1999 – INF – MP

ÉCOLE NATIONALE DES PONTS ET CHAUSSÉES,
ÉCOLES NATIONALES SUPÉRIEURES DE L'AÉRONAUTIQUE ET DE L'ESPACE,
DES TECHNIQUES AVANCÉES, DES TÉLÉCOMMUNICATIONS,
DES MINES DE PARIS, DES MINES DE SAINT-ÉTIENNE, DES MINES DE NANCY,
DES TÉLÉCOMMUNICATIONS DE BRETAGNE,
ÉCOLE POLYTECHNIQUE
(Filière T.S.I.)

CONCOURS D'ADMISSION 1999

ÉPREUVE D'INFORMATIQUE

Filière MP
(Durée de l'épreuve : 3 heures)

Sujet mis à la disposition de l'ENTPE

Les candidats et les candidates sont priés de mentionner de façon apparente sur la première page de la copie :
« INFORMATIQUE — Filière MP »

RECOMMANDATIONS AUX CANDIDATS ET CANDIDATES

L'énoncé de cette épreuve, y compris cette page de garde, comporte 11 pages.

Si, au cours de l'épreuve, un candidat ou une candidate repère ce qui lui semble être une erreur d'énoncé, il ou elle le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il ou elle a décidé de prendre.

Tout résultat fourni dans l'énoncé peut être utilisé pour les questions ultérieures même s'il n'a pas été démontré. Il ne faut pas hésiter à formuler les commentaires qui semblent pertinents même lorsque l'énoncé ne le demande pas explicitement.

COMPOSITION DE L'ÉPREUVE

L'épreuve comprend deux exercices et un problème.

- **Premier exercice :** l'exercice de théorie des automates, n° 1 page 2, à résoudre en 75 minutes environ.
- **Deuxième exercice :** l'exercice de logique des propositions, n° 2 page 4, à résoudre en 15 minutes environ.
- **Problème :** un problème de programmation **au choix parmi deux**, à résoudre en 90 minutes environ. Les deux choix possibles sont :
  - le problème de programmation en langage Pascal, n° 3 page 5, pour les candidats et les candidates désirant choisir ce langage ;
  - le problème de programmation en langage Caml, n° 4 page 8, pour les candidats et les candidates désirant choisir ce langage.

  Attention ! Pour le problème, un seul choix est possible. Le candidat ou la candidate précisera son choix sur sa copie et seul le problème correspondant sera corrigé.

Tournez la page S.V.P.

1 Exercice de théorie des automates, 75 mn environ

Sur la longueur des mots d'un langage reconnaissable

L'objet de ce problème est d'étudier deux propriétés de la distribution des longueurs des mots d'un langage reconnu par un automate fini.

Rappels pour fixer les notations et la terminologie

Un *alphabet* A est un ensemble fini d'éléments appelés *lettres*. A^* est l'ensemble des suites finies de lettres de A, appelées *mots*. Le *mot vide* est noté 1_{A^*}. La *longueur* d'un mot f, notée $|f|$, est le nombre de lettres qui le composent. Un *langage* est une partie de A^*.

Un *automate* $\mathcal{A}$ est un *graphe orienté et étiqueté*, décrit par une structure $\langle Q, A, E, I, T \rangle$. A est un alphabet. Q est un ensemble fini appelé ensemble des *états* de $\mathcal{A}$. Ce sont les *sommets* du graphe. $I \subseteq Q$ est appelé ensemble des *états initiaux* de $\mathcal{A}$. $T \subseteq Q$ est appelé ensemble des *états terminaux* de $\mathcal{A}$. $E \subseteq Q \times A \times Q$ est appelé l'ensemble des *transitions*. Une transition (p, a, q) est un arc du graphe, d'étiquette a et allant de l'état p vers l'état q. On pourra la noter $p \xrightarrow{a} q$. On représente graphiquement l'automate $\mathcal{A}$ ainsi :

- un état e est figuré par un cercle marqué en son centre par e ; si $e \in I$, cela est figuré par une flèche entrante sans origine ; si $e \in T$, cela est figuré par une flèche sortante sans but ;
- une transition (p, a, q) est figurée par une flèche allant de l'état p vers l'état q et étiquetée par la lettre a.

