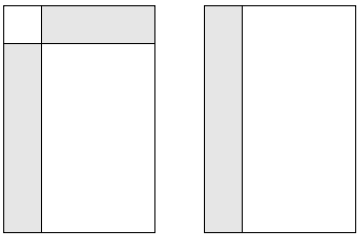
Concours blanc 2016: option informatique

Le sujet est constitué de deux parties, chacune à rédiger sur une copie séparée

- **Durée : 4 heures**
- **L'usage de tout dispositif électronique (calculatrice, téléphone portable, ...) est interdit.**
- La clarté et la lisibilité de la copie sont des qualités essentielles. Toute copie sale, peu claire ou à la rédaction approximative sera **pénalisée**. Numérotez vos feuilles doubles et indiquez votre nom sur chacune d'entre elles.



Première feuille Autres feuilles

La première page de la copie **doit laisser un cadre de 5cm**. Votre copie doit comporter une marge d'**au moins 3cm**. Toute copie ne respectant pas ces spécifications perdra des points.

- Un problème, ou un exercice à plusieurs questions, n'est pas un grand chelem, et il est approprié en l'absence de réponse à une question de supposer les résultats et de passer à la suivante. **Ne vous arrêtez pas à la première question dont vous n'avez pas la réponse !** Aucune copie ne sera pénalisée pour avoir tenté un début de réponse.
- S'il vous semble qu'une erreur s'est glissée dans un énoncé, justifiez-vous et poursuivez l'exercice *mutatis mutandis*, c'est-à-dire *en changeant ce qui doit être changé.*

1 Antichaînes d'entiers (à rédiger sur une copie séparée)

On s'intéresse ici à partitionner une liste d'entiers en listes d'entiers ne se divisant pas entre eux:
étant donnée une liste l, on cherche à déterminer des listes non vides $l_1, ..., l_p$ telles que

$$\forall i \in [\![1, p]\!], l_i \neq \emptyset$$

$$\forall i, j \in [\![1, p]\!], i \neq j \Rightarrow l_i \cap l_j = \emptyset$$

$$\bigcup_{i=1}^{p} l_i = l$$

$$\forall i \in [\![1, p]\!], \forall x, y \in l_i, x \notin \mathbb{Z}y \text{ et } y \notin \mathbb{Z}x$$

On dira que deux entiers x et y sont *comparables* si x divise y ou si y divise x.

Question 1	Écrire une fonction caml `comparables : int -> int -> bool` telle que `comparables x y = true` si, et seulement si, x et y sont comparables (i.e. x divise y ou y divise x).

Question 2	Écrire une fonction caml `incomparables_avec : int -> int list -> bool` telle que `incomparables_avec x l = true` si, et seulement si, x est incomparable avec chaque y de l.

Question 3	En modifiant la fonction précédente, écrire une fonction caml `separe : int -> int list -> (int list) * (int list)` telle que `separe x l` rend un couple (l_1, l_2) constitué de la liste l_1 des éléments de l incomparables avec x et de la liste l_2 des éléments de l comparables avec x. Par exemple, on aura `separe 4 [3;2;5;8;9] = ([3;5;9],[2;8])` La fonction `separe` aura une complexité linéaire en fonction de la longueur de la liste l.

Définition	On appelle *antichaîne* de l toute partie de l contituée d'éléments incomparables 2 à 2. Par exemple `[2;5;9]` est une antichaîne de `[3;2;5;8;9]`.

On considère la fonction caml suivante:

```
let rec une_antichaine l =
  match l with
    | [] -> [],[]
    | x::q -> let li,lc = separe x q in
              let c,l1 = une_antichaine li in
              x::c, l1 @ lc
;;
```

Question 4	Déterminer le type de la fonction `une_antichaine`.

Question 5	Démontrer que la fonction `une_antichaine` termine. Que calcule-t-elle?

Question 6	Montrer que la fonction `une_antichaine` a une complexité au plus quadratique en la longueur de la liste l.

Question 7	Écrire une fonction `antichaines : int list -> int list list` telle que `antichaines l` rend une partition de la liste l en antichaînes. Quelle est sa complexité en fonction de n, la longueur de la liste l?

Question 8	Donner un exemple de liste de la forme `[1;...;n]` qui admet au moins deux partitions en antichaînes différentes.

Question 9	(difficile) Décrire un algorithme permettant, pour une liste d'entiers, de calculer une partition en antichaînes de cardinal minimal (i.e. avec un minimum d'antichaînes). Donner sa complexité.

On s'intéresse maintenant au cas particulier des partitions en antichaînes des entiers de 1 à n:

$$l = [1; 2; ...; n]$$

On cherche à déterminer k, le nombre minimal d'antichaînes dans une partition en antichaînes de l.

Soit p la plus grande puissance de 2 telle que $1 \leqslant 2^p \leqslant n$

Question 10	Exprimer p en fonction de n.

Question 11	Pour tout $i \in \mathbb{N}$ on pose $A_i = [\![2^i, 2^{i+1}[\![$. Montrer que $$\{A_0, A_1, ..., A_{p-1}, A_p \cap [\![1, n]\!]\}$$ est une partition en antichaînes de $[\![1, n]\!]$.

