EPU GB 4ème année / Master SVS – BIM *Algorithmique pour la biologie*

Examen du 18 mai 2020 Devoir à renvoyer par mail.

Inscrivez **lisiblement** vos Nom et Prénom en tête de vos copies.

Exercice 1 : (Tri à bulles – 5 points)

Le *tri à bulles* ou *tri par propagation* est un algorithme de tri qui consiste à faire remonter progressivement les plus petits éléments d'un tableau, comme les bulles d'air qui remontent à la surface d'un liquide.

L'algorithme parcourt le tableau, et compare les couples d'éléments successifs. Lorsque deux éléments successifs ne sont pas dans l'ordre croissant, ils sont échangés. Après chaque parcours complet du tableau, l'algorithme recommence l'opération. Lorsqu'aucun échange n'a lieu pendant un parcours, cela signifie que le tableau est trié. On arrête alors l'algorithme.

Prenons un exemple. Soit la liste « 5 1 4 2 8 ».

Première étape : (on va du 1er élément à l'avant dernier)
(**51**428) → (**15**428) Les éléments 5 et 1 sont comparés, et comme $5 > 1$, l'algorithme les intervertit.
(1**54**28) → (1**45**28) Interversion car $5 > 4$.
(14**52**8) → (14**25**8) Interversion car $5 > 2$.
(142**58**) → (142**58**) Comme $5 < 8$, les éléments ne sont pas échangés.
Deuxième étape : (on va du 1er au « 2ème en partant de la fin »)
(**14**258) → (**14**258) Même principe qu'à l'étape 1.
(1**42**58) → (1**24**58)
(12**45**8) → (12**45**8)
À ce stade, la liste est triée, mais pour le détecter, l'algorithme doit effectuer un dernier parcours.
Troisième étape : (on va du 1er au « 3ème en partant de la fin »)
(**12**458) → (**12**458)
(1**24**58) → (1**24**58)

Comme la liste est triée, aucune interversion n'a lieu à cette étape, ce qui provoque l'arrêt de l'algorithme.

1. Ecrire une fonction python qui implémente le tri à bulle [4 points].
2. Donner la complexité de l'algorithme. Justifiez votre réponse [1 point].

Exercice 2 : (Prog. Dynamique, la plus longue sous-séquence palindromique – 18 points)

Une sous-séquence d'une séquence s donnée est une séquence de lettres qui apparaissent dans s dans le même ordre. Par exemple, pour s ="BBABCBCAB", les mots "BBBBB" et "ACCA" sont des sous-séquences, mais le mot "AABB" n'en est pas une. Une séquence *palindromique* est une séquence que l'on peut lire indifféremment de gauche à droite et de droite à gauche."BBBBB" et "ACCA" sont palindromiques.

Le but de l'exercice est de construire un algorithme permettant de chercher, pour une chaîne de caractères donnée, la plus longue sous séquence palindromique. Par exemple, pour la séquence "BBABCBCAB", la plus longue sous sous-séquence palindromique est de longueur 7 : "BABCBAB". Les sous-séquences "BBBBB" et "BBCBB" sont aussi palindromiques, mais ne sont pas de longueur maximale.

1. [7 points] La solution naïve pour ce problème est de générer toutes les sous-séquences de la séquence donnée, et d'en déduire la plus longue sous-séquence palindromique.
   (a) Ecrire une fonction python qui teste si une séquence est palindromique.
   (b) Ecrire une fonction qui construit à partir d'une séquence toutes les sous-séquences. Par exemple, pour la sequence ‘‘abc’’ , la fonction renvoie `["","a","b","c","ab","ac","bc","abc"]`.
   On remarquera que les sous-séquences d'une séquence de longueur 1, sont la chaîne vide et la séquence elle-même ; et que, pour calculer les sous-séquences d'une séquence de longueur n, on peut calculer les sous-séquences de la séquence privée de sa première lettre, et ajouter à cette liste, les mêmes sous-séquences auxquelles on aura ajouté la première lettre.

(c) Ecrire en python la fonction qui prend en entrée une séquence et qui renvoie une sous-séquence palindromique de longueur maximale.

(d) Montrer que la complexité de cette solution est au moins en $O(2^n)$ où n est la longueur de la séquence initiale.

2. [5 points] Nous allons maintenant éccrire un algorithme récursif pour résoudre ce problème. Pour cela, on introduit la fonction $L(i, j)$ qui représentera la longueur de la plus longue sous-séquence palindromique de la sous-séquence $S[i, j]$.

   (a) Ecrire une version <u>*récursive*</u> pour calculer la longueur de la plus longue sous-séquence palindromique. Pour cela on remarquera que :
   - toute séquence de longueur 1 est palindromique, autrement dit $L(i, i) = 1$ pour tout i,
   - toute séquence de longueur 2 est palindromique si les deux lettres sont identiques,
   - pour toute séquence de longueur supérieure ou égale à 3, la plus longue sous-séquence palindromique dépend de l'égalité de la première et dernière lettre.
     - Si les lettres aux extrémités sont identiques, alors la longueur de la plus longue sous-séquence palindromique de $S[i, j]$ est égale à 2 plus la longueur de la plus longue sous-séquence palindromique de $S[i + 1, j - 1]$.
     - Si les lettres sont différentes, alors la longueur de la plus longue sous-séquence palindromique de $S[i, j]$ est soit égale à la longueur de la plus longue sous-séquence palindromique de $S[i + 1, j]$, soit égale à la longueur de la plus longue sous séquence palindromique de $S[i, j - 1]$.

   Ne pas oublier les cas de base.

   (b) Montrer sur un exemple que les sous problèmes ne sont pas indépendants.

3. [6 points] Nous allons maintenant utiliser la programmation dynamique pour calculer la longueur de la plus longue sous-séquence palindromique.

   (a) Quelles sont les valeurs de i et j pour lesquelles on peut déterminer, sans calcul, la valeur de $L(i, j)$?

   (b) Ecrire un algorithme itératif qui stocke dans une structure de données toutes les valeurs $L(i, j)$.

   (c) En déduire la complexité de l'algorithme.