Algorithmes pour le traitement de séquences.

Alignement optimal et logiciel d'aide à la détection de plagiat.

Question 1 :

On peut s'inspirer de l'algorithme dynamique Edit Distance pour obtenir un algorithme de calcul du score de l'alignement optimal.

La différence avec celui-ci est que notre but n'est pas de partir d'un mot x pour le transformer en un mot y, mais d'obtenir des mots x' et y' de même longueur que x ou y, selon lequel est le plus grand, en ajoutant des blancs dans le mot le plus petit, de telle sorte à aligner le maximum de caractères entre x' et y'.

Il est nécessaire de montrer l'équivalence entre les deux problèmes :
Pour cela, il est nécessaire de montrer qu'il existe une bijection entre chaque transformation entre deux mots x et y et un alignement de x et y.
C'est-à-dire, comment peut-on passer d'une transformation à un alignement (F1) et comment peut-on passer d'un alignement à une transformation (F2) , le tout avec des coûts égaux.
De même, en passant d'une transformation à un alignement et de cet alignement à une transformation, on doit pouvoir retrouver les chaînes x et y telles qu'elles étaient au départ (F1oF2 = Id).

Reprenons l'exemple de l'énoncé : soient x = ACGA et y = ATGCTA.

On peut transformer x en y de la manière suivante :

Substituer le A dans x par le A dans y à la position 0 → x = ACGA (coût total = 0) (1)
Substituer le C dans x par le T dans y à la position 1 → x = ATGA (coût total = 1) (2)
Substituer le G dans x par le G dans y à la position 2 → x = ATGA (coût total = 1) (3)
Insérer C dans x à la position 3 → x = ATGCA (coût total = 2) (4)
Insérer T dans x à la position 4 → x = ATGCTA (coût total = 3) (5)
Substituer le A dans x par le A dans y à la position 0 → x = ATGCTA (coût total = 3) (6)

Si nous cherchons à construire notre alignement de la même manière :
On obtient x' = ACG__A
et y' = ATGCTA
On a donc une différence aux positions 2,3 et 4, ce qui implique un coût total de 3.
On retrouve donc bien le même coût !

De même si on reprend les alignements x' et y'.
En partant de x', il est possible d'atteindre y :
En remplaçant le A de x' par le A de y' à la position 0 → x' = ACG__A (coût total = 0) (1)
En remplaçant le C de x' par le T de y' à la position 1 → x' = ATG__A (coût total = 1) (2)

En remplaçant le G de x’ par le G de y’ à la position 2 → x’ = ATG__A (coût total = 1) (3)
En insérant C dans x’ à la position 3 → x’ = ATGC_A(coût total = 2) (4)
En insérant T dans x’ à la position 4 → x’ = ATGCTA (coût total =3) (5)
En remplaçant A de x’ par le A de y’ à la position 5 → x’ = ATGCTA (coût total = 3) (6).

On a donc atteint y en partant de x’ exactement avec les mêmes transformations que pour x avec le même coût. On vient donc de montrer l’équivalence entre la transformation et l’alignement.

On peut donc remarquer les équivalences suivantes :

- Une insertion de caractères dans les transformations (x) correspond à l’insertion d’un blanc dans l’alignement (x’).
- Une substitution d’un caractère dans x par un autre strictement identique au premier dans y correspond à une absence de modification dans x’.
- Une substitution d’un caractère de x par un autre de y cette fois différent du premier correspond à une différence de caractère entre x’ et y’ qui n’est pas un blanc.
- La suppression de caractère dans x correspond à un ajout de blanc dans y’.

Il est donc possible d’utiliser l’algorithme EditDistance (lui même en O(|x|*|y|) pour calculer le coût d’un alignement optimal.

