Épreuve d'informatique (Mines 2009)

Durée : 3 heures

L'épreuve comporte deux problèmes indépendants : un problème sur les automates et un problème d'algorithmique sur les arbres.

Partie I. Automates

Les quelques rappels de définitions qui suivent permettent de fixer la terminologie et les notations.
Un *alphabet* Σ est un ensemble fini d'éléments appelés *lettres*. Un *mot* sur Σ est une suite finie de lettres de Σ ; la longueur d'un mot est le nombre de lettres qui le composent ; le mot de longueur nulle est noté ε.
On désigne par Σ^* l'ensemble des mots sur Σ, y compris le mot ε. Un *langage* sur Σ est une partie de Σ^*.

Un *automate fini* A est décrit par une structure (Σ, Q, T, I, F) où :

- Σ est un alphabet ;
- Q est un ensemble fini et non vide appelé *ensemble des états* de A ;
- $T \subset Q \times \Sigma \times Q$ est appelé l'*ensemble des transitions* ; étant donnée une transition $(p, x, q) \in T$ on dit qu'elle est d'*origine* p, d'*extrémité* q et qu'elle est d'*étiquette* x ; on pourra la noter $p \xrightarrow{x} q$;
- $I \subset Q$ est appelé l'*ensemble des états initiaux* de A ;
- $F \subset Q$ est appelé *ensemble des états finaux* de A.

Dans ce problème on considérera uniquement des automates ayant un seul état initial noté q_0.

Un *chemin* de A est une suite de transitions de la forme $p_0 \xrightarrow{x_1} p_1 \xrightarrow{x_2} p_2 \cdots \xrightarrow{x_k} p_k$. On dit alors que ce chemin va de p_0 à p_k. Dans un automate fini, un état q est dit *utile* s'il existe à la fois un chemin de l'état initial q_0 à q et un chemin de q à un état final.

On rappelle le théorème de Kleene : un langage sur un alphabet Σ est rationnel si et seulement s'il existe un automate fini d'alphabet Σ qui le reconnait.

On ne considère dans tout le problème que l'alphabet $\Sigma = \{a, b\}$. Tous les mots et langages considérés seront définis sur cet alphabet.

On définit une application ϕ de Σ^* dans Σ^* de la façon suivante :

- $\phi(\varepsilon) = \varepsilon$;
- si un mot u de longueur $2k > 0$ s'écrit $u = u_1 u_2 \cdots u_{2k-1} u_{2k}$ où, pour $i \in [\![1, 2k]\!]$, u_i appartient à Σ, alors $\phi(u) = u_2 u_1 u_4 u_3 \cdots u_{2k} u_{2k-1}$;
- si un mot u de longueur $2k+1 > 0$ s'écrit $u = u_1 u_2 \cdots u_{2k} u_{2k+1}$ où, pour $i \in [\![1, 2k+1]\!]$, u_i appartient à Σ, alors $\phi(u) = u_2 u_1 u_4 u_3 \cdots u_{2k} u_{2k-1} u_{2k+1}$.

La fonction ϕ agit donc en échangeant chaque lettre d'indice pair avec la lettre (d'indice impair) qui la précède immédiatement. Ainsi, $\phi(a) = a$, $\phi(abba) = baab$, $\phi(aabab) = aaabb$.

Question 1. Soit u un mot dans Σ^*. Établir une condition nécessaire et suffisante pour que, quel que soit le mot v dans Σ^*, l'égalité $\phi(uv) = \phi(u)\phi(v)$ soit vérifiée.

Question 2. On note L_1 l'ensemble des mots u tels que $\phi(u) \neq u$. Caractériser les mots de L_1.

Question 3. Proposer, preuve à l'appui, une expression rationnelle décrivant L_1 ; on privilégiera une expression rationnelle simple.

Question 4. Dessiner un automate fini reconnaissant L_1 ; on privilégiera un automate ayant peu d'états.

Question 5. On note L_2 le langage décrit par l'expression rationnelle a^*b^*.
Proposer une expression rationnelle décrivant $\phi(L_2)$. Justifier brièvement la réponse.

Question 6. Dessiner un automate fini reconnaissant $\phi(L_2)$. Justifier brièvement la réponse.

On se propose de montrer que si L est un langage rationnel, alors $\phi(L)$ est aussi un langage rationnel. Les questions 7 à 14 permettent d'obtenir ce résultat.

