RAPPORT DE L'ÉPREUVE PRATIQUE D'ALGORITHMIQUE ET DE PROGRAMMATION ULC INFO 2013

Écoles concernées : Cachan, Lyon, Paris
Coefficients : Cachan 5, Lyon 4, Paris 4
Membres du jury : Jérémie Detrey, Pierre-Alain Fouque, Olivier Teytaud

Remarques générales à propos de l'épreuve

Organisation de l'épreuve et statistiques. Comme les années précédentes, cette épreuve demandait aux candidat·e·s de mettre en uvre la chaîne complète de résolution d'un problème informatique : analyse des spécifications, choix des structures de données, analyse de la complexité de la solution retenue, programmation, et tests. En plus de cette partie consacrée à la programmation proprement dite, une présentation orale permettait aux candidat·e·s d'une part de démontrer leur capacité à expliciter la méthodologie qu'illes avaient suivie, d'autre part d'aborder des questions qu'illes n'auraient pas eu le temps de programmer complètement.

Cette année le jury a examiné 132 candidat·e·s, réparti·e·s en 6 sessions dont l'organisation reste identique à celle adoptée les années précédentes. À savoir, les candidat·e·s sont admis·es dans la salle de composition par groupe de 3 toutes les demi-heures. Les candidat·e·s ont 10 minutes pour se familiariser avec l'environnement mis à leur disposition. Illes peuvent pendant cette période poser des questions aux surveillants de l'épreuve s'illes rencontrent des difficultés d'ordre pratique. Puis l'épreuve est préparée durant 3h30 sur machine. En plus des sauvegardes sur le disque dur de la machine, une clé USB est fournie aux candidat·e·s afin qu'illes puissent sauvegarder une seconde copie de leur travail sur cette clé. Cette préparation est suivie d'une interrogation orale. Le contenu de la clé USB n'est pas examiné durant l'interrogation orale. Au total, l'épreuve dure donc 4h10 heures. Le nombre de candidat·e·s est limité chaque jour de telle sorte que les dernièr·e·s candidat·e·s entré·e·s ne croiseront pas les premièr·e·s candidat·e·s sorti·e·s.

Comme les années précédentes, les langages et environnements suivants ont été proposés aux candidat·e·s :

- PC sous Windows XP avec Caml Light, Pascal (Delphi), Maple et Java.
- PC sous Debian/Linux avec Caml Light, Objective Caml, C/C++, Pascal et Java.

Les candidat·e·s ont choisi les langages et environnements dans les proportions suivantes :

Environnements	Windows		Linux			
Proportions	64%		36%			
Langages	Caml Light	Caml	Caml	Caml	Pascal	C/C++
Proportions	2%	61%	10%	21%	1%	5%

Conseils pratiques à destination des candidat·e·s et de leurs préparateurs/trices. Avant toute chose, le jury tient à rappeler qu'il maintient dans son barème un équilibre entre les points attribués à la partie programmation, c'est-à-dire aux réponses numériques que la/le candidat·e doit remplir en annexe de son sujet et qui sont corrigées de manière binaire (pas de demi-point pour une réponse approximativement exacte) et la partie algorithmique (interrogation orale à suivre). Il est **impératif** que les candidat·e·s aient préparé sérieusement cette seconde partie afin de ne pas perdre de temps (et donc de précieux points) durant l'oral. Les candidat·e·s sont

encouragé·e·s à présenter à l'oral leurs idées quant à la résolution de questions qu'illes n'auraient pas eu le temps de programmer durant l'épreuve pratique.

Qui plus est, la partie orale de l'épreuve peut porter sur n'importe quel point de la partie écrite et pas seulement sur ceux explicitement marqués comme tels dans le sujet. En particulier, le jury attend des candidat·e·s qu'illes puissent expliquer la façon dont illes ont obtenu les réponses aux questions et qu'illes aient une idée de la complexité de leur approche. Certain·e·s candidat·e·s (heureusement peu nombreux/euses) arrivent à l'épreuve orale en ayant complètement oublié ce qu'illes ont bien pu programmer deux heures plus tôt ; n'ayant pas non plus noté sur leurs brouillons de quoi se rafraîchir la mémoire, illes se retrouvent donc grandement handicapé·e·s pour la suite de l'épreuve.

