Le Réchauffement Climatique

Épreuve pratique d'algorithmique et de programmation
Concours commun des écoles normales supérieures

Durée de l'épreuve: 3 heures 30 minutes

Juillet 2007

ATTENTION !

N'oubliez en aucun cas de recopier votre u_0
à l'emplacement prévu sur votre fiche réponse

Important.
Lorsque la description d'un algorithme est demandée, vous devez présenter son fonctionnement de façon schématique, courte et précise. Vous ne devez en aucun cas recopier le code de vos procédures!
Quand on demande la complexité en temps ou en mémoire d'un algorithme en fonction d'un paramètre n, on demande l'ordre de grandeur en fonction du paramètre, par exemple: $O(n^2)$, $O(n \log n)$,...
Il est recommandé de commencer par lancer vos programmes sur de petites valeurs des paramètres et de ***tester vos programmes sur des petits exemples que vous aurez résolus préalablement à la main***.

1 Introduction

Comme chacun sait, le climat se réchauffe, ce qui a des conséquences sur la montée des eaux des océans. Les experts scientifiques mandatés par l'ONU aimeraient étudier plus précisement quelles villes seraient impactées dans les divers scénarios qu'ils envisagent. Pour ce faire, ils disposent de la liste des villes, et pour chacune, de l'intervalle fermé d'altitude sur lequel elle s'étend.
Ils aimeraient en particulier pouvoir déterminer, pour une prévision de montée des eaux à une altitude donnée, les villes qui seraient partiellement inondées, c'est-à-dire où le niveau d'eau serait compris dans l'intervalle d'altitude.

Nous allons donc étudier des structures de données permettant de répondre à ces questions de manière efficace, à savoir les listes à enjambements et des extensions pour la recherche d'intervalles. Ces structures permettent de faire des recherches rapides dans des ensembles d'objets triés, et elles peuvent être mises à jour facilement lors d'insertions de nouveaux éléments.

Les réponses à certaines questions ne peuvent être calculées que si vos algorithmes sont efficaces. Il est également important de suivre la méthode recommandée pour l'implémentation, car elle sera fondamentale dans la suite du sujet.

2 Recherche simple

Nous allons tout d'abord programmer la liste à enjambements classique, qui permet de faire des recherches rapides dans un ensemble d'éléments triés. L'idée de départ est simple. Considérons une liste simplement chaînée de n éléments triés en ordre croissant. Dans une telle liste, rechercher un élément nécessitera au pire n comparaisons. Insérer un élément prend le temps d'une recherche, plus un temps constant de mise à jour de la liste.
Considérons maintenant l'idée suivante : afin d'accélérer la recherche, nous ajoutons aux pointeurs classiques de la liste, une série de pointeurs qui permettent de faire des sauts plus longs dans la liste. Pour commencer, les éléments de rang pair dans la liste seront connectés eux aussi sous forme de sous-liste chaînée. De cette façon, la recherche peut maintenant commencer par une première étape de recherche dans cette sous-liste, qui ne compte que $n/2$ éléments, puis descendre dans la liste initiale où il n'y a plus qu'une seule comparaison à effectuer.
De manière récursive, cette sous-liste peut elle-même avoir une sous-liste, etc. On dit que la liste principale est de niveau 0, sa première sous-liste de niveau 1, etc.
Pour $n = 2^k$ (pour k entier), on obtient donc $k + 1$ listes de niveaux $0...k$. On définit le niveau d'un élément comme étant celui de la liste de plus haut niveau qui le contient. On peut aussi remarquer que la liste de niveau k contient exactement les éléments de niveau au moins k.

La figure 1 donne un exemple d'une telle structure de donnée.

Question à développer pendant l'oral : Quelle est la complexité en temps d'une recherche ?

