Composition d'Informatique B - 2h, Filière MP (XLSR)

Statistiques

La note moyenne des 403 candidats français et internationaux admissibles est 9,25/20, et l'écart-type est de 3,80.

Question	1	2	3	4	5	6	7
Moyenne*	0,89	0,93	0,92	0,62	0,73	0,56	0,28
Pourcentage de copies ayant traité la question	100%	100%	100%	94%	99%	81%	62%

Question	8	9	10	11	12	13	14
Moyenne*	0,79	0,51	0,36	0,22	0,38	0,66	0,25
Pourcentage de copies ayant traité la question	98%	83%	64%	44%	14%	19%	10%

* La moyenne est rapportée sur 1 point, et calculée parmi les copies qui ont traité la question.

Commentaires

Le sujet était long et comportait plusieurs questions assez subtiles (9, 12...). Les dernières questions (12, 13, 14) ont été très peu abordées. La question 13 était pourtant plutôt facile.

En grande majorité, les candidats maîtrisent la syntaxe du langage Python. À l'exception de la fonction récursive en question 6, le sujet se résolvait naturellement avec des fonctions écrites en style impératif : boucles for ou while, instructions conditionnelles, affectations.

Il ne s'agit pas simplement d'écrire du code fonctionnel, il faut aussi songer à sa lisibilité. Notamment, les noms de variables doivent aider le lecteur à deviner quel est leur rôle. Lorsqu'une fonction dépasse la dizaine de lignes de code, il est souhaitable d'expliquer son comportement en langage courant.

L'absence de test sur le dépassement de bornes de listes est une erreur trop fréquente et systématiquement pénalisée. Plusieurs candidats semblent également ignorer quand et comment copier une liste en Python.

Question 1

- La question est généralement bien traitée. Quelques candidats ne savent pas implémenter un parcours de liste de listes, ou oublient de donner la complexité.

Question 2

- La plupart des candidats ont pensé à réutiliser la fonction cases_noires. Beaucoup de copies ajoutent cependant des tests inutiles dans leur manipulation des booléens (par exemple, un test if suivi d'un return True). Il était suffisant d'écrire

  return cases_noires (cle_l) == cases_noires (cle_c)

  en faisant bien attention à utiliser le test d'égalité == et non le = de l'affectation.

Question 3

- En grande majorité, les candidats ont fait correctement le parcours de la liste.
- Quelques-uns ont cherché à réutiliser la fonction cases_noires de la question 1. C'est possible mais il faut faire attention au typage : cases_noires prend une liste de listes d'entiers, alors que taille_minimale doit travailler sur une liste d'entiers. Quand on a compris cela, on peut en effet écrire :

```
def taille_minimale(l):
    return cases_noires([l]) + len(l) - 1
```

- Certains candidats ne retournent pas la bonne formule (il était attendu la somme des entiers + taille de la liste – 1).

Question 4

- Une proportion significative de candidats n'a pas traité cette question de compréhension d'un morceau de code Python, ou n'a pas traité la partie 2.
- La question 4.2 est assez peu réussie. Quelques copies ont répondu que la fonction ne vérifie pas si la solution contient uniquement des 0 et 1. Pour avoir l'ensemble des points à cette question, la réponse d'avantage attendue était que la fonction ne vérifie pas s'il y a des blocs manquants.

Question 5

- Curieusement, des candidats relativement nombreux expriment la solution sous forme de partie entière au lieu de parler de quotient et de reste de la division euclidienne. D'autres se contentent d'énoncer l'égalité n = k*nc+l mais ne concluent pas sur la division euclidienne. Les copies qui se sont contentées d'indiquer cette égalité n'ont pas eu de points.

Question 6 :

- Cette question s'est révélée assez discriminante. Beaucoup de copies ont correctement exprimé la structure récursive de l'algorithme, mais peu ont su écrire la fonction sans aucune erreur.

- Les erreurs les plus fréquentes sont :
    - Une mauvaise compréhension de la spécification de la fonction auxiliaire : elle doit ajouter les solutions trouvées à la liste passée en argument ; notamment, elle ne doit pas retourner cette liste.
    - L'ajout de solutions à liste sans effectuer de copie au préalable (avec la fonction copy_sol() nommée dans le sujet).
    - Une mauvaise implémentation des indices k et l de la n-ième case à modifier.
- Les copies ayant traité la complexité ont généralement bien identifié le terme exponentiel en 2^{nc*nl} mais ont souvent oublié le facteur nc*nl dû à l'utilisation de la fonction verif.

