ÉCOLE POLYTECHNIQUE – ÉCOLES NORMALES SUPÉRIEURES

CONCOURS D'ADMISSION 2011 FILIÈRE MP SPÉCIALITÉ INFO

COMPOSITION D'INFORMATIQUE – A – (XULC)

(Durée : 4 heures)

L'utilisation des calculatrices n'est pas autorisée pour cette épreuve.
Le langage de programmation choisi par le candidat doit être spécifié en tête de la copie.

⋆ ⋆ ⋆

Transmission dans les arbres

On étudie dans ce problème des algorithmes de transmission d'informations dans les arbres, avec le modèle dit "du téléphone". Dans la partie I, on étudie les arbres binomiaux. Dans la partie II, on considère plusieurs calculs élémentaires sur les arbres. Dans la partie III, on étudie la diffusion de l'information à partir de la racine de l'arbre. Dans la partie IV, on s'intéresse à l'échange total d'informations entre tous les nœuds d'un arbre. Les parties I et II sont indépendantes.

Algorithmes et programmes

- Pour les questions qui demandent la conception d'un algorithme : il s'agit de donner une description concise mais précise, en français, d'un algorithme effectuant la tâche indiquée.
- Pour les questions qui demandent l'écriture d'un programme : il s'agit d'exprimer votre algorithme dans un langage de programmation de votre choix. La lisibilité de votre programme (notamment : mise en évidence de sa structure, indentation, commentaires pertinents) sera prise en compte dans l'évaluation.
- Lorsqu'elle est demandée, la complexité d'un algorithme ou d'un programme ne sera pas calculée exactement mais seulement estimée en ordre de grandeur, avec des expressions du type $O(m+n)$, $O(m^2 \log n)$, etc., où $m, n, \ldots$ sont des paramètres en entrée de l'algorithme.
- Lorsqu'une question demande d'écrire un programme, et qu'une certaine complexité est exigée, on ne demande pas de faire la preuve que le programme proposé a effectivement cette complexité.

Définitions

- Un *arbre* $\mathcal{T} = (r, \mathcal{L})$ est défini par la donnée d'un entier r, appelé nœud, et d'une liste d'arbres $\mathcal{L}$, éventuellement vide. Si la liste $\mathcal{L}$ est vide, on dit que r est un *nœud externe*. Sinon, la liste $\mathcal{L}$ comprend ℓ éléments $\mathcal{T}_i = (r_i, \mathcal{L}_i)$, $1 \leq i \leq \ell$, on dit que r est un *nœud interne*, que les nœuds r_i sont les *fils* de r, et que r est le *père* des r_i.

– Tous les nœuds de $\mathcal{T}$ sont supposés distincts. On note $\mathcal{N}(\mathcal{T})$ l'ensemble de tous les nœuds de $\mathcal{T}$ et $\texttt{nbn}(\mathcal{T})$ leur nombre.

– Dans un arbre, les *voisins* d'un nœud sont son père (s'il existe) et ses fils.

– Entre deux nœuds s et t de $\mathcal{T}$ il existe un unique *chemin*, composé de nœuds $x_0, x_1, \ldots, x_k$ deux à deux distincts, tels que $x_0 = s$, $x_k = t$, et x_i et x_{i+1} sont voisins pour $0 \leq i \leq k-1$. On note $\texttt{chemin}(s,t)$ ce chemin et on dit que k est sa *longueur*.

– Le nœud r est appelé la *racine* de $\mathcal{T}$.

– La *profondeur* d'un nœud s est la longueur de $\texttt{chemin}(s,r)$; en particulier, la racine r a pour profondeur 0. La profondeur de $\mathcal{T}$ est le maximum de la profondeur de ses nœuds.