Si L est un langage, on note P(L) l'ensemble des mots de L de longueur paire, et I(L) l'ensemble des mots de L de longueur impaire.

Question 7. Montrer que si L est rationnel, P(L) et I(L) sont rationnels.

Question 8. On considère un automate fini A reconnaissant un langage L ne contenant que des mots de longueur paire ; soit q un état utile de A. Montrer que A ne possède pas de transition dont l'origine et l'extrémité soient q.

Question 9. On considère un automate fini $A = (\Sigma, Q, T, \{q_0\}, F)$ reconnaissant un langage L ne contenant que des mots de longueur paire ; soit q un état utile de A. Montrer que les chemins de q_0 à q sont soit tous de longueur paire, soit tous de longueur impaire.

Soit A un automate fini. Soit q un état de A. On suppose que :

- q est un état utile ;
- q n'est ni l'état initial, ni un état final ;
- il n'existe dans A aucune transition dont l'origine et l'extrémité soient simultanément q, c'est-à-dire aucune transition qui s'écrive $q \xrightarrow{x} q$ quelle que soit l'étiquette x considérée ;
- il existe un moins deux transitions d'origine q ou au moins deux transitions d'extrémité q.

On considère l'automate obtenu à partir de A et q de la façon suivante : pour chaque quadruplet (q', q'', x, y) où q' et q'' sont deux états de A, x et y deux lettres de Σ distinctes ou non, et tel que A contienne les transitions $q' \xrightarrow{x} q$ et $q \xrightarrow{y} q''$ on ajoute un nouvel état r et les transitions $q' \xrightarrow{x} r$ et $r \xrightarrow{y} q''$; on ajoute donc autant d'états que de tels quadruplets ; chaque état ajouté est extrémité d'une unique transition et origine d'une unique transition. Enfin, on supprime l'état q et toutes les transitions d'origine ou d'extrémité q. On note $S(A, q)$ l'automate ainsi obtenu.

Question 10. Soit q un état vérifiant les hypothèses ci-dessus. Montrer que les automates A et $S(A, q)$ reconnaissent le même langage.

Question 11. Montrer que si L est un langage rationnel alors $\phi(P(L))$ est aussi un langage rationnel.

Soit L un langage rationnel. Soit x une lettre de Σ. On note $M(L, x)$ le langage défini comme suit : pour tout mot u sur l'alphabet Σ, le mot u appartient à $M(L, x)$ si et seulement si le mot ux est dans L.

Question 12. Montrer que si L est un langage rationnel et x appartient à Σ, le langage $M(L, x)$ est rationnel.

Question 13. Soit L un langage. Donner une relation entre $\phi(I(L))$, $\phi(M(I(L), a))$ et $\phi(M(I(L), b))$.

Question 14. Soit L un langage rationnel. Montrer que $\phi(L)$ est aussi un langage rationnel.

Question 15. Soit L un langage non rationnel. Indiquer si $\phi(L)$ peut être un langage rationnel.

Question 16. Il s'agit d'écrire la fonction ϕ en langage de programmation.
On utilise le type suivant pour représenter les lettres de l'alphabet Σ :

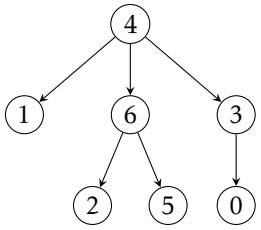
```
type lettre = a | b ;;
```

Un mot est codé par une liste de type *lettre list* ; par exemple le mot *abbab* est codé par la liste `[a;b;b;a;b]`. La liste vide `[]` code le mot de longueur nulle ε.
Écrire en Caml une fonction **`phi`** telle que, si un mot u sur l'alphabet Σ est codé par une liste **`u`** de type *lettre list*, alors **`phi u`** renvoie une liste de type *lettre list* codant $\phi(u)$.
Attention : l'emploi de références ou de vecteurs est interdit.

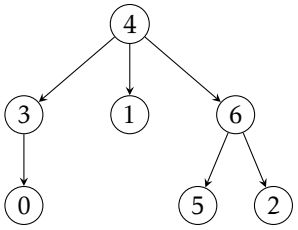
Partie II. Algorithmique

L'objectif de ce problème est de compter le nombre d'arbres enracinés, non ordonnés et étiquetés de nombre de nœuds donné. Pour cela, on étudie un codage particulier de ces arbres appelé codage de Prüfer.