Question 2 :

Pour établir cet algorithme, il est nécessaire de s’intéresser à la manière dont est produit le tableau d’entiers dans EditDistance :

On remarque qu'à chaque itération, l’entier à mettre dans T[i][j] est choisi en fonction du coût des précédents entiers, situés respectivement en haut, à gauche et en diagonale en haut à gauche par rapport à celui-ci. Or l’entier choisi dépend du minimum entre ces 3 plus le coût d’une transformation, qui est différente pour chaque :
Pour l’entier de la colonne précédente (à gauche) on choisit comme valeur à comparer l’entier plus le coût d’une insertion à la position j-1 dans y.
Pour l’entier de la ligne précédente (en haut) on choisit comme valeur à comparer l’entier plus le coût d’une suppression à la position i-1 dans x.
Pour l’entier à la ligne et colonne précédente (en diagonale en haut à gauche), on choisit comme valeur à comparer l’entier plus le coût d’un remplacement du caractère dans x à la position i-1 par celui dans y à la position j-1.
On choisit alors le plus petit entre les 3, et en cas d’égalité, c’est celui en diagonale qui est retenu (le dernier à être comparé), puisque pour un alignement, la substitution ne nécessite aucune modification.

Ainsi, en partant de T[n][m], on peut remonter jusqu’à T[0][0] en utilisant le résultat de cette sélection : pour chaque entier de la matrice testé, on regarde les 3 voisins (la colonne précédente située à gauche, la ligne précédente située en haut et la diagonale précédente) et on se déplace vers le plus petit. En cas d’égalité, comme dans EditDistance, on sélectionne par défaut la diagonale, puisque il n’y a pas de modification à faire pour l’alignement. Selon la transformation nécessaire pour passer d’un entier à l’autre de la matrice, on détermine ainsi ce que l’on doit faire dans l’alignement, comme dit ci-dessus. Il faudra seulement faire attention au fait que l’on construit ici nos chaînes de caractères en commençant par la fin.

Alignement (matrice T, string x, string y, int m, int n) renvoie deux strings x’ et y’ :

```
String x’, y’ = “”;
int i = m;
int j = n;
int k = 0
int min;

Tant Que (i>0 ET j >0)  {
        Si (T[i][j-1] <= T[i][j])
                {min = T[i][j-1];       k = 1;}
        Si (T[i-1][j] <= min)
                {min = T[i-1][j];       k = 2;}
        Si (T[i-1][j-1] <= min)
                { k =3;}

        Si ( k == 1):
                j--;
                x’= concaténation de “_” et x’;
                y’ = concaténation de y[j] et y’;

        Et Si ( k==2):
                i--;
                y’ = concaténation de “_” et y’;
                x’ = concaténation de x[i] et x’;

        Et Si ( k == 3 ) :
                i--; j--;
                x’ = concaténation de x[i] et x’;
                y’ = concaténation de y[j] et y’;

}

Si ( i == 0)      {
        Tant Que (j =/= 0)      {
                j--;
                x’ = concaténation de “_” et x’;
                y’ = concaténation de y[j] et y’;
        }
}
Et Si ( j == 0)   {
        Tant Que (i =/= 0)      {
                i--;
                y’ = concaténation de “_” et y’;
                x’ = concaténation de x[i] et x’;
        }
}
return x’, y’;
```

Question 3 :

Voici l'implémentation de l'algorithme EditDistance de la Question 1 :

```
int EditDistance (char * x, char * y, int m, int n, int ** T)
{
    T[0][0] = 0;
    /*========= Remplissage des bords gauche et haut de la matrice ===========*/
    for (int i = 1; i <= m; i++)
    {
        T[i][0] = T[i-1][0] + Del(x, i-1);
    }
    for (int j = 1; j <= n; j++)
    {
        T[0][j] = T[0][j-1] + Ins(x, y, j-1);
    }
    /*======================================================================*/
    int min;
    /*========== Remplissage du reste de la matrice ==========================*/
    for (int i = 1; i <= m; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            min = T[i][j-1] + Ins(x, y, j-1);

            if ((T[i-1][j] + Del(x, i-1)) <= min)
                min  = T[i-1][j] + Del(x, i-1);

            if ((T[i-1][j-1] + Sub(x, y, i-1, j-1)) <= min)
                min = T[i-1][j-1] + Sub(x, y, i-1, j-1);
            // Le test précédent étant placé en dernier avec une inégalité large, il est choisi
par défaut en cas d"égalité.
            T[i][j] = min;
        }
    }
    /*======================================================================*/
    return T[m][n];
}
```

Comme on peut utiliser le résultat de celui-ci pour obtenir par la suite des alignements, il n'est pas nécessaire d'y apporter de modifications. A noter que les fonctions Ins et Del renvoient toujours 1 et Sub renvoie 0 si les caractères sont identiques et 1 sinon.

