ÉPREUVE ORALE PRATIQUE D'ALGORITHMIQUE ET PROGRAMMATION
ENS : PARIS LYON CACHAN
COEFFICIENTS : PARIS 4 LYON 4 CACHAN 6
MEMBRES DE JURYS : Arnaud Legrand, Sylvain Pion, Frédéric Tronel

Organisation de l'épreuve

Généralités et statistiques

Comme l'année dernière, cette épreuve demandait aux candidats de mettre en œuvre la chaîne complète de résolution d'un problème informatique : analyse des spécifications, choix des structures de données, mise en forme et évaluation de l'algorithme et de sa complexité, programmation sur machine et test des programmes. En plus de la partie programmation, la présentation orale permettait d'évaluer les candidats sur leur capacité à expliquer leurs approches et solutions.

Le jury a examiné cette année 96 candidats, sur 6 sessions organisées comme les années précédente en «pipeline» : les candidats d'une session commune arrivent par groupe de 3 toutes les demi-heures, de sorte que les derniers arrivés rentrent avant que les premiers n'aient fini leur présentation orale. Une fois encore, le jury a fait un barème répartissant les points à égalité entre la partie «résultats pratiques» et la présentation des algorithmes à l'oral.

Comme l'année dernière, plusieurs langages et environnements de programmation étaient proposés :

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

Il était demandé aux candidats de choisir le langage et l'environnement à l'avance. Encore une fois, CAML Light sous Windows a été très majoritairement choisi : 60 % des étudiants ont opté pour cet environnement. 13,4 % des étudiants ont choisi CAML Light ou Ocaml sous Linux, 13,4 % ont choisi Pascal sous Windows ou sous linux, 6,2 % C/C++ sous Linux. Un candidat a ont choisi Maple cette année.

Comme l'an dernier, chaque sujet était accompagné d'une fiche réponse «type» correspondant à un $\widetilde{u_0} \neq u_0$ permettant ainsi à chaque candidat de comparer les réponses obtenues par son programme avec $\widetilde{u_0}$ avec celles de la fiche réponse «type». Une fois encore, cette formule a permis à la majorité des candidats d'éviter de donner des réponses incorrectes et donc de progresser régulièrement, à leur rythme.

De manière générale, les candidats semblent de mieux en mieux préparés à cette épreuve. De moins en moins de candidats perdent du temps avec la génération de nombres aléatoires (maintenant classique) et stockent bien leurs résultats au fur et à mesure[1].

[1] Se référer aux rapports des années précédentes pour une discussion plus complète sur ces problèmes.

Nouveautés

Cette année, une dimension supplémentaire a été apportée à certain des sujets. Dans bon nombre des sujets précédemment posés, on demandait aux candidats de programmer quelque chose, souvent délicat ou algorithmiquement fin, mais il s'agissait en général plus d'un «exercice de style» qu'un développement vraiment «utile» ou directement utilisable. La principale motivation derrière l'exécution du programme des candidats était la vérification des résultats attendus. Cette année, le programme développé par les candidats était parfois conçu et utilisé afin d'obtenir une meilleure compréhension du problème étudié. C'est le cas par exemple du sujet «ordonnancement» où certaines des questions permettaient de repérer des propriétés intéressantes (optimalité de FIFO, faible pertinence de certaine métriques, difficulté d'un problème par rapport à un autre) voire d'aider les candidats à trouver certain des contre-exemples dont la présentation était attendue à l'oral. Le sujet «pair-à-pair» fonctionnait sur le même principe puisqu'il proposait aux candidats d'observer des propriétés statistiques et de réfléchir à comment en tirer parti dans le contexte proposé.

Dans le cas où les candidats n'arrivaient pas à programmer eux-mêmes la réponse aux différentes questions, les réponses fournies dans la fiche réponse étaient suffisantes pour mener à bien l'analyse attendue. De manière générale, cette partie «interprétation» n'a pas trop déstabilisé les candidats et a donné lieu à des discussions intéressantes.

Commentaires spécifiques aux différents sujets

Distances entre courbes Ce sujet portait sur des algorithmes géométriques simples : des calculs de distances Euclidienne et de Hausdorff entre ensembles de points et de segments, puis entre lignes polygonales, et en particulier des convexes, autour desquels plusieurs questions étaient posées. C'est un des rares sujets de géométrie qui considère la distance Euclidienne, par opposition aux normes L_1 ou L_∞ qui sont plus simples à calculer.

De nombreux candidats ont rencontré des problèmes de dépassement de capacité et n'ont pas bien lu le sujet, qui recommandait simplement d'utiliser de l'arithmétique flottante pour remédier à ce problème. Ces candidats n'ont donc pas pu aller bien loin. À part ce problème bloquant, la majorité des candidats a su programmer correctement les primitives de distances et les algorithmes jusqu'à la question 5, qui ne présentaient pas d'autre difficulté spécifique.