Le problème de cette structure est la difficulté de sa mise à jour lors d'insertions de nouveaux éléments. Pour améliorer cet aspect, nous allons conserver la propriété importante

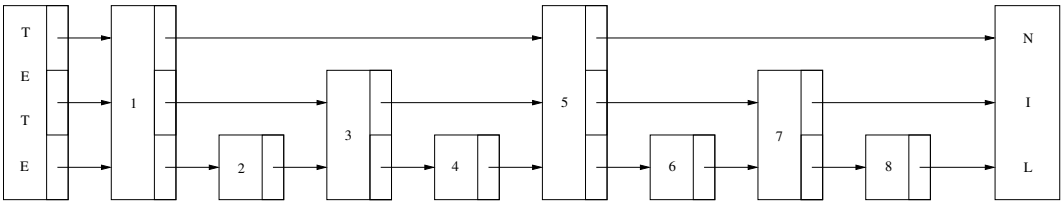

FIG. 1 – Exemple de structure de recherche fixe contenant les éléments $\{1...8\}$ de niveaux respectifs $\{2, 0, 1, 0, 2, 0, 1, 0\}$. La sous-liste de niveau 1 contient les éléments $\{2, 4, 6, 8\}$.

de la structure, à savoir le nombre exponentiellement décroissant d'éléments par niveau, mais au lieu d'avoir des sous-listes régulièrement espacées difficiles à maintenir, nous allons randomiser la structure, en utilisant une distribution aléatoire des éléments à chaque niveau. Ainsi, lors de l'insertion d'un élément, son niveau sera choisi aléatoirement selon une distribution adaptée, et il conservera toujours ce niveau quels que soient les éléments insérés plus tard.
Pour ce faire, nous choisissons donc un générateur aléatoire de niveaux qui fournisse une distribution des tailles de chaque niveau qui décroisse exponentiellement. En pratique, on bornera le niveau maximum généré par la valeur 32 afin de simplifier la programmation.

La figure 2 donne un exemple d'une telle structure de donnée.

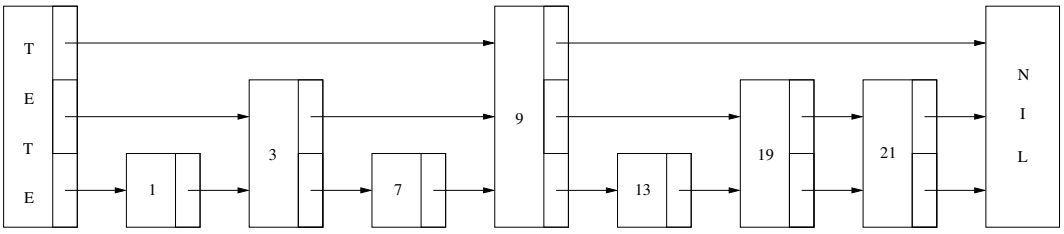

FIG. 2 – Exemple de liste à enjambements contenant les éléments $\{1, 3, 7, 9, 13, 19, 21\}$ de niveaux respectifs $\{0, 1, 0, 2, 0, 1, 1\}$. La sous-liste de niveau 1 contient les éléments $\{3, 9, 19, 21\}$.

3 Suite pseudo-aléatoire de niveaux

Considérons les suites d'entiers (u_n), (v_n) et (w_n) définies pour $n \geq 0$ par :

$$u_n = \begin{cases} \textit{votre } \mathtt{u_0} \textit{ (à reporter sur votre fiche)} & \text{si } n = 0 \\ 15\,091 \times u_{n-1} \mod 64\,007 & \text{si } n \geq 1 \end{cases}$$

$$v_n = \begin{cases} \mathtt{u_0} & \text{si } n = 0 \\ 1\,129 \times v_{n-1} \mod 15\,193 & \text{si } n \geq 1 \end{cases}$$

$$w_n = u_n + 64007 \times v_n$$

Attention : On choisira u_0 strictement positif et inférieur ou égal à 15 192.

