UE d'algorithmique (LU3IN003).
Licence d'informatique.

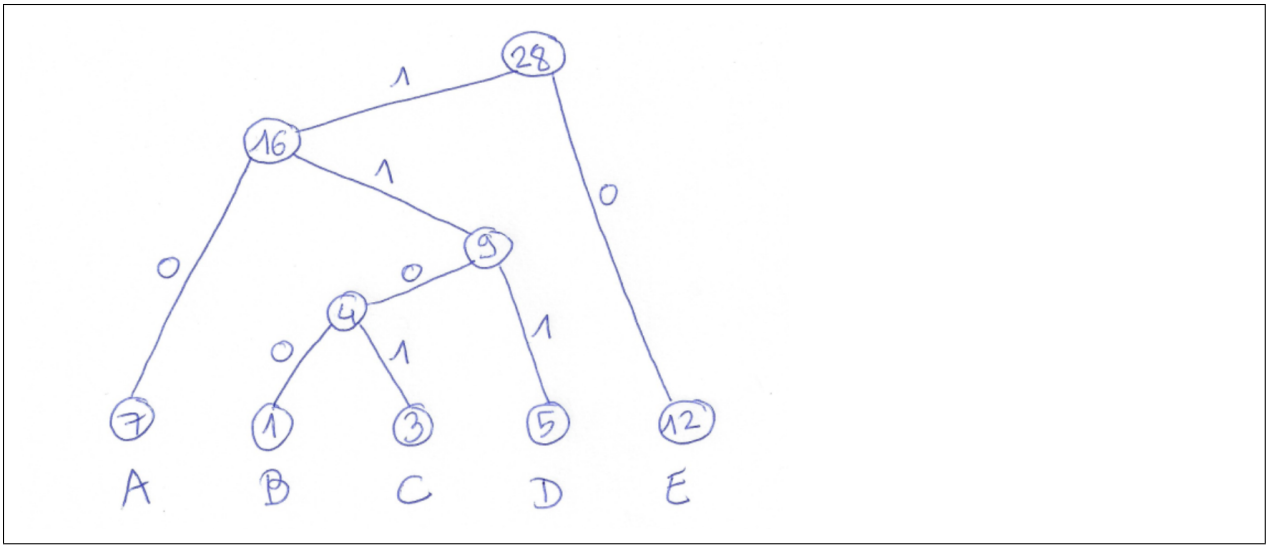
Examen du 20 décembre 2024.

Seule une feuille A4 portant sur les cours et les TD est autorisée, tout autre document est interdit. Téléphones portables éteints et rangés dans vos sacs. Le barème est indicatif et est susceptible d'être modifié. Toutes les réponses doivent être justifiées.

Exercice 1 (3 points)

Question 1 (1/3) — Donner le codage de Huffman des caractères A, B, C, D, E pour les fréquences ci-dessous. À chaque branchement de l'arbre, on affectera la valeur 0 (resp. 1) à la branche vers le fils de plus petite (resp. grande) étiquette.

A	B	C	D	E
7	1	3	5	12



Question 2 (1/3) — Coder la chaîne BAC avec le code trouvé dans la première question.

1100101101

Question 3 (1/3) — Décoder 11000111 avec le code trouvé dans la première question.

BED

Exercice 2 (3 points)

Supposons que l'on cherche un chemin de coût minimum entre deux sommets a et b dans un graphe connexe orienté $G = (S, A)$. On notera n le nombre de sommets et m le nombre d'arcs de G. Dans chacune des situations indiquées dans le tableau page suivante, vous indiquerez quel algorithme utiliser, en justifiant votre choix (si plusieurs algorithmes sont possibles, vous choisirez celui ayant la meilleure complexité temporelle). Les quatre premières colonnes du tableau caractérisent le type de graphe considéré. Par exemple, dans la première ligne, on considère des graphes dont les coûts des arcs sont identiques, positifs, qui peuvent comporter des circuits, et dont on ne connaît pas par avance le nombre maximum d'arcs dans un chemin de coût minimum. Pour chaque algorithme, vous indiquerez la complexité la plus précise possible.