Un *calcul* de $\mathcal{A}$ est un chemin $c = p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \cdots \xrightarrow{a_n} p_n$ dans le graphe orienté où $\forall i \in [1, n], p_{i-1} \xrightarrow{a_i} p_i \in E$. p_0 est l'*origine* du calcul, p_n son *extrémité*. L'*étiquette* de c est le mot formé par la suite des étiquettes des arcs successifs du chemin. Un calcul d'origine p, d'extrémité q et d'étiquette u peut être noté $p \xrightarrow{u} q$. Un calcul $p \xrightarrow{u} q$ de $\mathcal{A}$ est dit *réussi* si $p \in I$ et $q \in T$. $L(\mathcal{A})$, langage reconnu par $\mathcal{A}$, est l'ensemble des étiquettes des calculs réussis.

L'automate $\mathcal{A}$ est dit *déterministe* si I est un singleton et si pour tout $(p, a) \in Q \times A$, il existe *au plus* un $q \in A$ tel que $(p, a, q) \in E$. Sinon, il est dit *non-déterministe*. Tout automate est équivalent à un automate déterministe.

$\mathcal{A}$ est dit *émondé* si pour chaque état p de $\mathcal{A}$, il existe un état initial i, un état terminal t et deux mots u et v tels que $i \xrightarrow{u} p$ et $p \xrightarrow{v} t$ sont des calculs, i.e. tout état est réellement utile à la reconnaissance des mots de $L(\mathcal{A})$. Tout automate est équivalent à un automate émondé.

Attention ! Dans cet exercice, tous les automates sont supposés déterministes et émondés. Les deux parties de l'exercice sont indépendantes.

Première Partie

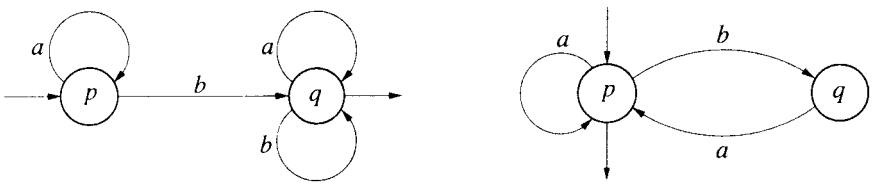
Nombre de mots d'une longueur donnée

1 – *Exemple.* Soit $\mathcal{B}$ l'automate décrit figure 1(a). Caractériser le langage reconnu par l'automate $\mathcal{B}$. En déduire, pour tout $n \in \mathbb{N}$, le nombre de mots de longueur n reconnus par $\mathcal{B}$.

2 – *Exemple.* Soit $\mathcal{F}$ l'automate décrit figure 1(b). Pour $n \in \mathbb{N}$, on note $f(n)$ le nombre de mots de longueur n reconnus par $\mathcal{F}$.

a – Exprimer $f(n+2)$ en fonction de $f(n+1)$ et $f(n)$.

b – Calculer $f(8)$.



(a) L'automate $\mathcal{B}$ (b) L'automate $\mathcal{F}$

Figure 1 – *Deux automates*

3 – *Généralisation.* Soit $\mathcal{A} = \langle Q, A, E, I, T \rangle$ un automate et $L = L(\mathcal{A})$. Pour tout $p \in Q$, on note $L_p = \{u \in A^* \mid \exists t \in T \quad p \xrightarrow{u} t\}$. Soit $n \in \mathbb{N}$ et $p \in Q$, on note $l(n)$ le nombre d'éléments de L de longueur n et $l_p(n)$ le nombre d'éléments de L_p de longueur n.

a – Soit $p \in Q$, montrer que $l_p(n)$ est une combinaison linéaire à coefficients entiers des $l_q(n-1)$ pour $q \in Q$.

b – Montrer qu'il existe un endomorphisme u de $\mathbb{R}^Q$ tel que pour tout $n \in \mathbb{N}$ l'on ait $(l_p(n+1))_{p\in Q} = u((l_p(n))_{p\in Q})$, endomorphisme représenté dans une base appropriée par une matrice à coefficients entiers.

c – En déduire que la suite $(l(n))_{n\geqslant 0}$ satisfait une relation de récurrence linéaire de la forme $\forall n \in \mathbb{N},\ l(n+k) = z_{k-1}l(n+k-1) + z_{k-2}l(n+k-2) + \cdots + z_1 l(n+1) + z_0 l(n)$ où k est le nombre d'états de $\mathcal{A}$ et les coefficients $(z_i)_{0\leqslant i<k}$ sont des entiers relatifs. On pourra introduire le polynôme caractéristique de l'endomorphisme u de la question précédente.

d – Expliquer comment déterminer les conditions initiales, i.e. $l(0), \ldots, l(k-1)$.