Les algorithmes de vérification de convexité proposés par les candidats étaient presque tous incomplets. La majorité a proposé un algorithme de complexité quadratique (alors qu'on demandait un algorithme «de complexité optimale», ici linéaire). Beaucoup ont proposé la bonne solution qui était de parcourir l'enveloppe convexe en faisant des tests d'orientation sur des triplets de sommets consécutifs. Peu ont vu que cela ne suffisait pas et qu'il y avait la possibilité de faire plusieurs tours, la question 7 permettant de détecter si l'algorithme était correct sur ce point. Très peu de candidats ont eu le temps de mettre en pratique leur

algorithme sur les questions 6 et 7.

Finalement, très peu de candidats ont abordé la question 8 concernant la séparation des convexes, mais ils s'en sont cependant bien sortis à l'oral. Aucun candidat n'a entamé les questions suivantes.

Nous recommandons aux préparateurs de sensibiliser les candidats aux problèmes de dépassement de capacité des entiers, particulièrement en CAML où les entiers sont limités à 31 bits de précision, ainsi qu'aux erreurs d'arrondis des flottants.

Ordonnancement avec préemption Ce sujet portait sur l'étude des performances d'algorithmes d'ordonnancement *online* de tâches préemptibles. Il ne présentait aucune difficulté algorithmique mais sa formulation peu classique a déstabilisé certains candidats qui ont eu du mal à intégrer les définitions et à se représenter ce qu'était un ordonnancement.

La question 2 portait sur la génération d'instances de type Poisson (typiques de celles que l'on peut rencontrer sur un serveur ou dans des *call centers*) et a été traitée correctement par presque tous les candidats. En revanche, seule la moitié des candidats a réussi à faire la question 3, c'est-à-dire à évaluer les performances d'un ordonnancement de liste. L'intérêt des ordonnancements de listes dans ce contexte est qu'ils sont au moins meilleur que n'importe quel autre ordonnancement. La démonstration de ce résultat repose sur de simples arguments d'échange et faisait l'objet d'une question à développer à l'oral.

Plus surprenant, seul un candidat a réussi à écrire une fonction générant l'ensemble des permutations de $\{1, \ldots, n\}$, trois candidats ayant préféré les calculer carrément à la main pour répondre à la question 4 ! Le rappel de la définition récursive de l'ensemble des permutations d'un ensemble X se prêtait pourtant très bien à une programmation récursive. Cette fonction permettait alors de chercher l'ordonnancement optimal pour chacune des quatre métriques proposées (question 4) et de rechercher pour une instance simple une structure éventuelle (et de se rendre compte par exemple que l'ordonnancement FIFO était optimal pour la métrique MF). Cette fonction permettait également de calculer des statistiques (meilleur ordonnancement, pire ordonnancement, ordonnancement moyen) pour chacune des quatre métriques et de se rendre compte que la métrique MF par exemple est beaucoup moins pertinente que la métrique MS et que la modélisation de ce type de système n'a finalement rien de trivial. Les candidats ont dans l'ensemble réussi à mener cette analyse en s'aidant des résultats de la fiche réponse.

Le reste du sujet n'a pratiquement été abordé qu'à l'oral et proposait des algorithmes optimaux simples pour trois des métriques. Encore une fois, les démonstrations d'optimalité reposaient sur des arguments d'échange très simples et ont parfaitement été trouvées par quelques candidats.

Nous recommandons aux préparateurs de familiariser les candidats aux algorithmes simples permettant d'énumérer les permutation d'un ensembles, les sous-ensembles d'un ensemble, les sous-ensembles de taille k d'un ensemble, …. La

programmation de ce type d'énumération découlant directement d'une définition récursive ne devrait pas surprendre les candidats.

Étude des pandémies Ce sujet portait sur des algorithmes de graphes classiques. On y retrouvait le classique parcours en largeur, Union-Find, et l'arbre couvrant minimal (Kruskal). Il est apparu qu'une des difficultés principales de certains candidats a été le choix d'une bonne structure de données pour représenter le graphe. Le graphe étant creux, une matrice d'adjacence ou une liste d'arêtes était une erreur, il valait mieux dans ce cas utiliser un tableau de listes. Les candidats qui n'étaient pas parfaitement à l'aise avec ces structures de bases n'ont pas réussi à les composer, et sont donc partis dans des directions les menant à l'échec dans les questions suivantes. À noter qu'un candidat a programmé toutes les questions (mais pas forcément toutes correctement), le sujet étant plus court que la moyenne. Parmi les questions à développer à l'oral, on notera celles sur la redécouverte de l'algorithme d'Union/Find qui n'était qu'esquissé ainsi que la démonstration de sa complexité (le cas simple logarithmique uniquement) qui a en général bien occupé les candidats.