Coûts des arcs identiques	Coûts des arcs positifs ou nuls	Existence de circuit dans G	Nombre maximum d'arcs dans un chemin de coût minimum	Algorithme préconisé (et justification du choix de cet algorithme)	Complexité de l'algorithme préconisé
oui	oui	oui	?	parcours en largeur (un parcours en largeur retourne un plus court chemin en nombre d'arêtes - et la complexité de cet algorithme est meilleure que celle des algorithmes de Dijkstra et de Bellman-Ford, que l'on peut aussi utiliser).	$O(n+m)$
non	oui	non	?	algorithme de Bellman (on peut utiliser cet algorithme car le graphe est sans circuit - et la complexité de cet algorithme est meilleure que celle des algorithmes de Dijkstra et de Bellman-Ford, que l'on peut aussi utiliser).	$O(n+m)$
non	oui	oui	$n-1$	algorithme de Dijkstra (on peut utiliser cet algorithme car les coûts sont positifs - et la complexité de cet algorithme est meilleure que celle de Bellman-Ford, en $O(mn)$, que l'on peut aussi utiliser)	$O(n^2)$ ou $O(m \log n)$ (on choisit le min des deux).
non	non	oui	$\log n$	algorithme de Bellman-Ford (limité à $\log n$ itérations). On ne peut pas utiliser les algorithmes de Dijkstra (coûts négatifs) ni de Bellman (présence de circuit), donc on choisit l'algorithme de Bellman-Ford. De plus $O(\log n)$ itérations suffisent car à l'itération i l'algorithme de Bellman-Ford a trouvé tous les chemins de coût min qui nécessitent au plus i arcs.	$O(m \log n)$

Coûts des arcs identiques	Coûts des arcs positifs ou nuls	Existence de circuit dans G	Nombre maximum d'arcs dans un chemin de coût minimum	Algorithme préconisé (et justification du choix de cet algorithme)	Complexité de l'algorithme préconisé
oui	oui	oui	?		
non	oui	non	?		
non	oui	oui	$n-1$		
non	non	oui	$\log n$		

Exercice 3 (8 points)

On dispose d'un groupe d'agents situés à différentes positions géographiques. Pour tout couple d'agent u et v, il est possible de connecter directement l'agent u et l'agent v, et ce avec un coût $c(\{u, v\})$. On s'intéresse au problème qui consiste à connecter l'ensemble des agents directement ou indirectement (c'est-à-dire en passant par d'autres agents) de façon à ce que le coût total de connexion soit minimum. Ce problème consiste donc à identifier un arbre couvrant de coût minimum dans le graphe non orienté G où chaque sommet représente un agent, chaque arête $\{u, v\}$ représente une connexion possible entre u et v, et le coût de l'arête est $c(\{u, v\})$.

On suppose que l'on dispose d'un arbre couvrant de coût minimum T pour le graphe $G = (S, A)$ valué par la fonction c. L'objet de cet exercice est de voir si après changement du coût de connexion entre deux agents, l'arbre T est toujours un arbre couvrant de coût minimum – et si ce n'est pas le cas, de calculer rapidement un arbre couvrant de coût minimum du graphe avec les nouvelles valuations. On note c' la nouvelle fonction de coût.

Question 1 (1/8) — On suppose dans cette question (et dans cette question seulement) que le coût de chaque arête est quintuplé : pour toute arête $\{x, y\} \in A$, on a $c'(\{u, v\}) = 5c(\{u, v\})$. L'arbre couvrant T est-il toujours un arbre couvrant de coût minimum avec ces nouvelles valuations ? Justifiez votre réponse.

T est toujours un arbre couvrant de coût minimum puisque le coût d'un arbre X avec la valuation c est quintuplé avec la valuation c' : un arbre couvrant de coût minimum avec c est un arbre couvrant de coût minimum avec c'.

On suppose maintenant qu'un seul coût de connexion – entre deux agents u et v – est modifié et on note x la valeur de ce nouveau coût.

Dans les questions 2 à 5, on considère que le coût de $\{u, v\}$ diminue. Ainsi, la nouvelle fonction de coût c' est définie de la manière suivante : $c'(\{u, v\}) = x < c(u, v)$ et pour toute arête e différente de $\{u, v\}$, $c'(e) = c(e)$.
Il s'agit de voir si l'arbre T est un arbre couvrant de coût minimum du graphe G valué par la fonction c', et si ce n'est pas le cas, de construire à partir de T un arbre couvrant de coût minimum T' pour le graphe G valué par la fonction de coût c'.

Question 2 (2.5/8) — On considère (dans cette question uniquement) le graphe G_1 de la figure 1.

