Composition d'Informatique A – (XULCR)

Filière MP spécialité Info

1 L'épreuve

Il s'agit dans cette épreuve d'étudier les facteurs des mots finis sur l'alphabet $\{0, 1\}$, c'est-à-dire l'ensemble des sous-mots d'un mot binaire. Dans une première partie, une structure de données arborescente, les arbres de mots, permettant une représentation succincte d'un ensemble de facteurs, est introduite. On demande dans cette partie d'expliciter cette structure sur quelques exemples, de prouver l'adéquation entre cette structure et l'ensemble représenté, et de programmer quelques fonctions simples : ajout d'un élément, calcul de la taille de l'ensemble représenté, énumération des éléments de cet ensemble, etc. Dans la deuxième partie, on restreint l'étude au cas particulier de l'ensemble des suffixes d'un mot. Il y est notamment demandé aux candidates et aux candidats de démontrer quelques bornes simples sur la taille d'un arbre des suffixes. Dans une troisième partie, cette structure de données est optimisée, en condensant encore un peu plus l'information représentée : ce sont les arbres de facteurs. Comme en première partie, il y est demandé de programmer des fonctions simples de manipulation de cette structure. En quatrième partie, cette structure est appliquée au cas de la recherche du plus long facteur répété d'un mot. Il y est demandé de programmer la fonction naïve, cubique, de recherche de ce plus long facteur répété, puis la version "programmation dynamique" de cette fonction, en temps quadratique, et enfin d'exploiter la structure de données pour programmer la fonction en temps linéaire. Enfin, la dernière partie s'intéresse au cas des mots contenant un maximum de facteurs répétés distincts. Cette dernière partie, plus théorique, permet de calculer quelques bornes sur ces mots, puis d'en démontrer l'existence et d'en donner la forme en utilisant des propriétés des graphes orientés.

La moyenne des 739 candidats français est de 9,77/20 avec un écart-type de 3,88.

La moyenne des 55 candidats étrangers est de 6,33/20 avec un écart-type de 2,80.

2 Remarques générales

Clarté des copies et des démonstrations. Trop de copies s'avèrent difficiles à lire pour les correcteurs. Les candidats qui produisent des réponses raturées, mal indentées ou encore non

structurées en paragraphe perdent nécessairement des points malgré tous les efforts de décodage des correcteurs. Les centres d'examen fournissent des feuilles de brouillon aux candidats.

On rappelle que pour être valide une preuve de programme doit suivre avec précision la structure du programme. Les preuves par induction, et notamment les preuves par récurrence, ne sont pas maîtrisées par un grand nombre de candidats. Outre l'oubli ou le mauvais choix du cas de base, le choix de la quantité ou la structure sur laquelle on effectue l'induction est souvent hasardeux. Pour les questions sur la correction d'un algorithme, on attend autre chose qu'une simple paraphrase de cet algorithme.

Erreurs dans les programmes. On rappelle qu'on parcourt les éléments d'un tableau de taille `n` avec une boucle `for` en écrivant `for i = 0 to n - 1 do` et non pas `for i = 0 to n do`. Cette erreur est présente dans de trop nombreuses copies.

De trop nombreuses erreurs de sémantique rendent incorrects les programmes proposés par les candidats. Par exemple : `let x = 4 in match y with x -> ...` n'est pas équivalent à `if y = 4 then ...`. Les erreurs de typage mettent aussi à mal la correction des programmes : en OCaml, les booléens ne sont pas des entiers, les tableaux ne sont pas des listes, et en particulier dans ce sujet les mots ne sont pas des listes de caractères.

On souhaite aussi alerter les candidats sur l'influence de Python dans l'écriture de programmes OCaml. Les candidats ont tendance à oublier le caractère immuable des listes OCaml : écrire `x :: l` ne modifie pas la liste `l` mais représente la liste formée par la valeur de `x` suivie de la valeur de `l`. Enfin, la syntaxe d'OCaml est très différente de celle de Python : l'indentation n'est pas significative (et donc l'oubli de `begin`/`end` ou de parenthèses peut être fatal), les itérations de la forme `for i in range ...` n'existent pas, les affectations multiples non plus, etc.

Concision des programmes. Le sujet indiquait que toute fonction de la bibliothèque standard d'OCaml pouvait être utilisée et rappelait l'existence de fonctions telles que `Array.of_list` pour faire gagner du temps aux candidats : les copies qui n'ont pas utilisé ces fonctions n'ont pas été sanctionnées mais se sont sanctionnées elles-mêmes en perdant du temps à les réimplémenter. D'une manière générale, les programmes attendus sont souvent courts : un candidat qui écrit un long programme avec un haut niveau d'imbrications et de cas doit s'interroger sur l'existence d'une solution plus simple.