Si l'examinatrice/teur interrompt le/la candidat·e, voyant que la bonne compréhension est établie, et qu'il n'est pas utile d'y passer plus de temps, il peut être une bonne idée d'accepter de passer à la question suivante plutôt que de gaspiller des minutes précieuses. De même, il est préférable de ne pas passer de longues minutes à essayer d'improviser une explication douteuse, lorsque l'on peut utiliser le temps restant pour d'autres questions où l'on a des choses à dire. Enfin, il est souvent pertinent de regarder les différentes parties, pour savoir sauter aux éléments que l'on sait traiter.

Il est important de stocker certaines valeurs intermédiaires, en apparence anodines, mais utiles de nombreuses fois, comme les u_n.

L'utilisation de récursivité n'est pas une nécessité, surtout pour des candidat·e·s peu aptes à éviter des très coûteux passages de paramètres (tables de valeurs) dans des fonctions récursives. La complexité en espace n'est pas forcément un élément négligeable d'une question de complexité.

Il ne faut pas renoncer à traiter une fonction sous prétexte que la complexité est exponentielle. Une exponentielle avec une petite constante peut être tout à fait traitable en quelques dizaines de secondes sur des entrées modérées. Certaines questions peuvent demander plusieurs secondes voire minutes suivant les langages et comment elles ont été codées.

Remarques spécifiques à chacune des épreuves

Registres à décalage à rétroaction linéaire

Ce sujet demandait aux candidat·e·s d'étudier certaines propriétés classiques des registres à décalage à rétroaction linéaire (LFSR). Ces objets, qui permettent de calculer efficacement des suites récurrentes linéaires définies sur $\mathbb{Z}/2\mathbb{Z}$, trouvent de nombreuses applications dans divers domaines de l'informatique, de la cryptographie (en tant que suites chiffrantes) aux télécommunications (comme randomiseurs de signal).

Une première partie du sujet cherchait à permettre aux candidat·e·s d'appréhender ces objets en leur faisant manipuler des LFSR aléatoires (questions 2 et 3), puis en leur faisant retrouver certaines propriétés mathématiques simples comme la périodicité de la suite de bits générée (questions 4, 5 et questions d'oral associées). En particulier, la question 5, qui demandait de calculer la période de la suite de bits générée par un LFSR donné, s'est avérée assez discriminante.

La deuxième partie se concentrait sur le problème du calcul de la complexité linéaire d'une suite de bits donnée, qui revient à trouver le plus court LFSR générant cette suite. À cet effet, trois méthodes étaient étudiées :

- La première, de complexité exponentielle, consistait à effectuer une recherche exhaustive parmi tous les LFSR de longueur donnée. De manière surprenante, plus de la moitié des candidat·e·s ont eu du mal avec l'énumération exhaustive qui leur était demandée.
- La seconde méthode proposée (question 7) était une résolution du système linéaire par la méthode du pivot de Gauss, de complexité cubique. Même si certain·e·s candidat·e·s ont correctement effectué l'analyse de complexité, aucun·e n'a implémenté cet algorithme.
- Enfin, la question 8 demandait aux candidat·e·s de retrouver l'algorithme de Berlekamp-Massey, de complexité quadratique. Cette dernière question, plus difficile, n'a été traitée que par un seul candidat.

Jeu de Hex

Ce sujet s'intéressait à la résolution du classique jeu de Hex. Le plateau de jeu est hexagonal, un joueur pose des X et l'autre des O sur le plateau et un critère de connection permet de décider qui gagne. La première question ne posait pas de difficultés particulières. La question 2 imposait de savoir détecter si un ensemble de cases marquées connectait la frontière nord et la frontière sud d'un domaine à pavage hexagonal ; une proportion importante de candidat·e·s a négligé cette question, utilisant des solutions qui ne résolvaient pas tous les cas possibles. Une approche en temps linéaire est possible.

La question 3 demandait de savoir dérouler des stratégies simples, et de détecter les victoires, ce qui imposait de savoir détecter des connections. Plusieurs solutions sont possibles, on pouvait par exemple maintenir une structure de données contenant toutes les cases X connectées au nord et toutes les cases O connectées à l'ouest. Le bon traitement de cette question a été rare. Les questions suivantes, jusqu'à 6, pouvaient être traitées très similairement, si du moins les choix algorithmiques rendaient la méthode efficace. Peu de candidat·e·s avaient pris le temps de bien vérifier la complétude de leur méthode pour les connections, d'où de nombreuses difficultés.

