ÉCOLE POLYTECHNIQUE CONCOURS D'ADMISSION 2023
ENS : ULM, LYON, PARIS-SACLAY, RENNES

Composition d'Informatique A (XULSR)

Filières MP spécialité Info et MPI

1 L'épreuve

Ce sujet porte sur la compression d'un texte en fonction de la fréquence des lettres qui le composent. La première partie s'intéresse au comptage des occurrences des lettres dans un texte. La deuxième partie s'intéresse à la représentation d'un alphabet comme feuilles d'un arbre binaire. La troisième partie montre qu'un tel arbre décrit un code préfixe et peut donc être utilisé pour encoder et décoder un texte. La quatrième partie s'intéresse à la construction d'un arbre optimal du point de vue de la taille du texte encodé ; l'approche suivie est similaire à celle de l'algorithme de Huffman, si ce n'est qu'elle ne nécessite pas d'utiliser des files de priorité (ce serait même contre-productif). Les parties cinq et six s'intéressent à des types d'arbre bien précis, les arbres canoniques et les arbres alphabétiques, la motivation étant de limiter le coût de stockage de l'arbre lui-même. La septième partie montre que les codes préfixes donnent parfois une compression assez éloignée de celle prévue par l'entropie de Shannon et s'intéresse donc à un autre type de compression : les codes arithmétiques de type *rANS*. Pour obtenir la note maximale, il n'était pas nécessaire de traiter l'intégralité du sujet.

La moyenne des 880 candidats français en MP est de 9,68/20 avec un écart-type de 3,60.
La moyenne des 228 candidats français en MPI est de 11,60/20 avec un écart-type de 3,43.

2 Remarques générales

Il est rappelé que la présentation de la copie est importante. Trop de candidats écrivent de façon illisible ou raturent leurs réponses, alors qu'ils ont accès à des feuilles de brouillon. Même si la propreté de la copie ne joue pas dans la note, une réponse ne peut rapporter des points que si le correcteur arrive à la déchiffrer. Par ailleurs, trop de candidats semblent se jeter dans l'écriture d'algorithmes complexes sans réflexion préalable. La plupart des codes attendus tiennent en moins de dix lignes, au pire une quinzaine de lignes (questions VI.1 et VII.2). Pourtant, les correcteurs ont régulièrement vu des algorithmes faisant plus d'une page !

Les listes et les tableaux sont deux structures de données très différentes. Les structures de données utilisées dans les énoncés des questions n'ont pas été choisies pour rendre les réponses artificiellement compliquées mais au contraire pour simplifier la vie des candidats. Malgré cela, trop de copies transforment des listes en tableaux ou vice-versa, ce qui conduit à des codes inutilement compliqués, voire faux. Par ailleurs, trop de candidats, probablement inspirés de Python, ne font pas la différence entre listes et tableaux et se permettent d'écrire `l[i]` avec `l` une liste, ou `x::t` avec `t` un tableau. De plus, ces opérations sont alors souvent considérées comme étant en temps constant, ce qui a évidemment été sanctionné.

L'ordre d'évaluation en OCaml n'est pas spécifié et rien ne garantit que les arguments d'une fonction seront évalués de gauche à droite ; le compilateur de bytecode OCaml les évalue d'ailleurs

de droite à gauche. Plusieurs questions (II.4, V.1, VI.2) peuvent être résolues élégamment en combinant parcours d'arbre et état mutable, mais cela nécessite alors d'être attentif à l'ordre d'évaluation des arguments. Ce genre de problème n'a pas été sanctionné. Mais vu le nombre élevé de copies concernées, il serait bon de sensibiliser les étudiants à cette problématique de l'ordre d'évaluation.

Enfin, concernant les analyses de complexité, le fait qu'une fonction effectue $O(n)$ appels récursifs n'implique en rien que la complexité temporelle totale soit aussi en $O(n)$. Encore faut-il pouvoir garantir que le nombre d'opérations élémentaires effectuées à chaque appel est borné. En particulier, la concaténation de liste n'est pas une opération élémentaire ; sa complexité est linéaire en la taille de la liste de gauche !

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, en fonction des filières.

Question I.1 [MP : 98%, 84% ; MPI : 100%, 84%] Il s'agit là d'une question très facile.

Question I.2 [MP : 97%, 51% ; MPI : 99%, 57%] Beaucoup de candidats ont écrit une fonction qui renvoie un tableau trié par ordre décroissant, contrairement à ce qui est demandé dans le sujet.

Question II.1 [MP : 90%, 76% ; MPI : 91%, 71%] Il n'est pas nécessaire de donner la formule générale du cardinal de $\mathcal{T}_{[1,n]}$ et de l'évaluer pour $n = 4$. Certaines copies ont d'ailleurs tenté de démontrer que la réponse pour un n quelconque était $n!$, ce qui est incorrect et ne fonctionne même pas pour $n = 2$. Comme un bon croquis vaut mieux qu'un long discours, dessiner 5 arbres binaires permet de répondre facilement à la question.

Question II.2 [MP : 94%, 49% ; MPI : 99%, 73%] Beaucoup de copies ont proposé un algorithme qui précalcule les profondeurs de chaque feuille, ce qui est correct mais inutilement compliqué.

Question II.3 [MP : 89%, 27% ; MPI : 98%, 45%] Beaucoup de candidats ont écrit une fonction qui renvoie un chemin partant d'une feuille plutôt que de la racine.

Question II.4 [MP : 55%, 13% ; MPI : 75%, 24%] Trouver une formule générale capable de donner l'étiquette de chaque feuille est un peu subtil ; il est bien plus simple d'incrémenter un compteur à chaque création de feuille.