On regrettera aussi l'usage de constructions lourdes là où une autre construction syntaxique équivalente existe : `match x with x when x <> y =>` à la place de `if x <> y then` ou bien `match e with (y, z) =>` à la place de `let (y, z) = e in` ou encore `if cond then () else` à la place de `if not cond then`.

Lorsqu'une structure est définie par induction, telle que c'était le cas pour les arbres ici, de nombreuses copies font une distinction de cas *avant* chaque appel récursif au lieu de gérer tous les cas dans un filtrage, ce qui rend le code beaucoup plus long et beaucoup moins clair. En voici un exemple :

```
let compter arbre =
  if arbre = V then 0 else
  let rec aux = function
  | N(b,g,d) -> let x = if b then 1 else 0 in
                let y = if g = V then 0 else aux g in
                let z = if d = V then 0 else aux d in
                x + y + z
  in
  aux arbre;;
```

3 Commentaire détaillé

Pour chaque question, sont indiqués entre crochets le pourcentage de copies ayant traité la question et le pourcentage de copies ayant obtenu la totalité des points.

Partie I

Question 1 [100% - 82%]. Il s'agissait là d'une question très facile, surtout destinée à aider à la compréhension des définitions.

Question 2 [88% - 17%]. Une induction naturelle sur la structure des mots de l'ensemble donnait une réponse simple, mais elle a été rarement utilisée. Beaucoup de copies se compliquent la vie avec une récurrence sur le nombre de mots ou encore une récurrence structurelle sur un arbre. En particulier, l'existence d'un objet ne peut pas être démontrée par une induction structurelle sur ce même objet !

Question 3 [94% - 58%]. Beaucoup de copies proposent des algorithmes qui sont légèrement faux (erreurs de ± 1). Il est pourtant très facile de procéder à une vérification en simulant l'algorithme pour $n = 0$ et 1, ce qui ne prend que quelques secondes.

Question 4 [83% - 28%]. Un certain nombre de copies ne font pas attention à l'énoncé et renvoient un arbre contenant au plus k caractères 1 au lieu de exactement k. Il est clair que beaucoup de copies utilisent la fonction `sN` sans la regarder en détail, en pensant qu'elle prend un arbre et renvoie un arbre réduit.

Question 5 [98% - 88%]. Il s'agissait là d'une question facile.

Question 6 [94% - 36%]. Beaucoup d'approximations sur cette question : les copies mélangent tableaux et listes, ce qui a été systématiquement sanctionné. Beaucoup de copies font des erreurs d'indices et dépassent de 1 la fin du tableau contenant le mot.

Question 7 [73% - 9%]. Très peu de copies utilisent le caractère réduit de l'arbre pour justifier la complexité de cette fonction. En particulier, les copies ont tendance à oublier qu'il y a un coût inhérent à l'appel d'une fonction, même si cette fonction ne calcule rien d'intéressant (dans le cas d'un arbre non réduit).

Beaucoup de copies supposent à tort que la concaténation se fait en temps constant, de même que les conversions entre listes et tableaux. Beaucoup de copies n'utilisent pas correctement l'accumulateur suggéré dans l'énoncé, avec pour résultat une duplication de mots dans la liste renvoyée. Il est préférable d'écrire un code simple, correct, sans accumulateur et avec la mauvaise complexité qu'un algorithme très compliqué et faux.

Question 8 [77% - 32%]. Un certain nombre de copies oublient des cas importants, notamment l'insertion dans un arbre vide ! On retrouve tous les problèmes des questions précédentes liés aux indices et la distinction tableau/liste. À noter qu'il était tout à fait acceptable, même si un peu inefficace, de convertir le mot en liste en début de fonction pour ensuite travailler sur une liste (mais attention alors à l'ordre des lettres).

Partie II

Question 9 [99% - 79%]. Il s'agissait là d'une question très facile, surtout destinée à aider à la compréhension des définitions. Le mot vide (la racine `T`) a souvent été oublié.

Question 10 [97% - 8%]. Un nombre très impressionnant de copies affirment que la hauteur de l'arbre est $\ell(m)$ (au lieu de $\ell(m)+1$). Et ce n'est même pas forcément un problème de définition de la hauteur puisque ces copies définissent elles-mêmes une fonction hauteur avec les bonnes propriétés. Par ailleurs, le cas `N(true,V,V)` est souvent traité incorrectement.

