	**Algorithmique**	**NSI T^{ale}**
	Programmation dynamique	Exos

*

1. Alignement de séquences

Dans cet exercice, on cherche à mesurer la distance entre deux chaînes de caractères. Un des usages importants en est la comparaison de séquences génétiques. Bien sûr, ces méthodes servent aussi aux correcteurs orthographiques, à la reconnaissance de mots lors de scan, etc.
La mesure la plus simple (**distance de Hamming**) est de compter simplement le nombre de caractères différents. Par exemple, `manger` et `mentir` sont à distance 3. On utilise cette distance par exemple dans l'algorithme k-nn.
On va ici utiliser une autre distance. A partir de deux mots, on construit un alignement de ces deux mots en insérant des blancs (notés avec le caractère undescore `_`). Avec ces blancs, on construit deux chaînes de la même longueur, sachant que deux blancs ne sont jamais alignés entre les deux mots. On cherche l'alignement donnant la ressemblance maximale entre les deux chaînes.
Dans cet exercice, on calculera le score de la manière suivante : si, au même endroit, deux caractères sont identiques, on ajoute un point ; si les caractères sont différents (y compris un caractère avec un `_`), on enlève un point.

Remarques :

- Dans le cas général, on utilise un tableau pour donner le score de chaque comparaison de caractères deux à deux.
- On pourrait tout aussi bien utiliser la distance d'édition (dite aussi distance de Levenshtein ou distance d'Ulam). Cette distance compte le nombre minimal de désaccords entre deux alignements, c'est-à-dire qu'on ne compte que les différences. On cherche alors le minimum plutôt que le maximum.

1. Sans tenir compte des accents ou autres caractères diacritiques), donner les scores des paires : `manger-mentir`, `pécheurs-écriture`, `niche-chien`, `algorithme-polyrythmie`, `génome-écriture`.
Écrire les alignements correspondants.

2. On va utiliser directement une méthode de programmation dynamique pour réaliser cet alignement de séquences. En effet, la méthode par force brute n'est pas facile à trouver (et est bien moins efficace).
Pour cela, on va construire un tableau où l'on va calculer par méthode ascendante (du bas vers le haut) tous les scores possibles, en travaillant sur un exemple, comme génome-énorme.
Le tableau aura autant de lignes que l'un des mots, auquel on peut rajouter un `_` soit en début, soit en fin.
De même pour le nombre de colonnes, avec le deuxième mot. L'objectif est de minimiser le nombre de pièces rendues pour un montant fixé.

	_	E	N	O	R	M	E
_	0						
G			-2				
E							
N							
O							
M					1		
E							

a) Pour initialiser le tableau, on compare le mot « _ » de longueur 1 avec tous les mots possibles.
Sur la première ligne :

- le score « _ »/« _ » est 0 : ce cas ne peut pas advenir.
- le score « _ »/« E » est -1.
- le score « _ »/« EN » est -2 : on rajoute un _ au premier mot, on calcule le score de « __ »/« EN ».
- etc.

Compléter la première ligne et la première colonne de ce tableau

b) A quelle comparaison correspond le -2 déjà présent dans le tableau (ligne G, colonne L) ? Idem avec le 1.

c) Pour compléter le tableau, on va se placer « au milieu », en rajoutant une lettre à un ou deux des mots dont le score est déjà calculé
On veut calculer le score de « GENO »/ «ENOR » à partir de mots plus petits.

i. Dans quelle case est-ce score ? De quelles cases du tableau peut-on parvenir ? Quelles sont les comparaisons correspondantes ?

ii. Vérifiez que le score de « GENO »/ «ENO » est 2, celui de « GEN »/« ENOR » est -1, celui de « GEN »/« ENO » est 0.

iii. On représente la partie du tableau où se passe le calcul O R

	O	R
N	GEN_/_ENO **0** ↘	GEN_/ENOR **-1** ↓
O	GENO/_ENO **2** →	GENO / ENOR ??

Calculer l'évolution du score, arrivant à GENO/ENOR, en partant de chacune des trois cases pouvant être à l'origine de cette comparaison. Quelle est la valeur finalement retenue ? Rajouter les _ manquants éventuellement.

iv. Reprendre les questions i à iii, pour construire la portion de tableau permettant d'obtenir le score de l'alignement GEN/EN. Quelle est la différence avec le cas précédent ?

	↘	↓
	→	GEN / EN ??

v. Finir de compléter le tableau. Que vaut le score d'alignement et où se trouve-t-il ?

d) Programmer la méthode précédente en Python. Donner la complexité de l'algorithme.
Résumé du d. :

- La 1ère ligne et 1ère colonne comprennent les valeurs `0`, `−1`, `−2`, etc.
- Pour calculer la valeur d'un élément du tableau, on prend le maximum de :
  - Valeur du dessus `− 1`
  - Valeur de gauche `− 1`
  - Valeur en diagonale dessus/gauche `± 1` suivant si la lettre à rajouter est identique ou non.

e) Est-il nécessaire de conserver tout le tableau en mémoire pour calculer le score d'alignement ? Justifier. On ne demande pas la programmation.

f) Comment peut-on retrouver l'alignement correspondant au meilleur score ? Expliquer la méthode grâce au tableau, et éventuellement la programmer.