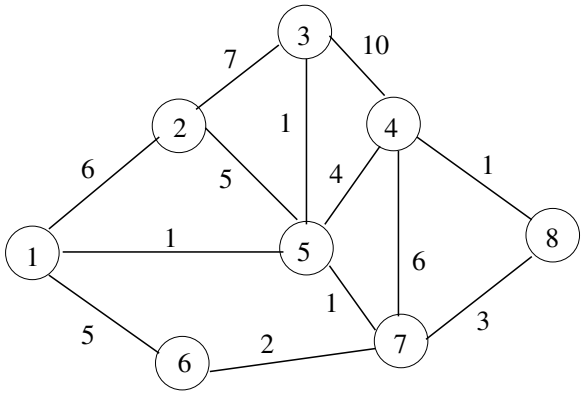


FIGURE 1 – Exemple de graphe de connexion entre agents

(a) Calculer un arbre couvrant de coût minimum pour G_1 (vous pouvez surligner les arêtes de l'arbre couvrant de coût minimum sur le graphe G_1, ou bien indiquer la liste des arêtes qui ont été sélectionnées). Quel algorithme avez-vous utilisé ? Indiquez l'ordre dans lequel les arêtes ont été ajoutées à l'arbre.

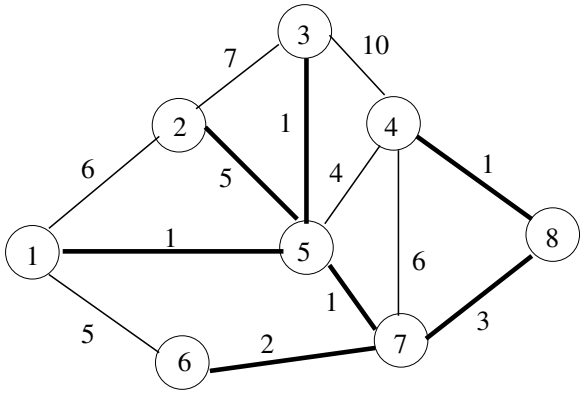


FIGURE 2 – Arbre couvrant de coût minimum de G_1

On peut par exemple utiliser l'algorithme de Prim ou l'algorithme de Kruskal pour déterminer un arbre couvrant de coût minimum (cf. arbre représenté par des arêtes en gras sur le graphe de la figure 2 de coût 14).

(b) On notera T_1 l'arbre couvrant de coût minimum obtenu dans la question précédente. Supposons que le coût de l'arête $\{6, 7\}$ passe de 2 à 1. T_1 est-il toujours un arbre couvrant de coût minimum ?

Oui.

(c) Supposons maintenant que le coût de l'arête $\{1, 2\}$ passe de 6 à 5. T_1 est-il toujours un arbre couvrant de coût minimum ?

Oui.

(d) Supposons maintenant que le coût de l'arête $\{1, 2\}$ passe de 6 à 4. T_1 est-il toujours un arbre couvrant de coût minimum ?

Non. En remplaçant l'arête $\{2, 5\}$ par l'arête $\{1, 2\}$ dans T_1, on obtient un arbre couvrant de coût inférieur au coût de T_1 ($13 < 14$).

Question 3 (1.5/8) — On suppose dans cette question que $\{u, v\} \in T$. Après diminution du coût de $\{u, v\}$, T est-il toujours un arbre couvrant de coût minimum ? Justifiez votre réponse.

Oui, T est toujours un arbre couvrant de coût minimum pour le graphe G valué par c', i.e. (G, c'). On le montre par l'absurde : supposons que T ne soit plus un arbre couvrant de coût minimum après diminution du coût de $\{u, v\}$, c.a.d dans (G, c') (le coût ayant diminué de δ). Soit T' un arbre couvrant de coût minimum dans (G, c'). Étant donné un arbre A on note $c(A)$ le coût de A dans (G, c) (avant changement du coût de $\{u, v\}$), et $c'(A)$ le coût de A dans (G, c') (après changement du coût de $\{u, v\}$). On a : $c'(T') < c'(T)$ (car T' est un arbre couvrant de coût minimum, ce qui n'est pas le cas de T dans (G, c')). On a $c'(T) = c(T) - \delta$, et $c'(T') \geq c(T') - \delta$ (seul le coût de $\{u, v\}$ a changé). Donc $c(T') \leq c'(T') + \delta < c'(T) + \delta \leq c(T) - \delta + \delta$. D'où $c(T') < c(T)$: T n'était donc pas un arbre couvrant de coût minimum dans (G, c). Contradiction.