4 – *Application.* Soit $n \in \mathbb{N}$, appliquer la méthode de la question précédente à l'automate $\mathcal{B}$ de la figure 1(a) pour déterminer le nombre de mots de longueur n reconnus par $\mathcal{B}$.

Deuxième Partie

Caractérisation des langages minces

Soit $\mathcal{A} = \langle Q, A, E, I, T \rangle$ un automate. À toute partie R de Q, on associe le sous-graphe $\mathcal{C}_R$ de $\mathcal{A}$ dont les sommets sont les éléments de R et les arcs sont les arcs de $\mathcal{A}$ dont l'extrémité et l'origine sont dans R.

Soit $R \subseteq Q$, $\mathcal{C}_R$ est dit *fortement connexe* si pour tous $p, q \in R$, il existe un calcul dans $\mathcal{C}_R$ dont l'origine est p et l'extrémité est q.

Soit $R \subseteq Q$, $\mathcal{C}_R$ est dit *fortement connexe maximal* s'il est fortement connexe et si pour toute partie S de Q strictement plus grande que R pour l'inclusion, $\mathcal{C}_S$ n'est pas fortement connexe.

Tournez la page S.V.P.

Un *circuit* est un calcul dont l'extrémité coïncide avec l'origine. Pour tout circuit, il existe $R \subseteq Q$ tel que $\mathcal{C}_R$ est fortement connexe maximal et contient le circuit.

Un *rayon* est un ensemble de mots de la forme uv^*w avec $u, v, w \in A^*$. Si $v = 1_{A^*}$ alors le rayon est réduit à un mot unique. Un langage est *mince* s'il est une union *finie* de rayons.

Soit L un langage. Pour $n \in \mathbb{N}$, on note $l(n)$ le nombre de mots de longueur n appartenant à L. L est dit à *croissance bornée* s'il existe $k \in \mathbb{N}$ tel que pour tout n on ait $l(n) \leqslant k$.

Attention ! On rappelle que l'automate $\mathcal{A}$ est supposé déterministe et émondé.

5 – Montrer qu'un langage mince est à croissance bornée.

6 – Soit $R \subseteq Q$ tel que $\mathcal{C}_R$ est fortement connexe maximal. Montrer que si $\mathcal{C}_R$ n'est pas réduit à un seul circuit, alors $L(\mathcal{A})$ n'est pas à croissance bornée.

7 – Soit $R \subseteq Q$ et $S \subseteq Q$ tels que $\mathcal{C}_R$ et $\mathcal{C}_S$ sont fortement connexes maximaux. Montrer que s'il existe un calcul de $\mathcal{A}$ d'origine dans R et d'extrémité dans S alors $L(\mathcal{A})$ n'est pas à croissance bornée.

8 – Montrer que si le langage reconnu par $\mathcal{A}$ est à croissance bornée, alors il est mince.

FIN DE L'EXERCICE DE THÉORIE DES AUTOMATES

2 Exercice de logique, 15 mn environ

Échange du contenu de deux variables informatiques

On note $\oplus$ le connecteur *ou exclusif (XOR)* de la logique des propositions.

1 – Rappeler la table de vérité du connecteur $\oplus$.

2 – Soit x et y des variables propositionnelles, construire les tables de vérité des formules logiques $(x \oplus (x \oplus y))$ et $((x \oplus (x \oplus y)) \oplus (x \oplus y))$.

On travaille à présent dans un langage de programmation quelconque. On dispose de l'unique instruction suivante :

$$a \leftarrow b \oplus c$$

où a, b et c sont des variables informatiques pouvant contenir des entiers codés sur 32 bits. L'exécution de cette instruction se déroule ainsi : soit $\overline{b_{31} \cdots b_1 b_0}^2$ la représentation binaire de l'entier contenu dans la variable b ; soit $\overline{c_{31} \cdots c_1 c_0}^2$ la représentation binaire de l'entier contenu dans la variable c ; alors l'instruction range dans la variable a l'entier dont la représentation binaire $\overline{a_{31} \cdots a_1 a_0}^2$ est définie par $\forall i \in [0, 31], a_i = b_i \oplus c_i$.

