S Informatique 1 MPI

Préliminaire : il s'agit d'une épreuve pratique de C. Comme il s'agit de code sur papier, il y a eu une indulgence relative concernant un « ; » ou une « } » manquant, mais par contre aucune indulgence si le code n'était pas du code C valide (comme par exemple mais non exclusivement : utilisation de () à la place de [] pour l'accès à un tableau, utilisation de « <- » comme opérateur d'affectation, appel de méthodes sur les structures, utilisation de constructions d'autres langages (`double_to_int()` notamment, passage de `void` en tant que paramètre, utilisation de `None` à la place de `NULL`, définition d'une nouvelle fonction dans le corps d'une autre fonction...).

Q1 - Question moyennement bien traitée. Il fallait évoquer, d'une manière ou d'une autre, le but poursuivi par les 3 lignes ensemble, à savoir la garde d'inclusion pour éviter des déclarations multiples. La simple explication du fonctionnement de chacune des lignes séparément n'était pas suffisante.

Q2 - Question généralement bien traitée. Confusion fréquente entre`"network.h"` et `<network.h>`. Par contre, l'inclusion du fichier .c ou l'utilisation de constructions inexistantes (`import, open`) n'est pas correcte.

Q3 - Question généralement bien traitée. Il fallait utiliser le mécanisme de précondition `assert` plutôt qu'un test `if`, et bien affecter n (n+1 ou n+2 tolérés si utilisation cohérente par la suite). La boucle d'initialisation du tableau était nécessaire, sauf si en cas de référence explicite au comportement standard du C (variable globale directement initialisée à 0, section 6.7.8 item 10 du standard C99). La portée des variables semble encore mal maîtrisée, beaucoup de copies comportant des `mallocs` inutiles.

Q4 - Question très bien traitée dans l'ensemble. Toutes les incrémentations (++, +=, +1, passage par variable temporaire) ont été acceptées.

Q5 - Question très bien traitée. Comme à la question 3, il fallait utiliser `assert` comme indiqué (un test if ne suffisait pas). Toutes les décrémentations (–, -=, -1, passage par variable temporaire) ont été acceptées. Quelques erreurs sur les conditions de l'`assert` (parfois inversées).

Q6 - Question très majoritairement bien traitée. Le point important était l'utilisation d'une boucle for et notamment le calcul correct de la borne supérieure de l'itération, en relation avec la question 3. T est une variable de type pointeur mais il ne fallait surtout pas allouer de la mémoire et modifier l'adresse contenue dans la variable T.

Q7 - Question qui a été souvent la plus difficile et, vu la quantité de code écrit par certains, la plus chronophage. Parmi les erreurs vues, on notera, sans ordre précis : l'écriture d'un parcours en DFS plutôt qu'en BFS, l'écriture d'un parcours ne faisant absolument rien, les boucles infinies, les `mallocs` sans `free` (particulièrement dans des fonctions récursives), et les parcours supposant l'existence d'une librairie de file prête à l'emploi. On attendait ici un code C fonctionnel le plus simple possible (utilisation d'un tableau pour gérer une file), écrire un texte explicatif de l'algorithme sans aucune ligne de code ou en pseudo-code était donc hors sujet.

Q8 - Question globalement bien traitée. Outre l'appel aux deux fonctions `remove` et `add`, on attendait ici un parcours « intelligent » de l'arborescence, en partant du puits, et non pas une simple itération sur les indexes. Attention à ne pas utiliser les variables s et t si elles ne sont pas définies par ailleurs.

Q9 - Certainement l'une des questions les plus mal traitées du sujet. On attendait, outre le fait de passer par récurrence, l'idée de transformation de chemin, prenant en compte le fait que la suppression d'un arc créée par définition un arc inverse, ce qui fait qu'il est possible de transformer 2 chemins ayant des arcs transformés en 2 autres chemins sans arcs transformés. L'utilisation de dessins pour expliquer l'idée a été valorisée.