Question 4 (1/8) — On suppose maintenant que $\{u, v\} \notin T$. Dans quel cas T est-il toujours un arbre couvrant de coût minimum après diminution du coût de $\{u, v\}$? On ne demande pas de justification.

Soit γ la chaîne de u à v dans T. T est toujours un arbre couvrant de coût minimum dans (G, c') si le coût $c(e)$ de chaque arête e de γ satisfait la condition $c(e) \leq x$.

Question 5 (1/8) — En déduire le principe d'un algorithme qui permet de construire un nouvel arbre couvrant de coût minimum T' à partir de T. Quelle est la complexité de cet algorithme ? Cet algorithme est-il plus efficace que l'algorithme utilisé à la question 1(a) ?

Pour obtenir un arbre de coût minimum de (G, c') à partir de T :
- Si $\{u, v\}$ appartient à T, alors T est un arbre couvrant de coût minimum de (G, c').
- Si $\{u, v\}$ n'appartient pas à T. Soit γ la chaîne de u à v dans T.
  - Si $c(e) \leq x$ pour toute arête e de γ, alors T est un arbre couvrant de coût minimum de (G, c'). Sinon l'arbre de coût minimum de (G, c') est obtenu à partir de T en supprimant l'arête de coût maximum de γ et en ajoutant l'arête $\{u, v\}$.

Si on ne tient pas compte du coût du calcul de l'arbre T, la complexité de l'algorithme de calcul de T' est $O(n)$ (ceci étant dû à la recherche de la chaîne entre u et v dans T, en faisant un parcours de racine u si T est codé par une liste d'adjacence). Il est donc moins coûteux de calculer T' à partir de T plutôt que de recalculer un arbre couvrant de coût minium dans (G, c') (coût $O(m \log n)$ ou $O(m \log m)$).

On considère maintenant que le coût de $\{u, v\}$ augmente. Ainsi, la nouvelle fonction de coût c' est définie de la manière suivante : $c'(\{u, v\}) = x > c(\{u, v\})$ et pour toute arête e différente de $\{u, v\}$, $c'(e) = c(e)$.

Question 6 (1/8) — Dans quel cas T est-il toujours un arbre couvrant de coût minimum après augmentation du coût de $\{u, v\}$? Vous distinguerez le cas où $\{u, v\} \in T$ du cas où $\{u, v\} \notin T$. On ne demande pas de justification.

Si $\{u, v\} \notin T$ alors T est toujours un arbre couvrant de coût minimum.
Si $\{u, v\} \in T$ alors soit T_u et T_v les deux composantes connexes obtenues quand on retire $\{u, v\}$ à T. T est un arbre couvrant de coût minimum de T si et seulement si il n'y a pas d'arête de coût inférieur à x dans le co-cycle de T_u.

Exercice 4 (6 points)

L'ADN, ou acide désoxyribonucléique, est le support de l'information génétique chez tous les organismes vivants. Cette molécule est une double hélice caractérisée par l'alternance de bases : l'adénine (notée A), la thymine (notée T), la cytosine (notée C), la guanine (notée G). Dans cet exercice, nous allons travailler sur l'alignement de deux séquences d'ADN, c'est-à-dire de deux chaînes de caractères définies sur l'alphabet $\{A, T, C, G\}$. L'objectif est de comparer ces deux séquences afin d'identifier les parties où les bases (A, T, C, G) correspondent. Ce type d'analyse est essentiel en bio-informatique pour étudier les relations entre des séquences, détecter des mutations ou comprendre les mécanismes évolutifs.

