MP

CONCOURS COMMUN INP

RAPPORT DE L'ÉPREUVE ÉCRITE D'INFORMATIQUE

1/ PRÉSENTATION DU SUJET

Ce sujet est composé de trois exercices et est conçu pour évaluer une large part du programme d'option informatique de la filière MP et du programme d'informatique commune des deux années. Les exercices comportent des questions de difficulté graduée (accessibles et proches du cours au début, plus ambitieuses en fin) de manière à distinguer les différents niveaux de maîtrise du programme de l'ensemble des candidates et candidats. Aucun thème ni mise en situation particulière n'a été mis en œuvre pour rattacher ces exercices entre eux et ceux-ci étaient donc totalement indépendants.

Le premier exercice traite de la représentation binaire de nombres entiers et de leur multiplication efficace à l'aide de l'algorithme de Karatsuba. Tout le code de cet exercice est attendu dans le langage Python et la représentation des nombres entiers relève du programme d'informatique commune, mais des éléments du programme de l'option informatique sont aussi évalués dans cet exercice (stratégie diviser pour régner et analyse de la complexité des algorithmes récursifs utilisant cette stratégie). Les questions se répartissent entre implémentation d'algorithmes du cours, compréhension et implémentation d'algorithmes nouveaux mais donnés (Karatsuba), analyse de complexité de programmes et questions de cours sur les manières de représenter les nombres entiers en base 2.

Le second exercice aborde principalement le programme d'option informatique, même si quelques points du programme d'informatique commune y sont présents (dictionnaires). Cet exercice porte sur l'usage de structures adaptées pour représenter des ensembles de mots, d'abord sous forme arborescente, puis en fin d'exercice avec des automates finis. La première structure abordée, les arbres préfixes, est décrite en détail et agrémentée d'exemples et a permis d'évaluer à la fois la bonne compréhension des structures arborescentes au-delà des arbres binaires et la manipulation de ces structures inductives (et des types récursifs associés) en OCaml. L'évaluation de la maîtrise de base du langage OCaml se fait via l'écriture de fonctions élémentaires de manipulation de listes, chaînes de caractères et arbres préfixes. Outre l'écriture de programmes en OCaml, une des questions (**Q21**) porte sur la compréhension d'un programme déjà écrit, avec pour objectifs d'évaluer la capacité à comprendre le code d'autrui et d'aider les candidates et candidats à aborder les questions d'implémentation plus difficiles (**Q22** et **Q23**) en proposant un schéma de manipulation des arbres préfixes réutilisable. La fin de l'exercice aborde la transformation de ces arbres préfixes en automates finis pour éliminer les redondances qui limitent l'efficacité de la structure du point de vue de son empreinte mémoire. Il s'agit d'une partie d'un plus haut niveau dont les questions nécessitent un peu de recul sur les structures manipulées.

Le troisième exercice porte sur l'intégralité des chapitres de logique au programme. L'objectif y est d'évaluer la capacité à manipuler des formules du calcul propositionnel, à bien distinguer la syntaxe et la sémantique de ces formules, à bien identifier le rôle de pont entre syntaxe et sémantique que remplit un système de preuve. La sémantique des formules y est d'abord abordée via la manipulation de table de vérités simples, puis de manière plus fine par la preuve par induction de la complétude d'un système de connecteurs. La manipulation de systèmes de preuve est évaluée par de petits arbres de preuve utilisant des règles rappelées par l'énoncé, y compris dans des variantes des règles de la déduction naturelle. La notion de correction d'un système de preuve et par extension des règles qui le composent y sont également évaluées. En fin d'exercice, des preuves plus ambitieuses sont proposées ainsi qu'une ouverture sur la notion d'équivalence (syntaxique) de systèmes de preuves, qui fait écho au caractère complet (sémantique) des systèmes de connecteurs.

