Sorbonne Université

Faculté des Sciences et Ingénierie

Rapport de projet LU3IN003

Fait par :

XIAO Carine - XU Audic

Olivier Spanjaard 2022 - 2023

Table des matières

1 Alignement de deux mots

1.1 Question 1

Soit $(\bar{x}, \bar{y})$ et $(\bar{u}, \bar{v})$ des alignements de (x, y) et (u, v) respectivement. Montrons que $(\bar{x} \cdot \bar{u}, \bar{y} \cdot \bar{v})$ est un alignement de $(x \cdot u, y \cdot v)$.

(i) Montrons que $\pi(\bar{x} \cdot \bar{u}) = x \cdot u$:

Par propriété de la concaténation, on a : $\pi(\bar{x} \cdot \bar{u}) = \pi(\bar{x}) \cdot \pi(\bar{u})$.

Et d'après les hypothèses sur $\bar{x}$ et $\bar{u}$: $\pi(\bar{x} \cdot \bar{u}) = x \cdot u$.

(ii) Montrons que $\pi(\bar{y} \cdot \bar{v}) = \bar{y} \cdot \bar{v}$:

De même par propriété de la concaténation, on a : $\pi(\bar{y} \cdot \bar{v}) = \pi(\bar{y}) \cdot \pi(\bar{v})$.

Et par hypothèses sur $\bar{y}$ et $\bar{v}$: $\pi(\bar{y} \cdot \bar{v}) = y \cdot v$.

(iii) Montrons que $|\bar{x} \cdot \bar{u}| = |\bar{y} \cdot \bar{v}|$:

Clair car $|\bar{x} \cdot \bar{u}| = |\bar{x}| + |\bar{u}| = |\bar{y}| + |\bar{v}| = |\bar{y} \cdot \bar{v}|$.

(iv) Montrons que $\forall i \in [\![1, |\bar{x}| + |\bar{u}|]\!], (\bar{x} \cdot \bar{v})_i \neq -$ ou $(\bar{y} \cdot \bar{v})_i \neq -$:

Soit $i \in [\![1, |\bar{x}| + |\bar{y}|]\!]$, on distingue 2 cas.

Si $i \leq |\bar{x}|$, alors $(\bar{x} \cdot \bar{u})_i = \bar{x}_i$ et $(\bar{y} \cdot \bar{v})_i = \bar{y}_i$.

Par (iv) des alignements $(\bar{x}, \bar{y})$ et $(\bar{u}, \bar{v})$, on obtient que $(\bar{x} \cdot \bar{v})_i \neq -$ ou $(\bar{y} \cdot \bar{v})_i \neq -$.

Sinon $i > |\bar{x}|$, en posant $i' = i - |\bar{x}| \in [\![1, |\bar{u}|]\!]$, on a : $(\bar{x} \cdot \bar{u})_i = \bar{u}_{i'}$ et $\bar{y} \cdot \bar{v} = \bar{v}_{i'}$.

Par (iv) des sous-alignements, on en conclut que : $(\bar{x} \cdot \bar{v})_i \neq -$ ou $(\bar{y} \cdot \bar{v})_i \neq -$.

D'où $(\bar{x} \cdot \bar{u}, \bar{y} \cdot \bar{v})$ est un alignement de $(x \cdot u, y \cdot v)$.

1.2 Question 2

Soit $x \in \Sigma^*$ de longueur $n \in \mathbb{N}$ et $y \in \Sigma^*$ de longueur $m \in \mathbb{N}$. Construisons un alignement $(\bar{x}, \bar{y})$ de la façon suivante. Pour $\bar{x}$, on concatène le mot x avec m gaps. Puis pour $\bar{y}$, on concatène n gaps avec le mot y. $(\bar{x}, \bar{y})$ vérifie $(i), (ii), (iii)$ et (iv) donc il s'agit bien d'un alignement de (x, y) de longueur $n + m$, qui est la longueur maximale. En effet, pour un alignement de longueur supérieure ou égale à $n + m + 1$, il existe $i_0 \in [\![1, |\bar{x}|]\!]$ tel que $\bar{x}_{i_0} = -$ et $\bar{y}_{i_0} = -$.

2 Méthode naïve par énumération

2.1 Question 3

Étant donné $x \in \Sigma^*$ un mot de longueur $n \in \mathbb{N}$, on cherche le nombre de façon de placer k gaps dans un mots de longueur $n+k$. Il y a donc $\binom{n+k}{k} = \binom{n+k}{n}$ possibilités de faire cela.