Sur les mots NICHE et CHIEN, l'algorithme renvoie la matrice suivante :

```
result : 4
012345
112344
222234
323334
432344
543334
```

Il aboutit donc au bon résultat, mais un terme de la matrice change par rapport à l'énoncé, ligne 4 colonne 1, qui est un 4 dans l'énoncé et qui devient un 3 ici, ce qui modifie par conséquent 2 termes suivant.

Selon moi, il ne peut y avoir un 4 à cet endroit car l'algorithme choisit forcément de partir du 2 situé au-dessus (coût le plus faible), mais aucune opération n'a un coût de 2…

Toujours est-il que l'algorithme trouve le bon résultat et nous allons voir par la suite qu'on aboutit au bon alignement.

```
Source Wikipedia. La distance de Levenshtein une distance mathematique donnant une mesure de la
similarite entre deux chaines de caracteres. Elle est egale au nombre minimal de caracteres qu il
faut supprimer, inserer ou remplacer pour passer d une chaine a l autre. Elle a ete proposee par
Vladimir Levenshtein en 1965. Elle est egalement connue sous les noms de distance d edition ou de
deformation dynamique temporelle, notamment en reconnaissance de formes et particulierement en
reconnaissance vocale1,2.Cette distance est d autant plus grande que le nombre de differences
entre les deux chaines est grand. La distance de Levenshtein peut etre consideree comme une
generalisation de la distance de Hamming. On peut montrer en particulier que la distance de
Hamming est un majorant de la distance de Levenshtein.Definition : on appelle distance de
Levenshtein entre deux mots M et P le cout minimal pour aller de M a P en effectuant les
operations elementaires suivantes : i) substitution d un caractere de M en un caractere de P ; ii)
ajout dans M d un caractere de P ; iii) suppression d un caractere de M. On associe ainsi a
chacune de ces operations un cout. Le cout est toujours egal a 1, sauf dans le cas d une
substitution de caracteres identiques. Exemples : si M = "examen" et P = "examen", alors LD (M, P)
= 0, parce qu aucune operation n a ete realisee. Si M = "examen" et P = "examan", alors LD (M, P)
= 1, parce qu il y a eu un remplacement (changement du e en a), et que l on ne peut pas en faire
moins.
```

```
Source Wikipedia modifie par un etudiant du cours IT-4301E, traitement algorithmique de l
information. La distance de d editon est une distance au sens mathematique donnant une mesure de
la similarite entre deux sequences. Elle est egale au nombre minimal de caracteres qu il faut
supprimer, inserer ou substituer pour passer d une sequence a l autre. Elle a ete proposee par
Vladimir Levenshtein en 1965. Elle est ainsi egalement connue sous les noms de distance de
Levenshtein ou de distance de deformation dynamique temporelle dans le domaine de la
reconnaissance de formes. Cette distance est est une fonction croissante du nombre de differences
entre les deux sequences. La distance d edition peut etre consideree comme une generalisation de
la distance de Hamming (donnee par le nombre de position en lesquelles les deux sequences
possedent des caracteres differents). On peut montrer en particulier que la distance de Hamming
est un majorant de la distance d edition.Definition formelle : on appelle distance d edition entre
deux mots M et P le cout minimal transformer M en P en effectuant les operations elementaires,
dites d edition, suivantes : i) substitution d un caractere de M par un caractere de P ; ii)
insertion dans M d un caractere de P ; iii) suppression (ou deletion) d un caractere de M. On
associe ainsi a chacune de ces operations un cout. On choisit souvent un cout egal a 1 pour toutes
les operations excepte la substitution de caracteres identiques qui a un cout nul.Exemples : si M
= "examen" et P = "examen", alors Lev(M, P) = 0 parce qu aucune operation n a ete realisee. Si M =
"examen" et P = "examan", alors Lev(M, P) = 1, parce qu il y a eu une substitution (changement du
e en a), et que l on ne peut pas en faire une transformation de M en P avec un moindre cout.
```

Sur texte1.c et texte2.c (ci -dessus)

Le score d'alignement optimal trouvé est de 577.

