CONCOURS COMMUNS POLYTECHNIQUES

ÉPREUVE SPÉCIFIQUE
FILIÈRE MP

INFORMATIQUE

Durée : 2 heures

Les calculatrices programmables et alphanumériques sont autorisées, sous réserve des conditions définies dans la circulaire n° 86-228 du 28 juillet 1986.

PARTIE I. EPREUVE DE LOGIQUE

Résolution d'une énigme par la logique des propositions

Vous êtes perdu sur une piste dans le désert. Vous arrivez à une bifurcation. Chacune des deux pistes est gardée par un sphinx que vous pouvez interroger. Les pistes peuvent soit conduire à une oasis, soit se perdre dans un désert profond (au mieux elles conduisent toutes à une oasis, au pire elles se perdent toutes les deux).

A. Le sphinx de droite vous répond :

Une au moins des deux pistes conduit à une oasis.

B. Le sphinx de gauche vous répond :

La piste de droite se perd dans le désert.

C. Vous savez que les deux sphinx disent tous les deux la vérité, ou bien mentent tous les deux.

Si **OD** est la proposition "Il y a une oasis au bout de la route de droite"
et si **OG** est la proposition "Il y a une oasis au bout de la route de gauche",
alors :

1. Exprimer par une formule de la logique des propositions les affirmations **A** et **B**.

2. Exprimer alors la connaissance **C**.

3. Résoudre l'énigme en utilisant les formules de Morgan ou les tables de vérité ou les deux si nécessaire.

PARTIE II. EXERCICE SUR LES AUTOMATES FINIS

Soit $L \subseteq X^*$ un langage reconnu par l'automate fini $A = \left(Q_A, X, \delta_A, q_0, F\right)$ avec Q_A ensemble des états de A, q_0 état initial de A, F ensemble des états terminaux ; X alphabet de A, δ_A (fonction de transition) application de $Q_A \times X$ dans Q_A.

δ_A^* désignera l'application de $Q_A \times X^*$ dans Q_A définie par :

$\delta_A^*(q, \varepsilon) = q, \quad \forall\, q \in Q_A$ où ε désigne l'élément neutre de X^*,

$\delta_A^*(q, a) = \delta_A(q, a) \quad \forall\, a \in X$ et $\delta_A^*(q, a_1 a_2 \ldots a_n) = \delta_A\left(\delta_A^*(q, a_1 a_2 \ldots a_{n-1}), a_n\right) \quad \forall\, a_1, a_2, \ldots, a_n \in X$.

Soit Préf(L), l'ensemble des préfixes des mots de L, défini par :

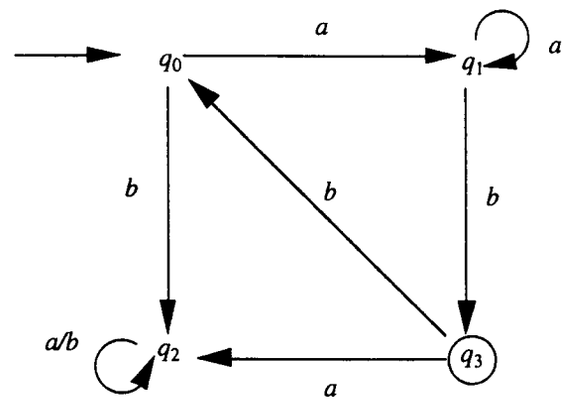
$\text{Préf}(L) = \left\{ w \in X^* \,\middle|\, \exists\, y \in X^* : wy \in L \right\}$.

Nous allons montrer que ce langage est reconnu par un automate fini.

1. Etude d'un exemple :

On considère l'automate fini ci-dessous $A = \left(\{q_0, q_1, q_2, q_3\}, \{a, b\}, \delta_a, q_0, \{q_3\}\right)$ où δ_a est défini par la table de transition suivante :

δ_A	a	b
q_0	q_1	q_2
q_1	q_1	q_3
q_2	q_2	q_2
q_3	q_2	q_0



a) Quel est le langage reconnu par A ?

b) Soit G l'ensemble des états de A accessibles en lisant un préfixe d'un mot de L :

$G = \left\{ q \in Q_A \,\middle|\, \exists w \text{ et } y \in X^* : wy \in L \text{ et } \delta_A^*(q_0, w) = q \right\}$.