2.2 Question 4

Soit (x, y) de longueur respectives n et $m \in \mathbb{N}$ tel que $n \geq m$.

1) Si on ajoute k gaps à x pour obtenir $\bar{x}$, il faudra en ajouter $n+k-m$ à y pour obtenir $\bar{y}$.

2) On cherche le nombre de façon de placer $n+k-m$ gaps dans $\bar{y}$ qui est de longueur $n+k$. Or on nous rappel qu'un gap de $\bar{y}$ ne doit pas être à la même position qu'un gap de $\bar{x}$. Il y a donc $\binom{n}{n+k-m} = \binom{n}{m-k}$ possibilités de faire cela.

3) On a vu qu'il y avait $\binom{n+k}{k}$ façon de placer k gaps dans x et qu'en ajoutant ces k gaps, il y avait $\binom{n}{n+k-m}$ alignements possible de (x, y). Or on a aussi vu qu'un alignement de x était de longueur au maximum $n+m$. On en déduit que le nombre d'alignement possibles de (x, y) est de : $\sum_{k=0}^{m} \binom{n+k}{k}\binom{n}{n+k-m}$.

Application numérique : pour $|x| = 15$ et $|y| = 10$, on trouve **298 199 265** alignements possibles.

2.3 Question 5

On a vu que le nombre d'alignement possible de (x, y) est de : $\sum_{k=0}^{m} \binom{n+k}{k}\binom{n}{n+k-m}$.

On cherche à majorer ce terme, on a :

$$\binom{n+k}{k}\binom{n}{n+k-m} = \frac{(n+k)!}{k!\ n!}\frac{n!}{(m-k)!(n+k-m)} = \frac{(n+k)!}{k!}\frac{1}{(m-k)!(n+k-m)!} = \frac{(n+k)(n+k-1)...(n+k-m+1)}{k!\ (m-k)!}$$

Or : $\frac{(n+k)(n+k-1)...(n+k-m+1)}{k!\ (m-k)!} \leqslant \frac{(n+k)^m}{k!\ (m-k)!} \leqslant \frac{(n+m)^m}{m}$.

Donc le nombre d'alignement possible est majoré par : $\frac{(m+1)(n+m)^m}{m} \leqslant 2(n+m)^m$.

De plus, pour trouver la distance d'édition entre x et y, ou un alignement de coût minimal, il faut parcourir tous les alignements qui sont au maximum de taille $(n+m)$. On en déduit que la complexité d'un tel algorithme naïf est en $O((n+m)(n+m)^m) = O((n+m)^{m+1})$, qui est donc exponentielle.

2.4 Question 6

Un algorithme naïf qui consisterait à énumérer tous les alignements de deux mots en vue de trouver la distance d'édition entre ces deux mots, aurait une compléxité spatiale de $O(n+m)$. En effet, les alignements ont besoin, dans le pire cas, d'une chaîne de taille $n+m$ d'après la question 1, qu'on pourra écraser pour les suivants. Il faut de plus, deux entiers : l'un pour stocker la distance d'édition de l'alignement qu'on vient de produire, et l'autre pour le minimum des distances d'édition qu'on a trouvé jusqu'à maintenant. De même, pour un algorithme en vue de trouver un alignement de coût minimal, sa compléxité spatiale sera en $O(n+m)$.

2.5 Tâche A

Nous avons testé notre implémentation sur les instances $nst\_0000010\_44.adn$, $Inst\_0000010\_7.adn$ et $Inst\_0000010\_8.adn$, et avons observé qu'elle est valide et renvoit bien 10, 8 et 2.
On cherche maintenant à savoir jusqu'à quelle taille d'instance on peut résoudre les instances fournies en moins d'une minute. On obtient le tableau suivant.

Instances	taille n	taille m	temps(s)
Inst_0000013_45.adn	13	12	26.00
Inst_0000013_56.adn	13	12	26.39
Inst_0000013_89.adn	13	12	26.84
Inst_0000014_7.adn	14	12	58.29
Inst_0000014_23.adn	14	10	7.54
Inst_0000014_83.adn	14	10	7.5
Inst_0000015_2.adn	15	13	316
Inst_0000015_4.adn	15	12	124
Inst_0000015_76.adn	15	13	307

Figure 1 : tableau des performances associées à DIST_NAIF

Le temps de calcul dépasse une minute lorsqu'on dépasse les instances de taille 14×12.

