Epreuve d'algorithmique - mathématiques appliquées

(Durée : 3 heures)

Important

Il est recommandé de lire l'ensemble du sujet avant de commencer la rédaction. Les trois parties sont largement indépendantes.

Introduction

On étudie dans ce problème des fonctions qui croissent vraiment très lentement (mais inexorablement !) vers $+\infty$. Comme l'a montré en 1975 le grand informaticien Robert E. Tarjan, de telles fonctions interviennent pour calculer le coût des algorithmes de gestion des partitions d'un ensemble. Elles apparaissent aussi pour déterminer la longueur maximale des suites de Davenport-Schinzel, problème posé en 1965 par ces deux mathématiciens qui étudiaient les solutions de certaines équations différentielles, et dont la solution a été donnée par Hart, Sharir et Wiernik en 1988.

Nous nous contentons de quelques propriétés simples pour l'étude de la gestion des partitions (partie I) et des suites de Davenport-Schinzel (partie II), sans montrer complètement leur lien avec la fonction d'Ackermann (partie III).

Définitions et notations

- $\mathbf{N}$ désigne l'ensemble des entiers naturels, $\mathbf{N}^*$ l'ensemble $\mathbf{N}$ privé de 0, $\mathbf{R}$ l'ensemble des réels et $\mathbf{R}^+$ l'ensemble des réels positifs ou nuls
- $log_2(x)$ désigne le logarithme en base 2 de x
- Si f et g sont deux fonctions de $\mathbf{N}$ dans $\mathbf{R}^+$, on note $g(n) = O(f(n))$ s'il existe $c \in \mathbf{R}^+$ et $n_0 \in \mathbf{N}$ tels que $g(n) \leq cf(n)$ pour tout $n \geq n_0$

Partie I. Gestion des partitions d'un ensemble

On veut gérer dynamiquement une partition d'un ensemble de référence $E = \{x_1, x_2, \cdots, x_n\}$, où $n \geq 2$ est un entier. A tout moment, on aura à traiter une collection de sous-ensembles non vides et disjoints deux à deux qui évolue dynamiquement. Au départ, il y a n sous-ensembles (réduits à un singleton) $\{x_i\}, 1 \leq i \leq n$. On va faire évoluer ces sous-ensembles en fusionnant certains d'entre eux. On fera un total de m opérations de fusion successives, où $m \leq n$. On connaît pour chaque sous-ensemble un représentant privilégié (un élément qui lui appartient). On note S_i le sous-ensemble dont le représentant est x_i.

On va examiner plusieurs procédures pour réaliser les opérations de fusion successives. Dans une première solution, pour tout élément x_j de E, on connaît directement $repres[j] = i$, le représentant de l'unique ensemble S_i auquel il appartient. Au début on a donc $repres[j] = j$ et $S_j = \{x_j\}$ pour $1 \leq j \leq n$. On définit le tableau des représentants comme suit :

```
const MAX = 1000 (* par exemple *);
type tab : array[1..MAX] of integer;
var repres : tab;
```

Chaque procédure de fusion fait appel à une structure de données qui utilise un certain nombre de tableaux auxiliaires. On exprime le coût d'une procédure par le nombre d'accès (en lecture ou écriture) à ce ou ces tableaux auxiliaires. Pour chacune des deux structures de données ci-dessous, on va demander de détailler l'initialisation des variables et d'écrire une procédure en PASCAL

```
procedure fusion(u,v : integer ; ...) ;
```

qui, étant donnés deux éléments x_u et x_v de E, réalise la fusion des deux sous-ensembles $S_{repres[u]}$ et $S_{repres[v]}$ auxquels ils appartiennent respectivement.

Question I.1. *On se limite ici au seul tableau des représentants repres.*

1. *Ecrire la procédure de fusion et donner son coût*
2. *Calculer en fonction de n le coût maximal de m fusions successives, où $m \leq n$ (on demande un ordre de grandeur en $O(\ldots)$, pas une formule exacte)*

Question I.2. *On veut que le coût d'une fusion soit proportionnel au cardinal du plus petit des deux ensembles à fusionner. A cet effet, en plus de repres, on utilise les tableaux auxiliaires*