Q10 - Question globalement bien traitée. ll fallait utiliser les 2 fonctions `bfs_tree()` et `residue()`. Tous les tests de boucles (appel direct à `bfs_tree`, passage par une variable locale) et toutes les boucles

(`while, do/while` avec un appel à `bfs_tree` avant) ont été acceptés.

Q11 - Question plutôt bien traitée, quand elle a été abordée. On attendait une bonne définition d'un variant de boucle, et l'utilisation soit du degré sortant de s soit du degré entrant de t.

Q12 - Question globalement bien traitée, quand elle a été abordée. Il fallait évoquer le fait qu'un chemin passait obligatoirement par un arc de l'entourage pour relier s à t, d'où l'inégalité.

Q13 - Question globalement bien traitée. On attendait une démonstration par l'absurde.

Q14 - idem que la question 13.

Q15 - Question globalement bien traitée. On attendait des références aux questions précédentes.

Q16 - Question finalement peu traitée. On attendait des références aux questions suivantes et l'écriture des inégalités résultantes. Souvent, seuls les calculs ont été faits, mais pas l'établissement de la conclusion.

Q17 - Question globalement mal traitée. On attendait 2 allocations, une pour le graphe et une pour le tableau de voisins, et une boucle d'initialisation à `NULL` du tableau de voisins. Le graphe a souvent été alloué statiquement, en variable locale, et soit retourné tel quel (pas de pointeur), soit par adresse. Pas mal d'erreurs aussi sur l'initialisation du tableau, vu parfois comme un tableau à 1 seule case, ou parfois comme un tableau de tableau. La boucle d'initialisation est celle qui a posé le plus de problème, beaucoup initialisant chaque case du tableau avec un `malloc` et remplissant ensuite la structure avec un peu n'importe quoi (0, -1, null).

Q18 - Question globalement mal traitée. Il fallait déjà penser à ajouter l'arc dans les 2 sens (u->v et v->u), le graphe étant non orienté. Ensuite, on attendait l'allocation dynamique de la structure (et non pas une variable locale passée statiquement), et la mise à jour de tous les liens nécessaires, en fonction de l'endroit d'insertion. Toutes les insertions (en début de liste, en fin de liste avec parcours jusqu'à la fin, ou même en deuxième place) ont été acceptées. Il fallait aussi penser à vérifier que les pointeurs n'étaient pas `null`, notamment lors du parcours de la liste ou de la recherche du précédent ou du suivant. Enfin, un certain nombre ont confondu liste doublement chaînée et liste doublement chaînée circulaire, et ont rajouté des contraintes sur le bouclage de la liste, ce qui posait des problèmes étant donné que la première case de la liste n'était pas utilisée. Beaucoup d'erreurs possibles, qui ont fait que cette question a rarement été bien traitée.

Q19 - Question simple, qui a pourtant été parfois mal traitée. Il fallait faire un parcours des voisins en utilisant la structure de liste chaînée (-> suivant). Toutes les possibilités (boucle for, boucle while, variables temporaires) ont été acceptées. L'erreur la plus fréquente a été de faire une itération sur les indices du tableau « *neighbours* », montrant ainsi une incompréhension totale de la structure de liste chaînée.

Q20 - Question globalement bien traitée. La subtilité était de penser à diviser le résultat par deux pour prendre en compte le lemme des poignées de main. Attention à bien utiliser l'opérateur de division du langage C, à savoir « / ».

Q21 - Question moyennement bien traitée. La subtilité était non seulement de penser à ne pas faire une division entière, mais surtout à savoir forcer une division en nombres flottants en C. Toutes les solutions valides en C (mise de chacun des 2 termes ou même d'un seul dans une variable de type float ou double avant la division, ou faire un cast en float ou double à la volée d'au moins un des termes de la division) ont été acceptées. Mettre le résultat après la division dans une variable de type float ou double ne fonctionne pas.