Notons, qu'un des candidats utilisant le langage Maple a utilisé la bibliothèque de graphes fournie par le langage pour calculer les composantes connexes, le diamètre, etc. Cela lui a permis d'obtenir les solutions pour les instances de petites tailles (a) mais qu'il n'a jamais pu obtenir une réponse pour les instances plus grandes (b et c). Il est du coup passé complètement à coté de tous les problèmes de calcul de complexité du sujet.

Nous recommandons aux candidats de vraiment s'entraîner avec les structures de bases comme les listes et les tableaux, afin de les maîtriser totalement et ainsi de pouvoir se concentrer sur les aspects plus compliqués des sujets.

Diagrammes de décisions binaires Ce sujet portait sur l'étude des diagrammes de décision binaire. Une première partie du sujet était consacrée à l'étude des formules de logique booléennes. Elle permettait aux candidats de programmer des algorithmes naïfs de résolution de problèmes classiques (tels la recherche de valuations satisfaisant une formule de logique du premier ordre (ANYSAT), ou bien encore le nombre de valuations satisfaisant une formule (#SAT)). La grande majorité des candidats a réussi à programmer correctement cette partie qui ne requérait pas de structure de données particulièrement complexe. Les candidats sont dans l'ensemble à l'aise avec les structures arborescentes. L'énumération de l'ensemble des valuations possibles pour n variables booléennes n'a pas non plus présenté de difficulté particulière (le sujet donnait cependant une indication très explicite sur une manière de procéder).

Une seconde partie était consacrée à la programmation de tables de hachages. Là encore les candidats s'en sont en général bien sorti. Pour une partie des candidats cette structure de données n'était pas une découverte. Il était même possible de traiter le reste du sujet sans suivre explicitement l'implémentation rudimentaire suggérée dans cette partie. Cependant afin de pouvoir répondre correctement

aux questions 7 et 8 (et donc d'obtenir des points assez facilement gagnés) il était préférable de traiter cette partie du sujet.

Une troisième partie du sujet était consacrée aux diagrammes de décision binaire réduits et ordonnés à proprement parler. La construction des diagrammes nécessitait de bien comprendre l'algorithme esquissé dans cette partie. Aucun candidat n'a réussi (dans le temps imparti) à traiter cette construction. Il faut cependant noter qu'une partie des candidats a su montrer à l'oral qu'ils n'étaient pas loin d'obtenir un algorithme fonctionnel. L'implémentation efficace de cette partie nécessitait une structure de données permettant de retrouver rapidement un élément à partir d'une clé (ici un triplet d'entiers). L'utilisation d'une table de hachage n'était qu'une des possibilités. La programmation des procédures pour les problèmes ANYSAT et #SAT ont pu être traitée à l'oral par certains candidats. Quelques candidats qui utilisaient des langages compilés ont pu répondre aux questions posées dans cette partie en réutilisant les algorithmes naïfs développés lors de la première partie.

Enfin une dernière partie permettait de mettre en application les algorithmes développés dans la partie précédente sur un cas d'école, le problème des n-dames. Il s'agissait essentiellement d'écrire une formule booléenne dont le nombre de variables augmente de manière quadratique en fonction de n pour laquelle chaque valuation code une solution du problème des n-dames. Le problème était sans doute attaquable dans le temps de l'épreuve en utilisant l'algorithme naïf pour $n = 5$, mais peu probablement pour les valeurs suivantes. Les seuls candidats à avoir traiter cette partie n'ont pas utiliser les diagrammes de décision binaire mais ont programmé un algorithme de recherche ad-hoc. Quelques candidats ont su répondre aux questions à développer à l'oral pour cette partie.

Réseaux de Petri Ce sujet se proposait d'étudier le fonctionnement de réseaux de Petri. La définition qui en était donnée était simplifiée puisque les transitions ne pouvaient produire ou consommer qu'au plus un jeton (même si une généralisation à plus de jetons ne posait aucun problème technique). La première partie se concentrait sur l'implémentation de la sémantique de réseaux construits aléatoirement. La très grande majorité des candidats a correctement traiter cette partie.

