Composition d'Informatique C (XULSR), Filière MPI

1 Méthode de correction et barème

Chaque question rapporte un certain nombre de *points* et est notée avec un *score* entre 0 et 5. Le nombre de points total est de 62, et un coefficient multiplicateur de 1,2 est utilisé pour obtenir la note finale, en tronquant à 20. En d'autres termes, 52 points étaient suffisants pour obtenir 20/20, et la note finale sur 20 d'un·e candidat·e est alors obtenue en appliquant la formule

$$\text{note(copie)} = \max\left(20,\ 1,2 \times \frac{20}{62} \times \sum_{\text{question } q} \text{points}(q) \times \frac{\text{score(copie}, q)}{5}\right).$$

La moyenne des 397 candidat·e·s est de 9,25/20 avec un écart type de 3,71.

2 Commentaire détaillé

Pour chaque question, sont indiqués entre crochets : le pourcentage de copies ayant répondu à la question, le nombre de points de la question, et la moyenne des scores (entre 0 et 5) des copies ayant traité la question. Les pénalités indiquées, qui portent sur le score, sont données de façon indicative à titre d'exemple et reflètent les erreurs les plus communes. Une pénalité de -2 points est appliquée pour les questions de code lorsque le·la candidat·e n'utilise pas le bon langage de programmation.

3 Remarques générales

De manière générale, si une question demandait un algorithme avec une complexité précise, il était important de justifier cette complexité pour avoir tous les points. Nous tenons aussi à attirer l'attention sur la qualité de la rédaction : il est nécessaire d'écrire proprement, de manière lisible, alignée. Quelques justifications sont toujours nécessaires, il faut éviter les mots "triviale" "évident" qui n'apportent aucune information à la correctrice ou au correcteur. À l'inverse, il faut aussi savoir être concis quand c'est nécessaire. Les questions nécessitant plus d'une page de justification sont rares voire inexistantes. Insistons : les candidates et les candidats devaient trouver le bon niveau de rédaction : il faut assez de détails pour convaincre la correctrice.teur, mais trop de détails vous fait perdre du temps et montre un manque de recul. Une bonne rédaction doit identifier les points clefs du raisonnement et les mentionner clairement. Ces points étaient particulièrement importants pour les questions II.18 à II.21, dont nous donnons des exemples de solutions plus loin.