Q22 - Question moyennement bien traitée. Il y avait principalement 3 difficultés : penser à appeler la fonction `nw_init()` pour initialiser la structure H, bien découper en 3 parties suivant les 3 critères donnés, et faire un parcours intelligent des voisins et non pas une itération sur les indexes (comme pour la question 19). Ont été acceptées autant les affectations regroupées dans une seule boucle unifiée que les affectations dans des boucles séparées. L'utilisation de variables temporaires pour éviter de refaire certains calculs n'a pas été prise en compte. Par contre, l'utilisation des fonctions `nw_add` et `nw_delete`

avec des boucles en $m*n*n$ au lieu d'affectations simples dans des boucles a été sanctionnée, l'explosion en complexité étant évidente.

Q23 - Question globalement bien traitée, lorsqu'elle a été abordée. On attendait l'indication des 2 ensembles à dénombrer.

Q24 - Question rarement traitée. On attendait l'indication des 3 ensembles à traiter pour la première égalité. Pour la seconde égalité, le dénombrement correct de l'ensemble, et non pas simplement la réutilisation de la première égalité en mettant $|X|$ à 0 a été valorisé. Lorsque cette question a été traitée, souvent seule la seconde inégalité a été traitée.

Q25 - Question rarement traitée. Pas de difficulté particulière, correctement traitée lorsqu'elle a été abordée.

Q26 - Question rarement traitée. Il fallait penser à appeler les fonctions `reduce()` et `disconnect()`. Presque toutes les réponses ont recalculé la multiplicité de la bordure pour la comparer à mn3, peu ont pensé à tester simplement si la source était isolée.

Q27 - Question globalement bien traitée. Souvent, les questions précédentes ont été esquivées et celle-ci traitée, vu sa simplicité. Aucune difficulté.

Q28 - Question rarement traitée. Il fallait penser à utiliser la question précédente et à bien exprimer $\delta(GX)$ en une fraction de la forme m/n. Lorsque cette question a été abordée, elle a été correctement traitée.

Q29 - Question très rarement traitée (de l'ordre d'une dizaine de copies), souvent assez superficiellement.

Q30 - Question globalement bien traitée. Souvent, les questions précédentes ont été esquivées et celle-ci traitée, vu qu'il s'agit de la même question, sous une autre forme, que la question 2. Les mêmes remarques s'appliquent.

Q31 - Question très rarement traitée, et souvent mal traitée ou très superficiellement, certainement par manque de temps. Le point important était le passage au maximum.

Q32 - Question très rarement traitée, et souvent mal traitée ou très superficiellement, certainement par manque de temps. Le point important était aussi le passage au maximum. Le fait d'indiquer le facteur d'approximation sans calcul ne constitue pas une réponse valable.

Q33 - Question très rarement traitée (de l'ordre de quelques copies), et presque jamais correctement, car beaucoup trop prospective.

↑RETOUR

T Informatique 2 MPI

Q1 - La première liste a été correctement représentée par la plupart des candidats, même si la qualité de la représentation des pointeurs était variable. De nombreuses copies n'ont en revanche pas réussi à représenter la deuxième liste. En effet, il ne s'agissait pas de représenter une infinité de maillons, mais bien un unique maillon pointant vers lui-même.

Q2 - Les correcteurs sont surpris que les candidats ne sachent pas tous traduire un message d'erreur très courant. Si le problème de la terminaison a souvent bien été repéré, les copies qui décrivent précisément le déroulé sont plus rares.

Q3 - Cette question n'a pas posé de problème particulier. Certains candidats ont critiqué l'ordre dans la commande proposée, alors que la commande était correcte.

Q4 - La question a été très bien réussie. De nombreux candidats se sont appuyés sur l'exemple fourni.