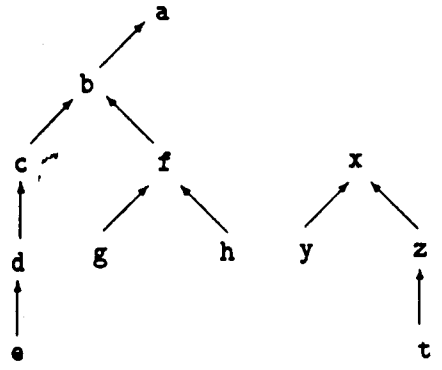
```
var taille, suivant, dernier : tab ;
```

définis ainsi :

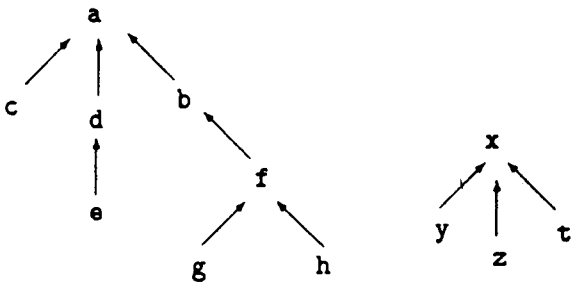
- *taille[i] est la taille de S_i si $i = repres[i]$ et vaut -1 sinon*
- *si $i = repres[i]$, on énumère $S_i = \{i, i_1, i_2, \ldots, i_s\}$ comme suit : $suivant[i] = i_1$, $suivant[i_k] = i_{k+1}$ pour $1 \leq k < s$ et $suivant[i_s] = -1$. On pose $dernier[i] = i_s$.*
- *si $i \neq repres[i]$, $dernier[i] = -1$*

1. *Préciser l'initialisation des tableaux auxiliaires (taille, suivant et dernier), écrire la procédure de fusion et donner son coût*
2. *Montrer que le coût maximal de m fusions successives, où $m \leq n$, est en $O(n\log_2(n))$*

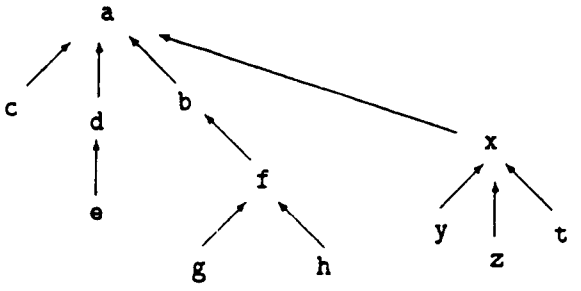
Dans une deuxième solution, les éléments n'ont plus un accès direct au représentant de l'ensemble auxquels ils appartiennent. On conserve seulement pour tout élément x_i un lien vers un autre élément $pere[i]$. Les éléments x_i qui n'ont pas de père ($pere[x_i] = -1$) sont exactement les représentants privilégiés des sous-ensembles que l'on fusionne: on les appelle des racines. Chaque sous-ensemble est maintenant représenté comme un arbre:



(i) situation initiale



(ii) après trouver(d) et trouver(t)



(iii) après union(a,x)

L'opération de fusion (illustrée dans la figure pour les deux arbres auxquels appartiennent les éléments d et t) s'effectue en deux temps:

- déterminer les racines des deux arbres à fusionner. C'est le rôle de la fonction **trouver** dont la figure illustre le mécanisme: on remonte le chemin jusqu'à la racine à l'aide du tableau *pere*, et on en profite pour mettre à jour le père de chacun des éléments rencontrés: pour chacun d'entre eux, leur nouveau père est la racine (intuitivement, il s'agit d'une "compression de chemin")
- donner pour nouveau père de la racine du plus petit arbre (ici x) la racine du plus grand (ici a: on pose donc $pere[x] := a$)

On utilise comme plus haut le tableau *taille*: $taille[i]$ est le cardinal de l'arbre dont i est la racine (et vaut -1 si $pere[i] \neq -1$). On a donc les tableaux auxiliaires:

```
var pere, taille : tab ;
```

Question I.3.