Pour réaliser cet alignement, nous pouvons introduire des "trous" dans les séquences. Un trou dans une séquence est représenté par le symbole $-$. Ces trous représentent des bases manquantes ou ajoutées, permettant de mieux aligner les parties communes entre les deux séquences. Un alignement est présenté sous forme d'une grille où chaque colonne correspond à une base (ou un trou) dans les deux séquences. Voici par exemple un alignement possible des séquences AGTCG et AGTAG, alignement que l'on note $(AGT - CG, AG - TAG)$:

$$\begin{array}{cccccc} A & G & T & - & C & G \\ A & G & - & T & A & G \end{array}$$

L'évaluation d'un alignement s'appuie sur l'identification des trous, des correspondances et des différences entre les chaînes (séquences) **après alignement** :

$$\begin{array}{lcccccc} \text{colonnes :} & 1 & 2 & 3 & 4 & 5 & 6 \\ \text{séquence 1 :} & A & G & T & - & C & G \\ & | & | & & & \times & | \\ \text{séquence 2 :} & A & G & - & T & A & G \end{array}$$

Précisons que, pour qu'un alignement soit valide, les deux chaînes obtenues suite à l'introduction (éventuelle) de caractères "−" dans les deux séquences doivent être de même longueur. Dans cet exemple, nous avons :

— 2 trous : colonne 3 et colonne 4.
— 3 correspondances : colonne 1, 2 et 6.
— 1 différence : colonne 5.

Pour évaluer un alignement, on attribue un score (coût) à chaque type d'appariement dans l'alignement :

— Correspondance : score s_C.
— Différence : score s_D.
— Trou : score s_T.

Le score d'un alignement comportant n_C correspondances, n_D différences, et n_T trous est égal à $n_C \times s_C + n_D \times s_D + n_T \times s_T$. Par exemple, en fixant $s_C = 0, s_D = 3, s_T = 2$, on trouve un score de 7 pour l'alignement précédent :

$$\begin{array}{cccccc} A & G & T & - & C & G \\ | & | & & & \times & | \\ A & G & - & T & A & G \\ 0 & 0 & 2 & 2 & 3 & 0 \end{array}$$

On supposera dans la suite de l'exercice que les valeurs s_C, s_D et s_T sont positives quelconques, et on notera $S(X, Y)$ le score d'un alignement optimal de X et Y (un alignement est optimal s'il conduit à un score minimal). Pour l'exemple ci-dessus, un alignement optimal est (AGTCG, AGTAG), de score 3.

Question 1 (0.5/6) — Soit une séquence $X = x_1x_2 \dots x_n$ (avec $x_i \in \{A, T, C, G\}$). Quel est l'unique alignement possible entre X et une chaîne vide noté $\emptyset$? En déduire l'expression de $S(X, \emptyset)$ en fonction de s_T.

Une séquence vide n'a aucune correspondance possible, chaque base alignée correspond à un trou. Le seul moyen d'aligner une séquence vide avec une séquence de taille n consiste a insérer n trous, ce qui donne un score de $S(X, \emptyset) = |X| \cdot s_T = n \cdot s_T$.

Question 2 (1.5/6) — Dans ce qui suit, on notera $X_i = x_1x_2 \dots x_i$ la sous-chaîne de X constituée des i premiers caractères de X, et $Y_j = y_1y_2 \dots y_j$ la sous-chaîne de Y constituée des j premiers caractères de Y. En décomposant le problème d'alignement des i premières bases (caractères) de X avec les j premières bases de Y en sous-problèmes plus petits, donner l'expression :

a) du score obtenu en alignant x_i et y_j (c'est-à-dire que y_j est la base faisant face à x_i, en dernière position de l'alignement) ; vous distinguerez les cas $x_i = y_j$ et $x_i \neq y_j$.
b) du score obtenu en alignant y_j avec un trou (c'est-à-dire le caractère −), en dernière position de l'alignement.
c) du score obtenu en alignant x_i avec un trou (c'est-à-dire le caractère −), en dernière position de l'alignement.

Vous pourrez utiliser la fonction $Eq(x_i, y_j)$ qui renvoie 1 si $x_i = y_j$, et 0 sinon.

a) si $Eq(x_i, y_j)$ alors $S(X_{i-1}, Y_{j-1}) + s_C$ sinon $S(X_{i-1}, Y_{j-1}) + s_D$.
b) $S(X_i, Y_{j-1}) + s_T$
c) $S(X_{i-1}, Y_j) + s_T$

Question 3 (1/6) — Déduire de la réponse à la question précédente la formule de récurrence permettant de calculer le score optimal $S(X_i, Y_j)$.