Q5 - De nombreuses copies n'ont pas traité le cas où $a \geq b$, ce qui donnait un flot vide. Cela posait des problèmes pour la terminaison de la fonction. Certaines copies traitaient $a = b$, mais pas $a > b$. Il n'était pas supposé que $a \leq b$. Il n'était pas nécessaire d'utiliser une fonction auxiliaire.

Q6 - Cette question a été très rarement réussie. Les correcteurs ont lu quelques réponses élégantes exploitant l'indécidabilité du problème de l'arrêt. Un grand nombre de copies a essayé de justifier leur réponse en évoquant des "chaînes aléatoires", ce qui avait peu de sens, où encore en s'appuyant sur les décimales de π, alors qu'il est tout à fait possible d'écrire une fonction déterminant la n-ième décimale de π et ainsi de construire un flot pour les décimales de π.

Q7 - Un grand nombre de candidats ignore que `failwith s` lève une exception `Failure s`.

Q8 - Il ne fallait pas oublier de faire un appel à la fonction donnant la suite du flot.

Q9 - Certains candidats ont essayé de modifier un flot, alors qu'ils étaient immuables. D'autres candidats ont utilisé un accumulateur et inséré les éléments de la liste dans cet accumulateur : cela posait un problème pour l'ordre des termes, mais également un problème de terminaison pour les listes définies comme `list_ones`.

Q10 - Il ne s'agissait pas de modifier le flot ou d'en renvoyer un nouveau. Un grand nombre de candidats a appliqué la fonction sur un élément de trop en traitant mal le cas $t = 0$. Il ne fallait pas lever d'erreurs si le flot était trop court.

Q11 - Le cas où le flot est vide a été parfois oublié. Il n'était pas nécessaire de faire une fonction auxiliaire.

Q12 - Les réponses à cette question comportaient souvent des erreurs. Dans l'annexe la fonction `Array.exists` était rappelée et n'a été que trop peu utilisée. Un tableau créé avec `Array.make` doit être initialisé avec un élément du bon type. Certains candidats ont utilisé un rattrapage d'exception : c'était une bonne idée à condition de connaître la bonne syntaxe.

Q13 - La réponse à cette question tenait sur trois lignes. De nombreux candidats ont écrit de très longues fonctions avec quelques éléments manquants.

Q14 - De nombreux candidats ont identifié le problème. Néanmoins, les modifications proposées ne convenaient pas toujours. En particulier, il ne suffisait pas d'échanger les arguments. Il s'agissait de rendre les appels récursifs paresseux.

Q15 - La question ne posait pas de problèmes particuliers. Néanmoins, un grand nombre de candidats pense que `return 0` n'est là que parce que la fonction doit renvoyer un entier et ne semble pas avoir été familiarisé avec la notion de code de retour ou de code d'erreur.

Q16 - La plupart des candidats ont identifié qu'il suffisait d'énoncer que le problème de l'arrêt est indécidable. Les correcteurs ont été surpris par le grand nombre de candidats qui ont essayé d'écrire une réponse à propos des machines de Turing qui ne sont pas au programme de MPI/MP2I.

Q17 - La question était difficile. Certains candidats ont proposé une fonction dont la terminaison n'était pas assurée : cela ne rapportait aucun point. Les meilleures copies ont utilisé la question précédente et réalisé une réduction. Il fallait être précis dans la réduction et ne pas confondre entiers et booléens.

Q18 - De nombreux candidats ont tenté de convertir en binaire l'entier alors que ce n'était pas nécessaire. Le traitement du cas $n = 0$ a souvent été négligé, menant à des problèmes de terminaison des fonctions. Si `a lsl 1` donne bien la même chose que `a*2`, `a lsl (-1)` donne `0`. Il faut parfois s'en tenir aux outils que l'on maitrise.

Q19 - De nombreuses copies présentent les bonnes complexités, mais ne les comparent pas. Il y avait une nuance entre s le nombre d'éléments et t le nombre de termes explorés. Quelques copies n'ont pas bien lu la question et donnent les complexités temporelles et non spatiales.

