Option informatique

Présentation du sujet

Le sujet comportait quatre parties et avait pour but d'étudier le problème de la construction d'un arbre de Steiner pour un ensemble de sommets X dans un graphe connexe G (c'est-à-dire un sous-graphe de G contenant les sommets de X et qui soit un arbre).

— La première partie demandait de démontrer certains résultats plus ou moins classiques sur les graphes ainsi que de programmer un tri-fusion sur des listes. Ces questions étaient amenées à servir dans la suite de l'épreuve.

— La deuxième partie s'attachait à programmer et justifier l'algorithme de Kruskal inversé qui permet d'obtenir un arbre couvrant de poids minimal dans un graphe pondéré connexe. Cette partie, assez algorithmique, demandait aux candidats d'adapter des techniques vues en cours (compilation de listes d'arêtes, parcours d'un graphe) pour être menée à bien.

— La troisième partie introduisait les arbres de Steiner et permettait de prouver que la recherche d'un tel arbre ayant le moins de sommets possibles pour un ensemble X de sommets donné est aussi difficile que celui de la satisfiabilité d'une formule logique. Il s'agissait donc ici de construire une correspondance entre les deux problèmes. Des connaissances élémentaires de logique suffisaient pour traiter cette partie, le sujet guidant les différentes étapes de la transformation.

— Enfin, la quatrième partie décrivait une technique pour construire en temps polynomial un arbre de Steiner dans un graphe connexe pondéré dont le poids était du même ordre de grandeur que celui d'un arbre optimal. Elle commençait par l'étude de l'algorithme de Floyd-Warshall pour trouver les poids des plus courts chemins entre tous les sommets du graphe avant d'aborder la technique proprement dite. Cette partie comportait des questions assez abordables à l'exception des dernières questions Q33 et Q35 qui demandaient un investissement plus conséquent de la part des candidats.

Les chapitres abordés étaient : algorithmique classique, théorie des graphes, logique élémentaire.

Ce sujet comportait malheureusement quelques coquilles, plus ou moins flagrantes, à la portée limitée, mais dont certaines ont pu faire perdre un peu de temps aux candidats.

— À la question 4, on demandait de montrer qu'un graphe pondéré avec une fonction de poids injective possède un unique arbre couvrant de poids minimal, ce qui est bien sûr impossible si le graphe n'est pas connexe. Le jury a légèrement valorisé les candidats le faisant remarquer sans pénaliser ceux qui considéraient cette hypothèse comme implicite, la principale difficulté de cette question plutôt ardue étant de toute façon la justification de l'unicité.

— L'algorithme 1 décrit à la suite de la question 9 est évidemment l'algorithme de Kruskal inversé et non l'algorithme de Kruskal classique malgré ce que dit la légende.

— À la question 17, on demande de déterminer « sur le graphe G_1 représenté sur la figure 3 » un arbre de Steiner associé à l'ensemble de sommets $X = \{s_1, s_4, s_5\}$ alors que la figure 3 ne comporte pas de sommet s_5. Il s'agit d'une erreur flagrante à laquelle la quasi-totalité des candidats a été confronté. Il se trouve que l'erreur pouvait être rectifiée car le graphe G_1 avait été décrit d'une autre façon (et de manière correcte) dans les préliminaires du sujet. Plus loin, la figure 6 qui s'appuyait elle aussi sur le graphe G_1 permettait de comprendre ce qu'aurait dû être la figure 3.

Analyse globale des résultats

Le sujet était de longueur raisonnable mais comportait quelques questions difficiles. Les meilleures copies ont pu aller au bout de l'épreuve en faisant quelques impasses. Pour avoir une bonne note, il était important de s'intéresser aux quatre parties, qui contenaient toutes des questions abordables.

Il y avait cette année une proportion un peu plus importante de questions théoriques que de programmation. La partie programmation, plus facile, a été traitée convenablement sur la plupart des copies. Les réponses apportées aux questions théoriques, en particulier celles portant sur les graphes dans les parties I et II, ont été, par contre, souvent trop confuses, illogiques ou s'appuyaient sur des cas trop particuliers.

Comme à chaque fois, beaucoup de temps ou de points ont été perdus par une lecture trop superficielle de l'énoncé. Certains candidats justifient des réponses alors que l'énoncé indique explicitement de ne pas le faire ou oublient de donner des complexités alors que la question les demande. Trop de candidats réécrivent des fonctions du module `List` comme `List.rev` alors qu'ils les ont sans doute rencontrées dans leur formation et que le sujet disait explicitement qu'on pouvait les utiliser.

