ÉCOLE POLYTECHNIQUE CONCOURS D'ADMISSION 2016
ENS : ULM, CACHAN, LYON, RENNES

Composition d'Informatique (4 heures), Filière MP
(XULCR)

1. L'épreuve

Il s'agissait dans ce sujet de résoudre le problème de la satisfiabilité des formules booléennes en forme normale conjonctive avec au plus k littéraux par clause (k-SAT). Le sujet s'articulait autour des différentes valeurs de k, des plus petites aux plus grandes, amenant de ce fait le candidat à découvrir la complexité algorithmique croissante du problème et ses effets de seuil. Dans sa forme, l'épreuve alternait des questions de programmation avec des questions relatives à la compréhension de la structure du problème et à sa modélisation.

La partie préliminaire présentait le problème k-SAT, proposant au passage deux petits exercices pour aider le candidat à se familiariser avec les objets mathématiques et algorithmiques introduits. La partie 1 se focalisait sur le cas $k = 1$, pour lequel une solution simple en temps linéaire en la taille n de la formule était mise en œuvre. Cette solution exploitait le fait que chaque clause de la formule impose un unique choix de valuation possible pour l'une des variables booléennes. La partie 2 se concentrait sur le cas $k = 2$, pour lequel une solution en temps linéaire existe également. Cette solution ramène le problème à la recherche de composantes fortement connexes dans un graphe orienté. La première sous-partie présentait l'algorithme de Kosaraju-Sharir pour résoudre ce nouveau problème. Cet algorithme effectue deux parcours en profondeur successifs, le premier sur le graphe original, le second sur le graphe transposé. La seconde sous-partie faisait le lien avec le problème original et complétait sa résolution. La partie 3 considérait le cas général où k est fixe mais arbitraire. Elle proposait un algorithme exponentiel pour sa résolution, à base de recherche exhaustive avec marche arrière et conditions d'arrêt prématuré. Enfin la partie 4 reliait k-SAT au problème général de la satisfiabilité des formules booléennes (SAT). L'objectif était d'analyser le début de la réduction de SAT à k-SAT établissant la NP-complétude de ce dernier. La fin de la réduction était décrite dans le commentaire final, qui donnait par ailleurs des éléments de contexte permettant une mise en perspective du contenu du sujet et de son articulation.

2. Remarques générales

Les notes se répartissent selon le tableau suivant, avec une moyenne de 9,73 et un écart-type de 3,52 (pour les candidats français de l'École polytechnique) :

$0 \leqslant \mathrm{N} < 4$	35	4,98 %
$4 \leqslant \mathrm{N} < 8$	175	24,89 %
$8 \leqslant \mathrm{N} < 12$	301	42,82 %
$12 \leqslant \mathrm{N} < 16$	162	23,04 %
$16 \leqslant \mathrm{N} \leqslant 20$	30	4,27 %
Total	703	100 %
Nombre de copies : 703		
Note moyenne : 9,73		
Écart-type : 3,52		

Concernant les questions de programmation, les candidats 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. Certains candidats 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.

Beaucoup de candidats n'ont pas utilisé les fonctions `do_list`, `map`, `it_list` et `list_it` indiquées au début de l'énoncé. Cela a souvent conduit à un code inutilement long et compliqué, les candidats réécrivant des fonctions récursives faisant exactement la même chose que les fonctions de bibliothèque ci-dessus. Dans certains cas, les candidats parcourent même les listes à l'aide de références et de boucles `while`, ce qui est encore plus lourd.

De nombreuses questions exigeaient des solutions de complexité linéaire. Des candidats proposent parfois des solutions dont il est évident qu'elles ne sont pas linéaires, car elles font usage de fonctions coûteuses comme un test d'appartenance ou encore une concaténation de listes. Ceci est bien évidemment sanctionné.

Si des variables sont numérotées de 0 à n inclus, alors il y en a $n+1$ au total et il faut donc allouer un tableau de taille $n+1$ pour les associer à des valeurs. Le nombre de copies qui allouent un tableau de taille n seulement est vraiment impressionnant.

3. Commentaire détaillé

Pour chaque question, sont indiqués entre crochets le pourcentage de candidats ayant traité la question et le pourcentage de candidats ayant obtenu la totalité des points. Pour obtenir la note maximale, il était nécessaire de traiter le problème en entier.

Préliminaires

Question 1 [100% - 88%]. Il s'agissait là d'une question très simple, de nature à se familiariser avec la notion de satisfiabilité.

Question 2 [99% - 46%]. Certains candidats ont confondu l'indice de variable maximal avec le nombre de clauses ou la taille maximale d'une clause. La clause vide a été parfois mal gérée.

Partie I

Question 3 [95% - 26%]. Beaucoup d'erreurs d'utilisation de `var_max`, qui donne l'indice de variable maximal et non le nombre de variables. Des confusions entre variables et littéraux (oubli des constructeurs `V` et `NV`).

Partie II

Question 4 [84% - 10%]. Une question fort mal traitée par les candidats. Dans de nombreuses copies, les arguments ont été ceux d'une complexité en $\mathcal{O}(V \times E)$ et non en $\mathcal{O}(V + E)$. Même lorsque les arguments principaux étaient là, ils étaient souvent très mal formulés.