Voici l'implémentation de l'algorithme de la Question 2 :

```
void Alignement (int ** T, char * x, char * y, char * xp, char * yp, int m, int n)
/* Remplit xp et yp, l'alignement optimal entre les chaînes x et y.
/*============================================================================ */
{
    int i = m;
    int j = n;
    int k = 0;
    int min;

    while (i > 0 && j > 0)  {
        /*========== Copie à chaque itération de xp et yp pour pouvoir concaténer par le début
de la chaine.*/
        char * xp_copy = (char *)malloc(2*m * sizeof(char));
        strcpy(xp_copy, xp);
        char * yp_copy = (char *)malloc(2*n * sizeof(char));
        strcpy(yp_copy, yp);
        /*====================================================================*/

        /*========== Test parmi les 3 nombres précédents dans le tableau pour trouver le plus
petit.*/
        if (T[i][j-1] <= T[i][j])
            {min = T[i][j-1]; k = 1;}
        if(T[i-1][j] <= min)
            {min = T[i-1][j]; k = 2;}
        if(T[i-1][j-1] <= min)
            {k = 3;}
        /*=========== En cas d'égalité, c'est le k=3 qui sera sélectionné (la diagonale) =*/
        if (k==1)
        {
            j--;
            sprintf(xp, "_%s", xp_copy);
            sprintf(yp, "%c%s", y[j], yp_copy);
        } // Ajout de blanc dans xp, équivalent à une insertion.
        else if (k==2)
        {
            i--;
            sprintf(yp, "_%s", yp_copy);
            sprintf(xp, "%c%s", x[i], xp_copy);
        } // Ajout de blanc dans yp, équivalent à une suppression.
        else if (k==3)
        {
            i--;
            j--;
            sprintf(xp, "%c%s", x[i], xp_copy);
            sprintf(yp, "%c%s", y[j], yp_copy);
        } // Equivalent de la substitution, on recopie les caractères sans ajout de blanc.
        printf("xp : %s\n", xp);
        printf("yp : %s\n", yp);
        free(xp_copy);
        free(yp_copy);
    }
```

(suite sur la page suivante).

```c
    /* Traitement des bords de la matrice, c'est à dire quand on a traité un des deux mots
entièrement.*/
    if ( i == 0)    {
        while (j != 0)
        {
            char * xp_copy = (char *)malloc(m * sizeof(char));
            strcpy(xp_copy, xp);
            char * yp_copy = (char *)malloc(n * sizeof(char));
            strcpy(yp_copy, yp);
            j--;
            sprintf(xp, "_%s", xp_copy);
            sprintf(yp, "%c%s", y[j], yp_copy);
            free(xp_copy);
            free(yp_copy);
            printf("xp : %s\n", xp);
            printf("yp : %s\n", yp);
        }

    }

    else if ( j == 0)   {
        while (i != 0)
        {
            char * xp_copy = (char *)malloc(m * sizeof(char));
            strcpy(xp_copy, xp);
            char * yp_copy = (char *)malloc(n * sizeof(char));
            strcpy(yp_copy, yp);
            i--;
            sprintf(yp, "_%s", yp_copy);
            sprintf(xp, "%c%s", x[i], xp_copy);
            free(xp_copy);
            free(yp_copy);
            printf("xp : %s\n", xp);
            printf("yp : %s\n", yp);
        }
    }
}
```

Testé sur NICHE et CHIEN il renvoie le résultat suivant (l'affichage des étapes a été retiré sur la version ci-dessus) :

```
xp : _
yp : N
xp : E_
yp : EN
xp : _E_
yp : IEN
xp : H_E_
yp : HIEN
xp : CH_E_
yp : CHIEN
xp : ICH_E_
yp : _CHIEN
xp : NICH_E_
yp : __CHIEN
Resultat final :
NICH_E_
__CHIEN
```

Il s'agit bien du résultat attendu.