3 – Soit u et v deux variables informatiques pouvant contenir des entiers codés sur 32 bits.

a – Donner une suite de trois instructions à exécuter consécutivement pour échanger les valeurs contenues dans les variables u et v sans utiliser d'autre variable informatique.

b – Justifier *formellement* que cette suite d'instructions a bien l'effet requis.

FIN DE L'EXERCICE DE LOGIQUE

3 Problème de programmation en Pascal — 90 mn environ

UNIQUEMENT POUR LES CANDIDATS ET LES CANDIDATES CHOISISSANT LE LANGAGE PASCAL

L'objet de ce problème est la conception et l'analyse de complexité en temps d'un algorithme de tri pour un tableau contenant des entiers. Ce problème comporte deux parties indépendantes. La *première partie* concerne la détermination de la complexité en temps minimale des algorithmes de tri. La *deuxième partie* est la programmation et l'évaluation de complexité en temps d'un algorithme de tri ayant cette complexité minimale.

Première partie

Évaluation de la complexité en temps minimale d'un algorithme de tri

On définit la *hauteur* d'un arbre binaire par: la hauteur d'un arbre réduit à une feuille est 0 et la hauteur d'un arbre non réduit à une feuille est 1 plus le maximum des hauteurs des sous-arbres fils de sa racine.

1 – Quel est le nombre maximal de feuilles d'un arbre binaire de hauteur $h \in \mathbb{N}$?

Soit $n \geqslant 1$ un entier, on désire faire un choix *unique* parmi n possibilités données dans un ensemble $\mathcal{C}$ de cardinal n. Nous n'avons pas besoin de définir la nature des éléments de $\mathcal{C}$. Le choix est à faire selon certains critères que nous n'avons pas non plus besoin de définir.

Arbre de décision. Un arbre de décision pour $\mathcal{C}$ est un arbre binaire tel que:

- le sommet de l'arbre est étiqueté par $\mathcal{C}$, l'ensemble de tous les choix envisageables;
- chacune des feuilles de cet arbre est étiquetée par un singleton inclus dans $\mathcal{C}$;
- chaque nœud qui n'est pas une feuille est étiqueté par un sous-ensemble C_1 de $\mathcal{C}$ et possède deux fils étiquetés par des sous-ensembles *stricts et non-vides* C_2 et C_3 de C_1 tels que $C_2 \cup C_3 = C_1$; à l'intérieur du nœud figure un test particulier à ce nœud; la flèche allant vers le nœud étiqueté par C_2 porte le label v pour *vrai* et celle allant vers le nœud étiqueté par C_3 porte le label f pour *faux*, cf. figure 2.

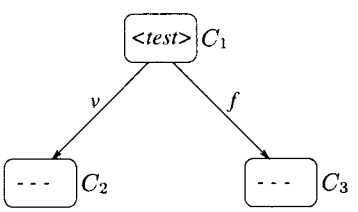


Figure 2 – *Nœud d'un arbre de décision*

Instruction définie par un nœud. L'instruction définie par un nœud d'un arbre de décision est la suivante: on sait que le choix unique à réaliser se trouve dans l'ensemble C_1, on effectue le test figurant dans le nœud; si le test est *vrai* alors le choix unique est à réaliser dans C_2, sinon le choix unique est à réaliser dans C_3. On descend alors vers le nœud correspondant.

Algorithme de choix. Un algorithme de choix parmi les éléments d'un ensemble C est défini par un arbre de décision pour cet ensemble. En partant du sommet de l'arbre et en exécutant l'instruction définie par chaque nœud rencontré, on finit par atteindre une feuille. Cette feuille est étiquetée par un singleton qui contient le choix.

Exemple. L'arbre dessiné figure 3 est un arbre de décision permettant de choisir le plus petit parmi trois nombres a, b et c.