Question 1 *Que valent :* ***a)*** w_{10} ***b)*** $w_{1\,000}$ ***c)*** $w_{100\,000}$

Étant donné un réel $p \in [0; 1]$, on considère la suite (p_n) à valeurs dans $\{0, 1\}$ définie par :

$$p_n = \begin{cases} 0 & \text{si } w_n < p \times (64007 \times 15193) \\ 1 & \text{sinon} \end{cases}$$

Dans la suite, nous considèrerons $p = \frac{1}{2}$.

Nous définissons maintenant la suite (l_n). Considérons les valeurs $\{p_{32n+i}\}$, pour i dans $\{0, ..., 31\}$. l_n est définie comme étant la valeur de i la plus petite pour laquelle la valeur de $\{p_{32n+i}\}$ vaut 1, ou 32 si aucune de ces valeurs ne vaut 1.
La suite (l_n) va servir de générateur pseudo-aléatoire de niveaux pour l'insertion de nouveaux éléments dans la liste à enjambements.

Question 2 *Que valent :* ***a)*** l_{10} ***b)*** $l_{1\,000}$ ***c)*** $l_{100\,000}$

Question 3 *Distribution des valeurs de* (l_n) *: parmi* $\{l_0, ..., l_{10\,000}\}$*, combien de fois apparaissent les valeurs :* ***a)*** *1* ***b)*** *3* ***c)*** *10*

Question à développer pendant l'oral : Quelle est la distribution des valeurs de (l_n) et qu'est-ce que cela suggère sur la complexité en moyenne de l'insertion d'un élément dans la liste à enjambements ?
Question à développer pendant l'oral : Dérandomisation : il est parfois intéressant d'introduire du déterminisme dans la structure de données, sans toutefois perdre les propriétés de distribution des niveaux. Proposez un moyen simple qui permette de garantir que le niveau d'un élément ne dépende que de sa valeur (on ne demande pas de le programmer).

4 Structure de recherche classique

Nous demandons maintenant de programmer la structure de données décrite, ainsi que la procédure de recherche d'un élément et celle d'insertion. La procédure de recherche retournera les noeuds de chaque niveau dont la valeur précède la valeur recherchée, de sorte à pouvoir être utilisée par la procédure d'insertion. À chaque insertion d'un élément, une valeur de la suite (l_n) sera utilisée comme niveau d'insertion (w_i aura donc le niveau l_i, dans les questions de cette section 4). Si un élément à insérer existe déjà dans la structure, il n'y sera stocké qu'une seule fois.

Pour tester, nous allons tout d'abord utiliser la structure pour y insérer successivement les valeurs $\{w_0, ..., w_{10}\}$. On suggère fortement de vérifier le contenu de la structure sur ce petit exemple (en affichant par exemple le contenu de chaque sous-liste après chaque insertion).

Question 4 *Dans la liste à enjambements ainsi obtenue, quel est le rang des éléments suivants (le premier élément étant de rang 0), dans la liste principale de niveau 0 ?* ***a)*** w_0 ***b)*** w_1 ***c)*** w_{10}

Utilisez maintenant la structure pour y insérer successivement les valeurs $\{w_0, ..., w_{100\,000}\}$.

Question 5 *Dans la liste à enjambements ainsi obtenue, quel est le rang des éléments suivants, dans la liste principale de niveau 0 ?* **a)** w_{10} **b)** $w_{1\,000}$ **c)** $w_{100\,000}$

Question 6 *Dans la liste à enjambements ainsi obtenue, quel est le rang des éléments suivants dans la sous-liste de leur propre niveau ?* **a)** w_{10} **b)** $w_{1\,000}$ **c)** $w_{100\,000}$

5 Extension à la recherche d'intervalles contenant une valeur

Considérons maintenant le problème réel, où la structure de données ne stocke pas simplement des valeurs, mais des intervalles de valeurs. Ces intervalles peuvent bien sûr se chevaucher, ils ne sont donc pas totalement ordonnés, même si l'ensemble de leurs bornes l'est. La structure de données doit maintenant pouvoir répondre rapidement l'ensemble des intervalles qui contiennent une valeur donnée.