Montrer que $q_0 \in G$.
Soit alors $G_0 = \{q_0\}$ et $\forall\, i \geq 0 \quad G_{i+1} = G_i \cup \left\{ q \in Q \,\middle|\, \exists\, q' \in G_i, \exists\, x \in X : q = \delta_A(q', x) \right\}$.
Montrer qu'il existe un rang j tel que $G_{j+1} = G_j$, et en déduire que $G = G_j$.

c) Soit l'automate fini non déterministe $B = \left(Q_A \cup \{t\}, X, \delta_B, q_0, \{t\}\right)$ construit à partir de A en ajoutant une transition de tout état $q \in G$ vers l'état t, étiquetée par ε, à savoir :

$\delta_B(q, a) = \delta_A(q, a) \quad \forall\, q \in Q_A$ et $\forall\, a \in X$
$\delta_B(q, \varepsilon) = t \quad \forall\, q \in G$.

Construire l'automate B puis le déterminiser.

2. Retour au cas général :

On considère un automate fini $A = \left(Q_A, X, \delta_A, q_0, F\right)$ quelconque défini comme dans l'introduction qui reconnaît le langage L. Montrer que l'automate fini non déterministe B, défini en 1.c, reconnaît Préf(L).

PARTIE III. EPREUVE D'ALGORITHMIQUE

Les candidats traiteront **au choix**, soit la **partie en langage Caml**, soit la **partie en langage Pascal.**

En langage Caml

I. Calcul des parties d'un ensemble fini

Les notations suivantes seront utilisées :

{ } pour l'ensemble vide
$1..n$ pour $[1,n]\cap\mathbb{N}$.

Soit E un ensemble fini non vide, on notera $|E|$ le cardinal de E et $\mathcal{P}(E)$ (respectivement $\mathcal{P}^{+}(E)$) l'ensemble des parties de E (respectivement des parties non vides de E). On remarquera que $\mathcal{P}(\{\}) = \{\{\}\}$.

Soit $x \in E$ et $A \in \mathcal{P}(\mathcal{P}(E))$.
On suppose que $\forall e \in A$ alors $x \notin e$.
On définit l'opérateur $\bullet$ de $E \times \mathcal{P}(\mathcal{P}(E))$ dans $\mathcal{P}(\mathcal{P}^{+}(E))$ par la formule :

$$x \bullet A = \{e \cup \{x\} \mid e \in A\}$$

$x \bullet A$ ajoute l'élément x à tous les sous-ensembles éléments de A

Exemple : $7 \bullet \{\{3,4\},\{\},\{2\}\} = \{\{7,3,4\},\{7\},\{7,2\}\}$

1. Soient $e \in \mathcal{P}(E)$ et $x \in E \setminus e$. En utilisant l'opérateur $\bullet$, donner une formule de récurrence permettant de calculer $\mathcal{P}(\{x\} \cup e)$ en fonction de $\mathcal{P}(e)$.

2. Soient $e \in \mathcal{P}(E)$ et $x \in E \setminus e$. Montrer que $\mathcal{P}(\{x\} \cup e)$ est composé d'autant d'ensembles qui contiennent x que d'ensembles qui ne le contiennent pas, c'est-à-dire que :

$$\left|\{Y \in \mathcal{P}(\{x\} \cup e) \mid x \in Y\}\right| = \left|\{Y \in \mathcal{P}(\{x\} \cup e) \mid x \notin Y\}\right|$$

3. Montrer que si $|E| = n$, avec $n > 0$, alors :

$$\sum_{f \in \mathcal{P}(E)} |f| = n\ 2^{n-1}$$

autrement dit que la somme des cardinaux des parties de E est $n\ 2^{n-1}$.

Dans la suite de l'énoncé, on supposera que les ensembles sont représentés par des listes Caml. Chaque élément de l'ensemble représenté doit figurer une fois et une seule dans la liste. L'ordre des éléments dans la liste n'est pas significatif.

Exemple : L'ensemble $\{\{7,3,4\},\{7\},\{7,2\}\}$, sera représenté par la liste Caml $[[7;3;4];[7];[7;2]]$, modulo la permutation des éléments des listes.

On rappelle que l'opérateur Caml @, de type 'a list - > 'a list - > 'a list, désigne la concaténation de listes.

Exemple : [2;4;5] @ [1;7] s'évalue en [2;4;5;1;7].

4. Ecrire la fonction récursive Caml **ajouter** qui prend deux arguments, un objet x (de type 'a) et une liste de listes $e = [l_1;...;l_n]$ (de type 'a list list) qui renvoie la liste de listes correspondant à l'ensemble $x \bullet e$.