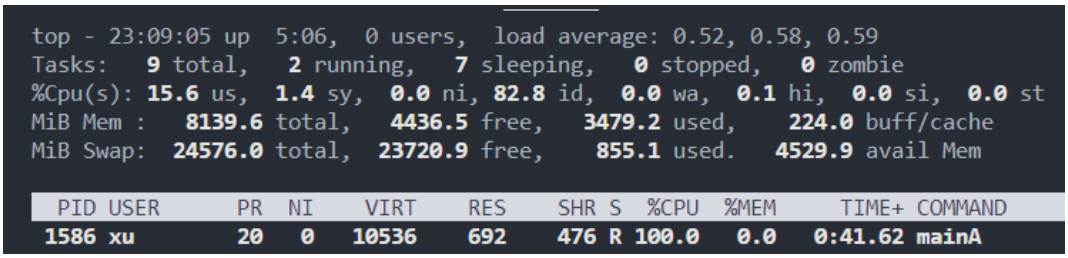

Figure 2 : Utilisation de top pour $|x| = 15$

Concernant la consommation mémoire utilisée, on voit sur la figure 2 qu'elle est assez conséquente. Ce qui est attendu car l'implémentation est naïve et qu'elle peut donc être améliorée.

3 Programmation dynamique

3.1 Question 7

Soit $(\bar{u}, \bar{v})$ un alignement de $(x_{[\![1,i]\!]}, y_{[\![1,j]\!]})$ de longueur l $\in \mathbb{N}$.

Si $\bar{u}_l = -$ alors $\bar{v}_l \neq -$ donc $_l = y_j$.

Si $\bar{v}_l = -$ alors $\bar{u}_l \neq -$ donc $\bar{u}_l = x_i$.

Si $\bar{u}_l \neq -$ et $\bar{v}_l \neq -$ alors $\bar{u}_l = x_i$ et $\bar{v}_l = y_j$.

3.2 Question 8

En distinguant les trois cas de la quesiton 7, on trouve le résultat suivant.

Si $\bar{u}_l = -$ alors $C(\bar{u}, \bar{v}) = C(\bar{u}_{[\![1,l-1]\!]}, \bar{v}_{[\![1,l-1]\!]}) + c_{ins} = C(\bar{u}_{[\![1,l-1]\!]}, \bar{v}_{[\![1,l-1]\!]}) + 2$.

Si $\bar{v}_l = -$ alors $C(\bar{u}, \bar{v}) = C(\bar{u}_{[\![1,l-1]\!]}, \bar{v}_{[\![1,l-1]\!]}) + c_{del} = C(\bar{u}_{[\![1,l-1]\!]}, \bar{v}_{[\![1,l-1]\!]}) + 2$.

Si $\bar{u}_l \neq -$ et $\bar{v}_l \neq -$ alors $C(\bar{u}, \bar{v}) = C(\bar{u}_{[\![1,l-1]\!]}, \bar{v}_{[\![1,l-1]\!]}) + c_{sub}$.

3.3 Question 9

Soit $i \in [\![1, n]\!]$ et $j \in [\![1, m]\!]$, d'après les questions 7 et 8, on en déduit que :

1. si $\bar{u}_l = -$ alors $D(i, j) = D(i, j-1) + c_{ins} = A$
2. si $\bar{v}_l = -$ alors $D(i, j) = D(i-1, j) + c_{del} = B$
3. si $\bar{u}_l \neq -$ et $\bar{v}_l \neq -$ alors $D(i, j) = D(i-1, j-1) + c_{sub} = C$.

D étant la distance d'édition entre deux sous-mots, il s'agit du coût minimum des alignements entre ces deux sous-mots, on a alors : $D(i,j) = min(A,B,C)$.

3.4 Question 10

On a $D(0,0) = d(x_\emptyset, y_\emptyset) = d(\varepsilon,\varepsilon) = 0$.

3.5 Question 11

Pour $j \in [\![1,m]\!]$, $D(0,j) = d(x_\emptyset, y_{[\![1,j]\!]}) = j \times c_{ins}$, ce qui correspond au coût pour insérer j lettres. De même, pour $i \in [\![1,n]\!]$, $D(i,0) = d(x_{[\![1,i]\!]}, y_\emptyset) = i \times c_{del}$, ce qui correspond au coût pour supprimer i lettres.