Question 11 [67% - 3%]. La question comporte plusieurs parties clairement identifiées. Pourtant, un grand nombre de copies oublient complètement de répondre à certaines d'entre elles. Indiquer "meilleur cas" et "pire cas" ne suffit pas comme justification ; il faut dire en quoi ce sont les meilleurs et pires cas.

Partie III

Question 12 [86% - 23%]. Comme à la question 11, un grand nombre de copies oublient de répondre à certaines des sous-questions, par exemple l'ensemble de mots ou le caractère réduit d'un arbre.

Question 13 [44% - 3%]. Cette question a été très mal traitée dans l'ensemble. Des copies proposent une preuve qui n'utilise jamais le fait qu'il y a au moins un 0 et 1 alors que l'énoncé donne un contre-exemple si ce n'est pas le cas. La deuxième partie était assez subtile et a été peu traitée.

Question 14 [54% - 18%]. Un nombre important de copies se contentent de recopier le code de la question 5 à l'identique. Il est pourtant improbable qu'un sujet de concours demande deux fois la même chose !

Question 15 [56% - 16%]. Beaucoup d'erreurs de bornes. Un nombre incroyable de copies écrivent ainsi un test $p+k \leq q$ avant d'accéder à $m[p+k]$ (au lieu d'un test $p+k<q$).

Question 16 [33% - 5%]. Cette question était assez délicate car il était facile d'oublier de traiter des cas, par exemple lorsque le mot est plus court que le facteur stocké dans le nœud de l'arbre.

Question 17 [12% - 2%]. Cette question difficile et traitée par peu de copies a donné lieu à des codes inutilement longs et compliqués. Beaucoup de copies se sont emmêlées les pinceaux entre l'utilisation d'un accumulateur et la concaténation des listes renvoyées.

Question 18 [41% - 7%]. Il s'agissait d'une question de compréhension mais qui demandait une certaine concentration pour ne pas se tromper dans les indices. Elle a rarement été traitée jusqu'au bout.

Question 19 [**27% - 6%**]. Bien que le sujet demandait explicitement aux candidats de ne pas recopier le code de la fonction afin de gagner du temps, nombreux sont ceux qui l'ont fait. Les réponses proposées ont souvent ignoré la valeur de la lettre courante, utilisant ainsi systématiquement $p + k + 1$ pour le sous-arbre gauche et $i + k + 1$ pour le sous-arbre droit.

Partie IV

Question 20 [**38% - 11%**]. On pouvait avantageusement se resservir de la fonction `plpc` mais très peu de copies l'ont fait. Beaucoup d'erreurs d'indices ont été commises. Elles sont pourtant faciles à éviter en regardant ce que fait l'algorithme pour les valeurs extrêmes des bornes. Une façon simple de s'assurer que la complexité est cubique est de ne pas imbriquer plus de trois boucles. Certaines copies sont allées jusqu'à imbriquer cinq boucles. Il fallait faire attention au cas de la diagonale, sous peine de se retrouver avec le mot lui-même comme plus long facteur répété.

Question 21 [**21% - 2%**]. Comme pour la question 20, de nombreuses erreurs d'indices ont été commises. En particulier, $c(i, j)$ concerne les mots qui finissent aux indices $i - 1$ et $j - 1$, et non pas i et j. Par ailleurs, il fallait là encore faire attention à ignorer le cas diagonal.

Question 22 [**6% - 1%**]. L'algorithme était un peu subtil mais le code extrêmement simple. Comme pour la question 10, il fallait faire attention à bien gérer le cas des sous-arbres vides.

Partie V

Question 23 [**40% - 29%**]. La réponse étant dans la question, le but n'était pas de la recopier mais de la justifier, ce qui pouvait se faire en quelques mots.

Question 24 [**27% - 2%**]. Peu de copies ont justifié (ou même essayé de justifier) que le nombre maximum de facteurs distincts était atteint.

Question 25 [**24% - 11%**]. La question était simple mais nombre de réponses étaient approximatives. En particulier, il fallait soit s'assurer que le cycle créé ne contient pas de sommets en double, soit décrémenter les degrés en conséquence.

Question 26 [**13% - 1%**]. Si la question était vraiment "triviale" ou "évidente", elle n'aurait pas été posée. Par ailleurs, retirer un cycle du graphe peut créer plusieurs composantes disjointes, ce qui a trop rarement été mentionné.

Question 27 [**6% - 1%**]. Cette question était assez simple mais n'a été que peu abordée.