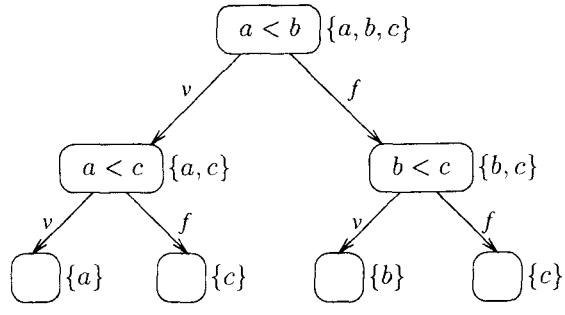


Figure 3 – *Un exemple d'arbre de décision*

2 – On s'intéresse à la hauteur d'un arbre de décision.

a – Quelle est la hauteur minimale en fonction de n d'un arbre de décision pour un choix dans un ensemble de cardinal n?

b – Quelle est la hauteur maximale en fonction de n d'un arbre de décision pour un choix dans un ensemble de cardinal n?

Notation. On se pose le problème du tri d'un tableau t contenant n entiers indicés entre 1 et n. n est une constante entière strictement positive. On appelle *indice* un entier i compris entre 1 et n, i.e. $1 \leqslant i \leqslant n$. Dans la suite du problème, si i est un indice, on notera $t(i)$ l'élément d'indice i du tableau t.
Attention ! Cette notation n'est pas liée à un langage de programmation. Elle sert uniquement à l'exposé du problème.

Un tableau q, contenant n entiers indicés entre 1 et n, est appelé une *permutation* de t s'il existe une bijection $\sigma : \{1, \ldots, n\} \to \{1, \ldots, n\}$ telle que $\forall i \in \{1, \ldots, n\}, q(i) = t(\sigma(i))$.

3 – Le tableau résultat du tri de t est une *permutation* de ce tableau. Combien existe-t-il de permutations du tableau t si les entiers contenus dans t sont deux à deux différents?

4 – On peut considérer le problème du tri du tableau t comme un algorithme de choix parmi les permutations du tableau t et dont les tests sont des comparaisons entre deux éléments du tableau. Donner une approximation de la hauteur minimale de l'arbre de décision correspondant à ce choix.

5 – En déduire que la meilleure complexité envisageable en temps dans le pire des cas d'un algorithme de choix pour le tri du tableau t est $O(n \ln(n))$.

Deuxième partie

Complexité en temps et programmation d'un algorithme de tri

Soit t un tableau de n entiers indicés entre 1 et n. n est une constante entière positive et non nulle. On reprend les notations de la première partie.

6 – Quel type de données faut-il utiliser pour représenter le tableau t pouvant contenir n entiers indicés entre 1 et n? Pour ce type de données, comment a-t-on accès à la valeur de $t(i)$ pour un indice i particulier? Programmer un algorithme echanger qui prend en arguments le tableau t et deux indices i et j et qui échange les valeurs de $t(i)$ et $t(j)$. Quelle est la complexité en temps de cet algorithme?

Attention ! Dans les programmes qui vous seront demandés, les tableaux ne devront être modifiés qu'avec l'algorithme echanger.

Arbres binaires définis dans un tableau. Un *arbre binaire* est un arbre où chaque nœud a *au plus* deux fils. Soit t notre tableau, pour des indices i et j tels que $i \leqslant j$, on définit l'arbre binaire noté $\alpha(t, i, j)$ par :

- si $2i > j$ alors $\alpha(t, i, j)$ est réduit à une feuille étiquetée par $t(i)$;
- si $2i = j$ alors la racine de $\alpha(t, i, j)$ est un nœud étiqueté par $t(i)$ et possède un seul fils qui est $\alpha(t, 2i, j)$;
- si $2i < j$ alors la racine de $\alpha(t, i, j)$ est un nœud étiqueté par $t(i)$ et elle possède un fils gauche qui est $\alpha(t, 2i, j)$ et un fils droit qui est $\alpha(t, 2i + 1, j)$.

7 – Soit $n = 10$ et le tableau t représenté ci-dessous, dessiner l'arbre binaire $\alpha(t, 1, 8)$.

23	12	3	4	67	10	9	6	7	12

On dit qu'un arbre binaire dont les nœuds sont étiquetés par des entiers est *ordonné* si et seulement s'il est réduit à une feuille ou bien si chacun des fils de la racine est ordonné et l'étiquette de la racine est supérieure ou égale aux étiquettes des racines de ses fils.