Les questions 7, 8 et 9 demandent de bien comprendre l'optimisation de stratégies. On note que l'approche récursive est possible, mais qu'il faut veiller à ne pas effectuer un grand nombre de copies inutiles pour éviter la saturation mémoire.

Clustering

Ce sujet s'intéressait au problème de clustering en dimension 1 et 2, c'est-à-dire étant donné un ensemble de N points, comment les classer en K classes afin de minimiser la somme des carrés des distances des points à la classe à laquelle ils appartiennent. Cette épreuve s'appuyait sur quelques propriétés arithmétiques que les candidat·e·s devaient justifier à l'oral.

La question 2 ne présentait pas de difficulté et demandait de trier des intervalles par taille croissante. La question 3 proposait une méthode gloutonne pour résoudre le problème en dimension 1 qui consistait à agréger en priorité les classes les plus proches, mais qui n'était pas optimale. Très peu de candidat·e·s sont arrivé·e·s à programmer cette question. Les questions 4, 5 et 6 permettaient de résoudre le problème en dimension 1 en utilisant un exercice de type programmation dynamique. Pour la question 6, il fallait programmer l'algorithme de meilleure complexité pour trouver la réponse quand les instances étaient grandes.

La seconde partie étudiait le problème en dimension 2 qui est NP-complet. La question 7 demandait de programmer une recherche exhaustive afin de trouver la solution optimale. Peu de candidat·e·s ont été capables de programmer cette recherche qui ne présentait pas de difficulté particulière à part d'énumérer efficacement toutes les partitions d'un ensemble en deux sous-ensembles. Enfin, les questions 8 et 9 considéraient le problème de trouver la meilleure approximation d'un ensemble de points par des segments de droites par la méthode des moindres carrés. Dans ce cas particulier où les classes sont des segments, un algorithme de type programmation dynamique était encore demandé pour résoudre le problème en temps polynomial.

Pavages apériodiques

Ce sujet étudiait les problématiques liées au pavage du plan $\mathbb{Z}^2$ par des tuiles de Wang (tuiles carrées dont les bords nord, sud, est et ouest sont de couleurs données). En particulier, il était question de déterminer si un jeu de tuiles donné pavait le plan et, si oui, s'il existait un tel pavage qui soit aussi périodique. Ces deux questions étant prouvées indécidables, le sujet proposait aux candidat·e·s deux méthodes permettant de résoudre rapidement les cas faciles, en testant si un jeu de tuile pavait une zone circonscrite du plan (un carré ou une bande horizontale) puis en vérifiant si un tel pavage était périodique.

La première partie du sujet avait pour but de familiariser les candidat·e·s avec les tuiles de Wang en leur faisant générer un jeu au hasard (question 1) puis en leur demandant d'en éliminer

les tuiles identiques (question 2). Il s'agissait pour cette dernière question de mettre en uvre un algorithme de tri.

La deuxième partie concernait le problème du pavage d'un carré de taille donnée. Il était demandé aux candidat·e·s d'énumérer de manière exhaustive l'ensemble des pavages possibles du carré avec un jeu de tuiles donné. Plusieurs méthodes étaient possibles pour répondre à cette question, l'idée essentielle étant de limiter l'explosion combinatoire due à l'exhaustivité de l'approche en imposant le plus tôt possible les contraintes de validité des pavages considérés.

Enfin, la troisième partie étudiait plus en détails le pavage de bandes horizontales bi-infinies à l'est et à l'ouest. À la question 4, la notion de « tuiles inutiles » permettait ainsi d'éliminer les tuiles d'un jeu qui ne pouvaient servir dans un tel pavage. Il était important pour cette question de se rendre compte qu'éliminer des tuiles inutiles d'un jeu pouvait en rendre d'autres inutiles : il fallait donc penser à itérer le procédé autant de fois que nécessaire. La question 5, très discriminante sur ce sujet, demandait aux candidat·e·s de considérer les empilements valides de deux tuiles comme un autre jeu de tuiles, et d'utiliser les résultats de la question précédente pour déterminer si le jeu de tuiles donné pavait une bande horizontale de deux tuiles de haut. Les questions 6 et 7 demandaient alors de généraliser ce résultat à des empilements de hauteur arbitraire, en procédant à une renumérotation des couleurs à l'est et à l'ouest pour éviter l'explosion combinatoire de celles-ci. Enfin, les questions 8 et 9 étudiaient la périodicité, horizontale et verticale, d'un jeu de tuiles pavant une bande horizontale. Malgré les questions d'oral destinées à les mettre sur la voie, les candidat·e·s n'ont pas abordé ces dernières questions.