2/ REMARQUES GÉNÉRALES

Le sujet aborde principalement les thèmes suivants : algorithmique, structures de données et logique dans des problèmes indépendants. La difficulté des questions est croissante dans chaque problème, avec des questions plus pointues à la fin de chacun, ce qui a laissé l'opportunité aux candidates et candidats de tous niveaux d'aborder une partie significative du sujet. Ce dernier a ainsi permis de bien classer les candidates et candidats avec une moyenne de 10,03 et un écart-type de 3,77.

Le sujet est relativement long, mais de nombreuses excellentes copies sont parvenues à en traiter une part très importante et dans de rare cas à en traiter la totalité. Les correcteurs recommandent dans ces conditions de privilégier la qualité à la quantité et de traiter proprement une partie substantielle du sujet plutôt que d'en bâcler l'intégralité. Nous en profitons pour rappeler que certaines questions en posent en fait plusieurs et qu'il ne faut pas oublier de répondre à toutes.

Avant toute chose et comme chaque année, nous tenons à appuyer et insister (oui, les deux) sur le fait que les copies rédigées sont destinées à être lues par des correcteurs et correctrices francophones et qu'à ce titre les copies doivent être rédigées dans un français intelligible. La communication par écrit avec autrui, qui est une compétence indispensable dans la vie d'une ingénieure ou d'un ingénieur, demande un effort minimal que trop de candidats ne fournissent pas. Les raturages excessifs, les questions non numérotées, les caractères spéciaux OCaml et Python difficilement identifiables, l'absence d'indentation ou la qualité de l'écriture discutable rendent certaines réponses partiellement ou totalement illisibles. Au-delà du soin à apporter à la copie, la qualité de l'expression écrite est bien trop souvent défaillante : phrases incomplètes, voire mots en dehors de toute phrase, ponctuation absente, accords ou orthographe si faux qu'ils changent le sens de la réponse, noms de fonctions et variables non signifiants dans les programmes. Nous soulignons le fait que connaître la bonne réponse à une question est inutile si l'on n'est pas capable de convaincre par écrit le correcteur qu'on la connaît bel et bien. Nous rappelons enfin que le barème prend explicitement en compte la présentation, le soin et l'orthographe et que les candidates et candidats s'exposent à perdre bien plus de points que ceux spécifiquement dédiés à ces items s'ils ne font pas l'effort de rendre leurs réponses compréhensibles.

Afin de pouvoir répondre correctement aux questions, il est également nécessaire de les lire avec attention. Il est recommandé ainsi de lire le sujet par blocs de quelques questions afin de mieux cerner le contexte dans lequel elles sont posées, de comprendre ce qui est attendu et de répondre aux questions avec un degré de précision adapté.

Les questions ”de cours” qui émaillent le sujet ont été reconnues comme telles par la grande majorité des candidates et candidats et ainsi très souvent traitées, mais parfois moins bien que ce à quoi les correcteurs pouvaient s'attendre. C'est en particulier le cas des questions de début du problème 2, ce qui montre un manque de recul des candidates et candidats quant aux structures de données à leur programme. Les correcteurs s'étonnent également du fait que plus de la moitié des copies échoue à garantir une complexité linéaire pour convertir une suite de bits en l'entier correspondant.

Le problème 3 a été très clivant : dans leur immense majorité, les candidates et candidats savent établir des tables de vérité ou dessiner des arbres de preuve mais semblent en revanche n'avoir aucune connaissance du lien entre syntaxe et sémantique qu'établit un système de preuve et ne pas savoir écrire une preuve par induction correcte. La notion de correction d'un système de preuve est au programme et la preuve de la correction de la déduction naturelle implique que la notion de correction d'une règle l'est également. Les correcteurs auraient pu comprendre que cette seconde notion soit peu familière en filière MP mais les réponses dénuées de sens apportées aux questions **36** et **37** montrent clairement que la notion de correction d'un système de preuve et celle de conséquence sémantique sont de toute façon inconnues d'une large majorité de candidates et candidats.

