Composition d'Informatique – A – (XULCR)

filière MP spécialité Info

1 L'épreuve

Il s'agissait dans cette épreuve d'explorer quelques algorithmes pour identifier et/ou dénombrer des sous-mots sur un alphabet fini. La première partie étudiait plus particulièrement la notion de sous-mot d'un mot donné. La première question proposait de démontrer une propriété récursive sur les sous-mots et de l'utiliser pour écrire un programme récursif polynomial pour déterminer si un mot est sous-mot d'un autre. Dans la suite, la première partie explorait le dénombrement des sous-mots d'un mot donné. Pour calculer le nombre de plongements d'un mot dans un autre, un programme récursif de complexité exponentielle était donné dans l'énoncé, les candidats devant en faire l'analyse. Il était ensuite demandé d'utiliser les techniques de programmation dynamique pour en tirer un programme polynomial. La même approche était ensuite suivie pour calculer le cardinal de l'ensemble des sous-mots d'un mot, puis pour calculer le plus petit sur-mot commun à deux mots donnés.

La deuxième partie étudiait ensuite les sous-mots des mots d'un langage rationnel donné par une expression régulière. Une première approche, décomposée en plusieurs questions, consistait à calculer l'expression régulière décrivant le résidu d'un langage rationnel par un mot d'une seule lettre, puis à l'utiliser récursivement pour déterminer si un mot est sous-mot d'un mot d'un langage rationnel. Une analyse de complexité était demandée. Une deuxième approche, plus efficace, était ensuite proposée : il s'agissait cette fois de calculer les facteurs d'un mot couverts par un langage rationnel, récursivement sur l'expression régulière décrivant ce langage, puis d'utiliser ce calcul pour déterminer là encore si le mot est sous-mot d'un mot du langage rationnel.

Les notes des 1112 copies se répartissent selon le tableau suivant, avec une moyenne de 9,55 et un écart type de 3,3. Pour obtenir la note maximale, il n'était pas nécessaire de traiter l'intégralité du sujet.

$0 \leq N < 4$	32	2,9 %
$4 \leq N < 8$	331	29,8 %
$8 \leq N < 12$	510	45,9 %
$12 \leq N < 16$	201	18,1 %
$16 \leq N \leq 20$	38	3,4 %
Nombre de copies	1112	100 %
Note moyenne	9,55	
Écart-type	3,3	

2 Remarques générales

Preuves de correction. Trop de copies se contentent de justifier la correction d'un algorithme en le paraphrasant en français ou en indiquant qu'« on voit bien que ». Les correcteurs attendent des preuves rigoureuses, avec une hypothèse de récurrence (un invariant de boucle,

dit autrement) correctement identifiée et formulée. Bien entendu, un raisonnement approximatif à base de « de proche en proche » ou de points de suspension n'est pas considéré comme une preuve par récurrence rigoureuse.

Programmation. Concernant les questions de programmation, les candidat(e)s doivent être conscients du fait qu'une réponse longue doit être expliquée en détail et que très souvent un programme très long contient un grand nombre d'erreurs. Découper un programme un peu long en plusieurs fonctions est une bonne initiative, mais appeler ces différentes fonctions `aux`, `aux2`, `aux3`, etc., ne l'est pas. Certaines copies utilisent une couleur différente pour écrire leurs programmes et cette initiative est très appréciée des correcteurs. Il en va de même pour le soin apporté à l'indentation du code, qui en facilite grandement la compréhension.

L'énoncé demandait de justifier les complexités en temps des algorithmes, mais beaucoup de candidat(e)s oublient de le faire. Affirmer que la complexité est en $O(n)$ sans en expliquer les raisons n'est pas une justification.

Une majorité de programmes contient des problèmes de bord : tableaux/matrices trop petits d'une unité, accès en dehors des bornes d'une unité (à gauche ou à droite), boucle/fonction qui s'arrête un cran trop tôt, code qui suppose une taille strictement positive alors qu'il devrait fonctionner également sur le mot vide, etc. Un exemple typique est la création d'un tableau de taille n ensuite parcouru avec un indice allant de 0 à n inclus.

3 Commentaire détaillé

Pour chaque question, sont indiqués entre crochets le pourcentage de candidat(e)s ayant traité la question et le pourcentage de candidat(e)s ayant obtenu la totalité des points. Très peu de candidats sont parvenus à la dernière question ; aucun n'a su traiter toutes les questions correctement. En particulier, les questions 5 et 8 ont été très mal traitées, quand elles n'ont pas été tout simplement ignorées par les candidat(e)s, et le sujet a été peu traité au delà de la question 11.

