdot v})_i$

Question 2

La longueur maximale d'un alignement de (x, y) est $n + m$ car on a alors un gap en face de chaque lettre, pour chacun des mots. On ne peut avoir deux gaps face à face, on ne peut donc augmenter la longueur du mot. Exemple avec les mots A et AG de taille n = 1 et m = 2

$$\begin{matrix} A - - \\ -AG \end{matrix}$$

On a bien une longueur de $3 = n + m$.

Question 3

En ajoutant k gaps à x, on obtient un mot de taille $n + k$. On va placer k gaps parmis ces $n + k$ places. On va donc choisir k parmis $n + k$, le nombre de mots $\bar{x}$ obtenus est donc $\binom{n+k}{k}$.

Question 4

Une fois ajoutés k gaps à x pour obtenir $\bar{x}$, on a un mot de taille $n+k$. Afin de l'aligner avec y, il faut que $\bar{y}$ soit aussi de taille $n + k$. Or, y est de taille m. Il faut donc ajouter $n + k - m$ gaps à y.

Le nombre de mots possible pour $\bar{x}$ est $\binom{n+k}{k}$, et le nombre de mots possible pour $\bar{y}$ est $\binom{n+k}{n+k-m}$. Or, on doit enlever les mots où des gaps se superposent. Il y a donc k choix en moins pour placer les gaps de $\bar{y}$. On obtient donc, pour un k donné, $\binom{n+k}{k} * \binom{n}{n+k-m}$ combinaisons possibles.

On va ajouter à n de 0 à, au plus, m gaps (on obtient ainsi le mot de taille maximale $n + m$).

Le nombre d'alignements possible pour (x, y) est donc $\sum_{k=0}^{k=m} \binom{n+k}{k}\binom{n}{n+k-m}$

En utilisant un calculateur en ligne, on trouve 298 199 265 alignements possibles pour $|x| = 15$ et $|y| = 10$.

Question 5

On peut trouver l'alignement de coût minimal en même temps que l'énumération de tous les alignements possibles à l'aide d'une variable qui stocke le minimum obtenu à chaque fois, et l'alignement qui en est à l'origine ?

Afin de calculer la distance d'édition de deux mots, il faut calculer la distance entre chaque lettre. Il y a $n + k$ lettres, c'est donc une opération en $O(n + k) = O(n)$(car $n > m \geq k$). On va effectuer cette opération pour chaque combinaison possible, la complexité obtenu est donc $O(n * \sum_{k=0}^{k=m} \binom{n+k}{k}\binom{n}{n+k-m})$, soit $O(n!)$.

Question 6

Il n'est nécéssaire de garder en mémoire uniquement les 2 mots en cours d'analyse et le meilleur alignement trouvé jusqu'ici. On a donc une complexité spatiale en $O(n)$.

La version de python utilisée pour tout le projet est Python 3.7.4
La version de matplotlib utilisée est la 3.1.2

TACHE A

Le code de la tâche A se trouve dans le fichier A.py. Certaines fonctions sont volontairement redondantes dans plusieurs fichier afin de n'avoir aucune dépendance de fichier entre chaque tâche.
Les résultats des tests se trouvent dans le fichier *res_tests/res_tache_A*.
Vous pouvez réalisez le graphe correspondant à ces données en lançant la commande *python graphe.py res_tests/res_tache_A'*(il faut installer la librairie matplotlib si nécéssaire avec la commande 'pip install matplotlib'.
Vous pouvez également créer vos données correspondantes aux performances de votre machine en éxecutant le fichier 'mesures-py'.
On observe que, sur notre machine, on obtient un temps suppérieur à une minute pour une instance de taille 12.
La consommation mémoire pour ces instances n'a pas dépassé 0.1% (tableau récap dans le fichier *utilisation*).