On rappelle que dans les hypothèses de définition de l'opérateur $x \bullet e$, x n'appartient à aucun des ensembles éléments de e.

Exemple : ajouter 1 $[[2;3];[5]]$ renvoie $[[1;2;3];[1;5]]$.

5. Ecrire une fonction récursive Caml **parties** qui à partir d'un argument e (de type 'a list) renvoie la liste de listes $[l_1;...;l_n]$ (de type 'a list list) qui correspond à $\mathcal{P}(e)$.

II. Application au problème dit du sac-à-dos

On dispose d'un sac-à-dos qui permet de transporter un volume V et dont la charge pondérale maximale est P. On possède n objets, numérotés de **1** à n. Deux fonctions que l'on supposera prédéfinies en Caml donnent le volume $v(i)$ et le poids $p(i)$ de l'objet i.

On veut transporter dans le sac le sous-ensemble de ces n objets qui représente le volume maximal inférieur ou égal à V (perte minimale d'espace), et à volume égal le sous-ensemble qui correspond au poids maximal inférieur ou égal à P. On qualifiera d'optimal un tel sous-ensemble.

On supposera que $\forall i, i \in 1..n$ on a $v(i) \leq V$ et $p(i) \leq P$. Ceci garantit l'existence d'au moins une solution. On remarquera qu'il peut y avoir plusieurs solutions optimales équivalentes. Dans la suite du problème on ne s'intéressera qu'à la recherche de l'une de ces solutions.

On admettra que la recherche de l'une des solutions optimales de la résolution du problème du sac-à-dos nécessite d'envisager tous les sous-ensembles possibles de l'ensemble des n objets.

On dit, dans ce cas, que le problème est NP-complet.

A. Utilisation du calcul des parties d'un ensemble

1. **parties** $[1;2;...;n]$ calcule la liste correspondant à $\mathcal{P}(1..n)$ avec $\mathcal{P}(1..n) = \{e \mid e$ est inclus dans $1..n\}$

Soit alors

$$\mathcal{P}_1(1..n) = \left\{ \left(e, \sum_{i \in e} p(i), \sum_{i \in e} v(i) \right) \middle| e \in \mathcal{P}_1(1..n)^+ \right\}$$

En s'inspirant de la fonction **parties** de la question I-5, écrire une fonction Caml récursive **chargements**, qui prend en argument une liste l (de type 'a list) telle que **chargements** $[1;2;...;n]$ renvoie la liste des triplets correspondant à l'ensemble :

$$\left\{ \left(e, \sum_{i \in e} p(i), \sum_{i \in e} v(i) \right) \middle| e \in \mathcal{P}_1(1..n)^+ \right\}$$

Exemple : si $l = [1;2]$ on doit obtenir : $\left[([1], p(1), v(1)); ([2], p(2), v(2)); ([1;2], p(1)+p(2), v(1)+v(2))\right]$

On sera amené à écrire une version modifiée de la fonction **ajouter** que l'on appellera **ajouter_objet**.

2. Ecrire une fonction **sac_a_dos** telle que

sac_a_dos P V $[1;2;...;n]$ renvoie le triplet qui correspond à l'une des solutions optimales.

B. Une stratégie alternative

Pour engendrer toutes les parties non vides de l'intervalle $1..n$, il existe une autre solution plus économique en espace mémoire. Elle consiste à calculer les 2^n-1 termes de la suite définie par :

$u_1 = [1]$;
si $1 < p < 2^n$ et $u_{p-1} = [x_1; x_2; ...; x_k]$, alors :

$$\text{si } k > 1, \quad u_p = \begin{cases} [x_1 + 1; x_1; x_2; ...; x_k] \text{ si } x_1 < n \\ [x_2 + 1; x_3; ...; x_k] \quad \text{sinon} \end{cases}$$

si $k = 1$, u_p n'est défini que si $x_1 < n$, par $u_p = [x_1 + 1; x_1]$.

1. Calculer manuellement la suite $u_1, ..., u_7$ pour $n = 3$.

2. Ecrire une fonction Caml **terme_suivant** qui calcule u_p à partir de n et u_{p-1}.

3. Soit les deux fontions Caml suivantes :

```
let rec deux_puissance =
        function
            0 -> 1
          | n -> 2* (deux_puissance (n - 1))
;;
let enumerer  n =
   let n1 = deux_puissance n and u = ref[1] and liste_parties = ref[[1]]
       in
            for i = 1 to n1-2
                do
                    u := terme_suivant n !u;
                    liste_parties := !u :: !liste_parties
                done;
                !liste_parties;;
```

En s'inspirant de la fonction **enumerer** et en utilisant **terme_suivant** écrire une fonction Caml **sac_a_dos'** telle que **sac_a_dos' P V** n renvoie une des solutions optimales pour n objets.