2. La Ruée vers l'Or

Un mineur d'or se situe face à une falaise. Grâce à son détecteur ultra sophistiqué, il connaît la densité des filons face à lui. Son but est de collecter le maximum d'or. Il est limité par la manière dont il creuse ses galeries : il ne peut pas retourner en arrière, cela entraînerait un effondrement de la mine. Il va donc de la gauche vers la droite, soit tout droit, soit en diagonale en montant, soit en diagonale en descendant. Il peut partir de n'importe quelle hauteur.

Exemple :

$$\begin{bmatrix} 4 & 2 & 0 & 1 \\ 2 & 3 & 5 & 1 \\ 5 & 0 & 1 & 2 \\ 3 & 6 & 1 & 1 \end{bmatrix}$$

Le mineur a récupéré 14 brezoufs d'or
(le brezouf étant la monnaie du Brezoufland)

1) Donner une « récolte » optimale pour l'exemple ci-contre.

2) On appelle `Or(i, j)` la quantité maximale d'or que le mineur a pu récupérer en arrivant au coefficient $a_{i,j}$ dans la mine `A` qui représente le sol derrière la falaise. La mine `A` a pour dimension `m` (nombre de lignes) et `n` (nombre de colonnes). Le but de cette question est d'écrire un algorithme en programmation dynamique pour optimiser la quantité d'or récoltée au final, notée `Or_max`.

$$\begin{bmatrix} 6 & 4 & 0 & 0 & 7 & 5 \\ 4 & 8 & 0 & 7 & 5 & 2 \\ 4 & 4 & 6 & 6 & 8 & 2 \\ 5 & 0 & 1 & 3 & 3 & 6 \\ 6 & 6 & 0 & 1 & 7 & 8 \\ 0 & 2 & 8 & 5 & 8 & 8 \end{bmatrix}$$

Récolte ?

a) Que vaut `Or(i,0)` (en première colonne) ?

b) Comment trouve-t-on la quantité maximale d'or extractible ?

c) On suppose que `j ≠ 0`. D'où peut provenir le mineur lorsqu'il est en $a_{i,j}$? On donnera la réponse en exprimant les différentes possibilités, qui sont des coefficients de la mine, en fonction de `i` et `j`.

d) En déduire une formule qui donne `Or(i, j)` en fonction de plusieurs valeurs de `Or(?, j - 1)` . On rappelle que `Or(i, j)` est la valeur maximale. Appeler le professeur pour vérification de la formule.

e) Écrire un algorithme donnant la quantité maximale d'or que le mineur peut extraire. On essaiera de minimiser la mémoire utilisée.

f) Compléter l'algorithme précédent en reconstruisant le « tracé de la mine », c'est à dire les coordonnées `(i, j)` des coefficients de la mine par lesquels le mineur est passé. On partira de la fin, en utilisant uniquement `Or_max` et `A`. On écrira l'algorithme sur papier

g) Donner la complexité temporelle (et spatiale si possible) de votre algorithme.

h) Programmer ces algorithmes.

3) Complément pour ceux qui vont vite :

Cette fois-ci, le mineur part du sol (le haut de la matrice) et creuse soit horizontalement, soit verticalement, vers le bas uniquement (vous pouvez aussi essayer sans la contrainte d'aller vers le bas).

Il a de plus une contrainte supplémentaire : il ne peut pas creuser dans plus de 3 cases/secteurs/coefficients adjacents (on ne peut pas creuser un carré complet de 4 cases).

Résoudre le problème de la récolte maximale.

On pourra réfléchir au problème avec les exemples suivants :

$$\begin{bmatrix} 6 & 4 & 0 & 0 & 7 & 5 \\ 4 & 8 & 0 & 6 & 5 & 8 \\ 4 & 4 & 6 & 6 & 8 & 2 \\ 5 & 0 & 1 & 1 & 6 & 6 \\ 6 & 6 & 1 & 4 & 1 & 0 \\ 0 & 2 & 8 & 5 & 8 & 0 \end{bmatrix} \qquad \begin{bmatrix} 2 & 5 & 8 & 1 & 6 & 0 \\ 2 & 7 & 6 & 7 & 8 & 7 \\ 2 & 5 & 8 & 7 & 3 & 5 \\ 7 & 1 & 2 & 4 & 2 & 6 \\ 3 & 5 & 2 & 7 & 0 & 8 \\ 5 & 8 & 5 & 6 & 0 & 1 \end{bmatrix}$$