Somme de sous-ensembles

Dans ce sujet, on s'intéressait au problème de la somme de sous-ensembles et de la somme de k ensembles. La question 2 demandait de trouver le nombre de collisions entre deux ensembles et de trouver le nombre de solutions de l'équation $x + y = t$ à t donné et $x \in L_1$ et $y \in L_2$. Les candidat·e·s devaient voir que ce second problème pouvait se résoudre par le même algorithme que de trouver les collisions et que la complexité de trouver les collisions se résolvait en temps quasi-linéaire. Dans le cas de trois ensembles, le problème avait une complexité en $O(N^2)$ si N est la taille des ensembles L_i et dans le cas à 4 ensembles, il fallait réduire le problème aux ensembles $L_1 \times L_2$ et $L_3 \times L_4$. Comme la taille de la cible t était petite, les candidat·e·s ont pu imaginer diverses stratégies comme le pruning pour limiter l'explosion combinatoire des ensembles intermédiaires et certains candidat·e·s ont utilisé des tables de hachage.

Les questions 5 et 6 s'intéressaient au problème d'approximation de la somme de sous-ensembles qui est NP-complet, mais où il existe un algorithme d'approximation en temps polynomial en utilisant du pruning afin de limiter l'explosion combinatoire des solutions. Un algorithme de seuillage d'une liste était suggéré et il fallait fusionner des listes triées en temps linéaire. La question orale qui permettait d'établir la complexité polynomiale a été bien traitée pour établir la propriété de récurrence, mais la complexité de la taille des listes était plus délicate.

Les questions 7 et 8 demandaient de résoudre le problème de la somme de sous-ensembles de manière exacte, en utilisant les algorithmes développés dans la première partie. L'idée était d'utiliser des compromis temps-mémoire par rapport à la recherche exhaustive afin d'avoir des algorithmes en temps $O(2^{n/2})$ si n est le nombre de valeurs. Là encore, comme t n'était pas trop grand, le pruning permettait de réduire le temps de calcul. Peu de candidat·e·s ont vu comment résoudre le problème algorithmique proposé, mais les valeurs numériques pouvaient être résolues avec d'autres techniques.

La dernière question s'intéressait à trouver une stratégie optimale dans un jeu lié à la somme de sous-ensembles. On proposait d'aborder le problème sous forme d'un exercice de programmation dynamique que les meilleurs candidat·e·s ont très bien traité.

Équilibre de Nash

La première question demandait simplement de savoir calculer des multiplications et des congruences. La deuxième question portait sur la construction de permutations. Un certain manque

de recul a parfois donné des réponses hasardeuses. Le principe du jeu utilisé pour exemple a été bien compris par l'ensemble des candidat·e·s ; le concept d'équilibre de Nash a aussi été compris par le plus grand nombre. La question 3 demandait de savoir implémenter les règles et la gestion d'une structure de données adéquates ; elle a été plutôt correctement traitée en général. La question 4 portait sur l'implémentation d'un tirage au sort et l'analyse de la complexité du programme correspondant ; elle a été très discriminante, certains candidat·e·s la traitant facilement avec une complexité satisfaisante, d'autres moins efficacement mais avec justesse, et d'autres enfin ne parvenant pas du tout à la traiter.

La suite mettait réellement en œuvre le concept d'équilibre de Nash. La simple compréhension du concept permettait de résoudre la question 5, qui permettait de tester l'implémentation d'un solveur approché proposé dans le sujet. Le sujet présentait la double originalité d'être numérique (utilisant des nombres flottants) et stochastique (utilisant des nombres aléatoires). Les questions 6, 7 et 8 étaient plus difficiles, une implémentation naïve étant très lente ; peu de candidat·e·s ont réussi à proposer une implémentation d'EXP3 qui ait un coût réduit par itération, permettant de traiter les questions jusqu'à la fin dans le temps imparti.