Note. Dans la suite du problème, si i et j sont deux indices tels que $i \leqslant j$, on confondra l'arbre binaire $\alpha(t, i, j)$ et les éléments du tableau t entre les indices i et j qui le composent.

8 – On désire ordonner des arbres en n'utilisant que les seules opérations consistant à échanger ou à comparer les valeurs de deux éléments du tableau t.

a – Programmer un algorithme ordonner prenant en arguments le tableau t et deux indices i et j tels que $i \leqslant j$. Sachant que les fils éventuels de la racine de $\alpha(t, i, j)$ sont initialement ordonnés, cet algorithme doit ordonner $\alpha(t, i, j)$.

b – Évaluer la complexité en temps dans le pire des cas de cet algorithme.

9 – Soit j un indice, démontrer que chacun des éléments du tableau entre les indices 1 et j est l'étiquette d'un nœud ou d'une feuille de $\alpha(t, 1, j)$.

Tournez la page S.V.P.

10 – On désire ordonner $\alpha(t, 1, n)$. Pour cela, on ordonne $\alpha(t, j, n)$ à l'aide de l'algorithme ordonner précédent, pour j décroissant de $\lfloor n/2 \rfloor$ à 1, où $\lfloor n/2 \rfloor$ désigne le résultat de la division entière de n par 2.

a – Programmer un algorithme ordonner_totalement prenant en argument le tableau t et ordonnant $\alpha(t, 1, n)$.

b – Justifier *rigoureusement* que cet algorithme a bien l'effet requis.

c – Évaluer sa complexité en temps dans le pire des cas en fonction de n.

11 – On désire trier les éléments du tableau en ordre croissant.

a – Programmer un algorithme bulle utilisant l'algorithme ordonner et prenant en arguments le tableau t et un indice $j \geqslant 2$. Supposant que $\alpha(t, 1, j)$ est ordonné, cet algorithme doit permuter les éléments du tableau t de manière à ce que $t(j)$ soit le plus grand des éléments du tableau t entre les indices 1 et j et que $\alpha(t, 1, j-1)$ soit ordonné.

b – En déduire et rédiger un algorithme de tri en ordre croissant du tableau t.

c – Montrer que cet algorithme de tri fait bien ce qu'il est censé faire.

d – En évaluer la complexité en temps dans le pire des cas en fonction de n.

FIN DU PROBLÈME DE PROGRAMMATION EN PASCAL

4 Problème de programmation en Caml — 90 mn environ

UNIQUEMENT POUR LES CANDIDATS ET LES CANDIDATES CHOISISSANT LE LANGAGE CAML

L'objet de ce problème est la conception et l'analyse de complexité en temps d'un algorithme de tri pour un tableau contenant des entiers. Ce problème comporte deux parties indépendantes. La *première partie* concerne la détermination de la complexité en temps minimale des algorithmes de tri. La *deuxième partie* est la programmation et l'évaluation de complexité en temps d'un algorithme de tri ayant cette complexité minimale.

Première partie

Évaluation de la complexité en temps minimale d'un algorithme de tri

On définit la *hauteur* d'un arbre binaire par : la hauteur d'un arbre réduit à une feuille est 0 et la hauteur d'un arbre non réduit à une feuille est 1 plus le maximum des hauteurs des sous-arbres fils de sa racine.

1 – Quel est le nombre maximal de feuilles d'un arbre binaire de hauteur $h \in \mathbb{N}$?

Soit $n \geqslant 1$ un entier, on désire faire un choix *unique* parmi n possibilités données dans un ensemble $\mathcal{C}$ de cardinal n. Nous n'avons pas besoin de définir la nature des éléments de $\mathcal{C}$. Le choix est à faire selon certains critères que nous n'avons pas non plus besoin de définir.

