École Polytechnique
Écoles Normales Supérieures

Concours d'admission 2017

Composition d'Informatique (4 heures), filière MP

Rapport de Didier Cassereau, Jean-Christophe Filliâtre, Paulin de Naurois, Vincent Pilaud, correcteurs

1 L'épreuve

Il s'agissait dans ce sujet d'étudier et de mettre en œuvre différents algorithmes permettant de trouver des solutions optimales, en nombre de coups, à des jeux à un joueur tels que le *Rubik's Cube*, le solitaire, ou encore le taquin.

La partie 1 introduisait la notion de jeu à un joueur, en donnant notamment un exemple très simple de jeu utilisant uniquement l'arithmétique. Un premier algorithme était proposé, à savoir un parcours en largeur. Il fallait en faire la preuve de correction ainsi que traduire son pseudo-code en langage Caml. La partie 2 étudiait un autre algorithme, de recherche itérée en profondeur. Outre des questions analogues à celles de la partie 1 pour cet algorithme, l'idée était de saisir la différence de complexité, notamment spatiale, entre cet algorithme et le parcours en largeur. La partie 3 introduisait une variante de la recherche itérée en profondeur, où une fonction minorant le nombre de coups restants est donnée et permet ainsi de s'épargner de nombreuses étapes de la recherche itérée présentée dans la partie précédente. Là encore, il s'agissait de mettre en œuvre cet algorithme mais aussi de montrer sa correction. Enfin la partie 4 appliquait l'algorithme de la partie précédente au jeu du taquin. L'une des difficultés était ici de correctement réaliser le retour sur trace dans le contexte de la recherche itérée en profondeur.

2 Remarques générales

Les notes des 1026 candidats se répartissent selon le tableau suivant, avec une moyenne de 9,37 et un écart type de 3,83.

$0 \leq N < 4$	82	8,0 %
$4 \leq N < 8$	289	28,2 %
$8 \leq N < 12$	393	38,3 %
$12 \leq N < 16$	213	20,7 %
$16 \leq N \leq 20$	49	4,8 %
Total	1026	100 %

Complexité. Une proportion non négligeable de candidats donne des complexités à l'unité près, ce qui a peu de sens en informatique, quand on attend seulement des estimations asymptotiques. Des candidats semblent ignorer purement et simplement l'existence des notations asymptotiques, pourtant rappelées en début de sujet.

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 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 semblent ignorer que Caml ne dispose pas d'une instruction `return`. La gestion des valeurs de retour des programmes proposés est trop souvent directement calquée sur le pseudo-code fourni dans l'énoncé et incorrecte. De même, les instructions « **pour chaque** » du pseudo-code sont souvent traduites de manière fantaisiste en Caml. Beaucoup d'instructions `else` manquent dans les programmes des candidats.

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. Beaucoup de candidats sont parvenus jusqu'à la dernière question mais aucun n'a su traiter toutes les questions correctement. En particulier, les questions 3, 9 et 12 ont été très mal traitées, quand elles n'ont pas été tout simplement ignorées par les candidats.

Partie I

Question 1 [100% - 90%]. Il s'agissait là d'une question très simple, de nature à se familiariser avec la notion de solution optimale. Aucune preuve d'optimalité n'était demandée pour cette question. De nombreux candidats perdent du temps et de l'énergie à prouver l'optimalité de leur solution.

Question 2 [100% - 26%]. Beaucoup de candidats identifient correctement l'invariant que A contient exactement les états à profondeur p mais peu en font une démonstration rigoureuse.

Question 3 [87% - 4%]. Des candidats affirment que A contient exactement 2^p éléments, oubliant qu'il s'agit d'un ensemble et qu'il y a des doublons. Les autres candidats identifient la majoration en 2^p, même si la justification n'est pas toujours totalement

valide, mais très peu minorent correctement le cardinal de A. C'est ce minorant qui fait de cette question l'une des plus difficiles du sujet.

Question 4 [97% - 41%]. Le pseudo-code proposé en figure 1 renvoie une valeur booléenne. Le programme demandé dans cette question doit en revanche renvoyer une valeur entière. Beaucoup de candidats se contentent de traduire le pseudo-code et omettent de respecter le type demandé. Par ailleurs, quand un candidat choisit de dévier du pseudocode proposé, il serait judicieux qu'il explique brièvement le comportement de ses fonctions.