3.6 Quesiton 12

Algorithme 1 Distance de manière naïve

```
fonction DIST_1(x, y) :
    n ← |x| ;
    m ← |y|;
    T ← [n + 1][m + 1];
    pour i = 0 à n faire
        pour j = 0 à m faire
            si i == 0 alors
                T[i][j] ← j × c_ins ;
            si j == 0 alors
                T[i][j] ← i × c_del ;
            A ← T[i][j − 1] + c_ins ;
            B ← T[i − 1][j] + c_del ;
            C ← T[i − 1][j − 1] + c_sub ;
            T[i][j] ← min(A, B, C) ;
    retourne T ;
```

3.7 Quesiton 13

La complexité spatiale de l'algorithme est en $\theta(n \times m)$

3.8 Quesiton 14

En supposant que le temps requis pour la comparaison, multiplication, addition, ainsi que la fonction min soit constant, la compléxité temporelle de l'algorithme est en $\theta(n \times m)$.

3.9 Quesiton 15

Nous allons montrer le premier cas.

Soit $(i, j) \in [\![0, n]\!] \times [\![0, m]\!]$. Supposons que $j > 0$ et $D(i, j) = D(i, j-1) + c_{ins}$.

On veux montrer que : $\forall (\bar{s}, \bar{t}) \in Al^*(i, j-1), (\bar{s} \cdot -, \bar{t} \cdot y_j) \in Al^*(i, j)$.

Soit $(\bar{s}, \bar{t}) \in Al^*(i, j-1)$, alors $(\bar{s}, \bar{t})$ est un alignement de $(x_{[\![1,i]\!]}, y_{[\![1,j]\!]})$ tel que :

$c(\bar{s}, \bar{t}) = d(x_{[\![1,i]\!]}, y_{[\![1,j-1]\!]}) = D(i, j-1)$.

Or $(\bar{s} \cdot -, \bar{t} \cdot y_j)$ est un alignement de $(x_{[\![1,i]\!]}, y_{[\![1,j]\!]})$.

On a de plus : $c(x_{[\![1,i]\!]}, y_{[\![1,j-1]\!]}) = c(\bar{s}, \bar{t}) + c_{ins} = D(i, j-1) + c_{ins} = D(i, j) = d(x_{[\![1,i]\!]}, y_{[\![1,j]\!]})$.

Donc $(\bar{s} \cdot -, \bar{t} \cdot y_j) \in Al^*(i, j)$.

3.10 Quesiton 16

Algorithme 2 Alignement minimal de (x,y)

fonction SOL_1($\boldsymbol{x}, \mathbf{y}, \mathbf{T}$) :
- $i \leftarrow |x|$;
- $j \leftarrow |y|$;
- $u \leftarrow \varepsilon$;
- $v \leftarrow \varepsilon$; ;
- **tant que** $i > 0$ **ou** $j > 0$ **faire**
  - **si** $j > 0$ et $T[i][j] == T[i][j-1] + c_{ins}$ **alors**
    - $u \leftarrow$ " $-$ " $\cdot\, u$;
    - $v \leftarrow y_j \cdot v$;
    - $j \leftarrow j - 1$;
  - **si** $i > 0$ et $T[i][j] == T[i-1][j] + c_{del}$ **alors**
    - $u \leftarrow x_i \cdot u$;
    - $v \leftarrow$ " $-$ " $\cdot\, v$;
    - $i \leftarrow i - 1$;
  - **si** $T[i][j] == T[i-1][j-1] + c_{sub}$ **alors**
    - $u \leftarrow x_i \cdot u$;
    - $v \leftarrow y_j \cdot v$;
    - $i \leftarrow i - 1$;
    - $j \leftarrow j - 1$;
- **retourne** (u, v) ;

3.11 Quesiton 17

On a vu que DIST_1 a une compléxité temporelle de $\theta(n \times m)$. Et SOL_1 fait $(n + m)$ itérations. En supposant que les tests de comparaisons et les additions se font en temps constants, on en déduit qu'on résout le problème ALI en $\theta(n \times m)$.

3.12 Quesiton 18

On a vu que DIST_1 a une compléxité spatiale de $\theta(n \times m)$. Pour SOL_1, dans le pire cas, on a besoin de deux chaînes de charactères de taille $n + m$, ainsi que de deux entiers. Sa complexité spatiale est donc de $\theta(n + m)$. On en déduit que celle pour répondre au problème d'ALI est en $\theta(n \times m)$.