Si $Y_j = \emptyset, S(X_i, Y_j) = i \cdot s_T$.
Si $X_i = \emptyset, S(X_i, Y_j) = j \cdot s_T$.
Si $X_i \neq \emptyset$ et $Y_j \neq \emptyset$:

$$S(X_i, Y_j) = \min \begin{cases} S(X_{i-1}, Y_{j-1}) + Eq(x_i, y_j) \cdot s_C + (1 - Eq(x_i, y_j)) \cdot s_D & \text{(Alignement direct)} \\ S(X_i, Y_{j-1}) + s_T & \text{(Trou dans } X\text{)} \\ S(X_{i-1}, Y_j) + s_T & \text{(Trou dans } Y\text{)} \end{cases}$$

Question 4 (2/6) — Écrire un algorithme qui calcule pour deux chaînes X, Y de longueurs n et m le *score* d'un alignement optimal. Quelle est sa complexité ?

Remarque : on suppose ci-dessous que le premier indice du tableau est 0.

Entrée : Deux chaînes X de taille n et Y de taille m.
Scores : s_C (correspondance), S_D (différence), s_T (trou).
Sortie : Le score d'alignement optimal.

1. **Initialisation :**
   — Créer une matrice S de dimensions $(n+1) \times (m+1)$.
   — Initialiser la première ligne et la première colonne pour tenir compte des trous :

$$S(i, 0) = i \cdot s_T \quad \text{pour } i \in [1, n]$$

$$S(0, j) = j \cdot s_T \quad \text{pour } j \in [1, m]$$

$$S(0, 0) = 0$$

2. **Remplissage de la matrice S :** pour chaque $i \in [1, n]$ et $j \in [1, m]$:
   — Calculer les scores des trois options :

$$\text{Diag} = S(X_{i-1}, Y_{j-1}) + Eq(x_i, y_j) \cdot s_C + (1 - Eq(x_i, y_j)) \cdot s_D$$

$$\text{Left} = S(i, j-1) + s_T$$

$$\text{Up} = S(i-1, j) + s_T$$

   — Mettre à jour $S(i, j)$ avec le minimum des trois :

$$S(i, j) = \min(\text{Diag}, \text{Left}, \text{Up})$$

3. **Retour** : l'algorithme retourne $S(n, m)$.

Complexité :
— **Temps :** $O(n \cdot m)$ pour remplir la matrice S.
— **Espace :** $O(n \cdot m)$ pour stocker la matrice S.

Question 5 (1/6) — Écrire un algorithme qui retourne un *alignement* optimal de X et Y. Vous pourrez expliquer comment adapter l'algorithme de la question précédente, ou proposer une fonction à appeler au terme de l'algorithme de la question précédente pour déterminer cet alignement à partir de la matrice des scores $S(X_i, Y_j)$.

Fonction : `AlignSequences(X, Y, S, i, j,` $\hat{X}$`,` $\hat{Y}$`)`
Entrée :
- X, Y : Les séquences originales.
- S : La matrice des scores d'alignement.
- i, j : Les indices actuels dans X et Y.
- $\hat{X}, \hat{Y}$: Les alignements partiels des séquences (listes vides au départ).

Sortie : Les alignements complets $\hat{X}$ et $\hat{Y}$.

1. **Si** $i = 0$ et $j = 0$:
   - Retourner `Reverse(`$\hat{X}$`)` et `Reverse(`$\hat{Y}$`)`.
2. **Sinon si** $i > 0$ et $j > 0$ et $S[i][j] = S[i-1][j-1] + \text{score}(X[i-1], Y[j-1])$:
   - Ajouter $X[i-1]$ à $\hat{X}$.
   - Ajouter $Y[j-1]$ à $\hat{Y}$.
   - Appeler récursivement `AlignSequences(X, Y, S, i-1, j-1,` $\hat{X}$`,` $\hat{Y}$`)`.
3. **Sinon si** $i > 0$ et $S[i][j] = S[i-1][j] + s_T$:
   - Ajouter $X[i-1]$ à $\hat{X}$.
   - Ajouter `"-"` à $\hat{Y}$.
   - Appeler récursivement `AlignSequences(X, Y, S, i-1, j,` $\hat{X}$`,` $\hat{Y}$`)`.
4. **Sinon** :
   - Ajouter `"-"` à $\hat{X}$.
   - Ajouter $Y[j-1]$ à $\hat{Y}$.
   - Appeler récursivement `AlignSequences(X, Y, S, i, j-1,` $\hat{X}$`,` $\hat{Y}$`)`.

Appel initial : `AlignSequences(X, Y, S, n, m, [], [])`, où n = longueur de X et m = longueur de Y.