Le jury rappelle qu'il apporte une grande attention à la présentation : une copie aérée (en particulier dans les codes de programme), avec des résultats encadrés, est bien plus facile à lire ce qui ne peut que profiter aux candidats. Les écritures illisibles ont été pénalisées.

Commentaires sur les réponses apportées et conseils aux futurs candidats

Sur la partie I, le jury a rencontré beaucoup de raisonnements approximatifs, en particulier sur les questions 2 et 3, pourtant classiques. La très grande majorité des candidats a choisi de faire des récurrences sur le nombre de sommets des graphes mais certains raisonnent mal. En admettant $\mathcal{P}(n)$, pour montrer $\mathcal{P}(n+1)$, il s'agit de considérer un graphe G ayant $n+1$ sommets et de se ramener à un graphe G' ayant un sommet de moins afin de pouvoir, le cas échéant, lui appliquer l'hypothèse de récurrence (et non de partir d'un graphe à n sommets vérifiant la propriété voulue et de lui rajouter un sommet et des arêtes d'une manière plus ou moins biaisée afin d'avoir facilement la propriété au rang $n+1$). Rappelons au passage que tout graphe n'a pas forcément un sommet de degré 1. On peut faire remarquer que des récurrences sur le nombre d'arêtes sont tout aussi efficaces et présentent quelques avantages (quand on enlève une arête, le nombre de sommets est inchangé, on crée juste une composante connexe de plus si l'arête ne fait pas partie d'un cycle).

Sur la question 7 qui demande d'écrire une fonction fusionnant deux listes triées dans l'ordre croissant, certains candidats se sont astreints à écrire une fonction récursive avec accumulateur mais se sont embrouillés sur l'ordre dans lequel doivent être mis les éléments dans l'accumulateur ainsi que sur la gestion que cela implique si l'une des listes est vide. On ne saurait trop conseiller aux candidats d'aller au plus simple (une récursivité non terminale ici).

Sur la question 8 qui demande d'écrire la fonction principale du tri fusion, nombreux sont ceux qui oublient le cas de la liste vide ou plus subtil, celui de la liste à un élément. Commencez toujours, lors d'un programme récursif, par traiter les cas de base afin de ne pas les oublier !

La partie II s'intéressait à l'algorithme de Kruskal inversé. La question 9, qui demandait de rappeler l'algorithme de Kruskal « ordinaire » (question de cours par excellence), a été dans l'ensemble bien traitée même si donner une complexité n'a pas beaucoup de sens si l'on ne précise pas avec quelles structures l'algorithme est programmé. Le jury s'est néanmoins montré conciliant sur ce point du moment que la complexité donnée était l'une de celles correspondant aux implémentations classiques ($O(|A|\log|S|)$ ou $O(|A|.|S|)$).

Le reste de cette partie amenait à écrire de nombreux programmes. Dans l'ensemble, le jury a constaté une bonne maitrise des bases du langage OCaml même si l'on rencontre encore trop souvent des confusions avec Python sur les listes en particulier (pour une liste `l` en OCaml, `l.(i)` ne permet pas d'accéder

au i-ième élément) ou sur les délimiteurs (indenter rend un code OCaml plus lisible mais ne permet absolument pas de délimiter des blocs, il faut utiliser par exemple des `begin/end`). Enfin, le code de certaines fonctions est souvent alourdi par l'usage de variables ou de références inutiles.

On rappelle que, la plupart du temps, la programmation de plusieurs fonctions intermédiaires pour répondre à une question donnée devrait être un signe qui alerte le candidat sur une méthode maladroite ou un algorithme à la complexité dégradée. En tout état de cause, pour toute fonction secondaire, il faut indiquer quels sont ses paramètres d'entrée et de sortie et ce qu'elle est censée faire, en choisissant des noms représentatifs (et non produire des `aux1 i j k`, `aux2 l m n` sans explication).

Sur la question 12 qui nécessitait de faire un parcours de graphe, trop de candidats ont oublié d'introduire une structure permettant de collecter les sommets par lesquels l'algorithme était déjà passé, ceci afin d'éviter de boucler indéfiniment sur un éventuel cycle du graphe. L'usage d'un tableau pour une telle structure est par ailleurs bien plus efficace que celui d'une liste.