3.13 Tâche B

Nous avons testé la fonction $PROG\_DYN$ sur les instances $nst\_0000010\_44.adn$, $Inst\_0000010\_7.adn$ et $Inst\_0000010\_8.adn$. Elle renvoit bien 10, 8 et 2 ainsi que des alignements valides de coût respectivement égal à leur distance d'édition.

Pour avoir un aperçu des perfomances de $PROG\_DYN$, on a tracé le graphique suivant.

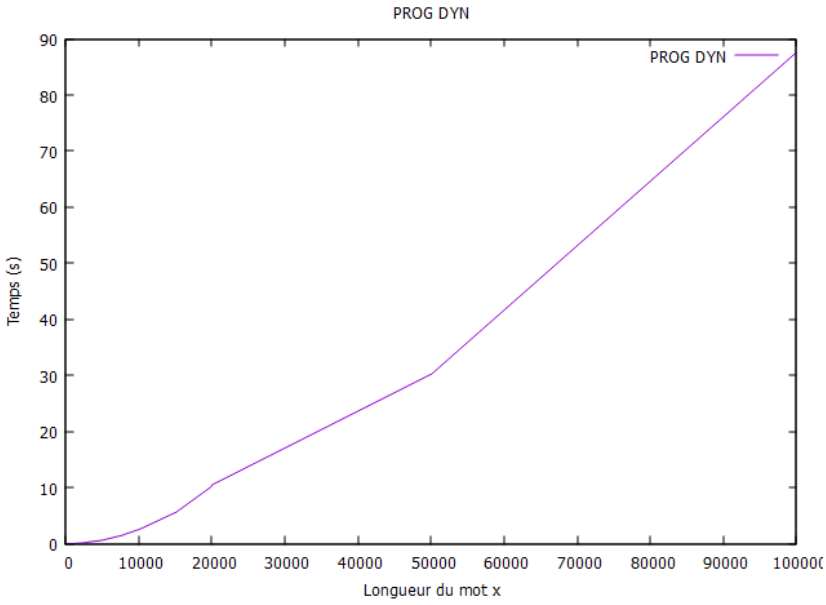

Figure 3 : Courbe de consommation de temps CPU en fonction de $|x|$

```
top - 00:17:02 up 11 min,  0 users,  load average: 0.52, 0.58, 0.59
Tasks:  11 total,   2 running,   9 sleeping,   0 stopped,   0 zombie
%Cpu(s): 11.0 us,  4.0 sy,  0.0 ni, 85.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   8139.6 total,   2899.3 free,   5016.3 used,    224.0 buff/cache
MiB Swap:  24576.0 total,  23949.5 free,    626.5 used.   2992.7 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  164 xu        20   0 1468340   1.2g    476 R  52.5  14.7   0:08.30 mainB
```

Figure 4 : Utilisation de top pour $|x| = 20000$

Pour une instance de très grande taille, la quantité de mémoire utilisée sera très importante. Plus l'instance est grande, plus la quantité de mémoire utilisée augmente.

4 Amélioration de la complexité spatiale du calcul de la distance

4.1 Question 19

Pour remplir la ligne $i > 0$ du tableau T :

— pour $j = 0$, on a $T[i][j] = i \times c_{del}$

— pour $j > 0$, on a vu à la question 12 que $T[i][j]$ est rempli à l'aide de $T[i][j-1]$, $T[i-1][j]$ ou $T[i-1][j-1]$

On remarque alors que pour remplir la ligne $i > 0$ du tableau T, il suffit d'avoir accès aux lignes $i-1$ et i du tableau.

4.2 Question 20

Algorithme 3 Calcul de la distance

```
fonction DIST_2(x, y) :
    Tc ← [] ;
    T ← [] ;
    pour j = 0 à m faire
        T[j] = j × c_ins ;
    pour i = 1 à n faire
        Tc[0] = i × c_del ;
        pour j = 1 à m faire
            A ← Tc[j − 1] + c_ins ;
            B ← T[j] + c_del ;
            C ← T[j − 1] + c_sub ;
            Tc[j] ← min(A, B, C) ;
        T ← Tc;
    retourne Tc[m] ;
```

4.3 Tâche C

Nous avons testé la fonction $DIST\_2$ sur plusieurs instances et elle renvoit le même résultat que $DIST\_1$.

