Composition d'Informatique C (XULSR)

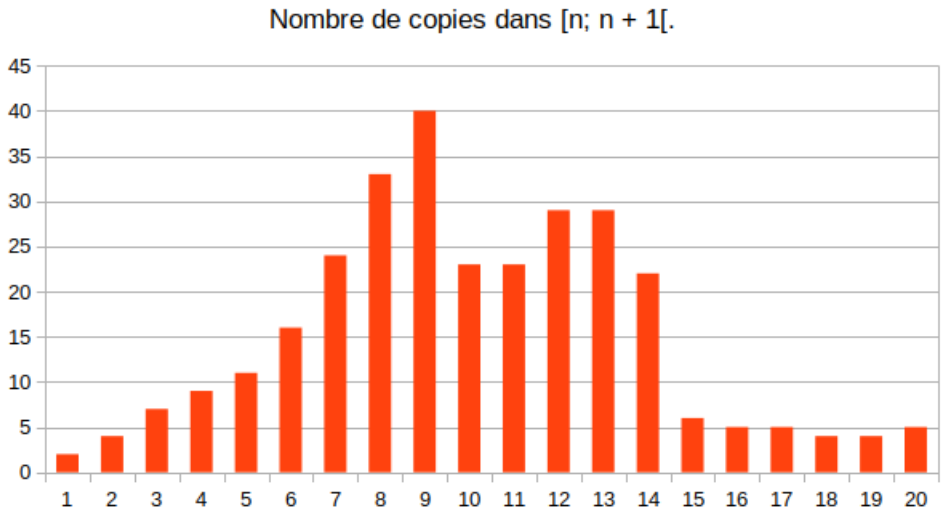
Filière MPI

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 60, et un coefficient multiplicateur de 1,2 est utilisé pour obtenir la note finale, en tronquant à 20. En d'autres termes, 50 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

$$\text{note(copie)} = \max\left(20,\ 1,2 \times \frac{20}{60} \times \sum_{\text{question } q} \text{points}(q) \times \frac{\text{score(copie}, q)}{5}\right).$$

La moyenne des 301 candidat·e·s est de 9.5/20 avec un écart type de 3.8.



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. Un pénalité de -2 points est appliquée pour les questions de code lorsque le·la candidat·e n'utilise pas le bon langage de programmation.

2.1 Partie I

Question I.1 [99% • 1 • 4,7]. RAS sur cette question très facile. Nous rappelons toutefois que le maximum de l'ensemble vide est à priori mal défini.

Question I.2 [93% • 2,5 • 3,6]. Question globalement bien traitée. Nous rappelons qu'il n'y a pas de nécessité absolue à introduire des fonctions auxiliaires lorsque le programme fonctionne sur des listes, et quelques copies font des programmes inutilement longs et compliqués. Ainsi, quelques copies introduisent des fonctions auxiliaires inutiles, de la forme

```
let fonction t =
    let rec aux t = ...
    in aux t
```

Quelques copies commencent par transformer les listes en tableaux, tentant probablement de dissimuler une mauvaise maîtrise de la manipulation de listes.

Tout cela n'a cependant pas été pénalisé tant que le résultat correct est calculé avec une complexité linéaire, sauf lorsque la solution proposée est excessivement longue avec de nombreuses fonctions auxiliaires sans explications.

Question I.3 [99% • 2 • 3,4]. Question également globalement bien traitée. Pénalité de -1 point par erreur d'indice (par exemple si le mot vide est produit, ou pour une erreur de 1 sur le nombre de 'a' écrit lors d'une itération), de -2.5 points si aucun `output`() n'était présent mais que l'ordre était bon. Aucun points si l'ordre n'est pas du tout bon (par exemple, à l'étape i, fait i^2 `pushR`($'a'$) puis `output`() en oubliant de pop).