Question 5 [97% - 57%]. Une question assez bien traitée dans l'ensemble, probablement vue en cours par de nombreux candidats. Certaines solutions de mauvaise complexité, cependant, lorsque les arêtes sont ajoutées en fin de liste plutôt qu'en début de liste.

Question 6 [85% - 16%]. Beaucoup de candidats ont ajouté à tort des composantes vides dans le résultat. Certains ont tenté de réécrire un parcours en profondeur, au lieu de réutiliser celui donné dans l'énoncé, ce qui conduisait à de nombreuses erreurs.

Question 7 [91% - 83%]. Aucune difficulté : il suffisait là de composer des fonctions données ou déjà écrites.

Question 8 [94% - 24%]. Beaucoup de candidats ont négligé de montrer proprement la réflexivité de la relation. Beaucoup ont oublié une des trois propriétés à vérifier.

Question 9 [77% - 1%]. Comme souligné dans quelques copies, le résultat de cette question était faux, la figure 2 fournissant un contre-exemple. Fort heureusement, cette question servait uniquement d'étape intermédiaire pour établir le résultat de la question 10 et n'avait donc pas de répercussions sur le reste du sujet. Les réponses soulevant le problème avec un contre-exemple clair (soit issu de la figure 2, soit inventé) ont été comptées comme justes, tout comme les réponses proposant une démonstration fausse mais plausible.

Question 10 [80% - 34%]. Question facile une fois le résultat (faux) de la question 9 établi. Toutefois, peu de copies ont pris le temps de justifier le résultat proprement, un certain nombre se contentant de paraphraser l'énoncé.

Question 11 [45% - 3%]. Question peu et mal traitée dans l'ensemble. Certaines copies n'ont pas vu qu'il fallait repartir de la question 10 et ont tenté de raisonner directement à partir de la question 9, menant à des preuves fausses et parfois contradictoires.

Question 12 [95% - 63%]. Il suffisait d'appliquer l'algorithme détaillé dans l'énoncé. Les erreurs provenaient souvent du non-respect du quatrième point indiquant d'éliminer la clause $(x_2 \vee \neg\, x_2)$.

Question 13 [87% - 16%]. Il s'agissait là de programmer un algorithme donné dans l'énoncé. Il fallait cependant être soigneux, pour ne pas oublier de cas, et créatif, pour factoriser le code autant que possible.

Question 14 [52% - 5%]. Question un peu délicate nécessitant de la rigueur pour être traitée proprement. L'équivalence des littéraux impliqués dans une même composante fortement connexe a été souvent négligée dans les copies, menant à des raisonnements bancals.

Question 15 [70% - 32%]. Une question facile, corollaire de la question précédente. Elle nécessitait cependant d'expliciter les arguments.

Question 16 [69% - 9%]. Beaucoup de candidats ont proposé une solution de mauvaise complexité en utilisant une fonction de recherche d'un élément dans une liste.

Partie III

Question 17 [88% - 73%]. Question très facile mais qui nécessitait un peu d'imagination pour être écrite de façon compacte et élégante. Une écriture laborieuse n'était pas sanctionnée, mais elle faisait perdre du temps au candidat.

Question 18 [80% - 74%]. Beaucoup de candidats n'ont pas réutilisé les trois fonctions écrites à la question précédente, ce qui aboutissait à un code plus long, plus compliqué et très souvent faux.

Question 19 [66% - 23%]. Question facile en principe, puisqu'il suffisait de transcrire l'algorithme de recherche exhaustive décrit en détail dans l'énoncé. En pratique, beaucoup de candidats n'ont pas suivi l'énoncé et ont structuré leur programme différemment, menant à des codes beaucoup plus longs et complexes que prévu. Le nombre d'erreurs d'indices ou de logique est impressionnant. On note toutefois un réel effort fait par les candidats pour commenter leur code.

Partie IV

Question 20 [62% - 34%]. Cette question demandait d'écrire des FNC équivalentes à deux formules proposées. Aucune méthode ou justification n'était demandée ; seul le résultat comptait. Il n'y avait pas de difficulté particulière, les propriétés des connecteurs logiques, en particulier la distributivité, ayant été rappelées dans l'introduction du sujet.

Question 21 [28% - 4%]. Certains candidats ont tenté de montrer l'équivalence (qui est fausse) au lieu de l'équisatisfiabilité comme demandé.

Question 22 [24% - 7%]. Programme assez simple, par traitement par cas. Il fallait bien évidemment ne pas oublier de cas. Beaucoup de candidats ont oublié par exemple le cas `Non` (`Non f`).

Question 23 [13% - 1%]. Question difficile. La manipulation des indices des variables fraîches demandait beaucoup de soin et a souvent été cause d'erreurs. Certains candidats ont écrit un programme renvoyant une `formule` au lieu d'une `fnc`.

Question 24 [8% - 1%]. Peu de candidats ont traité cette question et encore moins en ont fait la deuxième partie. La première partie demandait de justifier la complexité de la fonction de la question 22. Cette partie a été raisonnablement réussie. La deuxième partie demandait de justifier la complexité de la fonction de la question 23. Un indice était donné dans l'énoncé mais il n'a pas été bien utilisé, certains candidats ne comprenant pas qu'il fallait le prouver, d'autres qu'une fois prouvé, il fallait l'utiliser.