Les arbres seront représentés en Caml et en Pascal de la manière suivante :

```
        (* Caml *)
type arbre =
| Noeud of int * arbre list ; ;
```

```
                    { Pascal }
type arbre = ^noeud ;
liste_arbre = ^element ;
noeud = record n :integer ; l :liste_arbre ; end ;
element = record a :arbre ; r :liste_arbre ; end ;
```

Certaines questions font par ailleurs intervenir des listes d'entiers, qui seront représentées par le type `liste` suivant :

```
type liste == int list ; ;
```

```
type liste = ^entier ;
entier = record e :integer ; s :liste ; end ;
```

Partie I. Arbres binomiaux

Dans cette partie, on étudie des arbres particuliers, appelés "binomiaux".

Soit k un entier positif ou nul. Un arbre binomial d'ordre k est défini comme suit :

– un arbre binomial d'ordre 0 est réduit à sa racine ;

– si $k > 0$, un arbre binomial d'ordre k est de la forme $(r_k, (\mathcal{T}_{k-1}, \ldots, \mathcal{T}_1, \mathcal{T}_0))$ où chaque $\mathcal{T}_i$ est un arbre binomial d'ordre i.

Dans la suite, $\mathcal{B}_k$ désigne un arbre binomial d'ordre k.

Question 1 Dessiner $\mathcal{B}_4$, avec une numérotation des nœuds de votre choix.

Question 2 Quel est le nombre de nœuds de $\mathcal{B}_k$? Combien sont externes ?

Question 3 Pour $k > 0$, montrer qu'on peut aussi définir récursivement $\mathcal{B}_k$ à l'aide de deux copies de $\mathcal{B}_{k-1}$.

Question 4 Écrire une fonction `copie` qui prend en arguments un entier n et un arbre $\mathcal{T}$ et renvoie une copie de l'arbre $\mathcal{T}$ dans laquelle chaque nœud de numéro i est remplacé par un nœud de numéro $i + n$.

```
(* Caml *)  copie : int -> arbre -> arbre
{ Pascal }  function copie(n : integer ; t : arbre) : arbre ;
```

Question 5 Écrire une fonction `bin` qui prend en argument un entier $k \geq 0$ et qui renvoie l'arbre $\mathcal{B}_k$, avec une numérotation des nœuds de votre choix. On garantira une complexité proportionnelle au nombre de nœuds de $\mathcal{B}_k$.

```
(* Caml *)  bin : int -> arbre
{ Pascal }  function bin(k : integer) : arbre ;
```

Question 6 Quelle est la profondeur de $\mathcal{B}_k$? Quelle est la longueur maximale d'un chemin entre deux nœuds ?

Question 7 Combien de nœuds ont une profondeur donnée ℓ ?

Question 8 Pour $n \geq 1$, on définit $\mathcal{C}_n$ comme un arbre de racine r et à n nœuds, qu'on obtient avec la numérotation suivante des nœuds : la racine r a le numéro 1, et le père du nœud de numéro i, où $2 \leq i \leq n$, est le nœud de numéro $i - 2^{\lceil \log_2 i \rceil - 1}$, où la notation $\lceil x \rceil$ désigne le plus petit entier supérieur ou égal à x. Dessiner $\mathcal{C}_{16}$. Justifier que la définition de $\mathcal{C}_n$ conduit bien à un arbre. Donner, sans la justifier, une relation entre $\mathcal{B}_k$ et $\mathcal{C}_{2^k}$.

Question 9 Écrire une fonction `cn` qui prend en argument un entier $n \geq 1$ et qui renvoie l'arbre $\mathcal{C}_n$. On garantira une complexité $O(n)$.

```
(* Caml *)  cn : int -> arbre
{ Pascal }  function cn(n : integer) : arbre ;
```

Partie II. Fonctions élémentaires sur les arbres

Question 10 Écrire une fonction `profondeur` qui prend en argument un arbre $\mathcal{T}$ et renvoie sa profondeur. On garantira une complexité $O(\texttt{nbn}(\mathcal{T}))$.

```
(* Caml *)  profondeur : arbre -> int
{ Pascal }  function profondeur(t : arbre) : integer ;
```