Question 7

Si $\bar{u}_l = —$, alors $\bar{v}_l = y_j$, si $\bar{v}_l = —$, alors $\bar{u}_l = x_i$. Si $\bar{u}_l \neq —$ et $\bar{v}_l \neq —$, alors $\bar{u}_l = x_i$ et $\bar{v}_l = y_j$, car on ne peut changer l'ordre des lettres et on ne peut aligner deux gaps. Ainsi, la dernière lettre de $Al(i, j)$ est forcément, soit —, soit x_i (respectivement y_j).

Question 8

Si $\bar{u}_l = —$ ou $\bar{v}_l = —$, alors $C(\bar{u}, \bar{v}) = C(\bar{u}_{[1..l-1]}, \bar{v}_{[1..l-1]}) + c_{ins/del}$.
Sinon, $C(\bar{u}, \bar{v}) = C(\bar{u}_{[1..l-1]}, \bar{v}_{[1..l-1]}) + c_{sub}(x_i, y_j)$.

Question 9

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

Question 10

$D(0, 0) = 0$ car il n'y a pas de "coût initial" pour commencer l'alignement.

Question 11

Pour $j \in [1..m]$, on a $D(0, j) = c_{ins} * j$ car cela veut dire qu'il n'y aucune lettre de Σ dans $\bar{u}$, donc uniquement des insertions.

De même, pour $i \in [1..n]$, on a $D(i, 0) = c_{del} * i$ car cela veut dire qu'il n'y a aucune lettre de Σ dans $\bar{v}$, donc uniquement des suppressions.

Question 12

```
DIST_1(x,y):

    Pour i allant de 0 a n:

        Pour j allant de 0 a m :

            Si i = 0:
                Si j = 0:
                    D[0,0] ← 0
                Sinon:
                    D[0,j] ← j * c_ins

            Sinon:
                Si j = 0:
                    D[i,0] ← i * c_del

                Sinon :
                    c ← D[i-1, j-1] + c_sub(x_i, y_j)

                    Si D[i-1,j] + c_del < c:
                        c ← D[i-1, j] + c_del

                    Si D[i,j-1] + c_ins < c:
                        c ← D[i, j-1] + c_ins

                    D[i,j] ← c

        fin pour
    fin pour

    return D
```

Question 13

L'algorithme DIST_1 prend en mémoire un tableau imbriqué de taille n*m, sa complexité spatiale est donc de l'ordre $O(nm)$.

Question 14

On a deux boucles imbriquées allant jusqu'à n/m. Sa complexité temporelle est donc de l'ordre de $O(max(n, m)^2)$.

Question 15

Si on a $j > 0$ et $D(i, j) = D(i, j-1) + c_{ins}$, cela signifie que le meilleur coût de l'alignement de $(x_{[1..u]}, y_{[1..j]})$ est obtenu à partir de l'alignement $(x_{[1..u]}, y_{[1..j-1]})$ en ajoutant une insertion. Cela se traduit, donc, d'après l'énoncé, à rajouter un gap dans x, et se traduit mathématiquement par

$$\forall(\bar{s}, \bar{t}) \in Al * (i, j-1), (\bar{s} \cdot -, \bar{t} \cdot y_j) \in Al * (i, j)$$

On applique les même raisonnement pour les deux autres expressions, et on trouve ainsi

$$\forall(\bar{s}, \bar{t}) \in Al * (i-1, j), (\bar{s} \cdot x_i, \bar{t} \cdot -) \in Al * (i, j)$$

$$\forall(\bar{s}, \bar{t}) \in Al * (i-1, j-1), (\bar{s} \cdot x_i, \bar{t} \cdot y_j) \in Al * (i, j)$$

Question 16

```
SOL_1(x,y,D):

    (Les insertions dans x̄ et ȳ se font au debut)

    i ← |x|
    j ← |y|

    Tant que i > 0 et j > 0:
        Si D[i,j] = D[i−1,j−1] + c_sub(x_i, y_j):
            x̄ ← x_i
            ȳ ← y_j
            i ← i − 1
            j ← j − 1
        Sinon:
            Si D[i,j] = D[i,j−1] + c_ins:
                x̄ ← −
                ȳ ← y_j
                j ← j − 1
            Sinon:
                x̄ ← x_i
                ȳ ← −
                i ← i − 1
    fin tant que

    Tant que i > 0 :
        x̄ ← x_i
        ȳ ← −
        i ← i − 1
    fin tant que

    Tant que j > 0 :
        x̄ ← −
        ȳ ← y_j
        j ← j − 1
    fin tant que

    return (x̄, ȳ)
```