Dans une seconde partie, les candidats devait développer une procédure permettant la construction du graphe des marquages d'un réseau de Petri. Comme ce graphe peut être infini, il était nécessaire de se fixer une borne sur le nombre de sommets pouvant faire parti du graphe. Un parcours en largeur du graphe était ici préférable afin de faciliter le traitement des questions ultérieures (même si ce n'était pas une nécessité absolue pour répondre aux questions de cette partie). Une grande partie de candidats n'est pas très familier avec les parcours de graphe (en profondeur ou en largeur). Même si ces notions ne font pas partie du programme de l'épreuve, nous ne pouvons que conseiller aux préparateurs d'aborder ces notions avec leurs étudiants, tant le rôle des graphes est central en informatique. L'évaluation de la complexité des algorithmes développés par les étudiants a été

convenablement traitée, surtout pour ceux qui avaient développé une solution purement itérative.

Une troisième partie concernait la recherche de blocage dans les cas où le graphe des marquages est fini. Pour les candidats qui avaient choisi un parcours en largeur, il était facile de déterminer la longueur du plus court chemin vers un éventuel blocage.

Enfin dans une dernière partie, l'aspect dirigé du graphe des marquages était oublié. Cela permettait de développer un calcul de diamètre du graphe et obligeait les candidats à parcourir le graphe non plus désormais depuis le marquage initial, mais depuis l'ensemble des sommets. Quelques candidats ont su traiter cette question.

On notera que deux candidats ont ressenti le besoin d'utiliser des tables de hachage pour répondre efficacement à ce sujet. L'un a utilisé l'implémentation standard de camllight et l'autre en a pragmatiquement reprogrammé une version approximative entièrement statique (mais largement suffisante) en pascal pour accélérer son code… Enfin, avec certains candidats particulièrement réactifs, il a été possible d'aborder à l'oral une solution au problème de l'explosion du graphe de marquage : la notion d'ω-marquage proposée par Karp et Miller en 1969.

Nos recommandons aux préparateurs de ne pas se focaliser seulement sur les aspects purement fonctionnel de Caml, mais aussi d'aborder les traits impératifs du langage (références, structures de données mutables, boucles **for** et **while**).

Réseaux pair-à-pair non structurés Ce sujet portait sur l'application à la conception de réseaux pair-à-pair non structurés de théorèmes fondamentaux sur les graphes aléatoires. À part les questions de complexité très classiques du début, une des originalité de ce sujet résidait dans la demande d'analyse et d'interprétation des résultats obtenus par les programmes des candidats.

Les premières parties étaient dédiées à l'écriture de générateurs de nombres aléatoires un minimum solides ainsi qu'à la génération de graphes aléatoires. Au passage on vérifiait que le degré moyen du graphe $G(n,d)$ était bien d. La partie 4 demandait de programmer l'algorithme de Floyd-Warshall qui permettait d'obtenir d'un seul coup la connexité et le diamètre d'un graphe.

La partie 5 enfin permettait d'illustrer le théorème suivant :

Théorème (Erdős et Rényi, 1959)**.** *Soit* $c > 0$ *et* $p = \frac{1}{n}(\ln n + c + o(1))$.

$$P(G(n,p) \textit{ connexe}) \xrightarrow[n\to\infty]{} e^{-e^{-c}}$$

Cet effet de seuil surprenant à $\ln n$ était discuté et analysé à l'oral à l'aide des solutions à la question 6 données dans la fiche réponse. On demandait aux candidats quel type d'expériences ils auraient pu faire pour identifier et illustrer ce seuil à $\ln n$, ainsi que ce qui n'allait pas dans le «protocole expérimental» choisi. On demandait également d'interpréter l'intérêt d'un tel résultat dans le contexte des réseaux pair à pair.

La partie 6 illustrait un second résultat fondamental, qui est celui de l'émergence d'une composante géante au fur et à mesure que la probabilité p que deux sommets soient connecté augmente. Une autre façon de voir cette propriété : la probabilité qu'un graphe aléatoire soit connexe est à peu près la même que celle qu'aucun sommet ne soit isolé. Les expériences proposées en partie 6 illustraient donc le fait que quand le graphe n'est pas connexe, il y a en général une énorme composante et des sommets isolés, mais pas de petite composante. Ainsi, il devient très facile pour un pair de détecter qu'il n'appartient pas à la composante géante et d'en tirer les conséquences (en essayant de rejoindre le réseau s'il n'a plus aucun voisin).

Enfin, la dernière partie proposait un protocole d'inscription dans le réseau utilisant uniquement des informations locales et permettant d'obtenir un graphe tel que le voisinage des pairs est 1) bien de l'ordre de $c \log n$ et 2) uniformément échantillonné sur l'ensemble du graphe (grâce à une technique de marches aléatoires). Ces deux points ont été abordés à l'oral par deux candidats : le premier point en remarquant que le nombre d'arêtes A_n était défini par la récurrence $A_{n+1} \approx A_n + \frac{A_n}{n} + c$; le second point s'obtenait en estimant l'espérance de la longueur de la marche aléatoire proposée et en la comparant au diamètre du graphe observé en partie 5.