Un arbre possède un nombre fini d'éléments appelés *nœuds*. Les arbres considérés dans ce problème possèdent tous au moins un nœud. Un *arbre enraciné non ordonné* A est défini récursivement de la façon suivante : il est constitué d'un nœud particulier appelé *racine* de A et d'un ensemble fini **non ordonné**, éventuellement vide, d'arbres enracinés non ordonnés appelés *sous-arbres* de A. Les racines des sous-arbres de A sont les *fils* de la racine de A et la racine de A est le *père* de ces derniers. Dans un arbre, deux nœuds sont dits *frères* s'ils ont même père. L'*arité* d'un nœud est son nombre de fils ; dans ce problème, l'arité d'un nœud peut être quelconque. Les nœuds d'arité 0 sont les *feuilles* de l'arbre.
Un arbre est dit *étiqueté* si à chaque nœud est associé un entier positif ou nul, ces entiers étant deux à deux distincts ; l'entier associé à un nœud est l'*étiquette* du nœud. On pourra nommer un nœud par son étiquette ; si i est un entier, on pourra donc parler du nœud i pour le nœud d'étiquette i.
Dans ce problème, le terme d'arbre désignera toujours un arbre enraciné non ordonné étiqueté.
Les deux dessins ci-dessous sont deux représentations graphiques d'un même arbre nommé A_1. L'étiquette de la racine de A_1 est 4 ; l'ensemble des étiquettes des fils de la racine est $\{1,3,6\}$; l'ensemble des étiquettes des fils du nœud d'étiquette 6 est $\{2,5\}$; le nœud d'étiquette 3 possède un seul fils : le nœud d'étiquette 0 ; les nœuds d'étiquettes 0, 1, 2, 5 n'ont pas de fils. Les représentations graphiques d'un arbre donné diffèrent par l'ordre dans lequel on dessine les fils d'un même nœud.

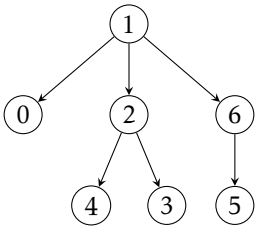


L'arbre A_1, première représentation.



L'arbre A_1, seconde représentation.

L'arbre A_2 représenté ci-dessous est différent de l'arbre A_1.



L'arbre A_2.

On dira qu'un arbre est un *arbre étiqueté consécutivement* s'il s'agit d'un arbre étiqueté et que l'ensemble de ses étiquettes forme un intervalle d'entiers de plus petite valeur 0 ; autrement dit, pour un arbre ayant n nœuds et étiqueté consécutivement, l'ensemble des étiquettes est $\{0,1,2,\ldots,n-1\}$. Les arbres A_1 et A_2 sont des arbres étiquetés consécutivement.

II.1 D'un codage racine-fils-frères d'un arbre au codage de Prüfer

Question 17. Donner la liste des arbres possédant trois nœuds et étiquetés consécutivement.

Soit A un arbre étiqueté consécutivement ayant n nœuds. Pour coder A, on définit un codage nommé *codage racine-fils-frères*. Pour cela, on fixe une représentation graphique de A ; on code A à l'aide de :

- l'étiquette de la racine (qui ne dépend pas de la représentation) ;
- un tableau nommé *fils* ; pour i compris entre 0 et $n-1$, la case d'indice i du tableau *fils* contient la valeur -1 si le nœud i est une feuille de l'arbre et, sinon, l'étiquette du fils du nœud i se situant le plus à gauche dans la représentation graphique choisie ;
- un tableau nommé *freres* ; pour i compris entre 0 et $n-1$, la case d'indice i du tableau *freres* contient la valeur -1 si le nœud i n'a aucun frère sur sa droite et, sinon, l'étiquette de son frère qui se trouve le premier sur sa droite.

Pour l'arbre A_1, si on choisit la première représentation, on obtient le codage suivant :

– la racine est le nœud 4 ;
– pour le tableau *fils* : les cases d'indices 0, 1, 2 et 5 contiennent la valeur −1, la case d'indice 3 contient 0, la case d'indice 4 contient 1, la case d'indice 6 contient 2 ;
– pour le tableau *freres* : les cases d'indices 0, 3, 4 et 5 contiennent la valeur −1, la case d'indice 1 contient 6, la case d'indice 2 contient 5, la case d'indice 6 contient 3.