Dans l'ensemble, en plus des nombreuses copies excellentes, les questions proches du cours ont permis aux candidats et candidates maîtrisant le programme d'assurer une note convenable.

La maîtrise des langages au programme est variable. Certaines spécificités de Python en particulier posent problème (voir remarques spécifiques sur le problème 1). Les candidates et candidats codant de manière impérative (références, boucles...) alors que le sujet propose de manipuler des structures inductives non mutables et ceux qui se restreignant à la récursivité terminale lors de l'écriture de fonctions récursives s'exposent régulièrement à des difficultés supplémentaires lors de la conception de leur code et dans la maîtrise de la complexité de ce dernier. L'énoncé n'impose pas de style particulier pour répondre aux questions de code OCaml et tout code correct et intelligible a été accepté, mais les correcteurs soulignent néanmoins l'importance de pouvoir se détacher du style impératif lorsque les circonstances s'y prêtent, comme ici. Enfin, quel que soit le langage de programmation utilisé, les correcteurs insistent sur le fait d'utiliser des identifiants qui sont sémantiquement cohérents avec les objets qu'ils désignent (une fonction qui somme les éléments d'une liste s'appelle bien plus volontiers `somme_liste` que `aux` par exemple) et de décrire dans un français clair les idées non triviales des codes les plus délicats.

3/ REMARQUES SPÉCIFIQUES

EXERCICE 1 – MULTIPLICATION DE GRANDS ENTIERS

La maîtrise globale de la syntaxe Python est plutôt bonne mais on dénote un manque de connaissance de certaines subtilités pourtant décisives (opérateurs %, //, ** et + sur les listes). La gestion précise et correcte des détails d'implémentation nuance la bonne compréhension du principe général des algorithmes utilisés dans cet exercice. Il est attendu des candidates et candidats assez de précision dans la maîtrise des algorithmes et de la syntaxe pour obtenir des implémentations correctes et non simplement une idée générale. L'analyse de complexité a posé des problèmes importants dans de trop nombreux cas.

Q1. Si le caractère borné des entiers machine dans le cas général est connu, le cas spécifique du langage Python l'est beaucoup moins. Les bornes proposées ne sont pas toujours correctes (elles n'étaient pas demandées) et on observe quelques confusions avec les flottants (mention de la précision).

Q2. La complexité de l'opérateur ** n'est pas connue et pousse les candidates et candidats à proposer un programme qui n'a pas la complexité attendue.

Q3. La question est souvent mal comprise du fait de candidates et de candidats n'ayant pas lu l'énoncé de la suivante, ce qui les pousse à répondre avec un niveau de détail inadapté.

Q4. Si le principe est connu, l'implémentation est souvent trop compliquée et manque des subtilités, comme la gestion de la dernière retenue.

Q5. L'usage de **assert** est visiblement mal connu et de trop nombreux candidates et candidats modifient la liste passée en argument malgré la mention spécifique de l'énoncé, parfois par ignorance des mécanismes de copie des listes Python. Affecter la "valeur" dans une nouvelle variable ne suffit pas à faire une copie dans le cas de listes.

Q6. L'algorithme est connu pour la majorité des candidates et candidats mais les opérateurs % et // le sont beaucoup moins. La gestion de l'ordre des bits obtenu pose souvent problème et l'usage de l'opérateur + pour étendre une liste rend parfois la complexité bien plus importante que nécessaire.

Q7. La question a été peu traitée, mais quand ce fut le cas, elle a été plutôt correctement. On voit cependant des copies dans lesquelles la gestion de la partie non récursive de la complexité est mal comprise et manque le fait qu'on manipule les complexités à une constante multiplicative près, soit via une constante abstraite, soit via un O(). La résolution peine parfois à aboutir à une forme simple alors qu'un terme dominant clair existe.

