Option informatique

Présentation du sujet

Le sujet comporte quatre parties, les trois dernières étant reliées par un même thème : l'étude de structures de données permettant de modéliser un ensemble d'entiers positifs.

La première partie, totalement indépendante du reste du sujet, étudie deux exemples de transformation de langages réguliers. Elle permet d'évaluer, entre autres, la capacité des candidats à utiliser les techniques classiques de la théorie des langages (raisonnement inductif, lemme de l'étoile, construction d'automates par « blocs »...).

La deuxième partie fait manipuler des représentations classiques d'ensembles (liste triée, vecteur trié, arbre binaire de recherche). Les candidats sont amenés à écrire de légères variantes d'algorithmes vus en cours en première ou deuxième année.

La troisième partie introduit une structure d'arbre binaire complet étiqueté par des booléens pour représenter un ensemble d'entiers positifs dont les feuilles servent à marquer les éléments de l'ensemble et les nœuds internes à stocker des informations permettant d'accélérer les opérations sur une telle structure. On propose d'écrire les fonctions de test d'appartenance, d'insertion ou de suppression d'un élément, de calcul du minimum et des fonctions permettant de « parcourir » les éléments de l'ensemble. On étudie à chaque fois leur complexité.

Enfin, la quatrième partie propose une autre structure d'arbre, plus complexe, particulièrement efficace pour de gros ensembles « denses ».

Les chapitres abordés sont : langage et automates, structures de données, algorithmique classique et complexité.

Analyse globale des résultats

Le sujet est relativement long et la fin de l'épreuve (question 37 et au delà) n'a quasiment jamais été abordée.

L'épreuve comporte cette année une proportion plus importante de questions de programmation que de questions théoriques. La partie programmation est traitée plutôt convenablement sur la plupart des copies, avec des algorithmes qui « marchent » mais dont l'efficacité n'est pas toujours optimale.

Les réponses apportées aux questions théoriques (tant sur les langages que dans la partie III) ont été, par contre, souvent trop confuses ou sans véritable justification. Sur un nombre non négligeable de copies, la partie I a été à peine abordée alors qu'elle représentait un gros quart de l'épreuve (et que langages et automates représentent une part importante du programme de deuxième année).

Beaucoup de temps ou de points ont été perdus par une lecture trop superficielle de l'énoncé. Certains candidats justifient des réponses alors que l'énoncé de la question indique explicitement de ne pas le faire, oublient de donner des complexités (pourtant élémentaires) alors qu'ils ont les bons programmes et que la question le demande, ou ne pensent pas, quand il faut donner un contre-exemple à une propriété, à regarder parmi les situations étudiées dans les questions précédentes du sujet.

Le jury rappelle qu'il apporte une grande attention à la présentation de la copie (une copie aérée avec des résultats encadrés est bien plus facile à lire, ce qui ne peut que profiter au candidat) et a pénalisé les écritures illisibles, en recrudescence. On rencontre aussi encore trop de fautes d'orthographe grossières (« on parcours... » « les languages rationels... »). On ne peut que conseiller de faire un effort sur ces points.

Commentaires sur les réponses apportées et conseils aux futurs candidats

Sur la partie I, le jury a rencontré beaucoup de confusions dans les objets manipulés :

- confusion entre a^*ba^* qui est un langage (ici l'ensemble des mots $\{a^nba^p,\ (n,p) \in \mathbb{N}^2\}$) et $\{a^*ba^p, p \in \mathbb{N}\}$ qui est un ensemble de langages (dont l'union vaut a^*ba^*) (**Q1**) ;
- dans le même ordre d'idée, confusion entre $\{L_{q,q'},\ (q,q') \in \cdots\}$ et $\bigcup_{(q,q')\in\cdots} L_{q,q'}$ (**Q7**) ;
- si u et v sont deux mots tels que $u.v \in K^*$, on peut avoir $u.v = \varepsilon$ (donc $u = v = \varepsilon$) et on n'a pas forcément u et v dans K^*.

À la question **Q3**, le jury attendait un raisonnement inductif dont les formules de la question précédente constituaient le socle. Dans les éléments de base qui construisent l'ensemble des langages réguliers, $\emptyset$ est trop souvent oublié et ε n'est pas nécessaire ($\varepsilon = \emptyset^*$).

À la question **Q4**, l'utilité du lemme de l'étoile est bien comprise mais la mise en pratique est souvent confuse. Soit on applique le lemme à partir d'un énoncé précis (et celui-ci ne fait pas référence à un automate), soit on introduit directement un automate ainsi qu'un mot adapté et on justifie par exemple l'existence d'une boucle dans un chemin reconnaissant le mot, ce qui permet d'aboutir à une contradiction. Dire qu'un automate « ne sait pas compter » n'est en aucun cas une preuve pour justifier qu'un langage tel que $\{a^nba^n, n \in \mathbb{N}\}$ n'est pas régulier.

Dans la construction de l'automate de la question **Q5**, les ε-transitions sont souvent données de manière trop floues : il faut indiquer quel est l'état de départ et d'arrivée. De plus, rappelons qu'un automate général n'a pas forcément qu'un seul état initial et qu'un seul état final. Il fallait par ailleurs, dans cette construction, utiliser plusieurs copies des automates K et L pour ne pas reconnaitre plus de mots que souhaité.

Enfin, on rappelle qu'une union quelconque de langages réguliers n'est pas forcément un langage régulier (sinon, tout langage serait régulier car $L = \bigcup_{m\in L}\{m\}$) (**Q10**).