Question 5 [93% - 44%]. Il s'agit d'une question facile dès lors que l'invariant de l'algorithme a bien été identifié (et prouvé) à la question 2. Bien entendu, il n'est pas trop tard pour le faire ici.

Partie II

Question 6 [93% - 15%]. Cette question n'est pas difficile dès lors que l'on pose correctement la récurrence. Beaucoup de candidats échouent sur ce point.

Question 7 [96% - 43%]. Certains candidats n'écrivent que le code de la fonction IDA, oubliant d'écrire le code de la fonction DFS. Même remarque qu'à la question 4 vis-à-vis du pseudo-code de la figure 2 et du type attendu.

Question 8 [93% - 52%]. Une question très facile. Une référence explicite à la question 6 est cependant la bienvenue.

Question 9 [74% - 9%]. Il y a huit résultats attendus dans cette question, avec une justification à chaque fois. Beaucoup de candidats en oublient une partie. Très peu de candidats ont l'intelligence de résumer les huit résultats dans un tableau. Peu de candidats évaluent correctement l'espace utilisé (sur la pile) par le parcours en profondeur.

Partie III

Question 10 [84% - 13%]. On peut tout à fait représenter ∞ par l'entier -1 mais il faut alors écrire soigneusement la comparaison $c < min$. Il est également possible de choisir un entier suffisamment grand, tel que la plus grande valeur du type `int`. Il n'est pas raisonnable en revanche de choisir une valeur telle que 2^{10000} lorsque l'on sait que les entiers de Caml sont représentés sur 64 bits au plus. Même remarque qu'aux questions 4 et 7 vis-à-vis du pseudo-code de la figure 3 et du type attendu.

Question 11 [82% - 48%]. Une justification était explicitement demandée. Elle est parfois inexistante ou complètement farfelue. Sans être appréciée, la fonction valant 0 en t et 1 sinon a été acceptée car elle répondait à la question posée. De nombreux candidats ont toutefois proposé une fonction plus réaliste impliquant $\log_2(t/n)$. De nombreux candidats confondent $\log(t-n)$ avec $\log(t/n)$.

Question 12 [**45% - 6%**]. Question peu traitée et très mal traitée en général. On trouve des argumentaires dans lesquels l'hypothèse d'admissibilité n'est pas utilisée.

Partie IV

Question 13 [**86% - 2%**]. De trop nombreux candidats se contentent de dire qu'il y a 16! états possibles et concluent à l'impossibilité d'un parcours en largeur. En particulier, l'exemple d'un chemin optimal de longueur 50 donné dans l'énoncé doit être utilisé pour justifier un nombre trop grand d'états visités par le parcours en largeur.

Question 14 [**59% - 53%**]. Il s'agit là d'une question bien plus simple qu'elle n'en a l'air.

Question 15 [**86% - 47%**]. Des candidats recalculent h intégralement, ce qui est correct mais inutile. D'autres candidats oublient tout simplement de mettre à jour h. Parmi les candidats qui mettent à jour h de manière différentielle par rapport à l'état précédent, ce qui est une bonne idée, certains oublient qu'ils ont déjà modifié `li` et `lj` juste avant.

Question 16 [**79% - 10%**]. Beaucoup de candidats accèdent à la première valeur de la liste `solution` sans penser au cas où cette liste est vide. Le cas particulier `lj` $= 3$ qui empêche le déplacement vers la gauche est assez souvent mal traité, `lj` étant comparée à tort avec 0 ou 4.

Question 17 [**57% - 4%**]. Là encore, des candidats recalculent h intégralement, ce qui est inutile. De même, des candidats testent explicitement si la grille est une solution, alors qu'il suffit de tester si h est nul. La difficulté de cette question tient par ailleurs dans la nécessité de bien revenir en arrière sur les choix infructueux, ce qui la distingue de la question 10. Très peu de candidats réalisent correctement ce retour sur trace.

Question 18 [**38% - 14%**]. Il s'agit ici d'adapter le code de la fonction IDA* écrit à la question 10, ce qui ne pose pas de difficulté particulière.