6. Option informatique

6.1. Introduction

Le sujet aborde différents problèmes liés à la recherche d'éléments minimaux pour une relation d'ordre partielle, en particulier le problème concret de l'optimisation d'un itinéraire dans un réseau ferroviaire en fonction de plusieurs critères simultanément (temps, nombre de correspondances, prix). Il n'existe pas forcément de parcours qui minimise les trois critères à la fois, mais on peut déterminer ceux qui sont minimaux pour la relation d'ordre produit sur les triplets de paramètres.

Il est constitué de quatre parties :

- une première qui étudie une structure de tas afin de modéliser une file de priorité ;
- une deuxième qui s'intéresse de manière générale aux relations d'ordre (élément minimal d'une relation d'ordre partielle, ordre lexicographique et ordre produit, notion d'optimum de Pareto) et étudie des méthodes pour calculer les éléments minimaux d'un ensemble ordonné dans différentes situations ;
- une troisième, conséquente, qui considère l'ensemble des éléments minimaux d'un langage pour une relation d'ordre donnée ;
- une dernière qui résout, en s'appuyant sur les résultats des parties I et II et à l'aide de l'algorithme de Dijkstra, le problème d'optimisation multi-critères dans le graphe formé par le réseau ferroviaire.

L'épreuve aborde donc les chapitres arbres, graphes et théories des langages en rapport avec le programme du tronc commun ainsi que celui de l'option informatique.

6.2. Analyse globale des résultats

Le sujet est de longueur raisonnable avec de nombreuses questions abordables par la grande majorité des candidats et chaque partie est assez progressive. Les meilleures copies vont au bout de l'épreuve en faisant quelques impasses. Pour avoir une bonne note, il est important de s'intéresser aux quatre parties.

Il y a une proportion plus importante de questions théoriques ou d'études d'exemples « à la main » que de questions de programmation pure. La partie programmation, plus facile, est globalement traitée plutôt convenablement sur la plupart des copies. Par contre, les réponses apportées aux questions théoriques, en particulier celles des parties II et III, sont souvent trop confuses, mal structurées, voire illogiques.

Comme à chaque fois, beaucoup de temps ou de points sont perdus par une lecture trop superficielle de l'énoncé. Certains candidats justifient des réponses alors que l'énoncé indique explicitement de ne pas le faire ou oublient de donner des complexités élémentaires alors que la question les demande.

Le jury rappelle qu'il apporte une grande attention à la présentation : une copie aérée (en particulier dans les codes de programme) est bien plus facile à lire. À l'inverse, les écritures illisibles sont pénalisées. On observe aussi cette année une recrudescence d'erreurs dans la numérotation des questions, toujours gênantes pour les correcteurs.

6.3. Commentaires sur les réponses apportées et conseils aux candidats

Signalons en préambule que, même si le sujet rappelle quelques fonctions du module `List`, les candidats ne sont pas obligés de s'en servir ou de se limiter à celles-ci. Trop de candidats réécrivent (parfois mal) des fonctions usuelles comme `List.rev` alors qu'elles devraient être connues. Inversement, l'usage de fonctions à la syntaxe plus complexe comme `List.fold_left` se fait à leurs risques et périls. Par ailleurs, la syntaxe des arbres en OCaml, pourtant au programme, n'est pas toujours bien maîtrisée.

La question **Q1** perturbe quelques candidats qui, bien que le sujet décrive en préambule la structure de tas d'appariement (qui est une structure de tas-minimum) et l'usage qu'il va en faire, interprètent la première question comme si elle porte sur un tas quelconque (donc éventuellement sur un tas-maximum, ce qui rend la question un peu surprenante voire artificielle dans le contexte, surtout pour une première question). On peut éventuellement avoir un doute, mais dans ce cas, une lecture rapide des questions suivantes devrait le lever. Prendre le temps de bien lire le sujet partie par partie, n'est jamais du temps perdu et permet d'éviter ce genre de méprise. Malgré tout, le jury pénalise peu cette maladresse, dans une question par ailleurs peu valorisée.

La plupart des erreurs dans la première partie sont liées à des questions d'ordre. Les candidats oublient que lors de la fusion, un tas est ajouté en tête (donc à gauche) de la liste des tas fils de l'autre tas (**Q4**). À la question **Q5**, suivant qu'on programme les différentes étapes avec des accumulateurs ou pas, il faut éventuellement retourner la liste des arbres à fusionner entre les deux étapes. À la question **Q8**, suivant l'ordre dans lequel on insère les éléments de la liste à trier dans un tas initialement vide pour ensuite extraire les éléments minimum un par un, la liste produisant un tas filiforme (qui correspond facilement à un meilleur cas en terme de complexité) est une liste triée dans l'ordre croissant ou décroissant.