C. Un calcul de complexité

1. Calculer la complexité $T(n)$ de la fonction **enumerer** en prenant uniquement en compte l'opérateur de liste ::.

2. Calculer en fonction de $n = |e|$ la complexité de la fonction **parties** e en prenant uniquement en compte l'opérateur de liste ::.

On supposera que la complexité de l @l' est donnée par la formule $T[l \,@\, l'] = |l|$, où $|l|$ désigne la longueur de l. On pourra utiliser la propriété démontrée à la question I-2.

3. Comparer les deux complexités calculées.

En langage Pascal

I. Calcul des parties d'un ensemble fini

Les notations suivantes seront utilisées :

{ } pour l'ensemble vide
$1..n$ pour $[1,n]\cap \mathbf{N}$.

Soit E un ensemble fini non vide, on notera $|E|$ le cardinal de E et $\mathcal{P}(E)$ (respectivement $\mathcal{P}^{+}(E)$) l'ensemble des parties de E (respectivement des parties non vides de E). On remarquera que $\mathcal{P}(\{\}) = \{\{\}\}$.

Soit $x \in E$ et $A \in \mathcal{P}(\mathcal{P}(E))$.
On suppose que $\forall e \in A$ alors $x \notin e$.
On définit l'opérateur $\bullet$ de $E \times \mathcal{P}(\mathcal{P}(E))$ dans $\mathcal{P}(\mathcal{P}^{+}(E))$ par la formule :

$$x \bullet A = \{e \cup \{x\} \mid e \in A\}$$

$x \bullet A$ ajoute l'élément x à tous les sous-ensembles éléments de A

Exemple : $7 \bullet \{\{3,4\},\{\},\{2\}\} = \{\{7,3,4\},\{7\},\{7,2\}\}$

1. Soient $e \in \mathcal{P}(E)$ et $x \in E \setminus e$. En utilisant l'opérateur $\bullet$, donner une formule de récurrence permettant de calculer $\mathcal{P}(\{x\} \cup e)$ en fonction de $\mathcal{P}(e)$.

2. Soient $e \in \mathcal{P}(E)$ et $x \in E \setminus e$. Montrer que $\mathcal{P}(\{x\} \cup e)$ est composé d'autant d'ensembles qui contiennent x que d'ensembles qui ne le contiennent pas, c'est-à-dire que :

$$\left|\{Y \in \mathcal{P}(\{x\} \cup e) \mid x \in Y\}\right| = \left|\{Y \in \mathcal{P}(\{x\} \cup e) \mid x \notin Y\}\right|$$

3. Montrer que si $|E| = n$, avec $n > 0$, alors :

$$\sum_{f \in \mathcal{P}(E)} |f| = n\ 2^{n-1}$$

autrement dit que la somme des cardinaux des parties de E est $n\ 2^{n-1}$.

Dans la suite de l'énoncé, on supposera que E est l'ensemble **N** des nombres entiers et que tous les ensembles sont représentés par des listes en Pascal. Chaque élément de l'ensemble représenté doit figurer une fois et une seule dans la liste. L'ordre des éléments dans la liste n'est pas significatif.

Deux types de listes seront considérées comme prédéfinies en Pascal :

- le type liste de nombres entiers appelé **Liste** servira à représenter un élément de $\mathcal{P}(\mathbf{N})$
  - **nil** représente la liste vide
  - function **cons** (*x*: integer; *l*: Liste): Liste
    est une fonction qui renvoie une liste dont la tête est *x* et la queue *l*
  - function **tete** (*l*: Liste): integer
    est une fonction qui renvoie la tête de *l*
  - function **queue** (*l*: Liste): Liste
    est une fonction qui renvoie la queue de *l*
  - function **concat** (*l*1, *l*2: Liste): Liste
    est une fonction qui renvoie la liste obtenue par concaténation de *l*1 et de *l*2

On rappelle que la concaténation de deux listes l_1 et l_2 est l'opération qui construit une liste contenant tous les éléments de l_1 suivis de tous les éléments de l_2.