1. *Ecrire en PASCAL la fonction*

```
function trouver(u : integer ; var pere : tab) : integer ;
```

qui renvoie la racine de l'arbre de u et met à jour les pères des éléments du chemin entre u et la racine

2. *Ecrire en PASCAL la procédure*

```
procedure fusion(u,v : integer ; var pere,taille : tab) ;
```

qui fusionne les deux arbres auxquels appartiennent u et v comme indiqué plus haut ; donner l'initialisation des tableaux pere et taille

Partie II. Suites de Davenport-Schinzel

Soient n et s deux entiers strictement positifs : $(n, s) \in \mathbb{N}^* \times \mathbb{N}^*$. Une suite finie de m entiers $U = (u_1, u_2, \ldots, u_m)$ est une suite de Davenport-Schnizel de type (n, s) (on notera $U \in DS(n, s)$) si elle vérifie les trois conditions suivantes :

1. $1 \leq u_i \leq n$ pour $1 \leq i \leq m$
2. $u_i \neq u_{i+1}$ pour $1 \leq i < m$
3. Il n'existe pas $s+2$ indices $1 \leq i_1 < i_2 < i_3 < \ldots < i_{s+2} \leq m$ tels que

$$u_{i_1} = u_{i_3} = u_{i_5} = \ldots = a,\ u_{i_2} = u_{i_4} = u_{i_6} = \ldots = b, \text{ et } a \neq b$$

On note $|U| = m$ la longueur de la suite U. Enfin, on note $\lambda_s(n) = \max\{|U|; U \in DS(n, s)\}$ (on montrera l'existence de $\lambda_s(n)$ question II.2).

Intuitivement, une suite dans $DS(n, s)$ est une suite d'entiers entre 1 et n telle que deux éléments consécutifs sont distincts et ne comprenant pas de sous-suite de valeurs alternées $a, b, a, b, \ldots$ de taille trop grande ($> s+1$). On étudie dans cette partie la longueur maximale de telles suites.

Question II.1. *Soit $d \in \mathbb{N}^*$. Soient $P_1, P_2, \ldots, P_n$ n polynômes distincts de degré inférieur ou égal à d et à coefficients réels. Soit $g : \mathbb{R} \longrightarrow \mathbb{R}$ leur minimum point-à-point : $g(x) = \min\{P_i(x), 1 \leq i \leq n\}$ pour tout $x \in \mathbb{R}$. Soit m le plus petit nombre d'intervalles ouverts $I_1, I_2, \ldots, I_m$ de $\mathbb{R}$ qui vérifie :*

- *les intervalles $I_1, I_2, \ldots, I_m$ sont deux à deux disjoints, $sup(I_j) = inf(I_{j+1})$ pour $1 \leq j < m$, et leur union est égale à $\mathbb{R}$ privé d'un nombre fini de points*
- *pour tout entier k entre 1 et m, il existe un indice i_k tel que $g(x) = P_{i_k}(x)$ pour tout $x \in I_k$.*

1. *Montrer que m existe bien*
2. *Montrer que $U = (i_1, i_2, \ldots, i_m) \in DS(n, d)$*

Question II.2.

1. *Calculer $\lambda_1(n)$*
2. *Soit $s \geq 1$ fixé. Donner un majorant en $O(n^2)$ de $\lambda_s(n)$*
3. *Calculer $\lambda_2(n)$*

Question II.3. *On définit le type de données*

```
const MAX = 1000 (* par exemple *) ;
type suite : array[0..MAX] of integer ;
var U : suite
```

où `U[0]` = $|U|$, la longueur de la suite U.

1. *Ecrire une fonction en PASCAL*

```
function DS(n,s : integer; U: suite) : boolean
```

qui vérifie si une suite U est dans $DS(n,s)$. On demande d'écrire une fonction dont le coût est en $O(n^2\lambda_s(n))$

2. *Peut-on écrire une procédure plus efficace si $s = 1$? On pourra éventuellement utiliser des tableaux auxiliaires*

Question II.4.

1. *Soit x un entier quelconque et $\mathcal{A}_x$ la suite de $2n$ termes $(1, x, 2, x, 3, x, \ldots, n, x)$. Soit $\mathcal{B}$ la suite de $n-2$ termes $(n-1, n-2, \ldots, 2)$. Soit U la suite de longueur $5n-2$ définie par $U = \mathcal{A}_{n+1}\mathcal{B}\mathcal{A}_{n+2}$. Montrer que $U \in DS(n+2, 3)$. En déduire un minorant de $\lambda_3(n)$*
2. *Montrer que $\lambda_3(n) \leq 2n(1 + 2log_2(n))$ pour $n \geq 1$*