La question 13 demandait d'écrire le programme principal de l'algorithme de Kruskal. De nombreux candidats n'ont pas vu qu'à chaque étape, lorsqu'on veut tester si l'arête (ij) de poids p peut être retirée du graphe B en cours, il suffit d'appeler `connectes g i j p` pour voir si le graphe reste connexe sans cette arête : ils ont testé à la place si tous les sommets restaient connectés au sommet 0 (ou pire si tous les sommets restaient connectés deux à deux) ce qui affecte bien sûr fortement la complexité de l'algorithme.

Enfin, signalons qu'à la question 15, plutôt difficile, on attendait une référence à la question 5 de la partie I et que la question 16 a été jugée au regard du programme écrit par le candidat et de sa complexité réelle.

La partie III faisait établir des correspondances entre la satisfiabilité d'une formule logique, la recherche d'un stable dans un graphe G_φ associé à la formule et celle de la recherche d'un arbre de Steiner dans un autre graphe G'_φ construit à partir de G_φ.

Pour la question 17, au demeurant plutôt facile, le jury a décidé d'accorder la moitié des points à tout candidat qui a signalé l'incohérence, le reste des points étant dévolu à la correction de la question pour les nombreux candidats qui ont pris l'initiative de corriger le dessin ou de modifier l'ensemble X pour être en accord avec la figure (la correction s'adaptant bien sûr au choix effectué).

La question 18 qui s'appuyait toujours sur le graphe G_1 avait l'avantage de pouvoir être traitée avec la « mauvaise » figure. Elle a été corrigée normalement avec le graphe traité par le candidat (celui de la figure 3 ou le « bon » graphe).

Une fois l'erreur d'énoncé digérée, les questions 17, 18 et 19 n'ont pas semblé poser de problème. Pour les questions 20 et 21, le principe est souvent compris mais les explications se sont révélées parfois confuses ou peu argumentées : en particulier, donner un modèle pour une formule revient à donner une distribution de vérité pour l'ensemble des variables (et non pour certains littéraux).

Les questions 22 et 23 ont par contre connus moins de succès et dans la question 24, la quasi-totalité des réponses oublient d'évoquer la construction des graphes G_φ et G'_φ qui doit aussi être faite en temps polynomial pour avoir le résultat. Rares aussi sont ceux qui comprennent pourquoi il est nécessaire d'avoir un arbre de Steiner minimal en nombre de sommets.

La partie IV-A sur l'algorithme de Floyd-Warshall a été globalement bien traitée même si le jury a noté trop souvent à la question 27 des explications qui se limitent à la paraphrase de la formule. On précise qu'à la question 28, deux matrices (une pour les distances et une pour les prédécesseurs) suffit pour tout le programme (car il n'y a pas de problème de recouvrement de valeurs possibles d'une étape à l'autre). Le jury, conscient que cette propriété n'est pas évidente pour une personne non familière avec l'algorithme de Floyd-Warshall, n'a pas pénalisé les candidats qui copiaient des matrices à chaque étape. Par contre, dans cette question, l'initialisation de ces matrices a été trop souvent oubliée et il fallait bien sûr partir d'une copie de la matrice du graphe reçue en entrée et non de la matrice elle-même.

La partie IV-B alternait entre questions simples qui rapportaient peu de points et questions difficiles rarement abordées de manière probante. On notera néanmoins quelques confusions sur la question 34 : il fallait proposer une renumérotation des éléments de X (la plus naturelle étant de renuméroter les éléments de X dans l'ordre où on les rencontre, en commençant à 0). Une part non négligeable des réponses a supposé que cette correspondance était donnée implicitement sous la forme d'une fonction `num`. Pour éviter ce genre d'erreurs, il suffit de bien respecter les spécifications (i.e le type) de la fonction demandée.

Conclusion

Malgré les erreurs d'énoncé, que nous ne pouvons que regretter, un candidat qui avait bien révisé son cours d'option informatique sur les graphes et qui prenait le soin de réserver du temps pour s'intéresser à toutes les parties avait tous les outils en main pour réussir. Le jury ne peut qu'encourager les futurs candidats à bien maitriser les algorithmes explicitement au programme.

De façon générale, afin de préparer au mieux cette épreuve, il convient de s'imposer un entrainement régulier sur machine, seul moyen d'acquérir de bons réflexes en programmation et de s'habituer à rédiger en détail des questions théoriques afin de gagner en aisance dans les argumentations.