Nous proposons maintenant de programmer une telle structure de données, avec notamment un algorithme d'insertion de nouvel intervalle en s'inspirant de la liste à enjambements. Pour cela, nous allons commencer par utiliser la liste à enjambements pour stocker les 2 bornes de chaque intervalle. Pour des raisons d'efficacité, il est impératif qu'un intervalle ne soit pas systématiquement représenté à un coût proportionnel au nombre d'éléments situés entre ses deux bornes. On utilisera donc la capacité d'"enjambement" de la structure dans ce but.

La figure 3 donne un schéma de la représentation attendue. En particulier, la recherche d'une valeur dans la structure, qui utilise le même algorithme que précédemment, fournit la liste des intervalles recherchés. Un intervalle est donc stocké tout le long d'une chaîne d'éléments, dont les niveaux montent puis redescendent, et ce, dans les niveaux les plus élevés possibles afin de minimiser le stockage. Dans l'exemple, l'intervalle $[1; 9]$ est ainsi stocké dans le noeud 1 au niveau 1, puis dans le noeud 3 au niveau 2, puis 7 au niveau 2, et finalement 9 au niveau 1. On a donc évité de le stocker dans les noeuds 5 et 6, ce qui est le but recherché. Lors de l'insertion d'un nouvel intervalle, on fera bien attention à mettre à jour les noeuds référençant les intervalles existants.

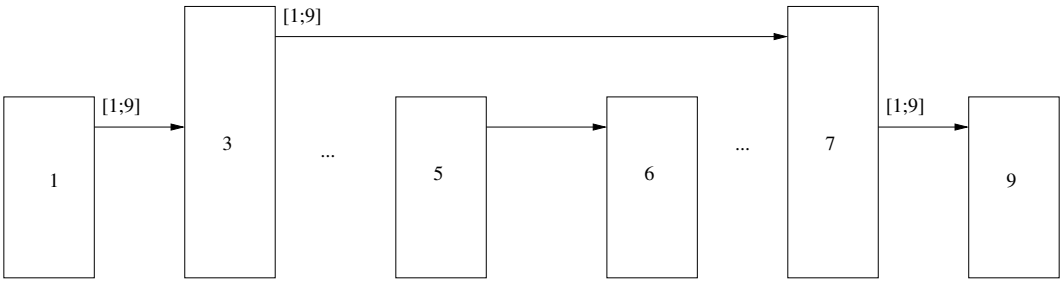

FIG. 3 – Exemple de liste à enjambements adaptée à la recherche d'intervalles, contenant les intervalles $\{[1; 9], [3; 7], [5; 6]\}$.

Génération aléatoire de villes. Nous allons considérer la suite (A_n) d'intervalles d'altitudes des villes suivantes. Pour cela nous utilisons les suites (m_n) et (M_n) définies comme suit : $m_n = \min(w_{2n}, w_{(2n+1)})$ et $M_n = \max(w_{2n}, w_{(2n+1)})$.

$$A_n = [m_n; M_n]$$

Insérer les valeurs $\{A_0, ..., A_{50\,000}\}$ dans la structure de données.

Question 7 *Utilisez votre algorithme de recherche pour calculer combien de villes seraient impactées par une montée des eaux aux altitudes suivantes :* ***a)*** w_{10} ***b)*** $w_{1\,000}$ ***c)*** $w_{100\,000}$

Question 8 *Afin de vérifier l'efficacité de votre algorithme, nous demandons maintenant de compter la somme des nombres de villes impactées respectivement par les altitudes :* ***a)*** $\{w_0, ..., w_{10}\}$ ***b)*** $\{w_0, ..., w_{1\,000}\}$ ***c)*** $\{w_0, ..., w_{10\,000}\}$