Sur les deux textes précédents :

```
Resultat final :
|--------------------------------------------------|--------------------------------------------------|
| Source Wikipedia_______. La_ ______di___________s_ | Source Wikipedia modifie par un etudiant du cours  |
| __________t_a_____n_____________ce de ____________ | IT-4301E, traitement algorithmique de l informatio |
| ___L_________e_venshtei__n ____une distance ______ | n. La distance de d editon est une distance au sen |
| __mathematique donnant une mesure de la similarite | s mathematique donnant une mesure de la similarite |
|  entre deux chaines de caracteres. Elle est egale  |  entre deux ____se_quenc_____e__s. Elle est egale  |
| au nombre minimal de caracteres qu il faut supprim | au nombre minimal de caracteres qu il faut supprim |
| er, inserer ou _remplacer pour passer d une _chain | er, inserer ou substituer pour passer d une sequen |
| _e a l autre. Elle a ete proposee par Vladimir Lev | ce a l autre. Elle a ete proposee par Vladimir Lev |
| enshtein en 1965. Elle est ______egalement connue  | enshtein en 1965. Elle est ainsi egalement connue  |
| sous les noms de distance d_ _e___dition ou de d__ | sous les noms de distance de Levenshtein ou de dis |
| ____e______formation dynamique temporelle, notamme | tance de deformation dynamique temporelle_ __da___ |
| nt en reconnaissance de formes et particulierement | ns le __do_mai___n_e de __________________l_______a |
|  en reconnaissance vocale1,2._Cette distance est _ |  ___reconnaissance de formes. Cette distance est e |
| _d _autant plus grande que le nombre de difference | st une fonction croissante du nombre de difference |
| s entre les deux chaines est grand. La distance de | s entre les deux _sequences_______. La distance d_ |
|  Levenshtein peut etre consideree comme une genera |  _e___dition peut etre consideree comme une genera |
| lisation de la distance de Hamming________________ | lisation de la distance de Hamming (donnee par le  |
| __________________________________________________ | nombre de position en lesquelles les deux sequence |
| ______________________________________. On peut mo | s possedent des caracteres differents). On peut mo |
| ntrer en particulier que la distance de Hamming es | ntrer en particulier que la distance de Hamming es |
| t un majorant de la distance de Levenshtein.Defini | t un majorant de la distance d_ _e___dition.Defini |
| tion _________: on appelle distance de Levenshtein | tion formelle : on appelle distance d_ _e___dition |
|  entre deux mots M et P le cout minimal pour aller |  entre deux mots M et P le cout minimal __transfor |
|  de M _a P en effectuant les operations elementair | mer M en P en effectuant les operations elementair |
| es_ ____s_____________uivantes : i) substitution d | es, dites d edition, suivantes : i) substitution d |
|  un caractere de M _en un caractere de P ; ii) _aj |  un caractere de M par un caractere de P ; ii) ins |
| out___ dans M d un caractere de P ; iii) suppressi | ertion dans M d un caractere de P ; iii) suppressi |
| on ____d________ __un caractere de M. On associe a | on (ou deletion) d un caractere de M. On associe a |
| insi a chacune de ces operations un cout. Le c_o__ | insi a chacune de ces operations un cout. On chois |
| ut ____est toujours egal a 1, sauf __dans le_ ___c | it souvent un cou_t egal a 1_ pour toutes les oper |
| a____s __d une ___substitution de caracteres ident | ations excepte la substitution de caracteres ident |
| iques___. ______________Exemples : si M = "examen" | iques qui a un cout nul.Exemples : si M = "examen" |
|  et P = "examen", alors LD (M, P) = 0, parce qu au |  et P = "examen", alors Lev(M, P) = 0_ parce qu au |
| cune operation n a ete realisee. Si M = "examen" e | cune operation n a ete realisee. Si M = "examen" e |
| t P = "examan", alors LD (M, P) = 1, parce qu il y | t P = "examan", alors Lev(M, P) = 1, parce qu il y |
|  a eu un_ remplacement (changement du e en a), et  |  a eu une substitution (changement du e en a), et  |
| que l on ne peut pas en faire ____________m___o___ | que l on ne peut pas en faire une transformation d |
| ___________________in_______s.                     | e M en P avec un moindre cout.                     |
|--------------------------------------------------|--------------------------------------------------|
```