- le type liste de liste de nombres entiers appelé **L_Liste** servira à représenter un sous-ensemble de $\mathcal{P}(\mathcal{P}(\mathbf{N}))$

  - **nil** représente la liste vide
  - function **L_cons** (*x*: Liste; *l*: L_Liste): L_Liste
    est une fonction qui renvoie une liste dont la tête est *x* et la queue *l*
  - function **L_tete** (*l*: L_Liste): Liste
    est une fonction qui renvoie la tête de *l*
  - function **L_queue** (*l*: L_Liste): L_Liste
    est une fonction qui renvoie la queue de *l*
  - function **L_concat** (*l*1, *l*2: L_Liste): L_Liste
    est une fonction qui renvoie la liste obtenue par concaténation de *l*1 et de *l*2

Exemples :

L'ensemble $\{7,3,4\}$ sera représenté par la liste de type Liste construite par l'expression $\text{cons}(7,\text{cons}(3,\text{cons}(4,\text{nil})))$ ou $\text{cons}(3,\text{cons}(7,\text{cons}(4,\text{nil})))$ ou toute autre permutation.

De même l'ensemble $\{\{7,3,4\},\{7\},\{7,2\}\}$ sera représenté par la liste de type L_Liste construite par l'expression :

```
L_cons ( cons(7, cons(3, cons(4, nil)))
       , L_cons( cons(7, nil)
              , L_cons(cons(7, cons(2, nil))
                     , nil)))
```

4. Ecrire la fonction récursive Pascal **ajouter** (*x*:integer; *e*: L_Liste):L_Liste qui renvoie la liste correspondant à l'ensemble $x \bullet e$.

On rappelle que dans les hypothèses de définition de l'opérateur $x \bullet e$, x n'appartient à aucun des ensembles éléments de *e*.

Exemple : ajouter (1, L_cons(cons(2, cons(3, nil)), L_cons(cons(5, nil), nil)))
renvoie L_cons(cons(1, cons(2, cons(3, nil))), L_cons(cons(1, cons(5, nil)), nil))

5. Ecrire une fonction récursive Pascal **parties**(*e*: Liste):L_Liste qui renvoie la liste qui correspond à $\mathcal{P}(e)$.

II. Application au problème dit du sac-à-dos

On dispose d'un sac-à-dos qui permet de transporter un volume V et dont la charge pondérale maximale est P. On possède n objets, numérotés de **1** à n. Deux fonctions que l'on supposera prédéfinies en Pascal donnent le volume $v(i)$ et le poids $p(i)$ de l'objet i.

On veut transporter dans le sac le sous-ensemble de ces n objets qui représente le volume maximal inférieur ou égal à V (perte minimale d'espace), et à volume égal le sous-ensemble qui correspond au poids maximal, inférieur ou égal à P. On qualifiera d'optimale une telle solution.

On supposera que $\forall i, i \in 1..n$ on a $v(i) \leq V$ et $p(i) \leq P$. Ceci garantit l'existence d'au moins une solution. On remarquera qu'il peut y avoir plusieurs solutions optimales équivalentes. Dans la suite du problème on ne s'intéressera qu'à la recherche de l'une de ces solutions.

On admettra que la recherche de l'une des solutions optimales de la résolution du problème du sac-à-dos nécessite d'envisager tous les sous-ensembles possibles de l'ensemble des n objets.

On dit, dans ce cas, que le problème est NP-complet.

A. Utilisation du calcul des parties d'un ensemble

1. **parties** (cons(1, cons(2, ... cons(n, nil)...))) calcule la liste correspondant à $\mathcal{P}(1..n)$ avec $\mathcal{P}(1..n) = \{e \mid e \text{ est inclus dans } 1..n\}$

Soit alors l'ensemble de triplets

$$\mathcal{P}_1(1..n) = \left\{ \left(e, \sum_{i \in e} p(i), \sum_{i \in e} v(i) \right) \mid e \text{ est inclus dans } 1..n \right\}$$

On supposera défini le type

```
type Triplet = record
                 ensemble: Liste;
                 somme_poids, somme_volume: real
               end;
```

ainsi que le type liste de triplets, T_Liste et les fonctions associées T_cons, T_tete, T_queue, T_concat.