Pour avoir un aperçu des perfomances de $DIST\_2$, on a tracé le graphique suivant.

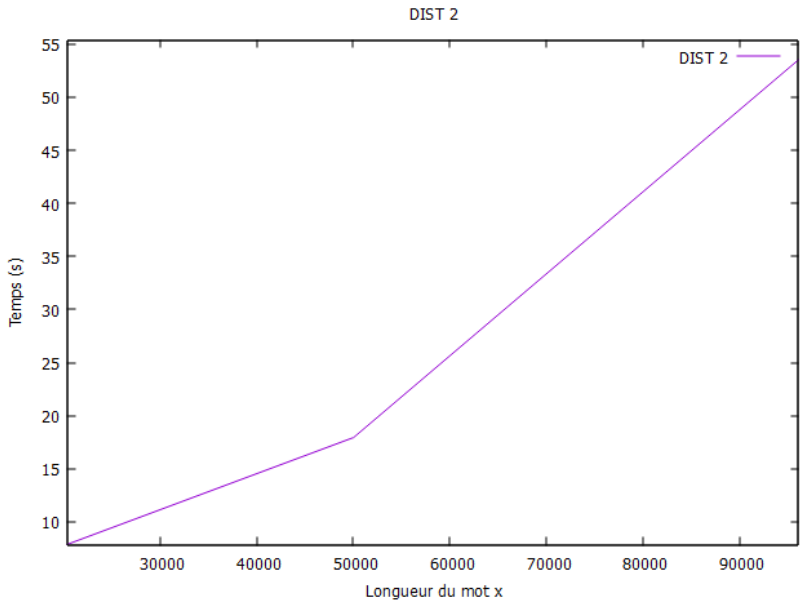

Figure 5 : Courbe de consommation de temps CPU en fonction de $|x|$

Pour comparer $DIST\_2$ à $DIST\_1$, nous avons le graphe des performances de $DIST\_1$.

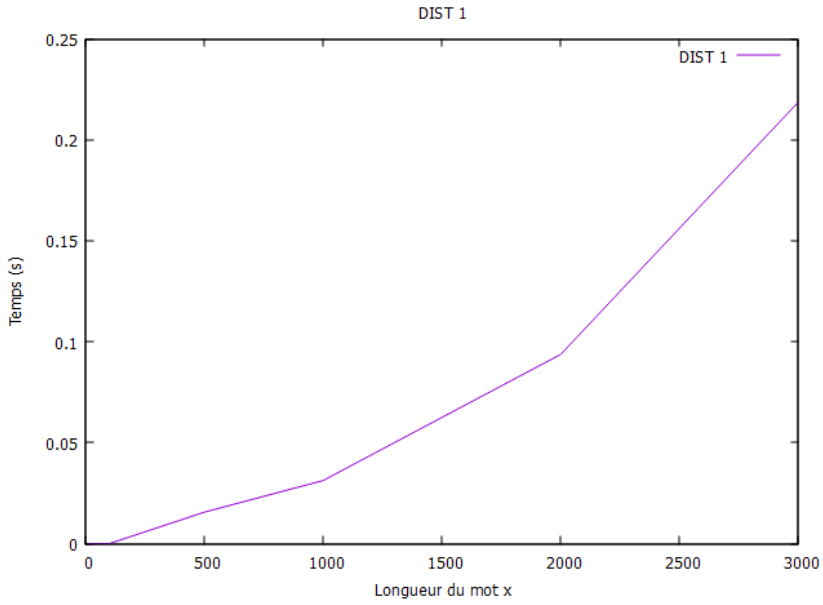

Figure 6 : Courbe de consommation de temps CPU en fonction de $|x|$

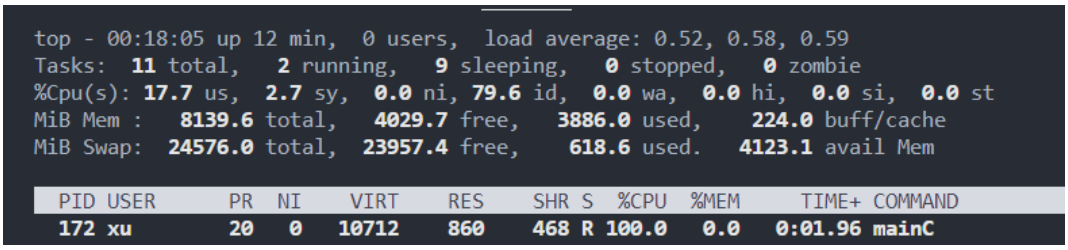

Figure 7 : Utilisation de top pour $|x| = 20000$

Pour une instance de très grande taille, la quantité de mémoire utilisée a été grandement améliorée comparée à $DIST\_1$.