Avec un affichage différent (les lignes sont accolées) :

```
x: Source Wikipedia_______. La_ ______di___________s___________t_a_____n_____________ce de ____________
y: Source Wikipedia modifie par un etudiant du cours IT-4301E, traitement algorithmique de l informatio
x: ___L_________e_venshtei__n ____une distance ________mathematique donnant une mesure de la similarite
y: n. La distance de d editon est une distance au sens mathematique donnant une mesure de la similarite
x:  entre deux chaines de caracteres. Elle est egale au nombre minimal de caracteres qu il faut supprim
y:  entre deux ____se_quenc_____e__s. Elle est egale au nombre minimal de caracteres qu il faut supprim
x: er, inserer ou _remplacer pour passer d une _chain_e a l autre. Elle a ete proposee par Vladimir Lev
y: er, inserer ou substituer pour passer d une sequence a l autre. Elle a ete proposee par Vladimir Lev
x: enshtein en 1965. Elle est ______egalement connue sous les noms de distance d_ _e___dition ou de d__
y: enshtein en 1965. Elle est ainsi egalement connue sous les noms de distance de Levenshtein ou de dis
x: ____e______formation dynamique temporelle, notamment en reconnaissance de formes et particulierement
y: tance de deformation dynamique temporelle_ __da___ns le __do_mai___n_e de __________________l_______a
x:  en reconnaissance vocale1,2._Cette distance est __d _autant plus grande que le nombre de difference
y:  ___reconnaissance de formes. Cette distance est est une fonction croissante du nombre de difference
x: s entre les deux chaines est grand. La distance de Levenshtein peut etre consideree comme une genera
y: s entre les deux _sequences_______. La distance d_ _e___dition peut etre consideree comme une genera
x: lisation de la distance de Hamming__________________________________________________________________
y: lisation de la distance de Hamming (donnee par le nombre de position en lesquelles les deux sequence
x: ______________________________________. On peut montrer en particulier que la distance de Hamming es
y: s possedent des caracteres differents). On peut montrer en particulier que la distance de Hamming es
x: t un majorant de la distance de Levenshtein.Definition _________: on appelle distance de Levenshtein
y: t un majorant de la distance d_ _e___dition.Definition formelle : on appelle distance d_ _e___dition
x:  entre deux mots M et P le cout minimal pour aller de M _a P en effectuant les operations elementair
y:  entre deux mots M et P le cout minimal __transformer M en P en effectuant les operations elementair
x: es_ ____s_____________uivantes : i) substitution d un caractere de M _en un caractere de P ; ii) _aj
y: es, dites d edition, suivantes : i) substitution d un caractere de M par un caractere de P ; ii) ins
x: out___ dans M d un caractere de P ; iii) suppression ____d________ __un caractere de M. On associe a
y: ertion dans M d un caractere de P ; iii) suppression (ou deletion) d un caractere de M. On associe a
x: insi a chacune de ces operations un cout. Le c_o__ut ____est toujours egal a 1, sauf __dans le_ ___c
y: insi a chacune de ces operations un cout. On choisit souvent un cou_t egal a 1_ pour toutes les oper
x: a____s __d une ___substitution de caracteres identiques___. ______________Exemples : si M = "examen"
y: ations excepte la substitution de caracteres identiques qui a un cout nul.Exemples : si M = "examen"
x:  et P = "examen", alors LD (M, P) = 0, parce qu aucune operation n a ete realisee. Si M = "examen" e
y:  et P = "examen", alors Lev(M, P) = 0_ parce qu aucune operation n a ete realisee. Si M = "examen" e
x: t P = "examan", alors LD (M, P) = 1, parce qu il y a eu un_ remplacement (changement du e en a), et
y: t P = "examan", alors Lev(M, P) = 1, parce qu il y a eu une substitution (changement du e en a), et
x: que l on ne peut pas en faire ____________m___o______________________in_______s.
y: que l on ne peut pas en faire une transformation de M en P avec un moindre cout.
```

On peut donc attester que l'algorithme fonctionne, cet alignement semblant tout à fait correct.