Partie III. Logarithme itéré et fonction d'Ackermann

Pour tout entier naturel i, on définit la fonction $lg^{(i)} : \mathbf{N}^* \longrightarrow \mathbf{R}$ par l'expression récurrente suivante :

$$lg^{(i)}(n) = \begin{cases} n & \text{si } i = 0 \\ log_2(lg^{(i-1)}(n)) & \text{si } i > 0 \text{ et } lg^{(i-1)}(n) > 0 \\ indéfini & \text{si } i > 0 \text{ et } lg^{(i-1)}(n) \leq 0, \text{ ou si } lg^{(i-1)}(n) \text{ est indéfini} \end{cases}$$

n définit la fonction $lg^* : \mathbf{N} \longrightarrow \mathbf{N}$ comme suit :

$$lg^*(n) = \min\left\{ i \geq 0 \mid lg^{(i)}(n) \leq 1 \right\}$$

tuitivement, $lg^*(n)$ est le nombre de fois qu'il faut itérer le logarithme pour "descendre" de n en ssous de 1. Enfin, on définit la fonction $F : \mathbf{N} \longrightarrow \mathbf{N}$ par :

$$F(0) = 1 \text{ et } F(i) = 2^{F(i-1)} \text{ pour } i \geq 1.$$

ıestion III.1.

1. *Montrer que la fonction lg^* est bien définie sur $\mathbf{N}$*
2. *Calculer $lg^*(10^{90})$ et $lg^*(2^{65536})$*
   Note : 10^{90} est une estimation du nombre d'atomes de l'univers !
3. *Montrer l'existence de $lim_{n\to+\infty} lg^*(n)$ et donner sa valeur*
4. *Comparer pour n grand les valeurs de $log_2(lg^*(n))$ et de $lg^*(log_2(n))$*
5. *Montrer que $lg^*(n)$ est le plus petit entier k tel que $F(k) \geq n$ (on dit que lg^* est "l'inverse fonctionnel" de F). Ecrire en PASCAL une fonction*

```
function log_etoile(n : integer) : integer;
```

qui calcule $lg^*(n)$ pour tout entier n

On définit la fonction d'Ackermann $A : \mathbf{N} \times \mathbf{N} \longrightarrow \mathbf{N}$ par l'expression récurrente suivante :

$$\begin{cases} A(1,0) = 2 & \\ A(p,0) = p-2 & \text{si } p \geq 2 \\ A(0,n) = n+1 & \text{si } n \geq 0 \\ A(p,n) = A(p-1, A(p,n-1)) & \text{si } p \geq 1 \text{ et } n \geq 1 \end{cases}$$

Question III.2.

1. *Montrer que la fonction d'Ackermann A est bien définie sur* $\mathbf{N} \times \mathbf{N}$
2. *Donner les expressions explicites de* $A(1,n)$, $A(2,n)$ *et* $A(3,n)$. *Que vaut A(4,n)?*
3. *Montrer que pour tout couple* $(p,n) \in \mathbf{N} \times \mathbf{N}$ *on a* $A(p,n) \geq n$, *avec égalité si et seulement si* $(p,n) = (2,0)$
4. *Montrer que* $A(p,n+1) > A(p,n)$ *pour tout couple* $(p,n) \in \mathbf{N} \times \mathbf{N}$
5. *Montrer que* $A(p+1,n) \geq A(p,n)$ *pour tout couple* $(p,n) \in \mathbf{N} \times \mathbf{N}^*$
6. *On définit la fonction* $\alpha : \mathbf{N} \longrightarrow \mathbf{N}$ *comme "l'inverse fonctionnel" de A :* $\alpha(n)$ *est le plus petit entier k tel que* $A(k,k) \geq n$. *Montrer que la fonction* α *est bien définie sur* $\mathbf{N}$. *Calculer* $\alpha(2^{65536})$
7. *Montrer l'existence de* $lim_{n \to +\infty} \alpha(n)$ *et donner sa valeur*
8. *Donner le signe de la quantité* $lg^*(n) - \alpha(n)$ *pour* $n \geq 4$
9. *Montrer l'existence de* $lim_{n \to +\infty} \frac{lg^*(n)}{\alpha(n)}$ *et donner sa valeur*

Question III.3.

1. *Montrer que pour* $m \leq n$, *le coût de m fusions avec la stratégie de la question I.3 est en* $O(n\alpha(n))$
   Note : c'est donc un coût "presque linéaire" en pratique !
2. *Montrer que* $\lambda_3(n)$ *(étudié question II.4) est de l'ordre de* $n\alpha(n)$