Question 11 Écrire une fonction `noeud_externe_max` qui prend en argument un arbre $\mathcal{T}$ et qui renvoie le numéro d'un nœud externe de $\mathcal{T}$ de profondeur maximale. En cas d'égalité, on choisira arbitrairement. On garantira une complexité $O(\texttt{nbn}(\mathcal{T}))$.

```
(* Caml *)  noeud_externe_max : arbre -> int
{ Pascal }  function noeud_externe_max(t : arbre) : integer ;
```

Question 12 Soit $\mathcal{T}$ un arbre de racine r et s un nœud de $\mathcal{T}$. Écrire une fonction `chemin` qui prend en arguments $\mathcal{T}$ et s et renvoie l'unique chemin de r à s sous la forme d'une liste d'entiers $[x_0; x_1; \ldots; x_k]$ avec $x_0 = r$, $x_k = s$ et x_i père de x_{i+1} pour $0 \leq i < k$. On garantira une complexité $O(\texttt{nbn}(\mathcal{T}))$.

```
(* Caml *)  chemin : arbre -> int -> liste
{ Pascal }  function chemin(t : arbre ; s : integer) : liste ;
```

Question 13 Soit $\mathcal{T}$ un arbre et s un nœud de $\mathcal{T}$. On définit l'arbre obtenu par changement de racine s, noté $\mathit{pivot}(\mathcal{T}, s)$, comme l'arbre $\mathcal{T}'$ de racine s dont les nœuds sont les mêmes que ceux de $\mathcal{T}$ et tel que deux nœuds sont voisins dans $\mathcal{T}'$ si et seulement s'ils le sont dans $\mathcal{T}$.

Écrire une fonction `pivot` qui prend en arguments $\mathcal{T}$ et s et renvoie l'arbre $\mathit{pivot}(\mathcal{T}, s)$. On garantira une complexité $O(\texttt{nbn}(\mathcal{T}))$. S'il y a plusieurs tels arbres $\mathcal{T}'$, on en choisit un arbitrairement.

```
(* Caml *)  pivot : arbre -> int -> arbre
{ Pascal }  function pivot(t : arbre ; s : integer) : arbre ;
```

Question 14 Étant donné un arbre $\mathcal{T}$, on définit son *diamètre* comme la longueur du plus long chemin dans $\mathcal{T}$. Soit r_1 l'un des nœuds externes de $\mathcal{T}$ de profondeur maximale, et $\mathcal{T}' = \mathit{pivot}(\mathcal{T}, r_1)$ l'arbre obtenu à partir de $\mathcal{T}$ par changement de racine r_1. Montrer que le diamètre de $\mathcal{T}$ est égal à la profondeur de $\mathcal{T}'$.

Question 15 En déduire une fonction `diametre` qui prend en argument un arbre et qui renvoie son diamètre.

```
(* Caml *)  diametre : arbre -> int
{ Pascal }  function diametre(t : arbre) : integer ;
```

Question 16 Soit $\mathcal{T}$ un arbre de diamètre D. Montrer que D est la plus grande profondeur d'un arbre obtenu par changement arbitraire de racine et que $\lceil \frac{D}{2} \rceil$ est la plus petite profondeur d'un arbre obtenu par changement arbitraire de racine.

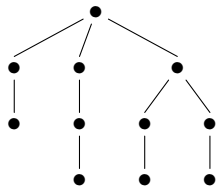
Partie III. Diffusion dans les arbres

On étudie dans cette partie le problème de la *diffusion* dans les arbres. La racine r de l'arbre $\mathcal{T} = (r, \mathcal{L})$ possède un message qu'elle doit transmettre à tous les autres nœuds. La diffusion procède par étapes, toutes de temps unitaire. À une étape donnée, chacun des nœuds déjà en possession du message le transmet à un et un seul de ses fils (sauf si tous ses fils l'ont déjà reçu). Une diffusion D est caractérisée par un ensemble de fonctions : à chaque nœud interne v de l'arbre, avec n_v fils, on associe une fonction injective f_v qui numérote ses fils de 1 à n_v dans l'ordre dans lequel v leur transmet le message.