Ainsi, l'arbre A_1 est représenté par la valeur 4 pour la racine et par les deux tableaux ci-dessous :

indice	0	1	2	3	4	5	6
fils	−1	−1	−1	0	1	−1	2

indice	0	1	2	3	4	5	6
freres	−1	6	5	−1	−1	−1	3

On définit aussi deux tableaux qui peuvent être calculés à partir du codage racine-fils-frères :

– un tableau nommé *peres* ; pour i compris entre 0 et $n-1$, la case d'indice i contient la valeur −1 s'il s'agit de la racine de l'arbre et, dans les autres cas, l'étiquette du père du nœud i ; pour l'arbre A_1, la case d'indice 4 contient la valeur −1, la case d'indice 0 contient 3, les cases d'indices 1, 3 et 6 contiennent 4, les cases d'indices 2 et 5 contiennent la valeur 6 ;
– un tableau nommé *arites* ; pour i compris entre 0 et $n-1$, la case d'indice i de ce tableau contient l'arité du nœud i ; pour l'arbre A_1, les cases d'indices 0, 1, 2 et 5 contiennent la valeur 0, la case d'indice 3 contient 1, la case d'indice 4 contient 3, la case d'indice 6 contient 2.

Pour l'arbre A_1, les tableaux *peres* et *arites* sont représentés ci-dessous :

indice	0	1	2	3	4	5	6
peres	3	4	6	4	−1	6	4

indice	0	1	2	3	4	5	6
arites	0	0	0	1	3	0	2

Question 18. Écrire en Caml une fonction `calculer_peres` telle que, si on considère un arbre A possédant n nœuds et étiqueté consécutivement et si :

– `racine` est un entier qui contient l'étiquette de la racine de A ;
– `fils` et `freres` sont deux vecteurs de longueur n qui représentent respectivement les tableaux *fils* et *freres* d'un codage racine-fils-frères de A,

alors `calculer_peres racine fils freres` renvoie un vecteur de longueur n correspondant au tableau *peres* défini plus haut.

Question 19. Indiquer, en fonction du nombre de nœuds de l'arbre, la complexité de la fonction `calculer_peres`.

Question 20. Écrire en Caml une fonction `calculer_arites` telle que, pour un arbre A possédant n nœuds et étiqueté consécutivement, si `fils` et `freres` sont deux vecteurs de longueur n qui représentent respectivement les tableaux *fils* et *freres* d'un codage racine-fils-frères de A, alors `calculer_arites racine fils freres` renvoie un vecteur correspondant au tableau *arites* défini plus haut.

Question 21. Indiquer, en fonction du nombre de nœuds de l'arbre, la complexité de la fonction `calculer_arites`.

Question 22. Il s'agit d'écrire une fonction `inserer` qui prend en arguments un tableau *table* d'entiers non nécessairement distincts triés par valeurs décroissantes et un entier d ; cette fonction modifie le tableau table pour insérer l'entier d en respectant l'ordre décroissant. L'entier d est inséré même s'il figure déjà dans table.
Écrire en Caml une fonction `inserer` telle que, si :

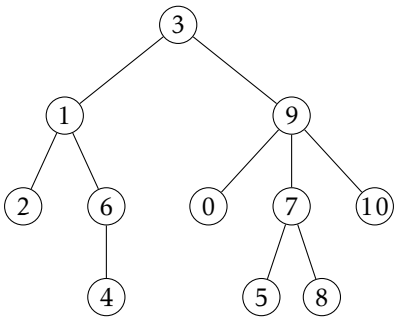
– `table` est un vecteur d'entiers,
– `nb` est un entier positif ou nul ne dépassant pas la dimension du vecteur `table` diminuée de 1,
– `d` est un entier,
– on suppose que le vecteur `table` contient des entiers classés par valeurs décroissantes dans les cases d'indices compris entre `0` et `nb - 1`, les autres cases du vecteur `table` étant ignorées,

alors `inserer table nb d` insère la donnée `d` dans le vecteur `table` en respectant l'ordre décroissant.

Question 23. Indiquer, en fonction du nombre *nb* d'entiers contenus dans un tableau trié *table*, la complexité de la fonction `inserer` quand elle insère un nouvel entier dans *table*.