Question 12	En considérant l'ensemble $\{1, 2, 2^2, ..., 2^p\}$, montrer que $k \geqslant p + 1$.

Question 13	Déduire de ce qui précède que $k = p + 1$, et exprimer simplement k en fonction de n.

Le résultat qu'on obtient ici est en fait une conséquence du théorème de Dilworth, qui concerne les *ordres partiels sur des ensembles finis*. En effet la relation de divisibilité sur les entiers est une relation d'ordre partielle.

2 Multiplication d'une suite de matrices (à rédiger sur une copie séparée)

Soit $(A_0, \cdots, A_{n-1})$ une suite de n matrices telles que, pour tout $0 \leqslant i < n$, A_i est une matrice $d_i \times d_{i+1}$.

Le but de ce problème est de calculer le produit

$$A_0 * A_1 * \cdots * A_{n-1}$$

2.1 Préambule

Soient $p, q \in \mathbb{N}$. Une matrice $p \times q$ est un *tableau* de $p * q$ éléments (ici des entiers relatifs), indicés par deux indices $0 \leqslant i < p$ et $0 \leqslant j < q$. On note $A = [a_{i,j}]_{0 \leqslant i < p, 0 \leqslant j < q}$ où $a_{i,j}$ est l'élément de la *ligne* i et de la *colonne* j.

Le produit de deux matrices $A = [a_{i,j}]_{0 \leqslant i < p, 0 \leqslant j < q}$ et $B = [b_{i,j}]_{0 \leqslant i < q, 0 \leqslant j < r}$ résulte en la matrice $A * B = C = [c_{i,j}]_{0 \leqslant i < p, 0 \leqslant j < r}$ définie par $c_{i,j} = \sum_{0 \leqslant k < q} a_{i,k} b_{k,j}$. Notons que la matrice $A * B$ n'est définie que si les dimensions sont compatibles, c'est-à-dire que le nombre de colonnes de A doit être égal au nombre de lignes de B.

Question 14	Écrire en **Caml** un algorithme, appelé $mult\_matrice$, qui calcule le produit d'une matrice $p \times q$ avec une matrice $q \times r$ en $O(prq)$ opérations élémentaires (multiplications, additions et soustractions d'entiers). Prouver la correction et la complexité de l'algorithme. *En Caml, une matrice peut être représentée par un vecteur de vecteurs (lignes).*

2.2 Premiers exemples de parenthésage

On rappelle que la multiplication de matrices est associative. Ainsi, $((A_0 * A_1) * A_2) = (A_0 * (A_1 * A_2))$.

Pour pouvoir appliquer l'algorithme de la Question 1 pour calculer le produit $A_0 * A_1 * \cdots * A_{n-1}$, il faut définir dans quel ordre effectuer les multiplications. En d'autres termes, il faut choisir un *parenthésage* du produit de matrices. Par exemple, pour une suite de 4 matrices, il faut choisir parmi:

$$\begin{array}{l} (A_0 * (A_1 * (A_2 * A_3))) \\ (A_0 * ((A_1 * A_2) * A_3)) \\ ((A_0 * A_1) * (A_2 * A_3)) \\ (((A_0 * A_1) * A_2) * A_3) \\ (((A_0 * (A_1 * A_2)) * A_3) \end{array} \tag{1}$$

Question 15	Pourquoi est-ce que tous les parenthésages aboutissent au même résultat ?

Définition	Le *coût* d'un parenthésage de $(A_0, \cdots, A_{n-1})$ est la complexité du calcul du produit de cette suite de matrices en suivant l'ordre défini par le parenthéage.

Question 16	Soit $A = \begin{bmatrix} 2 & 7 \\ 3 & -5 \\ 2 & 0 \end{bmatrix}$, $B = \begin{bmatrix} 7 & -4 & 8 \\ 1 & 4 & 4 \end{bmatrix}$ et $C = \begin{bmatrix} 8 & 7 \\ -3 & -5 \\ 2 & 9 \end{bmatrix}$. En utilisant l'algorithme de la Question 1, quelle est le coût de $((A*B)*C)$? et de $(A*(B*C))$? Conclusion ?

Question 17	On rappelle que A_i est une matrice $d_i \times d_{i+1}$. Calculer le coût du parenthésage $(((\cdots(A_0 * A_1) * A_2) * \cdots * A_{n-2}) * A_{n-1})$ en fonction de $d_0, \cdots, d_n$. Même question pour $(A_0 * (A_1 * (A_2 * \cdots * (A_{n-2} * A_{n-1}))) \cdots)$.

Ainsi, pour calculer efficacement le produit d'une suite de matrices, il s'agit de choisir un "bon" ordre dans lequel faire les multiplications. En d'autres termes, il faut choisir un parenthésage de coût minimum.

Question 18	Quel est le nombre de parenthésages possibles pour une séquence de 5 matrices ? de 6 matrices ?