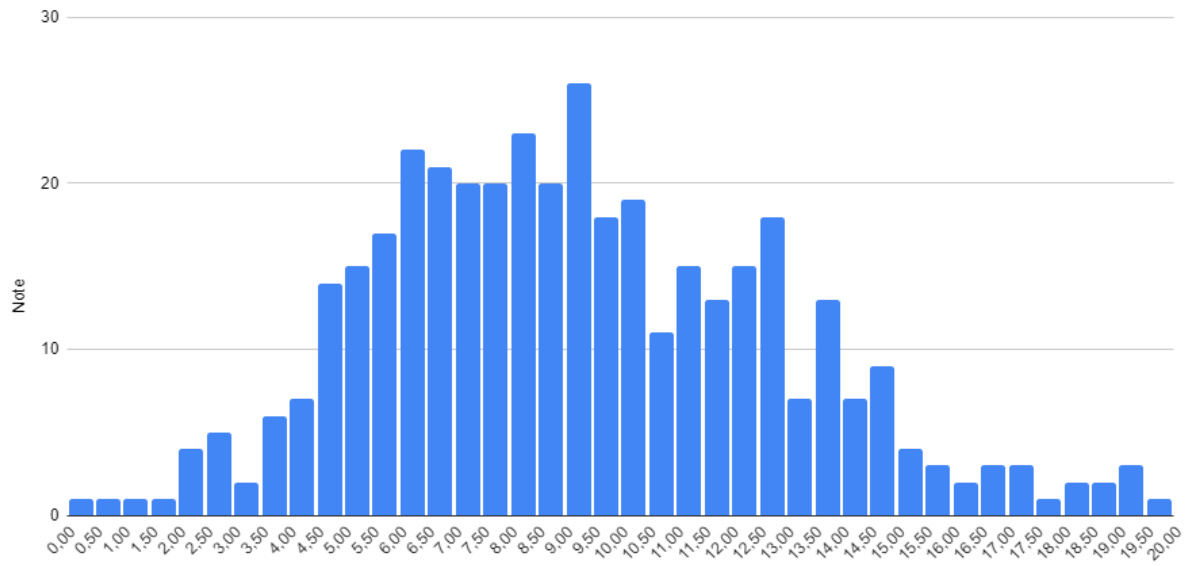

FIGURE 1 – Histogramme des notes

4 Commentaires détaillé

4.1 Partie I

De manière générale, trop de copies ont essayé de recalculer n dans certaines fonctions, quand c'est un paramètre global qui pouvait être directement utilisé.

4.1.1 Question I.1, [99%, 0,5, 3.9]

Question globalement bien traitée. Beaucoup de personnes ont oublié d'utiliser le fait que $X_1 = \top$ pour simplifier l'expression (-1/5), et quelques personnes ont mis deux fois certaines expressions (-1/5). Il n'était pas nécessaire de préciser les étapes intermédiaires qui menaient au résultat.

4.1.2 Question I.2, [96%, 1, 4.7]

Question globalement bien traitée. Il ne fallait pas oublier de justifier (succinctement) pourquoi la transformation est linéaire.

4.1.3 Question I.3, [100%, 1, 4.6]

Question globalement bien traitée. Quelques oublis sur les bornes (en particulier, oubli de renvoyer une exception quand l'entier est ≤ 0). Nous avons vu plusieurs tentatives de trouver une formule explicite pour transformer un littéral en sa variable, par exemple en renvoyant $i[n]$, ce qui ne fonctionne pas pour les littéraux n et $2n$. On conseille d'utiliser les formules plus directes et qui marchent ($i - n$ si $i > n$) plutôt que de passer du temps à trouver une formule générale qui pourrait sembler être plus astucieuse mais risque d'être fausse.

4.1.4 Question I.4, [99%, 0,5, 4.7]

Question globalement bien traitée. 2,5 points par graphe, perte d'un point par arête en trop ou manquante. Quelques oublis de mettre les sommets 1 et 6.

Plusieurs personnes se sont contentées de donner la liste des arêtes au lieu de représenter le graphe, ce qu'on déconseille fortement.

4.1.5 Question I.5, [100%, 1,5, 4.5]

Question globalement bien traitée. Plusieurs copies/personnes ont créé un graphe de taille n au lieu de $2n$, ou oublié une des deux arêtes à créer pour chaque clause.

4.1.6 Question I.6, [99%, 1, 4.4]

Question globalement bien traitée. Il fallait penser à justifier la complexité, et expliciter la complexité de la création du graphe, même si elle est moins importante que celle de l'ajout des arêtes.

4.1.7 Question I.7, [92%, 2, 4]

Beaucoup d'erreurs pour un algorithme de cours. Parmi les erreurs les plus fréquentes :

— Un algorithme trop compliqué et trop long sans explication ni commentaire.
— Le nombre de composantes renvoyé est faux de 1.
— Complexité trop élevée. En particulier, faire une boucle linéaire à chaque fois qu'on cherche un nouveau sommet non exploré.

4.1.8 Question I.8, [93%, 2, 3.6]

Question globalement bien traitée. Certaines personnes ont considéré que même composante connexe veut dire que les sommets sont forcément voisins.

En général, beaucoup d'énergie a été dépensée sur cette question. Il n'était pas nécessaire de prouver la transitivité de l'équivalence. Il n'était pas non plus utile de détailler une preuve pas récurrence sur la longueur du chemin dans le graphe. Globalement, expliquer qu'être voisin impliquait qu'on était équivalent et que l'équivalence est transitive ce qui suffisait à marquer les points. Un point (sur cinq) était consacré au fait de traiter le cas où la formule n'était pas satisfiable.