Soit A un arbre possédant n nœuds ; on note E(A) l'ensemble des étiquettes de A ; les étiquettes de A étant toutes distinctes, l'ensemble E(A) possède n éléments. Le codage de Prüfer d'un arbre étiqueté ayant n nœuds est une suite de $n-1$ entiers appartenant à E(A), suite notée Pr(A) ; ce codage est défini récursivement de la façon suivante. Si A est réduit à un nœud, sa racine, son codage de Prüfer est la suite vide. Sinon, soit f la feuille de A d'étiquette minimum et soit p le père de f ; on note A′ l'arbre obtenu en enlevant de A la feuille f ; par définition, le codage de Prüfer de A est la suite dont le premier élément est l'étiquette de p, ce premier élément étant suivi du codage de Prüfer de A′.
Ainsi, le codage de Prüfer de l'arbre A_1 est : 3, 4, 6, 4, 6, 4 ; le codage de Prüfer de l'arbre A_2 est : 1, 2, 2, 1, 6, 1.

Question 24. Indiquer le codage de Prüfer de l'arbre A_3 ci-dessous.



L'arbre A_3.

Question 25. On considère un arbre A étiqueté consécutivement. Il s'agit d'écrire une fonction qui calcule le codage de Prüfer de A. La fonction commencera par calculer les tableaux *peres* et *arites* ; puis elle construira un tableau contenant les feuilles de l'arbre initial classées par étiquettes décroissantes ; après cette partie préparatoire, la fonction calculera le codage de Prüfer.
Écrire en Caml une fonction `calculer_Prufer` telle que, si on considère un arbre A étiqueté consécutivement et si :

- `racine` est un entier qui contient l'étiquette de la racine de A,
- `fils` et `freres` sont deux vecteurs de longueur n qui représentent respectivement les tableaux *fils* et *freres* d'un codage racine-fils-frères de A,

alors `calculer_Prufer racine fils freres` renvoie un vecteur de longueur $n-1$ contenant le codage de Prüfer de l'arbre A.

Question 26. Indiquer la complexité du calcul du codage de Prüfer d'un arbre A possédant n nœuds, étiqueté consécutivement et codé avec le codage racine-fils-frères.

II.2 D'un codage de Prüfer d'un arbre à un codage racine-fils-frères

Question 27. On suppose qu'on connaît le codage de Prüfer d'un arbre A étiqueté consécutivement. Écrire en Caml une fonction `calculer_arites_par_Prufer` telle que, pour un arbre A possédant n nœuds et étiqueté consécutivement, si `prufer` est un vecteur de longueur $n-1$ contenant le codage de Prüfer de A, alors `calculer_arites_par_Prufer prufer` renvoie un vecteur de longueur n contenant les arités des nœuds de A.
Avant d'écrire la fonction `calculer_arites_par_Prufer`, on en donnera rapidement le principe.

Question 28. Déterminer un arbre A étiqueté consécutivement dont le codage de Prüfer Pr(A) est : 2, 3, 0, 2, 2. On détaillera la démarche utilisée.

Question 29. On considère un arbre A ; on suppose que l'ensemble des étiquettes E(A) de A est $\{1,3,5,6,7,9,10,12,13\}$; l'arbre A n'est donc pas étiqueté consécutivement ; on suppose enfin que le codage de Prüfer Pr(A) de A est : 3, 10, 3, 7, 7, 5, 7, 5. Déterminer l'arbre A. On décrira succinctement la démarche utilisée.

Question 30. Il s'agit d'écrire une fonction qui, à partir du codage de Prüfer d'un arbre A étiqueté consécutivement, calcule un codage racine-fils-frères de A. Écrire en Caml une fonction `calculer_arbre` telle que, pour un arbre A possédant n nœuds et étiqueté consécutivement, si :

- `prufer` est un vecteur de longueur $n-1$ contenant le codage de Prüfer de A,
- `fils` et `freres` sont deux vecteurs de longueur n,

alors `calculer_arbre prufer fils freres` modifie les vecteurs `fils` et `freres` pour qu'ils correspondent respectivement aux tableaux *fils* et *freres* d'un codage racine-fils-frères de A et renvoie l'étiquette de la racine de A.

Soit E un ensemble de n entiers distincts positifs ou nuls ; soit $\mathcal{S}(E)$ l'ensemble des suites de longueur $n-1$ dont tous les éléments sont dans E, distincts ou non ; soit enfin $\mathcal{A}(E)$ l'ensemble des arbres enracinés non ordonnés, possédant n nœuds et étiquetés par les éléments de E.

Question 31. Montrer que l'application Pr qui, à un arbre appartenant à $\mathcal{A}(E)$, associe son codage de Prüfer est une bijection entre $\mathcal{A}(E)$ et $\mathcal{S}(E)$.

Question 32. Déterminer le cardinal de $\mathcal{A}(E)$.

♦ ♦
♦