5 Amélioration de la complexité spatiale du calcul d'un alignement optimal par la méthode "diviser pour régner"

5.1 Question 21

Algorithme 4 Renvoie k gaps

fonction GAPS(k) :
    $x \leftarrow \varepsilon$;
    pour $i = 1$ **à** k **faire**
        $x \leftarrow x \cdot$ " $-$ " ;
    retourne x ;

5.2 Question 22

Algorithme 5 Align lettre mot

fonction ALIGN_LETTRE_MOT($\boldsymbol{x}, \mathbf{y}$) :
  $m \leftarrow |y|$;
  $i_0 \leftarrow -1$;
  pour $i = 0$ **à** $m - 1$ **faire**
    si $x == y[i]$ **alors**
      retourne $(GAPS(i) \cdot x \cdot GAPS(m - i - 1), y)$;
    si $y[i]$ et x sont concordantes **alors**
      $i_0 \leftarrow i$;
  si $i_0 \,! = -1$ **alors**
    retourne $(GAPS(i_0) \cdot y[i_0] \cdot GAPS(m - i_0 - 1), y)$
  retourne $(x \cdot GAPS(m - 1), y)$;

5.3 Question 23

Soit x^1 = BAL, x^2 = LON, y^1 = RO et y^2 = ND.

On pose $\bar{s}$ = BAL, $\bar{t}$ = RO$-$, $\bar{u}$ = LON$-$ et $\bar{v}$ = $--$ND, alors $(\bar{s}, \bar{t})$ et $(\bar{u}, \bar{v})$ sont respectivement les alignements optimaux de (x^1, y^1) et (x^2, y^2).

Pourtant $(\bar{s} \cdot \bar{u}, \bar{t} \cdot \bar{v})$ = (BALLON$-$, RO$---$ND) n'est pas un alignement optimal de (x,y). En efffet, le coût de cet alignement est de 22, tandis que (BALLON$-$, $---$ROND) est aussi un alignement de (x,y), mais de coût 17.

5.4 Question 24

Algorithme 6 Alignement minimal de (x,y)

fonction SOL_2($\boldsymbol{x}$, y) :

$n \leftarrow |x|$;
$m \leftarrow |y|$;
si $n = 0$ **alors**
    retourne $(GAPS(m), y)$;
si $y = 0$ **alors**
    retourne $(x, GAPS(n))$;
si $n = 1$ **alors**
    retourne $ALIGN\_LETTRE\_MOT(x, y)$;
si $m = 1$ **alors**
    retourne $ALIGN\_LETTRE\_MOT(y, x)$;
$i^* \leftarrow n/2$;
$j^* \leftarrow coupure(x, y)$;
$(\bar{s}, \bar{t}) \leftarrow SOL\_2(x_{[\![1,i^*]\!]}, y_{[\![1,j^*]\!]})$
$(\bar{u}, \bar{v}) \leftarrow SOL\_2(x_{[\![i^*+1,n]\!]}, y_{[\![j^*+1,m]\!]})$
retourne $(\bar{s} \cdot \bar{u}, \bar{t} \cdot \bar{v})$;

5.5 Question 25

Algorithme 7 Déterminer la coupure j^* associée à i^* pour (x,y)

```
fonction COUPURE(x, y) :
    n ← |x| ;  m ← |y| ;
    i* ← |x|/2 ;
    T[i] ← [] ;  Tc ← [] ;
    I[i] ← [] ;  Ic ← [] ;
    pour i = 0 à m faire
        T[i] ← i × c_ins ;
        I[i] ← i ;
    pour i = 1 à n faire
        Tc[0] ← i × c_del ;
        pour j = 1 à m faire
            A ← Tc[j − 1] + c_ins ;
            B ← T[j] + c_del ;
            C ← T[j − 1] + c_sub ;
            Tc[j] ← min(A, B, C) ;
            si i > i* alors
                si Tc[j] = A alors
                    Ic[j] ← Ic[j − 1] ;
                si Tc[j] = B alors
                    Ic[j] ← I[j] ;
                si Tc[j] = C alors
                    Ic[j] ← I[j − 1] ;
        T ← Tc ;
        si i > i* alors
            I ← Ic ;
    retourne Ic[m]
```

5.6 Question 26

Durant l'éxécution de la fonction $COUPURE$, on a eu besoin de quatre tableaux de taille $(m + 1)$, ainsi que de huit variables. La complexité spatiale de la fonction est donc en $\theta(m)$.