Question 1 [100% - 32%]. Une fois n'est pas coutume, c'est là une première question assez longue. La première partie (1a) est rarement rédigée avec soin, les candidats écrivant souvent « $a = a'$ » au lieu de « le plongement envoie le dernier caractère de ua sur le dernier caractère de $u'a'$ » (il pourrait y avoir plusieurs occurrences de a et de a'). La seconde partie (1b) est plutôt bien traitée, les candidats donnant un code de complexité exponentielle étant peu nombreux.

Les correcteurs ont constaté avec étonnement que relativement peu de réponses à la question 1b suivaient précisément la structure donnée par la question 1a, une majorité des candidats préférant parcourir les mots de gauche à droite et pour beaucoup de manière itérative. Toutes ces variantes correctes ont été acceptées.

Partie I

Question 2 [100% - 45%]. Une question plutôt bien traitée. Mais beaucoup de candidats ne justifient pas le caractère disjoint derrière la somme de la question 2c.

Question 3 [98% - 20%]. Les preuves de terminaison (3a) sont souvent peu rigoureuses. Pour la question 3b, très peu de candidats explicitent clairement ce que calcule la fonction `aux`. Il en résulte souvent des preuves par récurrence très approximatives.

Il faut se méfier des preuves qui se limitent à « trivial », à part si ce terme est accompagné d'un minimum de justification. Le terme « trivial » seul est perçu par les correcteurs comme une tentative de bluff de la part du candidat.

Question 4 [88% - 3%]. Pour la question 2a, l'argument derrière la majoration en $2^{|u|}$ est bien compris (*i.e.*, au plus deux appels à chaque fois) mais la preuve en est rarement rigoureuse, l'immense majorité des candidats oubliant de compter effectivement l'appel à `aux` lui-même dans leur formule récursive de T (*i.e.*, il manque le +1).

Pour la question 4b, beaucoup de candidats montrent que $T(u, v)$ ne peut pas être polynomial en la taille de u et v, ce qui ne répond absolument pas à la question.

Question 5 [60% - 14%]. Une question plutôt bien traitée, quand elle n'est pas ignorée par les candidats qui n'aiment pas la programmation. La remarque générale concernant les problèmes de bord s'applique en particulier à cette question.

Question 6 [93% - 31%]. Une question plutôt bien traitée. Pour prouver la réciproque de la question 6b, certains candidats oublient que w peut contenir aa.

Question 7 [76% - 7%]. Beaucoup de candidats oublient le cas de base dans leur formulation récursive de $Card(u)$.

Question 8 [75% - 2%]. Les questions 8b et 8c ont été très mal traitées dans l'ensemble. En particulier, il fallait programmer une fonction de comparaison des chaînes de caractères correspondant à l'énoncé (*i.e.*, comparaison des longueurs d'abord, puis ordre lexicographique à longueur égale), ce que très peu de candidats ont fait.

Partie II

Question 9 [92% - 26%]. Beaucoup de candidats échouent à identifier que le langage $L(e_1)$ est vide. Cette question était là pour aider à sensibiliser les candidats à la présence de l'expression `Empty` au sein des expressions régulières, en vue des questions suivantes.

Question 10 [82% - 2%]. Une moitié des candidats prend soin de tester l'appartenance du mot vide au langage de e_1 dans le cas d'un produit $e_1 \cdot e_2$. En revanche, très peu de candidats se soucient du fait que $L(e_2)$ puisse être vide.

Question 11 [81% - 6%]. Cette question a été très mal traitée par les candidats. À leur décharge, elle était assez trompeuse car c'est souvent le cas particulier du langage vide qui mettait l'égalité en défaut.

Question 12 [39% - 0%]. Comme pour la question 10, le cas de sous-expressions dont le langage est vide est en général ignoré. La taille quadratique de l'expression renvoyée est rarement identifiée.

Question 13 [19% - 0%]. Les mêmes commentaires que pour la question 12 s'appliquent.

Question 14 [14% - 1%]. Les rares candidats qui abordent cette question ont en général la bonne intuition (*i.e.*, appliquer successivement la fonction `char_residu_ratexp`) mais les détails sont souvent incorrects (ordre d'application, test de vacuité à la fin).

Question 15 [9% - 0%]. Les rares candidats qui abordent cette question la traitent plutôt bien.

Question 16 [7% - 6%]. Une question très facile, traitée par les tous meilleurs candidats et quelques opportunistes.