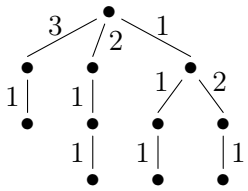
Pour $v \in \mathcal{N}(\mathcal{T})$, on note $t_D(v)$ le numéro de l'étape à laquelle v reçoit le message :

$$t_D(v) = \begin{cases} 0 & \text{si } v = r, \\ t_D(v') + f_{v'}(v) & \text{si } v' \text{ est le père de } v. \end{cases}$$

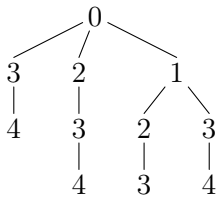
Voici un exemple :



Un arbre



Valeurs des fonctions f_v



Valeurs des numéros des étapes $t_D(v)$

La *durée* de la diffusion est son nombre d'étapes, soit $t(D) = \max_{v \in \mathcal{N}(\mathcal{T})} t_D(v)$. Une diffusion est *optimale* si sa durée est minimale parmi les durées de toutes les diffusions.

Diffusion dans les arbres binomiaux

On considère l'arbre binomial $\mathcal{B}_k$ défini dans la partie I. Tout nœud interne v a pour fils les racines $r_1, \ldots, r_{n_v}$ d'arbres binomiaux $\mathcal{B}_0, \mathcal{B}_1, \ldots, \mathcal{B}_{n_v-1}$. La numérotation naturelle s'obtient en posant $f_v(r_i) = i$ pour chaque nœud v et chaque i, $1 \leq i \leq n_v$, tandis que la numérotation renversée s'obtient en posant $f_v(r_i) = n_v - i + 1$.

Question 17 Quelle est la durée de la diffusion qui choisit la numérotation naturelle pour chaque nœud ?

Question 18 Même question pour la diffusion qui choisit la numérotation renversée pour chaque nœud.

Question 19 Quelle est la durée d'une diffusion optimale dans $\mathcal{B}_k$? Justifier votre réponse.

Diffusion dans un arbre quelconque

Question 20 Proposer un algorithme pour déterminer une diffusion optimale dans un arbre $\mathcal{T}$ quelconque. Quelle est sa complexité en fonction de $\texttt{nbn}(\mathcal{T})$?

Question 21 Écrire une fonction `diffusion_optimale` qui prend en argument un arbre $\mathcal{T}$ et qui calcule la durée d'une diffusion optimale pour $\mathcal{T}$. (On pourra supposer que le langage de programmation contient une fonction qui trie un tableau ou une liste d'entiers.)

```
(* Caml *)  diffusion_optimale : arbre -> int
{ Pascal }  function diffusion_optimale(t : arbre) : integer;
```