Question I.4 [94% • 1,5 • 3,9]. 1,5 points par propriété (symétrie, séparation, inégalité triangulaire), 0,5 points à l'appréciation du correcteur. La question étant facile, une attention particulière a été portée sur la rédaction. Il s'agissait de savoir rédiger précisement sans se perdre dans une rédaction inutilement longue. Pénalité de -1 points pour les candidat·e·s parlant de « le plus court chemin de u à v » (qui est mal défini car il peut y en avoir plusieurs). Quelques candidat·e·s se lancent dans des preuves par induction douteuses pour montrer la symétrie.

Question I.5 [46% • 3 • 2,7]. Question demandant un peu de créativité. 1,5 point pour l'explication, 3.5 points pour le code. Les candidat·e·s ont parfois répondu à la question I.6 avant, pour s'en inspirer en revenant à la question I.5. Ce n'était pas forcément une bonne idée, et il valait mieux réfléchir immédiatement à cette question. Il y avait essentiellement deux façons de procéder : par taille croissante de la longueur des mots, en déplaçant la jonction entre les 'a' et les 'b' avec des (`popL`(), `pushR`(x), `output`()) successifs (obtenant un 2-ordre), ou par nombre croissant du nombre maximal de a ou de b (ce qui était un peu plus compliqué mais pouvait donner un 1-ordre). Les dessins accompagnant le code pour expliquer l'ordre choisi étaient appréciés. Pénalité de -1 point si le mot vide est oublié, -1 point si le même mot est produit plusieurs fois, -1 point par erreur d'indices.

Question I.6 [87% • 2 • 3,3]. Question bien traitée. 1,5 points pour l'explication et 3,5 points pour le code. -1 points pour oubli du mot vide et par erreur d'indice. 2.5 points si aucun `output`() mais l'ordre de parcourt est bon. Pas de points si parcours des diagonales (obtenant un 2-ordre), ou si l'ordre reste sur une ligne/colonne (par exemple, quelques copies tentent de produire d'abord tous les mots de b^*, puis tous les mots de ab^*, puis de aab^*, etc.).

Question I.7a [82% • 1,5 • 4]. Question plus simple et souvent bien traitée. -1 point s'il manque un mot, -1,5 s'il en manque 2, 2,5 points s'il en manque plus mais que l'ordre est globalement bon, 0 point si l'ordre ne convient pas du tout.

Question I.7b [63% • 3 • 2,2]. Question demandant de la réflexion et une bonne rédaction. Il y avait essentiellement deux façons de la traiter : soit en considérant le moment où tous les mots de taille $\leq d+1$ ont déjà été produits et en argumentant qu'à partir de ce moment, les mots produits seront soit tous de la forme b^*, soit tous de la forme ab^*, ou alors en justifiant qu'il doit y avoir une infinité d'indices i tels que w_i est dans b^* et w_{i+1} dans ab^*, et donc que comme il n'y a pas de répétition un tel w_i doit être de la forme a^n pour $n \geq d+1$. Pas de points pour les copies qui montrent seulement qu'il n'y a pas de 1-ordre.

Question I.8 [51% • 3,5 • 2,7]. Trop de copies font l'impasse sur cette question de com-plexité, alors qu'elle est importante et proche du programme. 1,5 point pour l'appartenance à NP, 3,5 points pour la réduction. 3 points seulement si la preuve ne marche que pour $t = 1$. Pour l'appartenance à NP, de nombreuses copies oublient de justifier pourquoi la vérification de $\delta_{\text{pp}}(w_i, w_{i+1}) \leq t$ est en temps polynomial, ou font référence à la question I.2 qui concerne δ_{ppr}. Très peu de copies font la réduction à l'envers.

2.2 Partie II

Le sujet n'indiquait pas si `lg.premier->prec` et `lg.dernier->suiv` sont toujours `NULL`, ou si au contraire la liste « cycle » avec `lg.premier->prec` étant égal à `lg.dernier` et `lg.dernier->suiv` égal à `lg.premier` (quand la liste n'est pas vide). Nous avons alors accepté les deux solutions, en pénalisant si la copie n'était pas cohérente sur ce choix tout du long, en pénalisant seulement à la première incohérence. De même les erreurs de confusion sur l'utilisation des pointeurs (`p*`, `&p`, `p.champs`, `p->champs`, etc) n'ont été pénalisées (généralement d'un -1) qu'à leur première occurrence et non sur toutes les questions. Cette partie assez proche du cours a globalement été assez bien traitée.