Le reste de l'épreuve amenait à écrire de nombreux programmes. Dans l'ensemble, le jury a constaté une bonne maitrise des bases du langage OCaml (sauf parfois dans la manipulation des types créés comme les arbres binaires de recherche des questions **Q16** à **Q19** ou des arbres `veb` de la partie IV). On rencontre encore trop souvent des confusions avec Python sur les opérateurs (/ et non // pour la division entière, `2**p` qui ne permet pas de calculer 2^p en OCaml) ou sur les délimiteurs (indenter rend un code OCaml plus lisible mais ne permet absolument pas de délimiter des blocs, il faut utiliser par exemple des `begin/end`). Certains candidats ne semblent pas connaitre `List.length` ou `List.mem` (par ailleurs totalement inutiles dans cette épreuve) et perdent du temps à les reprogrammer. Enfin, le code de certaines fonctions est souvent alourdi par l'usage de variables ou de références inutiles.

On précise qu'à de très rares exceptions près, la plupart des programmes s'écrivaient en peu de lignes et sans fonction auxiliaire (sauf si celle-ci constituait le « moteur » principal de l'algorithme). La programmation de *plusieurs* fonctions intermédiaires pour répondre à une question donnée ou l'usage de conversion de types (tableau ↔ liste) devrait être un signe qui alerte le candidat sur une méthode maladroite ou un algorithme à la complexité dégradée. En tout état de cause, pour toute fonction secondaire, il faut indiquer quels sont ses paramètres d'entrée et de sortie et ce qu'elle est censée faire, en choisissant des noms représentatifs (et non produire des `aux1 i j k`, `aux2 l m n` sans explication).

Dans la mesure où l'épreuve portait sur l'étude de structures non classiques, il fallait produire des fonctions optimisées pour ces structures. Par exemple, pour insérer un élément dans un arbre de la partie III (**Q24**), il est maladroit de recalculer tous les nœuds internes après l'insertion de l'élément sur la feuille appropriée. Il suffisait de recalculer l'étiquette de ses ascendants et ce jusqu'à tomber sur un nœud déjà étiqueté par `true`. Ce genre de réponses fonctionnelles, mais pas assez réfléchies (**Q12**, **Q13**, **Q15**, **Q25**, **Q26**, **Q37**), a

été sanctionné tout comme les réponses amenant à une complexité ne correspondant pas à celle demandée par l'énoncé (**Q23**, **Q24**).

Le jury a été un peu déçu de constater que sur un tableau d'entiers trié dans l'ordre croissant, seuls 15 % des candidats proposent de faire une dichotomie pour tester la présence d'un élément dans le tableau (**Q12**). Dans le même ordre d'idée, la dichotomie de la question **Q13** est très souvent mal faite alors qu'il s'agit d'une technique de programmation classique : on écrit par exemple une fonction auxiliaire récursive qui prend en entrée deux indices (nommons les *debut* et fin) délimitant la zone du tableau sur laquelle on veut travailler. On rappelle que le « milieu » de la zone se calcule par $(debut + fin)/2$ et non $(fin - debut)/2$!

Toute hypothèse simplificatrice introduite par le candidat s'expose à une perte de points. On peut citer par exemple « je considère que `N` ou `p` est une constante globale » (on doit être capable de récupérer ces valeurs dans les paramètres d'entrée) ou « je suppose que les ensembles sont disjoints » (le programme doit tenir compte d'éventuels doublons). Dans la partie III, l'énoncé précisait bien que la valeur 2^p pouvait être obtenue via la taille du tableau : celui-ci étant de taille 2^{p+1}, 2^p s'obtenait systématiquement par `(Array.length e)/2`. On rappelle à ce propos que l'instruction `Array.length` s'exécute en temps constant (alors que `List.length` est en temps linéaire par rapport à la taille de la liste). Il n'y avait donc pas besoin de programmer de fonction puissance (ni de fonction logarithme...).

Dans les questions théoriques de la partie III (**Q27**, **Q29**), pour justifier une propriété associée à un algorithme donné sous la forme d'une boucle, le plus simple est de produire un invariant de boucle. Trop de réponses se sont contentées de paraphraser l'énoncé. Par exemple, pour la question **Q27**, à tout moment, le sous-arbre de racine i code une partie de l'ensemble dont x est le plus grand élément.

Enfin, toute personne ayant pris soin de lire attentivement le préambule, avait au moins la moitié de la réponse pour la question **Q32**.

Dans la partie IV, abordée seulement dans les meilleures copies, la fonction **Q34**, simple sur le principe, a donné lieu à beaucoup d'erreurs : pour créer un arbre `veb` associé à p, il fallait créer un tableau de taille $2^{2^{p-1}} + 1$ d'arbres `veb` associés à $p - 1$ en veillant à ce que ces arbres soient « physiquement » différents. Il fallait donc obtenir la puissance adaptée (soit par calcul avec une fonction auxiliaire, soit en se servant de la récursivité, soit en pré-calculant au départ toutes les valeurs nécessaires) et utiliser, par exemple, la fonction `Array.init` au lieu de `Array.make`.

Conclusion

L'option informatique n'est pas qu'une agglomération d'études de cas. On attend des étudiants, comme dans toutes les autres disciplines, la mise en œuvre et la restitution précise d'un certain nombre de résultats vus en cours, en particulier sur les structures standards ou la théorie des langages (qu'on sait être plus difficile). Le jury ne peut qu'encourager les futurs candidats à bien maitriser leur cours sur ces chapitres.

De façon générale, afin de préparer au mieux cette épreuve, il convient de s'imposer un entrainement régulier sur machine, seul moyen d'acquérir de bons réflexes en programmation et de s'habituer à rédiger en détail des questions théoriques afin de gagner en aisance dans les argumentations.