Q8. Là aussi peu traitée. Dans certains cas une résolution de l'équation a été tentée alors que celle-ci n'était pas demandée.

Q10. Sur cette question aussi la modification de la liste passée en argument a posé problème.

Q11. C'est avant tout la gestion des cas de base qui a posé problème du point de vue du type. Il était ici indispensable d'y renvoyer des listes et ça n'a pas toujours été le cas. Le cas A n'a pas toujours été traité alors qu'il était explicitement identifié. Il y a parfois eu des confusions entre les valeurs de h et n dans de trop nombreux cas, les variables intermédiaires nécessaires pour réaliser les calculs se sont vues attribuer des noms peu significatifs rendant la compréhension du code proposé difficile. L'énoncé avait omis le cas où les soustractions intermédiaires pouvaient donner lieu à la représentation d'entiers négatifs, non prévus par l'énoncé. Certaines copies ont remarqué ce point et proposé une manière de le résoudre. Ce n'était pas attendu des candidates et candidats mais n'a pas manqué de faire bonne impression quant au recul que cela dénotait.

EXERCICE 2 – ENSEMBLES DE MOTS IMPLÉMENTÉS PAR *TRIES*

L'exercice et la représentation des objets qu'il manipule ont généralement été compris. Les résultats sont hétérogènes : certaines questions sont souvent traitées et très réussies, d'autres le sont beaucoup moins. On compte parmi ces dernières celles nécessitant un effort de rédaction (**Q21**, **Q25**, **Q28** par exemple) et celles pour lesquelles l'écriture du code en OCaml idiomatique facilite grandement voire conditionne la réussite.

Q12. De manière surprenante, cette question de cours a été mal réussie. Les tables de hachage sont régulièrement citées, mais la deuxième proposition est soit vague, soit complètement hors de propos (piles, files). Les correcteurs s'attendaient à ce que les arbres binaires de recherche soient cités, mais la mention d'une liste de couples clé-valeur rapportait également les points.

Q13. Souvent traitée et moyennement réussie. Certaines copies proposent des solutions... n'utilisant pas de dictionnaire. La réponse attendue était d'utiliser un dictionnaire dont les clés sont les mots de l'ensemble et la valeur associée est non signifiante. Cependant, la question pouvait légitimement (quoique la suite de l'énoncé aurait dû écarter cette interprétation) être comprise ainsi : on considère une collection d'ensembles et on se demande auxquels de ces ensembles appartient un mot donné. Les réponses des quelques candidates et candidats qui ont travaillé avec cette seconde interprétation ont été acceptées lorsqu'elles faisaient sens.

Q14. Généralement réussie malgré quelques erreurs d'inattention notamment sur les nœuds terminaux. Une minorité significative produit toutefois une structure qui n'est même pas un arbre, malgré la description et l'exemple fournis par l'énoncé.

Q15. Très souvent traitée, avec des degrés de réussite variés. De nombreuses copies montrent une mauvaise maîtrise du langage OCaml dès cette première question de code, en particulier tentent de modifier des listes avec :: ou utilisent :: et @ de manière interchangeable. La question n'imposait pas de style pour cette question mais les copies travaillant de manière impérative (de l'ordre de la moitié) ont régulièrement échoué à obtenir la complexité linéaire demandée en utilisant @ sans maîtriser la complexité de cet opérateur. D'autres ont tout simplement oublié de justifier la complexité du code, alors même qu'elle était bien linéaire.

Q16. Cette question demandait d'écrire une fonction ayant comme entrée un type **unit** et non de déclarer une constante. Environ un tiers des candidates et candidats a buté sur cette subtilité. Quelques-uns ont aussi confondu l'ensemble vide avec l'ensemble contenant le mot vide.