5.7 Question 27

La fonction SOL_2 est une fonction résursive de type diviser pour régner. A chaque appel, elle fait deux appels récursifs où la taille du premier argument est réduit de moitié. La profondeur de son arbre des appels est donc en $O(log_2(n))$. De plus, à chacun de ses appels, elle utilise la fonction $COUPURE$ qui a une complexité spatiale en $\theta(m)$, ainsi que de deux chaînes de caractères tous majorées par $(n+m)$. On en déduit que la compléxité spatiale de SOL_2 est en $O((m+n+m).log_2(n)) = O((n+m).log_2(n))$

5.8 Question 28

En supposant toujours que les opérations de comparaisons, de multiplications, d'additions et la fonction min se font en temps constant, alors la fonction $COUPURE$ fait $(m+n \times m)$ itérations. Sa compléxité temporelle est donc en $\theta(n \times m)$.

5.9 Tâche D

Nous avons testé la fonction SOL_2 sur plusieurs instances et avons vérifié que la fonction retourne bien le résultat attendu. Pour avoir un aperçu des perfomances de SOL_2, on a tracé le graphique suivant.

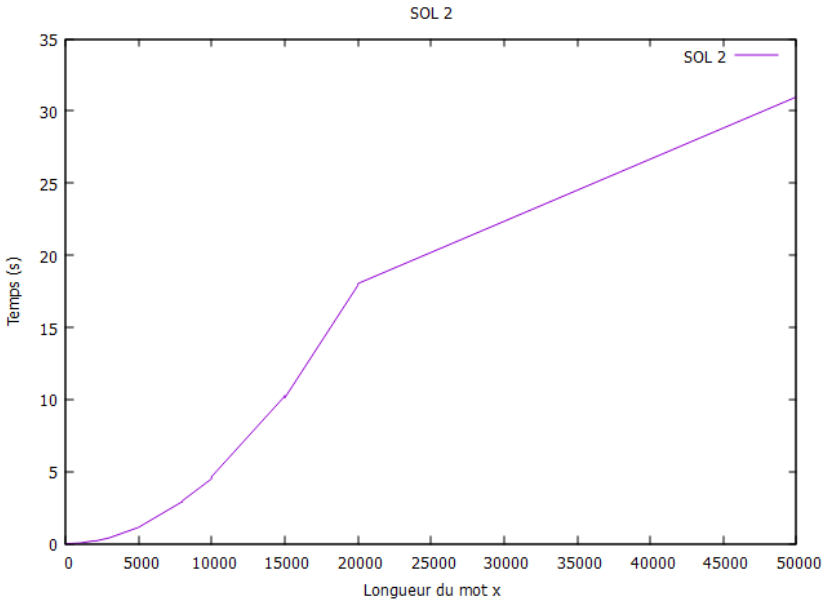

Figure 8 : Courbe de consommation de temps CPU en fonction de $|x|$

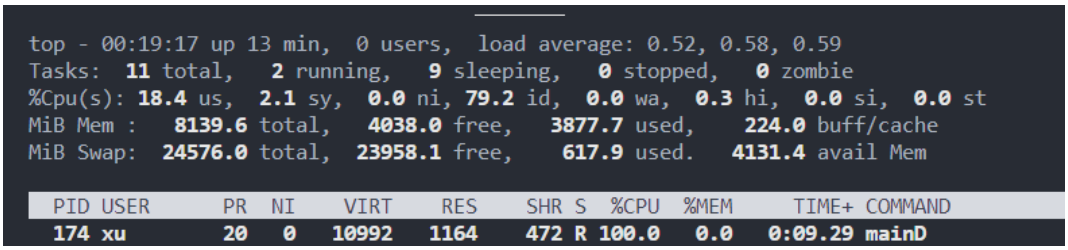

Figure 9 : Utilisation de top pour $|x| = 20000$

5.10 Question 29

En comparant les performances de SOL_2 à celles de SOL_1, nous remarquons qu'en effet, nous avons perdu en complexité temporelle en améliorant la complexité spatiale.