On rappelle que l'opérateur @ n'est pas de complexité constante et que son utilisation vient souvent ruiner tous les bénéfices qu'aurait pu apporter une structure de données élaborée. Par exemple, dans la question **Q7**, accumuler les éléments retirés un par un du tas par une commande du type `acc@[x]` produit certes une liste triée dans l'ordre croissant, mais donne une complexité en $\Theta(n^2)$. Il faut à la place accumuler par `x::acc`, ce qui produit une liste triée dans l'ordre décroissant, et retourner cette liste à la toute fin du programme.

La deuxième partie, plus mathématique, fait l'objet de nombreuses confusions voire incohérences, à commencer par de nombreuses imprécisions entre inégalités strictes et larges. De façon générale, les candidats ont souvent de bons arguments mais les livrent dans le désordre et ne les inscrivent pas dans le cadre d'une démonstration bien structurée et clairement annoncée (une récurrence sur la longueur d'une liste, un raisonnement par l'absurde, etc). Par exemple, à la question **Q10**, une démonstration ne peut pas commencer par « Soit $x \neq y$ » et terminer par « donc $x = y$ » : il faut dire qu'on fait un raisonnement par l'absurde et constater à la fin l'absurdité. Par ailleurs, dans cette question, de nombreux candidats s'appuient pour leur démonstration sur des éléments qui ne sont pas dans la partie X considérée. De même, à la question **Q18**, la propriété qui va être démontrée sur la fonction `w` n'est pas clairement énoncée. Cette question, par ailleurs plus compliquée qu'il n'y parait, voit trop souvent les candidats paraphraser le code de `w`. Il faut expliquer clairement pourquoi les éléments minimaux de la liste `y::z` renvoyés par `w (y::z)` sont aussi des éléments minimaux de `x::y::z`.

À la question **Q14**, on voit des confusions entre la notion théorique d'ensemble et l'éventuelle représentation informatique qu'on pourrait en donner (sous forme de liste par exemple). La phrase, lue trop souvent, « soit X un ensemble à n éléments ayant n fois le même élément » est incohérente.

La question **Q20** est souvent mal traitée car les candidats ont choisi d'extraire de la liste les éléments plus petits que u au lieu de supprimer ceux plus grands ce qui, dans une relation d'ordre partielle, n'est pas la même chose. Enfin, la question **Q21** demande de combiner dans le bon ordre les fonctions précédentes avec un appel récursif, afin d'assurer la complexité demandée. Plus subtile, elle n'est que très rarement bien réussie.

La partie III voit les erreurs classiques sur la théorie des langages :

- un langage n'est pas forcément un ensemble fini de mots (**Q22**) ;
- la façon la plus naturelle de prouver qu'un langage est régulier est d'en donner une expression régulière (**Q23**) ;
- le lemme de l'étoile ne permet pas de montrer qu'un langage est régulier. Il donne juste une propriété des langages réguliers et il s'agit d'une propriété sur des mots. Sa démonstration fait référence à un automate mais pas son énoncé qui dit juste « il existe un entier N tel que pour tout mot $w \in L$ de longueur supérieure à N, … » . On peut répondre à la question **Q25(b)** en raisonnant directement avec un automate ou utiliser le lemme mais trop de candidats mélangent les deux, produisant une réponse confuse. De plus le facteur itérant identifié dans w est souvent mal choisi (on ne peut pas le choisir directement, par contre le lemme de l'étoile dit, sous couvert d'hypothèses, où l'on peut en trouver un dans le mot w) ;
- l'algorithme d'élimination des états, nouveauté du dernier changement de programme, semble inconnu de la plupart des étudiants (**Q27**) ;
- quand on utilise une propriété de stabilité de l'ensemble des langages réguliers, il faut le dire (**Q30**) ;
- dans la mesure où l'énoncé donne dans les parties III.3 et III.4 les constructions des automates, on attend aux questions **Q29**, **Q32** et **Q33** des démonstrations rigoureuses et formalisées à l'aide de chemins explicites et non des arguments en ordre plus ou moins dispersés s'appuyant sur les exemples traités dans les questions qui précédent.

Dans la partie IV, les questions **Q35** à **Q39**, relativement abordables, sont généralement bien traitées par les candidats qui s'y intéressent. Les questions **Q40** et **Q42** ne sont quasiment jamais abordées.

6.4. Conclusion

Dans l'ensemble, le niveau constaté sur cette épreuve tant sur le plan de la compréhension du sujet que celui de la programmation effective se révèle satisfaisant, bien que subsiste la difficulté que constitue la rédaction de réponses claires en temps limité. De nombreuses parties du sujet s'appuient sur des structures et résultats classiques (tas, lemme de l'étoile, élimination) et le jury ne peut qu'encourager les futurs candidats à bien maîtriser leur cours d'informatique. De façon générale, afin de préparer au mieux cette épreuve, il convient de s'imposer un entraînement régulier sur machine, seul moyen d'acquérir de bons réflexes en programmation et de s'habituer à rédiger en détail des questions théoriques (en particulier celles sur les automates) afin de gagner en aisance dans les argumentations.