Q.17. et **Q18.** Très souvent traitées et bien réussies. De nombreuses copies présentent des solutions utilisant une fonction auxiliaire et un accumulateur plutôt qu'une fonction directement récursive. Cela alourdit le code et complexifie sa conception. C'est particulièrement le cas pour `max_list` puisque cette façon de faire ajoute la difficulté d'initialiser correctement l'accumulateur : la valeur `min_int` a été quelques fois lue et acceptée, mais pas zéro, car on n'a pas supposé dans cette question la positivité des termes. Nous rappelons que la récursivité terminale n'est pas au programme et estime que le recours systématique à cette stratégie témoigne d'un manque de maîtrise de la récursivité.

Q.19. et **Q20.** Le principe général, consistant à appliquer récursivement la fonction demandée aux enfants d'un nœud à l'aide de `list_map` a été généralement bien compris. La principale erreur rencontrée est l'absence de gestion du fait que la liste des fils comprend des couples, et qu'il faut donc ignorer les premiers éléments de ces derniers lors de l'usage de `list_map`. Écrire une fonction auxiliaire sommant les éléments d'une liste pour répondre à la question **20** rapportait une partie des points.

Q21. Diversement réussie et régulièrement mal interprétée : les candidates et candidats estiment souvent à tort qu'on ne compte que les **true** aux feuilles.

Q22. Cette question a été moins traitée que les précédentes. Il était attendu que les candidates et candidats s'inspirent du code fourni à la question précédente et notamment de l'astuce de construction d'un nœud fictif pour gérer les appels récursifs, mais il était bien sûr possible de faire autrement.

Q23. Peu traitée, mais plutôt correctement lorsque c'est le cas. La reconstruction au fur et à mesure de l'arbre a posé problème, les codes proposés montrent régulièrement que les candidates et candidats se comportent comme si les *tries* étaient des structures mutables. Cette fonction appelait l'explication de son algorithme, et, dans les trop rares cas où celle-ci était présente, elle a permis de valoriser des programmes qui n'étaient pas parfaits au plan technique.

Q24. Souvent abordée et bien traitée. Quelques problèmes de rédaction mais peu de problèmes de compréhension du phénomène, à savoir la redondance des suffixes.

Q25. Souvent abordée. Le passage du *trie* à l'automate à été compris dans les grandes lignes, mais la précision de la rédaction laisse à désirer. Les transitions de l'automate demandé ne sont pas des lettres. Ses états ne sont pas "les booléens du *trie*" mais les sommets du *trie*. Ses états ne sont pas non plus "les booléens true et false", ce qui ne ferait que deux états.

Q26. Généralement bien traitée sauf par les candidates et candidats qui pensaient qu'il fallait fusionner des états dès cette question ou ceux qui ont dessiné l'automate correspondant au *trie* de la figure 1 fournie par l'énoncé. Les correcteurs se demandent pourquoi certaines copies utilisent une autre convention que celle du sujet pour noter les états initial et acceptants.

Q27. Peu traitée, mais en général correctement. Quelques candidates et candidats semblent ne pas comprendre ce qu'est une transition sortante.

Q28. Très peu traitée. La difficulté qu'ont les candidates et candidats à s'exprimer clairement les pénalise grandement pour aborder ce type de question. Ne traiter que l'exemple était insuffisant, de même que se contenter d'une idée vague de fusion d'états : une description raisonnablement algorithmique du procédé était attendue.

EXERCICE 3 – LOGIQUE IMPLICATIONNELLE

Une distinction nette est observée entre les questions consistant à appliquer une technique (construction de table de vérité ou d'arbre de preuve) et celles nécessitant un raisonnement (en particulier la preuve par induction demandée en **Q32**), ces dernières ayant été très mal réussies et rédigées. La mauvaise maîtrise du vocabulaire et des notations étudiées dans les chapitres de logique, et en particulier la méconnaissance de la notion de conséquence sémantique, amplifie les difficultés rédactionnelles génériques des candidates et candidats.