Question II.1 [93% • 0,5 • 4,5]. Plusieurs façons d'initialiser `lg` : comme indiqué plus haut, il s'agissait surtout d'être cohérent dans la suite.

Question II.2 [94% • 0,5 • 4,7]. Cette question très facile a pratiquement toujours été traitée correctement, en cohérence avec les définitions choisies à la question précédente.

Question II.3 [93% • 3 • 3,5]. 1 point pour la complexité, 4 points pour le code. -1 point si le `malloc` n'est pas fait correctement, -1 point si l'`assert` n'est pas fait correctement, -1 point par erreur de chaînage, -1 point si `lg.dernier` n'est pas mis à jour dans le cas particulier où la liste était vide.

Question II.4 [92% • 2 • 3,7]. Notation similaire à la question précédente.

Question II.5 [89% • 2 • 4,2]. 2 points pour `output`, 2 points pour `vide_liste` et 1 point pour la complexité. Certaines copies ne pensent pas à réutiliser `popL()` pour `vide_liste`, et en le réécrivant prennent le risque de perdre des points. -1 point s'il manque un caractère, ou si le programme pourrait tenter d'accéder un champ du pointeur nul. -0,5 point s'il manque le '\n' final pour `output`.

2.3 Partie III

Question III.1 [95% • 1 • 4,3]. Notation similaire à la question I.4. La symétrie était plus facile à rédiger que pour la question I.4, il suffisait de mentionner que le graphe était non orienté.

Question III.2 [95% • 0,5 • 4,8]. 2.5 points pour (a) et 2.5 points pour (b), en tout ou rien pour chaque. Cette question servait principalement aux candidat·e·s pour se familiariser avec la partie.

Question III.3 [76% • 3 • 3]. -1 point par erreur d'indice. -1 point si la copie fait une boucle pour vérifier si l'enfant a déjà un parent ou pas plutôt que d'utiliser un tableau de Booléens (ce qui fonctionne mais nous estimons que le calcul d'un arbre couvrant en temps linéaire est un exercice classique). Nous attirons l'attention sur le fait que trop de candidat·e·s proposent le code suivant (ici reproduit en pseudo code) :

```
vu = un tableau initialisé à false
vu[0] = true
push(0)
   while (lg not vide){
     n = popL()
     vu[n] = true
    pour les voisins j de n, si not vu(j):
        arbre[n][j]=true
        pushL(j)
```

qui ne marche pas car un nœud peut se retrouver avec plusieurs parents (par exemple sur le graphe $0 \to 1$, $0 \to 2$, $2 \to 1$). Nous avons tout de même mis 2 points lorsque c'était fait proprement. Il fallait évidemment utiliser les fonctions pushL et popL au cœur du sujet.

Question III.4 [83% • 1,5 • 4,1]. 2,5 points pour (a) et 2,5 points pour (b) (en tout ou rien pour (b)). Quelques candidat·e·s ont été perturbé·e·s par l'absence d'accolades autour des `if(!mystere)` et `if(mystere)`.

Question III.5 [49% • 3 • 2,3]. Pour montrer la borne de 3 sur la distance, deux solutions étaient possibles : faire une preuve par induction (l'hypothèse d'induction était alors trop souvent mal définie), ou alors considérer deux nœuds u, v consécutifs dans l'ordre et analyser l'algorithme pour montrer qu'ils sont à distance ≤ 3. La deuxième solution, bien que pouvant être correcte, était plus risquée car il était facile d'oublier des sous-cas (par exemple quand un nœud de la preuve n'a pas de fils, ou quand un parent n'en a qu'un seul plutôt que plusieurs, etc).

Question III.6 [70% • 1,5 • 4]. 3 points pour la complexité, 2 points pour la suggestion d'utiliser des listes d'adjacence.

