TD-TP
Programmation dynamique

I. Algorithmes de programmation dynamique

1. Plus longue sous-séquence commune

On considère deux séquences $X_m = (x_1, \ldots, x_m)$ et $Y_n = (y_1, \ldots, y_n)$.

En utilisant le principe d'optimalité de la programmation dynamique, on souhaite écrire l'algorithme qui permet de trouver la plus longue sous-séquence commune aux deux séquences X et Y. L'algorithme LCS a été vu en cours. Il est décrit par la relation de récurrence suivante :

$$LCS\left(X_m, Y_n\right) = \begin{cases} 0 & \text{si } \ n = 0 \text{ ou } m = 0 \\ LCS\left(X_{m-1}, Y_{n-1}\right) + 1 & \text{si } x_m = y_n \\ \max\left(LCS\left(X_m, Y_{n-1}\right), LCS\left(X_{m-1}, Y_n\right)\right) & \text{si } x_m \neq y_n \end{cases}$$

On avec un tel algorithme remarque qu'une ligne ne dépend pas de la précédente.

1.1. Quel est le nombre de sous-problèmes possibles ?
1.2. Dans un premier temps, vous traiterez l'exemple suivant, dans lequel on prend les 2 chaînes X = {C, A, T, G, T} et Y = {A, C, G, C, T, G}. L'algorithme consiste à remplir une matrice, vide initialement, que l'on remplit ligne par ligne.

1.3. Quelle est la plus longue sous-séquence commune aux deux chaînes ?

1.4. Écrivez en Python l'algorithme *construire_table*(x, y) permettant de remplir la matrice. Les paramètres x et y sont les deux séquences de caractères prises sur l'alphabet {A, C, G, T}.

1.5. Quelle est la complexité de cet algorithme ?

1.6. À partir de la matrice précédente, écrivez l'algorithme qui permet de déduire le chemin *lcs* obtenu en partant du coin en bas à droite et en allant vers le coin en haut à gauche.

1.7. Quelle est la complexité de l'algorithme précédent ?

2. Distance de Levenshtein et variantes

Soit la distance d'édition de Levenshtein définie par la formule de récurrence suivante :

$$LEV\left(X_m, Y_n\right) = \begin{cases} n & \text{si } \ m = 0 \\ m & \text{si } \ n = 0 \\ \min \begin{cases} LEV(X_{m-1}, Y_n) + 1 & \\ LEV(X_n, Y_{n-1}) + 1 & \\ LEV(X_{m-1}, Y_{n-1}) + 1 & \text{si } x_m \neq y_n \\ LEV(X_{m-1}, Y_{n-1}) & \text{sinon} \end{cases} & \end{cases}$$

2.1. Reprenez l'exemple précédent en suivant l'algorithme et remplissez la matrice.
2.2. Écrivez le programme Python permettant de calculer la distance entre deux chaines X et Y en utilisant la distance d'édition de Levenshtein.
2.3. On modifie l'algorithme en considérant que les coûts de substitution dépendent des caractères. On définit ainsi une matrice des coûts suivante :

	A	C	G	T
A	0	1	2	3
C		0	1	2
G			0	1
T				0

Modifiez l'algorithme en conséquence.

3. Calcul du produit de matrices par minimisation des coûts de calcul (exercice à ne pas faire ici)

On considère 4 matrice d'entiers M1, M2, M3 et M4 de dimensions respectives 100x1, 1x50, 50x20 et 20x1.

3.1. Le produit des matrices étant associatif, de combien de façons peut on associer les opérateurs x pour évaluer le produit M1… M4 ? Soit M et N deux matrices *n*x*p* et *p*x*q*. On définit le calcul du coût du produit MxN comme étant *npq*.
3.2. Ecrivez et testez l'algorithme EN PYTHON qui permet de calculer le coût minimal du calcul M1x… x M4.

II. Dynamique Time Warping

1. DTW appliquée à des signaux temporels de synthèse

On s'intéresse à 3 classes de signaux représentant de simples tracés en 2 dimensions. Pour chaque classe, on a la suite de points (*xi,* yi), pour *i* variant de 1 à *n* (récupérer l'archive DATA.zip). L'objectif de cet exercice est de proposer des méthodes pour classifier ces signaux temporels.

Deux méthodes sont proposées pour catégoriser ces tracés :

- Une distance euclidienne, entre les 2 séries de points appartenant aux classes *i* et *j*.

- Une distance basée sur l'algorithme de programmation dynamique DTW.

Chacune des méthodes permet de calculer une distance entre les signaux.

1.1. Distance Euclidienne.
   a. Ecrire une fonction Python qui charge en mémoire deux signaux s1 et s2 (vecteurs), les trace et calcule une matrice de coûts locaux *C* telle que *c(i,j) = d(s1(i), s2(j)),* pour *i*=1..*n* et *j*=1..*m*, *d* étant la distance Euclidienne ou une autre distance (Manhattan par exemple).
   b. Dessiner la matrice *C*
   c. Ecrire une fonction Python qui calcule pour chaque paire de signaux *sk, sl* la distance Euclidienne globale cumulée sur l'ensemble des échantillons de la série temporelle.

1.2. Distance élastique
1.3. Ecrire une fonction Python qui, à partir de deux signaux s1 et s2, permet de construire une matrice de distances à partir de l'algorithme de programmation dynamique suivant :

$$D(i,j) = c(i,j) + \min\{ D(i-1,j-1) , D(i-1,j) , D(i,j-1)$$

1.4. Dessiner la matrice issue de cet algorithme.
1.5. Tracer le plus court chemin.

1.6. Déterminer un représentant pour chaque classe de signaux. Donner les plus proches voisins de ce représentant, pour les 2 distances, Euclidienne et programmation dynamique.

2. DTW appliquée à des tracés en 2D

Reprendre les questions précédentes pour plusieurs classes de tracés obtenus à partir d'une tablette Wacom.