Master 1 SVS – BIM *Algorithmique pour la biologie*

Examen du 10 décembre 2015 Documents non autorisés.

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.
(**154**28) → (14**52**8) Interversion car $5 > 4$.
(14**52**8) → (142**58**) 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 : (**Programmation Dynamique et le père Noël – 15 points**)

Cette année, la mère Noël s'est exceptionnellement proposée pour aider le père Noël. Ils vont donc devoir à eux deux passer par toutes les cheminées pour déposer leurs cadeaux. Ils aimeraient pouvoir réduire la longueur de leurs trajets. Le problème est le suivant : étant donnés un point de départ x_0 et n cheminées $x_1, \ldots, x_n$, trouver deux chemins $p_0, p_1, \ldots, p_{n_p}$ et $m_0, m_1, \ldots, m_{n_m}$ avec $p_0 = m_0 = x_0$ et $\{x_1, \ldots, x_n\} = \{p_1, \ldots, p_{n_m}\} \cup \{m_1, \ldots, m_{n_m}\}$ qui minimisent

$$\sum_{i=1}^{n_p} d(p_{i-1}, p_i) + \sum_{j=1}^{n_m} d(m_{j-1}, m_j)$$

où $d(x_i, x_j)$ désigne la distance entre x_i et x_j.

En fait, pour profiter du décalage horaire, le père Noël a l'habitude de voyager plutôt d'est en ouest, en partant de la ligne de changement de date. Pour simplifier le problème, nous allons donc supposer que ni le père Noël, ni la mère Noël ne pourront revenir vers l'est, et que la prochaine cheminée à visiter est donc la cheminée la plus à l'est parmi celles qu'il reste à visiter. De plus, nous supposons notre connaissance des positions des cheminées si précise que 2 cheminées n'ont jamais la même longitude.

1. Expliquez brièvement pourquoi il est intéressant de considérer que les x_i sont triés par longitude décroissante.

2. Supposons que le père Noël ait visité en dernier la cheminée x_i et la mère Noël la cheminée x_j. La prochaine cheminée à visiter est la cheminée x_k avec $k = max(i, j) + 1$. Nous nous trouvons dans un sous-problème du problème initial, sous-problème que l'on note $\mathcal{L}(i, j, [k..n])$: il s'agit de la somme des longueurs des chemins optimaux du père et de la mère Noël, partant de la situation où le père Noël est à la cheminée x_i et la mère Noël à la cheminée x_j. En particulier, $\mathcal{L}(0, 0, [1..n])$ est la longueur de la solution optimale du problème initial.

   Exprimez $\mathcal{L}(i, j, [k..n])$ en fonction des distances $d(*, *)$ et d'expressions de la forme $\mathcal{L}(*, *, [k+1..n])$. Il s'agit de remplacer les $*$ par des valeurs pertinentes.

3. Déterminez les cas de base de la récurrence. Ils correspondent au cas où il n'y a plus de cheminée à visiter.

4. Ecrivez un programme python qui résoud le problème de manière récursive.

5. Expliquez pourquoi l'algorithme récursif n'est pas efficace dans ce cas (on ne déterminera pas avec précision la complexité de l'algorithme de la question précédente).

6. Décrivez un algorithme de programmation dynamique qui permet de trouver la longueur optimale des chemins pour le problème posé. On utilisera la notation $d(x_i, x_j)$ pour désigner la distance entre les sommets x_i et x_j.

   Quelle est la complexité de cet algorithme (en nombre de calcul de distance $d(x_i, x_j)$) ?

7. Comment à la suite de l'algorithme précédent peut-on trouver les deux chemins $p_0, p_1, \ldots, p_{n_p}$ et $m_0, m_1, \ldots, m_{n_m}$? Quelle est la complexité de cette étape ?