Question III.1 [MP : 89%, 15% ; MPI : 91%, 18%] Beaucoup de candidats raisonnent par récurrence en indiquant simplement "et ainsi de suite" ou "on procède récursivement", sans même indiquer sur quoi procède la récurrence ; cela a été sanctionné. Par ailleurs, beaucoup trop de copies ont en réalité prouvé que l'algorithme de la question III.2 est déterministe, ce qui est vrai mais n'implique en rien l'unicité de la décomposition et ne répond donc pas à la question.

Question III.2 [MP : 82%, 12% ; MPI : 93%, 23%] Dans beaucoup de copies, le mot renvoyé est incorrectement à l'envers. Par ailleurs, il y a deux petites subtilités dans cette question. D'une part, la liste vide représentant le mot vide, elle doit être traitée sans erreur. D'autre part, la plupart des algorithmes de décompression proposés ne consomme aucun bit en entrée quand ils arrivent sur une feuille ; dès lors, affirmer que leur complexité totale est linéaire parce que la liste décroît de 1 à chaque appel récursif est incorrect.

Question IV.1 [MP : 92%, 45% ; MPI : 95%, 52%] Cette question ne présente pas de difficulté particulière et permet de se familiariser avec le type de raisonnement nécessaire pour les questions suivantes.

Question IV.2 [MP : 79%, 26% ; MPI : 80%, 31%] Plusieurs copies ont cherché à prouver la propriété par l'absurde en construisant un contre-exemple invalide, par exemple un arbre ayant trois nœuds attachés à la racine. D'autres copies ont quant à elles calculé un majorant (ou un minorant) de $c_q(t)$ et ont prétendu que, puisque cette borne est minimale pour $\ell_t(\sigma_0) = 1$, alors $c_q(t)$ l'est aussi, ce qui ne fait aucun sens.

Question IV.3 [MP : 59%, 10% ; MPI : 68%, 19%] Plusieurs copies ont commencé par prouver la deuxième propriété puis en ont trivialement déduit la première. C'est correct mais cela rend le raisonnement vraiment compliqué.

Question IV.4 [MP : 74%, 25% ; MPI : 88%, 35%] Un nombre trop important de copies proposent un algorithme d'une page entière pour la fonction `insert` alors qu'il s'agit d'un simple parcours de liste qui se fait en moins de 5 lignes de code.

Question IV.5 [MP : 41%, 9% ; MPI : 58%, 16%] Cette question très simple semble avoir été mal comprise par beaucoup de candidats.

Question IV.6 [MP : 20%, 3% ; MPI : 37%, 14%] Le code, non demandé, peut être assez subtil à écrire mais il s'explique facilement. En particulier, il ne nécessite absolument pas d'utiliser des files de priorité, contrairement à l'algorithme d'Huffman, puisque l'énoncé de la question précédente montre que les arbres sont produits dans le bon ordre et qu'ils peuvent donc être stockés directement dans un tableau, par exemple.

Question V.1 [MP : 33%, 3% ; MPI : 59%, 11%] Trop de copies ont proposé un code relativement long sans aucune explication. Si le fonctionnement de l'algorithme n'est pas évident, il est recommandé de donner l'intuition de l'algorithme en quelques lignes ou de faire des dessins des grandes étapes de l'algorithme. Il est à noter qu'il est possible d'écrire un code très court et simple en se souvenant du nombre de feuilles restant à insérer à chaque profondeur.

Question VI.1 [MP : 21%, 2% ; MPI : 40%, 3%] Beaucoup de copies donnent la bonne formule mais parcourent le tableau dynamique dans le mauvais sens. En effet, on ne peut pas remplir le tableau par i croissant puisque la valeur de $m_{i,j}$ dépend de celle de $m_{k,j}$ avec $k > i$. Une façon d'éviter cette erreur est par exemple de dessiner les dépendances de chaque case afin de s'assurer que l'ordre du parcours est le bon. Une autre façon est d'utiliser une approche de type mémoïsation, c'est-à-dire de définir m comme une fonction récursive qui remplit les cases du tableau à la volée au fur et à mesure de ses besoins, ce qui simplifie grandement le code au prix d'une petite perte d'efficacité.

Question VI.2 [MP : 10%, 3% ; MPI : 31%, 12%] Cette question est en fait un simple parcours en profondeur d'un arbre, si ce n'est que ce dernier est implicite.

Question VII.1 [MP : 47%, 13% ; MPI : 59%, 25%] Beaucoup de copies n'ont pas correctement dénombré les mots de $\mathcal{S}^q$. Par ailleurs, le sujet indique clairement que la réponse attendue ne dépasse pas 2^{17}, ce qui permet dans une certaine mesure de vérifier sa réponse. Enfin, le sujet propose de donner la réponse sous forme d'un produit de nombres premiers, ce qui évite d'avoir à faire le moindre calcul.

Question VII.2 [MP : 5%, 0% ; MPI : 16%, 2%] Cette question, peu abordée, ne présente pourtant pas de difficulté particulière puisqu'il s'agit juste d'inverser la formule mathématique donnée dans le sujet.

Question VII.3 [MP : 1%, 0% ; MPI : 2%, 0%] Puisque le sujet indique que les x_i ne doivent pas dépasser 2^B, cela suggère que l'invariant à préserver est que tous les x_i vivent dans l'intervalle $I = [2^{B-1}; 2^B - 1]$. L'algorithme de décompression n'a alors qu'à choisir k_i de telle sorte que $x_i = y_i 2^{k_i} + \ldots$ tombe dans I. La vraie difficulté de la question réside dans la preuve que, au moment de la compression, pour tout $x_i \in I$ et σ_i, il existe bien une valeur de k_i telle que x_{i+1} tombe dans I.