Question III.7 [68% • 2 • 2,2]. 2 points pour (a), 3 points pour (b). -1 point pour (a) pour les copies mentionnant seulement un vague problème de « dépassement de mémoire » sans mentionner les mots-clés de *pile d'appel*, *pile d'exécution* ou *Stack Overflow*. 1 point seulement pour (b) si la copie se contente de dire qu'il suffit d'utiliser une pile sans expliquer plus en détails.

2.4 Partie IV

Question IV.1 [55% • 3,5 • 1,5]. 2 points pour la vérification qu'il n'y a qu'un seul cycle, 3 points pour la vérification qu'il n'y a qu'un seul chemin strict vers le cycle. 1 point pour les solutions qui proposent de visiter tous les chemins possibles de taille $\leq |V|$ (qui peuvent être en nombre exponentiel). Pour la détection de cycle, pas de points pour les copies qui expliquent

de manière floue, par exemple « on fait un DFS et quand on retombe sur un nœud déjà vu on incrémente un compteur de cycles et si le compteur est ≥ 2 on rejette ». Très peu de copies proposent un algorithme correct et compréhensible pour détecter s'il n'y a qu'un seul cycle (par exemple avec un DFS à 3 couleurs bien expliqué, en vérifiant que pour chaque arête du cycle trouvé, si on l'enlève alors il n'y a plus de cycle ; ou alors via un calcul des composantes connexes puis en vérifiant que dans l'unique composante ayant plus d'un sommet, les sommets ont tous un unique voisin dans la composante). Quelques copies se contentent de répéter les définitions, comme « on trouve tous les cycles et on vérifie s'il n'y en a qu'un à permutation près ».

Question IV.2 [60% • 2,5 • 2,9]. 1 point pour la définition de u, 1 point pour celle de v, 1 pour F et un pour F', et 1 point si la double inclusion est montrée proprement. La question était relativement simple mais beaucoup de copies sont trop imprécises sur les définitions.

Question IV.3 [27% • 3,5 • 3]. 1 point pour les explications et la structuration du programme, 4 points sur le code. Cette question permettait de voir comment les candidat·e·s organisent du code simple mais un peu long. Quelques candidat·e·s proposent des solutions élégantes comme par exemple la fonction auxiliaire suivante

```
let rec affiche_mot w =
  match w with
  | [] -> output ()
  | t::q -> pushR t; affiche_mot q; popR ()
```

-2 s'il n'y a aucun `popR`, 1 si le mot u est produit, -1 si dans la boucle while l'écriture de v vient avant l'itération sur F'.

Question IV.4 [27% • 1,5 • 2,2]. Question assez tout-ou-rien, peu de façons de prouver le résultat. Beaucoup de copies tentent ici de raisonner par l'absurde, mais cela se termine toujours mal, ou avec un « donc » magique énonçant le résultat et en espérant que le·la correcteur. trice n'y voie que du feu. Pas de points pour les quelques copies qui considèrent que le langage est régulier et utilisent du pompage.

Question IV.5 [16% • 2 • 1,4]. Quelques copies ont vu la similarité de la question avec la question I.7b, et la réponse devenait alors assez facile à rédiger. Il fallait comme pour la question I.7b donner de la précision dans la réponse, en construisant le mot où les deux branches divergent.

Question IV.6 [16% • 3 • 2,1]. 1 point pour (a), 4 points pour (b). Une preuve simple pour (b) était par exemple de dire que u et u' mènent à des cycles et donc à des langages infinis (car l'automate est émondé), puis d'utiliser la question IV.5, ce qui donne deux branches lourdes car u et u' ne sont pas préfixes l'un de l'autre.

Question IV.7 [6% • 3,5 • 1,8]. 3,5 points pour la preuve, 1,5 point pour recoller tous les morceaux (points comptés seulement si les questions précédentes étaient bien traitées). Attention, le deuxième cycle ne commence pas nécessairement à l'état d'entrée (-1 point). Cette dernière question n'était pas la plus difficile du sujet mais demandait d'avoir une compréhension globale de la dernière partie.