En s'inspirant de la fonction **parties** de la question I-5, écrire une fonction Pascal récursive **chargements**(*l*: Liste): T_Liste telle que **chargements** (cons(1, cons(2, ...cons(n, nil)...))) renvoie la T_Liste de tous les triplets correspondant à l'ensemble

$$\left\{ \left(e, \sum_{i \in e} p(i), \sum_{i \in e} v(i) \mid e \in \mathcal{P}_1(1..n)^+ \right) \right\}$$

Exemple : si l = cons(1, cons(2, nil)) on doit obtenir T_cons(r_1, T_cons(r_2, T_cons(r_3, nil)))

où r_1, r_2 et r_3 désignent les enregistrements contenant respectivement
cons(1, nil), $p(1)$, $v(1)$
cons(2, nil), $p(2)$, $v(2)$
cons(1, cons(2, nil)), $p(1)+p(2)$, $v(1)+v(2)$

On sera amené à écrire une version modifiée de la fonction **ajouter** (*x*: integer; *l*: L_Liste): L_Liste que l'on appellera **ajouter_objet**(*x*: integer; *l*: T_Liste): T_Liste.

2. Ecrire une procédure **sac_a_dos**(*P*, *V*: real; *l*: Liste; Var *t*:Triplet) telle que **sac_a_dos**(*P*, *V*, cons(1, cons(2, ... cons(*n*, nil)...)),*t*) affecte à *t* le triplet correspondant à une des solutions optimales.

B. Une stratégie alternative

Pour engendrer toutes les parties non vides de l'intervalle 1..*n*, il existe une autre solution plus économique en espace mémoire. Elle consiste à calculer les 2^n-1 termes de la suite définie par :

$u_1 = [1]$;
si $1 < p < 2^n$ et $u_{p-1} = [x_1; x_2; ...; x_k]$, alors :

$$\text{si } k > 1, \quad u_p = \begin{cases} [x_1 + 1; x_1; x_2; ...; x_k] \text{ si } x_1 < n \\ [x_2 + 1; x_3; ...; x_k] \quad \text{sinon} \end{cases}$$

si $k = 1$, u_p n'est défini que si $x_1 < n$, par $u_p = [x_1 + 1; x_1]$

où $[x_1; x_2; ...; x_k]$ désigne la liste $\text{cons}\left(x_1, \text{cons}\left(x_2, ..., \text{cons}(x_k, \text{nil})...\right)\right)$

1. Calculer manuellement la suite $u_1, ..., u_7$ pour $n = 3$.

2. Ecrire une fonction Pascal **terme_suivant**(n: integer; u: Liste): Liste qui calcule u_p en fonction de n et de u_{p-1}.

3. Soient la fonction et la procédure Pascal suivantes :

```
function deux_puissance(n: integer): integer;
   begin
       if n=0      then deux_puissance:=1
            else  deux_puissance:=2*deux_puissance (n-1);
   end;

procédure enumerer (n:integer);
   var n1: integer; u: Liste; liste_parties: L_Liste;
         begin
            n1:=deux_puissance(n);
            u:=cons(1, nil);
            liste_parties:=L_cons(cons(1,nil),nil);
            for i = 1 to n1-2
                  begin
                        u:=terme_suivant(n,u);
                        liste_parties:=L_cons(u, liste_parties)
                  end;
            enumerer:=liste_parties;
         end;
```

On supposera que $0 < n \leq 14$. En s'inspirant de la procédure **enumerer** et en utilisant **terme_suivant** écrire une procédure Pascal **sac_a_dos_bis**(P, V: real; n: integer; Var t:Triplet) telle que **sac_a_dos_bis**(P, V, n, t) affecte à t le triplet correspondant à une des solutions optimales pour n objets.

C. Un calcul de complexité

1. Calculer la complexité $T(n)$ de la fonction **enumerer** en prenant uniquement en compte le nombre d'appels des fonctions cons et L_cons.

2. Calculer en fonction de $n = |e|$ la complexité de la fonction **parties**(e) en prenant uniquement en compte le nombre d'appels des fonctions cons et L_cons.

On supposera que la complexité de concat ($l1$, $l2$) est donnée par la formule T[concat($l1$, $l2$)]= $|l1|$, où $|l1|$ désigne la longueur de $l1$. On pourra utiliser la propriété démontrée à la question I-2.

3. Comparer les deux complexités calculées.

Fin de l'énoncé

www.tunisie-etudes.info

Ce document a été téléchargé depuis
www.tunisie-etudes.info

Des documents gratuits, devoirs, examens, cours, exercices, corrigés... Ainsi que toute une rubrique pour vous aider à trouver un emploi sans oublier les avis de concours en direct

Notre page Twitter :
http://www.twitter.com/TunisieEtudes
Notre page FaceBook :
http://www.facebook.com/TunisieEtudes



Merci d'avoir choisi www.tunisie-etudes.info
Bonne lecture et bon travail