RAPPORT DE L'ÉPREUVE PRATIQUE D'ALGORITHMIQUE ET DE PROGRAMMATION ULC INFO 2014

Écoles concernées : Cachan, Lyon, Paris
Coefficients : Cachan 5, Lyon 4, Paris 4
Membres du jury : Jérémie Detrey, Pierre-Alain Fouque, Michaël Rao, 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 œuvre 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 organisé 5 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. Le nombre de candidat·e·s est limité chaque jour de telle sorte que, modulo une courte période de quarantaine si nécessaire, 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.

Comme fréquemment, les variantes de Caml ont été les plus utilisées. Peu de candidats choisissent Pascal/Delphi ; il s'avère en général que les candidats faisant ce choix rare ont beaucoup de difficultés pratiques ; le jury recommande donc aux candidats de ne s'orienter vers ce langage qu'en parfaite connaissance de cause et en ayant bien vérifié leur maîtrise de l'environnement proposé pour l'épreuve.

Le jury, enfin, insiste sur le caractère pratique de l'épreuve ; les difficultés de maîtrise de l'environnement font partie du problème et il est légitime que des points soient gagnés ou perdus en fonction de l'aptitude à gérer la machine. De même, une mauvaise gestion de mémoire conduisant à une défaillance du système ou l'absence de sauvegarde sont des fautes tout autant qu'une mauvaise compréhension algorithmique.

Conseils pratiques à destination des candidat·e·s et de leurs préparateurs/trices. Le jury rappelle que, comme tous les ans, 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.

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, notamment lorsque le texte du sujet le suggère.

Il est important de stocker certaines valeurs intermédiaires, en apparence anodines, mais utiles de nombreuses fois, comme, dans beaucoup de sujets, 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 la qualité de l'implémentation.

Remarques spécifiques à chacune des épreuves

Décimales de π

Ce sujet demandait aux candidat· e· s de programmer différents algorithmes qui permettent de calculer au fil de l'eau les décimales de π en base 16 et en base 10. Ce faisant, des algorithmes de théorie des nombres élémentaires étaient nécessaires comme des multiplications, exponentiations et inversions modulaires et de factorisation. Enfin, le sujet demandait aussi aux candidat· e· s de bien connaître les différentes bases et décompositions ainsi que de manipuler les rationnels jusqu'à une certaine précision.

La première partie du sujet demandait aux candidat· e· s de décomposer des nombres en base 16 et en binaire et de faire quelques calculs de multiplication et exponentiation modulaires binaires en tenant compte de la précision des entiers, c'est-à-dire moins de 32 bits sur les machines à disposition. Les indications étaient données, mais peu de candidat· e· s sont parvenus à multiplier alors que l'exponentiation était mieux maî trisée pour les petites valeurs. Les questions 3 et 4 demandaient aux candidats d'afficher les décimales en base 16 en utilisant la méthode de Bailey, Borwein et Plouffe (BBP) et ne présentait pas de difficulté particulière si l'exponentiation fonctionnait. Très peu de candidat· e· s sont arrivé.e.s à calculer les décimales correctement.

La deuxième partie demandait de calculer en base 10 les décimales de π en suivant l'algorithme proposé par Xavier Gourdon et la convergence d'une série alternée. La méthode est similaire à celle de BBP et nécessite de calculer des sommes modulaires de binômes. Aucun candidat.e n'est parvenu.e à les calculer de façon exacte en utilisant la méthode proposée. Une fois cette difficulté résolue la fin était similaire à la partie 1.

Les candidat· e· s ont eu des difficultés à décomposer les entiers ou flottants dans les bases 16 et 2 et avec l'algorithme de multiplication rapide. C'est étonnant car cet algorithme, aussi appelé multiplication à la russe, est très similaire à celui d'exponentiation binaire et permet de

contourner les problèmes de petite précision. En conclusion, le calcul avec des flottants et la notion de précision ne sont pas connus chez de nombreux/ses candidat· e· s.

Multiplication rapide de polynômes

Le sujet proposait l'implémentation de multiplication de polynômes par la méthode de Karatsuba et la transformée de Fourier.

Une première question était dédiée à la génération de polynômes aléatoires ; elle sera utile dans toute la suite. La seconde question portait sur une fonction de hachage, utilisée pour signature. Les questions 3 et 4 portent sur des algorithmes naïfs et leur analyse en complexité.

La question 5 porte sur la méthode de Karatsuba. Une question orale porte sur le cas de polynômes dont la taille n'est pas une puissance de 2. Les complexités sont à discuter.

En utilisant les racines (primitives) n-ièmes de l'unité et le procédé d'évaluation-interpolation, on arrive alors à la transformée de Fourier discrète et à la transformée de Fourier rapide.

Contamination de cellules

Le sujet demandait de calculer un diagramme de Voronoï discret, puis de colorier les régions de manière à ce que deux régions voisines aient des couleurs différentes.