Arbre de décision. Un arbre de décision pour $\mathcal{C}$ est un arbre binaire tel que :

- le sommet de l'arbre est étiqueté par $\mathcal{C}$, l'ensemble de tous les choix envisageables ;
- chacune des feuilles de cet arbre est étiquetée par un singleton inclus dans $\mathcal{C}$;
- chaque nœud qui n'est pas une feuille est étiqueté par un sous-ensemble C_1 de $\mathcal{C}$ et possède deux fils étiquetés par des sous-ensembles *stricts et non-vides* C_2 et C_3 de C_1 tels que $C_2 \cup C_3 = C_1$; à l'intérieur du nœud figure un test particulier à ce nœud ; la flèche allant vers le nœud étiqueté par C_2 porte le label v pour *vrai* et celle allant vers le nœud étiqueté par C_3 porte le label f pour $faux$, cf. figure 4.

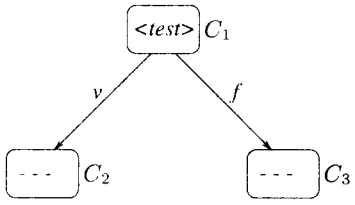


Figure 4 – *Nœud d'un arbre de décision*

Instruction définie par un nœud. L'instruction définie par un nœud d'un arbre de décision est la suivante : on sait que le choix unique à réaliser se trouve dans l'ensemble C_1, on effectue le test figurant dans le nœud ; si le test est *vrai* alors le choix unique est à réaliser dans C_2, sinon le choix unique est à réaliser dans C_3. On descend alors vers le nœud correspondant.

Algorithme de choix. Un algorithme de choix parmi les éléments d'un ensemble $\mathcal{C}$ est défini par un arbre de décision pour cet ensemble. En partant du sommet de l'arbre et en exécutant l'instruction définie par chaque nœud rencontré, on finit par atteindre une feuille. Cette feuille est étiquetée par un singleton qui contient le choix.

Exemple. L'arbre dessiné figure 5 est un arbre de décision permettant de choisir le plus petit parmi trois nombres a, b et c.

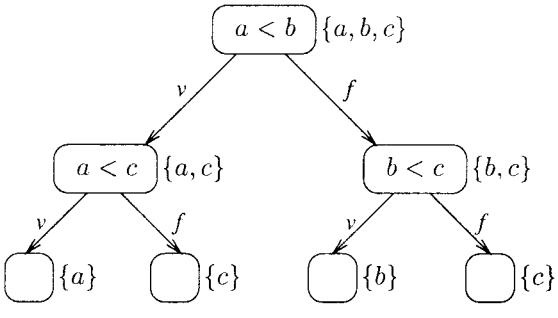


Figure 5 – *Un exemple d'arbre de décision*

2 – On s'intéresse à la hauteur d'un arbre de décision.

a – Quelle est la hauteur minimale en fonction de n d'un arbre de décision pour un choix dans un ensemble de cardinal n ?

b – Quelle est la hauteur maximale en fonction de n d'un arbre de décision pour un choix dans un ensemble de cardinal n ?

Notation. On se pose le problème du tri d'un tableau t contenant n entiers indicés entre 1 et n. n est une constante entière strictement positive. On appelle *indice* un entier i compris entre 1 et n, i.e. $1 \leqslant i \leqslant n$. Dans la suite du problème, si i est un indice, on notera $t(i)$ l'élément d'indice i du tableau t.
Attention ! Cette notation n'est pas liée à un langage de programmation. Elle sert uniquement à l'exposé du problème.

Un tableau q, contenant n entiers indicés entre 1 et n, est appelé une *permutation* de t s'il existe une bijection $\sigma : \{1,\ldots,n\} \to \{1,\ldots,n\}$ telle que $\forall i \in \{1,\ldots,n\}, q(i) = t(\sigma(i))$.

3 – Le tableau résultat du tri de t est une *permutation* de ce tableau. Combien existe-t-il de permutations du tableau t si les entiers contenus dans t sont deux à deux différents ?

4 – On peut considérer le problème du tri du tableau t comme un algorithme de choix parmi les permutations du tableau t et dont les tests sont des comparaisons entre deux éléments du tableau. Donner une approximation de la hauteur minimale de l'arbre de décision correspondant à ce choix.

5 – En déduire que la meilleure complexité envisageable en temps dans le pire des cas d'un algorithme de choix pour le tri du tableau t est $O(n\ln(n))$.

Deuxième partie

Complexité en temps et programmation d'un algorithme de tri

Soit t un tableau de n entiers indicés entre 1 et n. n est une constante entière positive et non nulle. On reprend les notations de la première partie.