Q29. Réussie par quasiment toutes les copies l'ayant traitée. On note toutefois que certaines utilisent les symboles $\top$ et $\bot$ pour désigner les booléens, alors que ce sont les notations au programme pour désigner des formules. Ceci est l'un des symptômes d'une mauvaise distinction entre syntaxe et sémantique, qui est un défaut récurrent dans tout cet exercice.

Q30. Réussie par quasiment toutes les copies l'ayant traitée. Peu de copies font le lien avec la question précédente (qui n'était certes pas demandé).

Q31. Réussie par quasiment toutes les copies l'ayant traitée. Certains candidats ont oublié de répondre à la deuxième partie de la question, d'autres se sont contentés d'indiquer que la formule est toujours vraie ce qui ne suffisait pas à obtenir les points, d'autres enfin ont fait montre d'inventivité quant à l'orthographe du mot "tautologie", régulièrement orthographié "totologie" voire "toutologie".

Q32. Presque aucun candidat ne sait rédiger une preuve par induction correcte. Beaucoup ne font pas la différence entre une preuve par induction sur les formules et une preuve par récurrence sur la taille des formules. On compte parmi les erreurs fréquentes : oubli de cas de base (souvent les variables propositionnelles), oubli d'un ou plusieurs cas inductifs, inclusion des formules de hauteur 1 dans les cas de base, pas de mention de comment est appliquée l'hypothèse inductive et à quoi, absence de l'annonce de ce qu'on prétend prouver (ce qui est impératif afin de communiquer efficacement avec les correcteurs). Donner les arguments centraux sans se plier au format de preuve imposé par l'énoncé n'accordait pas de point.

Q33. Cette question demandant moins de formalisme que la précédente, elle a été mieux réussie lorsqu'elle était traitée.

Q34. Les deux premiers items ont souvent été correctement traités ; le dernier en revanche a posé problème à beaucoup de candidates et candidats. Certaines copies ne mentionnent pas le nom des règles utilisées dans les arbres de preuve, ce qui a été pénalisé. Nous rappelons qu'un effort minimal de présentation est requis pour obtenir un arbre de preuve lisible et qu'il est possible de découper un arbre de preuve trop large en plusieurs morceaux en écrivant "prouvé ci-dessous" ou "prouvé ci-dessus" au bout d'une branche.

Q35. et **Q36.** Les notions de règle correcte ou de système de déduction correct ne sont pas connues ce qui n'est pas étonnant puisque la notion de conséquence sémantique est presque systématiquement inconnue ou très mal restituée. Si Γ est un ensemble de formules et F une formule, on trouve dans les copies à la place de l'assertion "$\Gamma = F$" les affirmations suivantes : "$\Gamma \to F$" ou "$\Gamma \to F$ est une tautologie". La rédaction de ces questions étant impossible sans maîtriser la définition d'une conséquence sémantique et les notations associées, ces dernières ont logiquement été très mal réussies par les rares candidates et candidats les ayant abordées.

Q37. Les candidates et candidats ayant abordé cette question ont souvent indiqué qu'une des règles dans le prétendu arbre était mal utilisée, ce qui dénote une mauvaise compréhension de la question. Celle-ci imposait en effet de répondre via un argument sémantique, donc en exhibant une valuation montrant que $A \to C$ n'est pas conséquence sémantique de Γ et en utilisant (par exemple par l'absurde) la correction de la déduction naturelle établie à la question précédente.

Q38. Q39. et **Q40.** Très peu traitées, ce qui s'explique sans doute au moins en partie par la longueur du sujet. Des arbres de preuve partiels ont été valorisés. On souligne qu'il n'est pas légitime d'utiliser l'équivalence sémantique établie en **Q30** pour faciliter la construction de l'arbre de preuve de la **Q40**, puisqu'un tel arbre est un objet syntaxique. Peu de candidates et candidats ont fait cette erreur toutefois.

Q41. Presque jamais traitée. La question était abordable en admettant l'existence des arbres de preuve demandés aux questions précédentes.