La première partie demandait de calculer le diagramme de Voronoï selon plusieurs distances d'un ensemble de sites (généré pseudo-aléatoirement) dans un tableau $n \times n$. Après des questions sans difficulté particulière, il était demandé d'implémenter un algorithme donné pour calculer le diagramme selon la distance de Manhattan. Pour être implémenté de façon efficace (en temps $O(n^2)$, c'est à dire linéaire en la taille du tableau), l'algorithme devait utiliser une queue, programmée avec une liste ou un tableau. Seul.e.s 10 candidat.e.s (sur 36) ont abordés cette dernière question de la première partie.

La deuxième partie demandait de construire le graphe des voisinages des différentes régions du diagramme, puis de le colorier avec un algorithme donné. On peut noter la similitude avec le problème du "Théorème des 4 couleurs", notée par beaucoup de candidat.e.s à l'oral. Une question sur le degré moyen était là pour aiguiller les candidat.e.s sur le fait que le graphe est creux, et qu'il fallait de préférence utiliser une structure par liste d'adjacence. Bien que cela n'était pas nécessaire pour réussir le sujet, on peut noter que le graphe construit est un graphe planaire, et a donc un degré moyen inférieur à 6 et est 5-dégénéré (c'est à dire que tout sous graphe a un sommet de degré au plus 5). L'algorithme de coloration donné était un algorithme glouton qui colorie les graphes k-dégénérés avec au plus $k+1$ couleurs. Il y avait plusieurs difficultés à considérer lors de l'implémentation : un simple appel récursif peut rapidement mener à un dépassement de pile, une recherche naïve du sommet de degré minimum mène à un temps en $O(n^2)$ qui ne permet pas de terminer en temps raisonnable sur les derniers graphes... Un candidat est arrivé à la fin de cette partie.

Retour sur trace

Le sujet demandait de programmer des algorithmes de retour sur trace (« *Backtraking* »en anglais) pour trois problèmes indépendants : les n dames, le tour du cavalier, et la règle de Golomb.

Dans le problème des n dames, il est demandé de placer n dames sur un échiquier $n \times n$ de telle manière que deux dames ne peuvent s'attaquer mutuellement. Il était demandé de générer toutes les solutions pour n de 4 à 18. Un algorithme naïf (on rajoute une dame ligne après ligne sur une case non attaquée) permettait sans problème d'arriver jusqu'à l'instance $n = 15$. Il fallait être un peu plus fin pour arriver à $n = 18$, comme par exemple garder l'ensemble des colonnes et diagonales libres dans des tableaux. La dernière partie du sujet revenait sur le problème des n dames, en demandant de calculer la solution de signature minimale. Il fallait remarquer qu'on n'a plus besoin de générer toutes les solutions, et qu'on peut se servir de la fonction de signature comme fonction d'optimisation. Pour réussir les dernière instances, il fallait brancher en priorité sur les lignes de plus grand v_i, car ce sont elles qui jouent le plus dans la signature.

Le deuxième problème était le problème du tour du cavalier, où il est demandé de parcourir toutes les cases d'un échiquier $n \times n$ avec un cavalier (mouvement en L). Pour réussir les instances difficiles, il fallait une heuristique pour savoir s'il est impossible de visiter l'ensemble des cases restantes. Cela est le cas par exemple si il y a plus de 2 cases non visitées qui sont adjacentes (par le mouvement du cavalier) à au plus une case non visitée.

Enfin, le dernier problème était le problème de la règle de Golomb où il est demandé de placer n « marques »sur une règle telles que les différences soient deux à deux différentes. Seul.e.s 10 candidat.e.s (sur 36) ont traité le problème du cavalier ou de la règle de Golomb, alors que les première instances étaient facilement traitables par des algorithmes naïfs.

Plus courts chemins

Il s'agissait ici de calculer les plus courts chemins d'un point à un autre dans des graphes. La première question était du simple maniement de graines pseudo-aléatoires. Les questions 2 et 3 étaient basées sur la construction de graphes et ne devraient pas poser de problèmes particuliers. L'algorithme de Dijkstra était alors introduit et donnait lieu à la question 4. Il s'agit de calculer le plus court chemin d'un nœud appelé source, à un autre noeud appelé cible. L'algorithme de Bellman-Ford occupait les questions 5 et 6. Il permet de calculer les plus courts chemins de tous les nœuds à une cible donnée. La méthode heuristique A*, généralisant Dijkstra, donnait lieu à plusieurs questions orales et une implémentation pour la question 7. Il s'agissait d'introduire une heuristique accélérant le calcul. L'algorithme de Johnson permettait de déterminer tous les plus courts chemins (de chaque nœud à chaque nœud du graphe). La dernière partie demandait plus de créativité, l'algorithme étant seulement suggéré dans ses plus grandes lignes. Il s'agissait de déterminer un arbre couvrant de poids minimal. Certains candidats sont parvenus à donner les grandes lignes d'une preuve de consistance de l'algorithme qu'ils avaient proposé.

Dans l'ensemble du sujet, une bonne implémentation se devait d'utiliser les structures particulières des graphes proposés. La construction des graphes prenait elle même un temps non-négligeable, et il convenait donc de la soigner, notamment en ne recalculant pas tous les termes des séries "u".