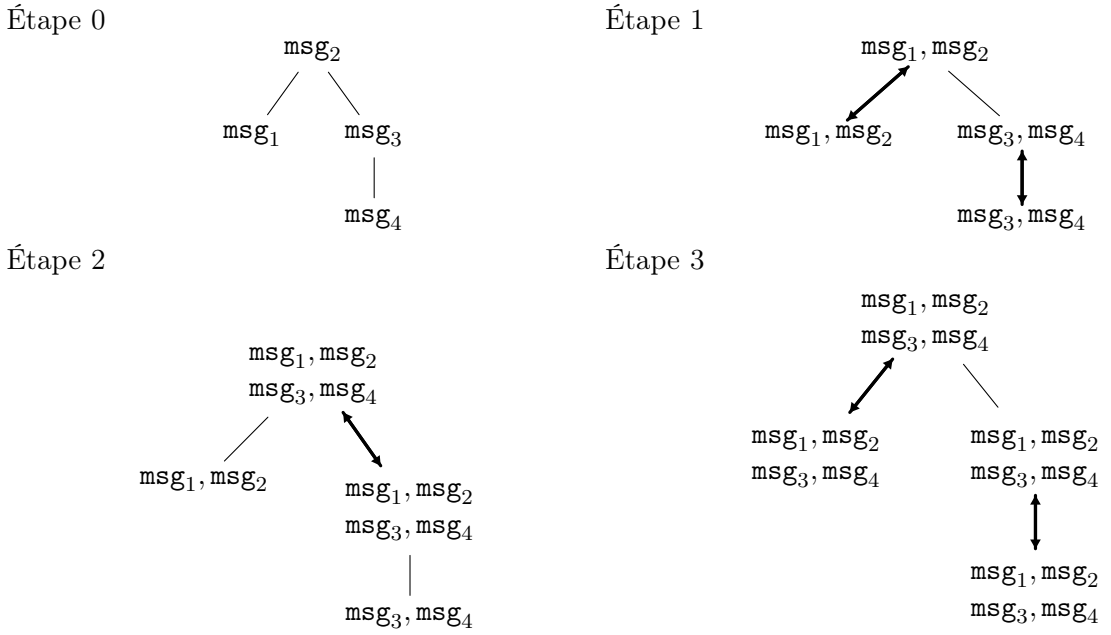
Durée de la diffusion optimale

Question 22 Donner une borne supérieure pour la durée d'une diffusion optimale dans un arbre arbitraire à n nœuds, et exhiber pour tout n un arbre pour lequel cette borne est atteinte.

Question 23 Donner une borne inférieure pour la durée d'une diffusion optimale dans un arbre arbitraire à n nœuds, et exhiber pour tout n un arbre pour lequel cette borne est atteinte.

Partie IV. Échange total dans les arbres

On étudie dans cette partie le problème de *l'échange total* dans les arbres. Chaque nœud v de l'arbre possède un message $\texttt{msg}_v$ qu'il doit transmettre à tous les autres nœuds. L'échange total procède par étapes, toutes de temps unitaire. Lors d'une étape, certaines paires de voisins s'échangent tous les messages qu'ils ont reçus dans les étapes précédentes ; chaque nœud ne peut communiquer qu'avec un seul voisin lors d'une étape donnée. La *durée* d'un échange total est son nombre d'étapes, et son *trafic* est le nombre d'échanges entre voisins ayant eu lieu au cours de l'algorithme. Voici un exemple pour un arbre à 4 nœuds, dont la durée est 3 et le trafic 5 :



Échange total dans les arbres binomiaux

On revient dans cette question sur l'arbre binomial $\mathcal{B}_k$ d'ordre $k \geq 1$ introduit dans la partie I.

Question 24 Proposer un algorithme d'échange total dans $\mathcal{B}_k$ dont la durée est $2k - 1$.

Question 25 Montrer que tout échange total dans $\mathcal{B}_k$ a une durée au moins égale à $2k - 1$.

Question 26 Plus généralement, donner une borne inférieure pour la durée de tout échange total dans un arbre $\mathcal{T}$ quelconque à $n \geq 2$ nœuds.

Échange total de trafic minimal

Soit $\mathcal{T}$ un arbre à $n \geq 2$ nœuds. On considère l'échange total suivant, où il y a un seul échange à chaque étape :

1. Tant que l'arbre contient au moins trois nœuds :
   (a) On choisit un nœud externe s (arbitrairement s'il y en a plusieurs), qui effectue un échange avec son père ;
   (b) On efface le nœud s de l'arbre.
2. Les deux derniers nœuds échangent leur information et possèdent désormais les messages de tous les nœuds de $\mathcal{T}$.
3. On effectue à nouveau tous les échanges de l'étape 1, dans l'ordre inverse, pour propager l'information à tous les nœuds.

Question 27 Montrer que le trafic de l'échange total précédent est égal à $2n - 3$.

Question 28 Écrire une fonction `echange_total` qui prend en argument un arbre $\mathcal{T}$ et renvoie la liste des $n-2$ nœuds externes successivement retirés de l'arbre $\mathcal{T}$ par l'étape 1 de l'algorithme d'échange total ci-dessus. On garantira une complexité $O(\texttt{nbn}(\mathcal{T}))$.

```
(* Caml *)  echange_total : arbre -> liste
{ Pascal }  function echange_total(t : arbre) : liste;
```

Question 29 Montrer que le trafic de tout échange total dans un arbre à $n \geq 2$ nœuds est au moins égal à $2n - 3$.

* *
*