4.1.9 Question I.9, [92%, 2, 3.6]

3 points pour montrer que C' est connexe, 2 points pour montrer qu'elle est maximale. Il y a eu beaucoup d'oublis sur le second cas.

Quelques copies ont voulu utiliser la réciproque de la question précédente, sans la justifier. Elles se sont contentées de montrer que la négation des littéraux sont équivalents, et ont invoqué la question 9 pour justifier q u'ils s ont connectés.

4.1.10 Question I.10, [89%, 3.5, 2.5]

1 point était consacré à bien montrer la réciproque. Beaucoup de personnes ont annoncé qu'une contradiction impliquait forcément l'existence de `i` et `neg_lit(i)` dans la même composante sans aucune réelle preuve ou des arguments fallacieux. La preuve la plus simple était d'expliquer l'existence d'une valuation si aucun i tel que `i` et `neg_lit(i)` sont dans la même composante. On a vu très peu de preuves qui fonctionnaient avec une autre approche.

4.1.11 Question I.11, [90%, 1, 4.7]

Question globalement bien traitée. Il suffisait d'utiliser la propriété précédente. Beaucoup de personne ont arrêté de traiter ce problème à la question précédente, alors qu'en lisant bien le sujet, il était facile de récupérer les points de celle-ci.

4.1.12 Question I.12, [67%, 4, 2.6]

Question difficile. Trop de personnes qui ont pensé à comment créer la valuation dans la question 10 n'ont pas su traiter cette question qui était la mise en pratique. On a eu quelques complexités fausses (en particulier, en présupposant que p est une borne supérieure au nombre de composantes connexes, ou que $p \leq n$).

Quelques unes des solutions proposées se contentaient en pratique de mettre "vrai" à toutes les variables.

Un point était attribué à la justification de la complexité quand l'algorithme était bon.

4.2 Partie II

4.2.1 Question II.13, [85%, 1, 4.6]

Question globalement bien traitée. Quelques personnes se sont arrêtées avant l'application de la distributivité.

4.2.2 Question II.14, [73%, 1, 4.3]

Question globalement bien traitée. Il y avait plusieurs solutions qui fonctionnaient. On pouvait penser également à regrouper vos formules dans une variable pour vous économiser le besoin de tout recopier à chaque ligne.

4.2.3 Question II.15, [85%, 1, 3.5]

Il y a eu une grande majorité d'oublis du cas où `e2.lr1` ou `e2.lr2` est égal à $n+1$ pour renvoyer l'exception. Certaines personnes ont eu la bonne idée d'appliquer l'algorithme de la Partie I sur la 1-contrainte à renvoyer, pour vérifier si elle n'est pas une contradiction avant de la renvoyer. Cela permettait de manière élégante de ne pas avoir à réfléchir exhaustivement à toutes les options de contradiction possibles.

4.2.4 Question II.16, [74%, 1.5, 3.9]

2 points pour l'algorithme qui a été plutôt bien réussi. Beaucoup d'oublis pour prouver l'équivalence qui était sur 3 points. En particulier, il fallait penser à bien préciser qu'ajouter la 1-contrainte $(i,\ 1)$ (respectivement $(i,\ n+1)$) revenait à forcer l'affectation de i à 1 (respectivement 0).

4.2.5 Question II.17, [41%, 1.5, 3]

La plupart des réponses ont montré que la taille de `l2` décroissait, mais une partie non négligeable ne s'est pas rendue compte que ce n'était pas strictement (cela rapportait 1/5). Il fallait bien remarquer que l'assignation faisait deux tentatives qui réduisaient le nombre de variables non assignées. Certaines personnes ont remarqué que, dans ce cas, la taille de `l2` allait décroitre de 1 à l'appel suivant, et dans ce cas il fallait préciser que c'était avec les équivalences (1) et (2) pour avoir tous les points.