Question 17

SOL_1 est en $O(n+m)$ et DIST_1 en $O(max(n,m)^2)$, ainsi l'éxecution séquentielle de ces deux algorithmes a une complexité temporelle de $O(max(n,m)^2)$.

Question 18

Ces fonctions utilisent des chaines et tableaux de taille allant jusqu'à n+m et un tableau de taille n*m. La complexité spatiale totale est donc $O(nm)$.

TACHE B

Le code de la tâche B se trouve dans le fichier B.py. Certaines fonctions sont volontairement redondantes dans plusieurs fichier afin de n'avoir aucune dépendance de fichier entre chaque tâche.
Les résultats des tests se trouvent dans le fichier res_tests/res_tache_B1 et res_tests/res_tache_B2.
Vous pouvez également créer vos données correspondantes aux performances de votre machine en éxecutant le fichier 'mesures-py'.
On observe que, sur nos machine, on ne peut dépasser des tailles de 15000 et 10000 lettres, après quoi l'utilisation mémoire devient trop grande et l'OS tue notre programme
La consommation mémoire pour ces instances est allée de 0.1% à >100% (tableau récap dans le fichier *utilisation*).
En réalisant les graphes correspondants à ces données avec la commande *python graphe.py* res_tests/res_tache_B1, on observe une courbe polynomial, ce qui correspond à notre complexité théorique.

Question 19

Lorsqu'on remplit la ligne i du tableau, on n'a besoin de regarder uniquement 3 cases : D(i-1,j-1), D(i-1,j) et D(i,j-1) (cf l'algorithme). Ainsi, on n'utilise pas les lignes $D(i', j)_{(i'<i-1, j \in [1..m])}$

Question 20

```
DIST_2(x,y):
    Pour j allant de 0 a m:
        prec[j] ← j * c_ins
    fin pour

    Pour i allant de 1 a n:

        Pour j allant de 0 a m:

            Si j = 0:
                D[0] = i * c_del

            Sinon:
                c ← prec[j − 1] + c_sub(x_i, y_j)

                Si prec[j]+ c_del < c:
                    c ← prec[j] + c_del

                Si D[j−1] + c_ins < c:
                    c ← D[j − 1] + c_ins

                D[j] ← c
        fin pour
        prec ← D
    fin pour

    return D
```

TACHE C

Le code de la tâche C se trouve dans le fichier C.py. Certaines fonctions sont volontairement redondantes dans plusieurs fichier afin de n'avoir aucune dépendance de fichier entre chaque tâche.
Les résultats des tests se trouvent dans le fichier res_tests/res_tache_C1 et res_tests/res_tache_C2.
Vous pouvez également créer vos données correspondantes aux performances de votre machine en éxecutant le fichier 'mesures-py'.
La consommation mémoire pour la plus grosse instance était de 0.2% (tableau récap dans le fichier *utilisation*). Ceci est bien meilleur que DIST_1 qui utilisait trop de mémoire et rendait l'éxecution de notre programme impossible pour les trop grosses instances. Avec cette méthode on a cependant uniquement la distance d'édition, et non l'alignement optimal.
En réalisant les graphes correspondants à ces données avec la commande *python graphe.py* res_tests/res_tache_C1', on observe une courbe polynomial, ce qui correspond à notre complexité théorique.