Q20 - La principale difficulté de la question était la manipulation des flottants en OCaml.

Q21 - Certains candidats ont cherché le terme à modifier avec une boucle `for` alors que l'indice se calculait à l'aide de `mod` : ils ont été pénalisés. Il faut toujours écrire un code le plus efficace possible.

Q22 - L'annexe du sujet donnait la syntaxe pour `lsl`. Il fallait l'utiliser, les fonctions de calculs de puissance étant bien moins efficaces. Les correcteurs ont rencontré de nombreuses erreurs sur la manipulation des flottants.

Q23 - Les réponses doivent être justifiées, il ne suffit pas de donner les lignes de codes concernées. Les candidats ont, dans la grande majorité, repéré qu'il fallait utiliser un verrou (mutex).

Q24 - Les arbres de preuves étaient souvent bien présentés avec un bon espace de sécurité en haut. Les correcteurs rappellent aux candidats qu'ils disposent de brouillon. S'il est possible de présenter un arbre de preuve en plusieurs arbres proprement, les candidats ayant omis une partie de l'arbre avec un « idem » ont été pénalisés. Certains candidats ont essayé de simplifier la formule avant de faire l'arbre de preuve, ces candidats n'ont pas eu de points.

Q25 - Certains candidats ont cherché à simplifier les formules ou utiliser des règles de déduction naturelle inspirées des lois de Morgan : ces candidats ont été pénalisés. Les correcteurs rappellent que $\neg\bot \neq \top$ et que $\neg(a \vee b) \neq (\neg a \wedge \neg b)$

Q26 - La question pouvait être abordée de plusieurs façons. Les meilleures copies ont utilisé la correction de la déduction naturelle, qu'il fallait mentionner pour utiliser les résultats admis ou les questions précédentes. Certains candidats ont proposé des tables de vérité : celles-ci ont été acceptées dès lors qu'elles étaient suffisamment précises.

Q27 - La question était difficile. `^` n'est pas un pas opérateur de *ou exclusif*. Il convenait d'écrire une fonction pour cela. Les booléens ne sauraient être confondus avec les entiers. Peu de candidats ont écrit un code organisé et propre. Il fallait faire preuve de rigueur dans les indices.

Q28 - Un grand nombre de candidats a écrit une fonction d'évaluation correcte. La négation logique n'est pas `!`, on utilise `not`.

Q29 - Les candidats ayant bien compris que $\oplus$ était un *ou exclusif* ont bien réussi cette question. Il s'agissait de bien justifier.

Q30 - Le mot vide s'écrit ε conformément au programme de MPI/MP2I.

Q31 - Si un grand nombre de candidats a identifié qu'il fallait prouver que la grammaire était non ambiguë, seuls quelques rares candidats ont réalisé une preuve correcte. En particulier, il fallait prouver que la décomposition $w = uavb$ d'un mot engendré, avec u et v des mots engendrés, était unique.

Q32 - Les candidats ont généralement donné la bonne réponse pour cette question, il ne fallait pas oublier de justifier.

Q33 - Il ne suffisait pas de donner les grandes lignes d'une récurrence. Les correcteurs ont été attentifs à la rigueur et aux détails dans les raisonnements exposés. Il fallait opérer une distinction de cas et détailler l'hérédité avec une étude précise des opérations réalisées dans la boucle.

Q34 - La question a été très peu traitée par les candidats. Il s'agissait d'une question difficile dans laquelle il fallait faire preuve de rigueur.

Q35 - Cette question ne présentait pas de difficultés et de nombreux candidats n'ont fait que cette question dans la quatrième section.

Q36 - La dernière question était difficile. Les correcteurs ont lu quelques rares réponses correctes dans les meilleures copies. Décrire ses intentions sans écrire de code ne rapportait pas de points.

↑RETOUR