Question 19	Prouver que le nombre $P(n)$ de parenthésages possibles pour une séquence de n matrices satisfait: $$P(n) = \begin{cases} 1 & \texttt{si } n = 1 \\ \sum_{1 \leqslant k < n} P(k)P(n-k) & \texttt{si } n \geqslant 2 \end{cases} \quad (2)$$

On admet que $P(n+1)$ défini ci-dessus est le nombre de Catalan $C(n)$ et que $C(n) = \Omega(\frac{4^n}{n^{3/2}})$.

Ainsi, tester la complexité de chacun des parenthésages (afin de choisir un meilleur) prendrait un temps exponentiel en n, ce que nous voulons éviter.

Le but de ce qui suit est de trouver un parenthésage optimal (de coût minimum) dans un temps raisonnable.

2.3 Formule de récurrence

Notation. À partir de maintenant, on note $A_{i\cdots j} = A_i * A_{i+1} * \cdots * A_j$ pour tout $0 \leqslant i \leqslant j < n$.

Question 20	Supposons que le produit $A_{i\cdots j}$ est obtenu par un parenthésage $\mathcal{P}$ optimal (de coût minimum). Soit k_{ij} ($i \leqslant k_{ij} < j$) tel que la dernière multiplication effectuée est $A_{j\cdots k_{ij}} * A_{k_{ij}+1\cdots j}$. Montrer que le parenthésage de $A_{i\cdots k_{ij}}$ induit par $\mathcal{P}$ est optimal. Même question pour $A_{k_{ij}+1\cdots j}$.

Notation. Pour $0 \leqslant i \leqslant j < n$, on pose $c[i,j]$ le coût d'un parenthésage optimal de $(A_i * \cdots * A_j)$.

Question 21	Soit $0 \leqslant i < n$. Que vaut $c[i,i]$? (Expliquer en une phrase)

Question 22	Montrer que, pour tout $i \leqslant k < j$, $$c[i,j] \quad \leqslant \quad c[i,k] + c[k+1,j] + d_{i-1} * d_k * d_j. \quad (3)$$

Question 23	Déduire des questions précédentes que, pour tout $0 \leqslant i \leqslant j < n$, $$c[i,j] = \begin{cases} 0 & \texttt{si } i = j \\ \min\limits_{i \leqslant k < j} \{\, c[i,k] + c[k+1,j] + d_{i-1} * d_k * d_j \,\} & \texttt{si } i < j \end{cases} \quad (4)$$

2.4 Programmation dynamique

Dans ce qui suit, pour tout $0 \leqslant i \leqslant j < n$, on pose $0 \leqslant k_{ij} < n-1$ un entier tel qu'il existe un parenthésage optimal de $(A_i * \cdots * A_j)$ de la forme $((A_i * \cdots * A_{k_{ij}}) * (A_{k_{ij}+1} * \cdots * A_j))$

Question 24	Donner un exemple simple pour lequel k_{ij} n'est pas unique.

Question 25	Écrire en **Caml** un algorithme, appelé $meilleur\_parenthesage$, qui prend: **en entrée :** un tableau $T = [A_0, \cdots, A_{n-1}]$ de matrices, **et renvoie :** deux matrices $n \times n$, appelées cout_opt et position_opt, telles que, pour tout $0 \leqslant i \leqslant j < n$ $$\begin{array}{rcl} \text{cout\_opt[j][i]} & = & c[i,j] \\ \text{position\_opt[j][i]} & = & k_{ij} \end{array} \quad (5)$$ *On rappelle qu'en Caml, on peut repésenter une matrice par un tableau de tableaux.*

Question 26	Quelle est la complexité de votre algorithme ?

2.5 Conclusion

L'algorithme suivant prend en entrée un tableau de matrices $T = [A_0, \cdots, A_{n-1}]$ tel que le produit $A_0 * \cdots * A_{n-1}$ est bien défini.

```
let Alg T =

        let rec Alg_aux T pos i j =
                if j>i then
                        let X = Alg_aux T pos i pos[j][i]  in
                        let Y = Alg_aux T pos (pos[j][i]+1) j in
                        mult_matrice X Y
                else    T[i]   in

        let cout_opt,pos_opt = meilleur_parenthesage T in
        let n = vect_length T in

        Alg_aux T pos_opt 0 (n-1) ;;
```

Question 27	Que fait l'algorithme ci-dessus ? (Expliquer son exécution en détail. En particulier, décrire les entrées de la fonction Alg_aux et ce que retourne cette fonction [a])

[a] "Décrire en détail" ne signifie pas "écrire un roman"...

2.6 Post Scriptum : culture générale

Le meilleur algorithme connu actuellement pour calculer le produit de deux matrices $p \times p$ est dû à François Le Gall (en 2014) et a une complexité de $O(p^{2.3728639})$.

Question 28	En utilisant l'algorithme de François Le Gall, quelle est la complexité de calculer le produit d'une suite de n matrices $p \times p$?

Note culturelle. Trouver un meilleur algorithme que celui de Le Gall pour multiplier deux matrices est un des challenges algorithmiques actuels.

Notez qu'on ne peut espérer un algorithme en $o(p^2)$ puisqu'il faut au moins lire les deux matrices en entrée, ce qui demande un temps $\Omega(p^2)$.