Question 21

```
mots_gaps(k):

    M ← ''
    Pour i allant de 1 a k:
        M ← M +'-'
    fin pour

    return M
```

Question 22

```
align_lettre_mot(x,y):

    c ← ∞

    Pour j allant de 1 a |y|:
        Si c_sub(x_1, y_j) < c:
            c ← c_sub(x_1, y_j)
            i ← j
        fin pour

    x̄ ← mots_gaps(i-1) + x + mots_gaps(|y| - i)

    return (x̄, y)
```

Question 23

On a $(\bar{s}, \bar{t})$ l'alignement optimal de (x^1, y^1) suivant de coût 13 :

$$\begin{array}{ccc} B & A & L \\ R & O & - \end{array}$$

et$(\bar{u}, \bar{v})$ l'alignement optimal de (x^2, y^2) suivant de coût 9 :

$$\begin{array}{cccc} L & O & N & - \\ - & - & N & D \end{array}$$

Il en vient donc $(\bar{s} \cdot \bar{u}, \bar{t} \cdot \bar{v})$ un alignement de (x,y) de coût 21 :

$$\begin{array}{ccccccc} B & A & L & L & O & N & - \\ R & O & - & - & - & N & D \end{array}$$

Or, on trouve l'alignement suivant de (x,y) :

$$\begin{array}{ccccccc} B & A & L & L & O & N & - \\ - & - & - & R & O & N & D \end{array}$$

Cet alignement est de coût 17, donc $(\bar{s} \cdot \bar{u}, \bar{t} \cdot \bar{v})$ n'est pas un alignement optimal de (x,y).

Question 24

```
SOL_2(x,y):

    Si |y| = 0:
        y ← mots_gaps(|x|)
        return (x,y)

    Si |x| = 0:
        x ← mots_gaps(|y|)
        return (x,y)

    Si |x| = 1:
        return align_lettre_mot(x,y)

    i* ← |x| / 2
    j* ← coupure(x,y)

    (x1, y1) ← SOL_2(x_[1..i*], y_[1..j*])
    (x2, y2) ← SOL_2(x_[i*1..|x|], y_[j*+1..|y|])

    return(x1+x2,y1+y2)
```

Question 25

```
coupure(x,y):

    Pour j allant de 0 a m:
        prec[j] ← j * c_del
    fin pour

    stop ← i/2

    Pour i allant de 1 a stop:

        Pour j allant de 0 a |y|:

            Si j = 0:
                D[0] = i * c_del
                min_c ← i * c_ins
                indice_min ← j

            Sinon:
                c ← prec[j - 1] + c_sub(x_i, y_j)

                Si prec[j]+ c_del < c:
                    c ← prec[j] + c_del

                Si D[j-1] + c_ins < c:
                    c ← D[j - 1] + c_ins

                Si c <= min_c :
                    min_c ← c
                    indice_min ← j

                D[j] ← c
        fin pour
        prec ← D
    fin pour

    return indice_min
```

Question 26

Trois tableaux de taille n, donc O(n).

Question 27

Quatre tableaux de taille max n/m, donc O(max(n,m)). On suppose que n>= m, ainsi on note O(n) la complexité de SOL_2.

Question 28

$O(n^2)$ car il y a deux boucles imbriquées de taille max n.

TACHE D

Le code de la tâche D se trouve dans le fichier D.py. Certaines fonctions sont volontairement redondantes dans plusieurs fichier afin de n'avoir aucune dépendance de fichier entre chaque tâche.
Les résultats des tests se trouvent dans le fichier res_tests/res_tache_D1 et res_tests/res_tache_D2.
Vous pouvez également créer vos données correspondantes aux performances de votre machine en éxecutant le fichier 'mesures-py'.
La consommation mémoire pour la plus grosse instance était de 0.2% (tableau récap dans le fichier *utilisation*). On obtient un résultat identique à la tâche C car la méthode est très similaire.
En réalisant les graphes correspondants à ces données avec la commande *python graphe.py* res_tests/res_tache_D1, on observe une courbe polynomial, ce qui correspond à notre complexité théorique.

Question 29

En comparant expérimentalement nos résultats, on constate une complexité temporelle meilleure pour SOL_2 que pour SOL_1.