4.2.6 Question II.18 à II.21, [34%, 1.5, 4.1], [28%, 1.5, 4.1], [22%, 1.5, 3.6], [20%, 1.5, 3.9]

Cette suite de question a été trop peu traitée, alors qu'il suffisait de comprendre un programme donné. Il fallait ne pas oublier de traiter le cas où une valuation existe ET le cas où on a une contradiction pour avoir tous les points. Pour la question II.19, il fallait penser à rappeler la preuve d'équivalence de la Question 16. Pour II.20, il fallait utiliser la correction de l'algorithme `contr2_simplify_valp`. Pour II.21, il fallait bien préciser qu'une valuation existe si et seulement si il existe une valuation avec `xlr1=0` ou une valuation avec `xlr1=1`.

Voici un exemple de rédaction pour la question 19 qui permettait d'obtenir la totalité des points :

Si la liste $l2$ est vide, alors une valuation satisfait $F_2 \wedge F_1$ si elle satisfait F_1. Le programme `eform2_sat_aux` *appelle* `eform1_of_valp` *pour obtenir une liste l'_1 représentant une formule F'_1 à partir des assignations de vp. La propriété démontrée pour* `eform1_of_valp` *implique que toute valuation (totale) est compatible avec vp. Puis, on appelle* `eform1_sat ($l_1'@l_1$)` *qui doit renvoyer une valuation v tel que $v \models F1 \wedge F'1$ si elle existe, et lever l'exception* `Exn_unsat` *sinon. Donc si une telle v existe, alors $v \models F1$ et $v(F'_1) = 1$, donc v est compatible avec vp.*

4.2.7 Question II.22, [19%, 4, 1.4]

Il fallait remarquer que le cas de la ligne 19 impliquait un embranchement de 2 possibilités, et que du coup la complexité avec une puissance de 2 en facteur. Un point sur 5 a été donné aux copies qui connaient une complexité de 2^n, et 2 points sur 5 pour celles qui ont vu que ça dépendait de la taille de $l2$, i.e. un facteur 2^k. Il fallait ensuite avoir un bon découpage et en particulier le facteur n^2 pour avoir les autres points.

4.3 Partie III

Certaines personnes ont manipulé la liste chaînée comme si c'était un tableau, en présupposant que soit sa taille était donnée, soit que la première case non utilisée contenait `NULL`. On a pénalisé légèrement cette erreur en enlevant des points une seule fois.

4.3.1 Question III.23, [85%, 1, 3.4]

Quelques personnes n'ont pas mis d'élément suivant dans le type, qui était implicitement attendu du fait qu'on construisait des listes chaînées. On a également pénalisé d'un point l'oubli de mettre `typedef`, et d'un autre point si le type était `int` pour les variables, alors qu'on attendait des entiers non signés.

4.3.2 Question III.24, [84%, 0.5, 4.2]

Beaucoup de personnes ont pensé à faire un algorithme récursif. Malheureusement, la plupart des solutions proposées dans ce cas traitaient le cas `NULL` pour les listes non vides, et du coup affichaient `"true"` dans tous les cas. Il y a eu quelques oublis de mettre un saut à la ligne à la fin de l'affichage de chaque clause.

4.3.3 Question III.25, [74%, 1.5, 4.8]

Question globalement bien traitée. RAS.

4.3.4 Question III.26, [68%, 2, 4.1]

L'algorithme (noté 2/5) a généralement été bien écrit. Il manquait souvent la preuve attendue. Dans la preuve, il manquait généralement l'argument que si on renvoyait vrai, il existait une assignation satisfaisant les restrictions de différence en donnant des valeurs différentes à chaque classe.

4.3.5 Question III.27, [59%, 1.5, 3.5]

Il y a eu divers oublis :
— coût de la création de l'union find en $O(n)$.
— Pour `eform_to_uf`, même si on ne fait aucune action sur les contraintes de différence, la boucle de parcours est en $O(e \log n + d)$ (pareil pour la boucle de `eform3_sat_int` en $O(e + d \log n)$).
— certaines personnes ont pensé que $e + d = n$.