6 – Quel type de données faut-il utiliser pour représenter le tableau t pouvant contenir n entiers indicés entre 1 et n ? Pour ce type de données, comment a-t-on accès à la valeur de $t(i)$ pour un indice i particulier ? Programmer un algorithme `echanger` qui prend en arguments le tableau t et deux indices i et j et qui échange les valeurs de $t(i)$ et $t(j)$. Quelle est la complexité en temps de cet algorithme ?

Attention ! Dans les programmes qui vous seront demandés, les tableaux ne devront être modifiés qu'avec l'algorithme `echanger`.

Arbres binaires définis dans un tableau. Un *arbre binaire* est un arbre où chaque nœud a *au plus* deux fils. Soit t notre tableau, pour des indices i et j tels que $i \leqslant j$, on définit l'arbre binaire noté $\alpha(t,i,j)$ par :

- si $2i > j$ alors $\alpha(t,i,j)$ est réduit à une feuille étiquetée par $t(i)$;
- si $2i = j$ alors la racine de $\alpha(t,i,j)$ est un nœud étiqueté par $t(i)$ et possède un seul fils qui est $\alpha(t,2i,j)$;
- si $2i < j$ alors la racine de $\alpha(t,i,j)$ est un nœud étiqueté par $t(i)$ et elle possède un fils gauche qui est $\alpha(t,2i,j)$ et un fils droit qui est $\alpha(t,2i+1,j)$.

7 – Soit $n = 10$ et le tableau t représenté ci-dessous, dessiner l'arbre binaire $\alpha(t, 1, 8)$.

23	12	3	4	67	10	9	6	7	12

On dit qu'un arbre binaire dont les nœuds sont étiquetés par des entiers est *ordonné* si et seulement s'il est réduit à une feuille ou bien si chacun des fils de la racine est ordonné et l'étiquette de la racine est supérieure ou égale aux étiquettes des racines de ses fils.

Note. Dans la suite du problème, si i et j sont deux indices tels que $i \leqslant j$, on confondra l'arbre binaire $\alpha(t, i, j)$ et les éléments du tableau t entre les indices i et j qui le composent.

8 – On désire ordonner des arbres en n'utilisant que les seules opérations consistant à échanger ou à comparer les valeurs de deux éléments du tableau t.

a – Programmer un algorithme ordonner prenant en arguments le tableau t et deux indices i et j tels que $i \leqslant j$. Sachant que les fils éventuels de la racine de $\alpha(t, i, j)$ sont initialement ordonnés, cet algorithme doit ordonner $\alpha(t, i, j)$.

b – Évaluer la complexité en temps dans le pire des cas de cet algorithme.

9 – Soit j un indice, démontrer que chacun des éléments du tableau entre les indices 1 et j est l'étiquette d'un nœud ou d'une feuille de $\alpha(t, 1, j)$.

10 – On désire ordonner $\alpha(t, 1, n)$. Pour cela, on ordonne $\alpha(t, j, n)$ à l'aide de l'algorithme ordonner précédent, pour j décroissant de $\lfloor n/2 \rfloor$ à 1, où $\lfloor n/2 \rfloor$ désigne le résultat de la division entière de n par 2.

a – Programmer un algorithme ordonner_totalement prenant en argument le tableau t et ordonnant $\alpha(t, 1, n)$.

b – Justifier *rigoureusement* que cet algorithme a bien l'effet requis.

c – Évaluer sa complexité en temps dans le pire des cas en fonction de n.

11 – On désire trier les éléments du tableau en ordre croissant.

a – Programmer un algorithme bulle utilisant l'algorithme ordonner et prenant en arguments le tableau t et un indice $j \geqslant 2$. Supposant que $\alpha(t, 1, j)$ est ordonné, cet algorithme doit permuter les éléments du tableau t de manière à ce que $t(j)$ soit le plus grand des éléments du tableau t entre les indices 1 et j et que $\alpha(t, 1, j-1)$ soit ordonné.

b – En déduire et rédiger un algorithme de tri en ordre croissant du tableau t.

c – Montrer que cet algorithme de tri fait bien ce qu'il est censé faire.

d – En évaluer la complexité en temps dans le pire des cas en fonction de n.

FIN DU PROBLÈME DE PROGRAMMATION EN CAML

FIN DE L'ÉPREUVE