4.3.6 Question III.28, [52%, 1.5, 3.3]

Beaucoup de personnes ont utilisé `uf_find` qui du coup ne donnait pas la bonne complexité. Certaines personnes ont pensé à adapter l'utilisation de `uf_find` pour ne visiter les sommets qu'une fois avant de trouver le représentant, s'assurant ainsi de garder une complexité linéaire.

Il y a eu beaucoup de solutions fausses avec la bonne complexité où on ne comptait pas la bonne chose. La plus fréquente était de compter combien d'éléments sont parents d'au moins un sommet, qui est en général strictement plus grand que le nombre de classes d'équivalences.

4.3.7 Question III.29, [40%, 1, 4.5]

Algorithme en général bien traité. L'erreur la plus fréquente a été de donner comme valeur aux éléments d'une classe l'indice du représentant (cette solution ne donne en général pas des valeurs dans $[1, k]$. Un point était attribué à la justification d e l a complexité.

4.3.8 Question III.30, [31%, 1, 4.5]

Question globalement bien traitée. Il fallait bien penser à traiter les deux cas (satisfiable et non satisfiable). Des personnes se sont contentées de pointer vers leur preuve de la question 26, ce qui était suffisant si celle-ci était bonne.

4.3.9 Question III.31, [30%, 1.5, 4.3]

Question plutôt facile mais qui a été peu traitée. Elle aurait probablement été bien plus traitée si elle était apparue plus tôt. Lire le sujet en intégralité permettait d'identifier cette question comme faisable sans avoir à comprendre et traiter les autres questions.

Il fallait bien penser à justifier que la valeur était minimale (i.e. il n'était pas possible de faire mieux) et justifier qu'il y a toujours une solution de taille 2 pour le cas d'une seule différence.

4.3.10 Question III.32, [19%, 2.5, 3.2]

Il fallait penser à parler de bipartition ou de 2-coloration, et rappeler que c'est linéaire. Plusieurs personnes ont voulu réutiliser l'algorithme de la partie I, ce qui ne correspondait pas aux réponses attendues, étant donné que ce n'est pas un algorithme de graphes. Plusieurs personnes ont voulu chercher une bipartition où les sommets correspondent aux littéraux et en ne mettant que les contraintes de différence. C ela n e fonctionnait p as, c ar i l faut s'assurer que

deux éléments dans une même classe d'équivalence aient la même valeur. Un moyen de pallier à cela était d'ajouter, pour chaque contrainte $X_i = X_j$, un sommet supplémentaire, et deux arêtes qui relient ce nouveau sommet aux sommets i et j.

4.4 Partie IV

Mise à part la première question où beaucoup de copies sont peut-être venues chercher des points, cette partie a été très peu traitée.

4.4.1 Question IV.33, [35%, 0.5, 4.9]

Question globalement bien traitée. Il suffisait de comprendre et appliquer les définitions.

4.4.2 Question IV.34, [14%, 1.5, 3.9]

Question dont la réponse est très simple une fois que l'on a compris les définitions et le modèle. La plupart des personnes qui l'ont traitée l'ont réussie, avec parfois l'erreur de mettre une égalité plutôt qu'une différence.

4.4.3 Question IV.35, [10%, 2, 2.7]

Il y avait beaucoup d'éléments auxquels penser :
— Vérifier que les deux noeuds ont le même label (1/5)
— Vérifier que soit les deux noeuds sont des feuilles, soit aucun ne l'est (1/5)
— Vérifier que les deux enfants sont équivalents, et la correction globale (2/5)
— Justifier la complexité (1/5)

4.4.4 Question IV.36, [4%, 3, 2.8]

Question très peu traitée, mais l'idée générale était plutôt bien comprise par les personnes qui s'y sont essayées.

4.4.5 Question IV.37, [2%, 1.5, 1.8]

Il était important de remarquer qu'on unit deux éléments à chaque merge, et que cela ne peut avoir lieu que $n-1$ fois. Il y a eu souvent des réponses linéaires ($n-1$, $n/2$) qui, justifiées, donnaient quelques points.

4.4.6 Question IV.38, [2%, 2, 1.3]

Personne n'a trouvé la bonne complexité.