Question 7 :

- La plupart des copies sont restées assez vagues sur la manière d'améliorer la fonction, parfois en se contentant de suggérer l'utilisation de verif_ligne à chaque appel récursif.
- On attendait des précisions sur les lignes à modifier dans la fonction de la question 6, et idéalement un algorithme qui n'ait pas à recalculer le nombre de cases noires à chaque étape.
- Certains candidats se contentent de mettre à jour un compteur du nombre total de cases noircies. C'est une amélioration correcte mais plus grossière que ce qui est demandé.
- Très peu de candidats ont écrit un peu de code, ce qui était attendu.

Question 8 :

- Cette question n'est pas difficile mais a fait l'objet de fréquentes erreurs d'inattention :
    - Les conditions (a) et (c) ne s'appliquent que dans les cas où la plage considérée ne touche pas le bord de la ligne. Il faut vérifier cela pour éviter les dépassements de liste lorsque c = 0 ou c+s = nc.
    - La fonction doit retourner la première position de conflit. Cela s'obtient naturellement si l'on teste les conditions séquentiellement, en sortant avec un return dès que l'une d'elles n'est pas vérifiée. En revanche, si l'on continue de tester après avoir rencontré un conflit, on retourne généralement la position du dernier conflit et non du premier.

Question 9 :

- Question difficile et rarement réussie.
- La longueur et la subtilité des algorithmes proposés rend indispensable d'expliquer un minimum leur fonctionnement en langage courant.
- L'utilisation de la fonction conflit (ou, plus rarement, d'une fonction récursive) débouchait généralement sur une complexité trop grande en O(nc*s). Quelques points ont malgré tout été attribués aux copies qui ont bien implémenté cette approche.

- Beaucoup de copies ne testent pas correctement si les cases situées avant et après le bloc candidat (lorsqu'elles existent) ne créent pas de conflits. Sur le plan algorithmique, il faut remarquer qu'un conflit dû à une case noire en position i n'implique pas que la première position valide soit après la position i (si la case i-s est de couleur indéterminée, le bloc peut potentiellement commencer en i-s+1).
- Une solution possible était de parcourir la liste en maintenant un compteur du nombre de cases noires ou indéterminées consécutives jusqu'à la position i courante. A chaque fois que ce compteur dépasse s, on teste si les cases aux positions i-s et i+1 (lorsqu'elles existent) ne sont pas noires, et on retourne i-s+1 dans ce cas.

Question 10 :

- A nouveau, beaucoup de candidats ne testent pas les dépassements d'indices avant d'accéder aux cases M[c-1][b] et M[c-s-1][b-1] du tableau.
- La preuve de complexité ignore souvent le coût de la fonction conflit, qui est pourtant conséquent. L'argument à utiliser est qu'on effectue une boucle for sur les blocs d'une ligne en appelant conflit une fois (au plus) par bloc. La complexité de cette boucle est donc au plus la somme des tailles des blocs, qui est en O(nc).

Question 11 :

- La question a été modérément comprise par les candidats qui l'ont abordée. Il fallait observer que le premier bloc ne peut pas débuter après la première case noire. Cela modifie la contrainte 2.(b) qui doit aussi vérifier $p = -1$ ou $p \geq c - s + 1$ avant d'affecter $M[c][0] = c - s + 1$.

Question 12 :

- Question très peu abordée.
- Il fallait remarquer qu'une fois trouvée la position c du bloc b, le bloc b-1 doit être placé en position M[c-2][b-1].

Question 13 :

- Cette question était plutôt facile (quand on a réussi à lire le sujet jusque là !).
- Lorsqu'on écrit

```
for c in range(liste_dp[i], liste_pp[i] + cle_l[i_ligne][i])
```

il n'est pas nécessaire de tester préalablement si liste_dp[i] < liste_pp[i] + cle_l[i_ligne][i]. En effet, dans le cas contraire, le corps de la boucle ne sera pas exécuté, ce qui convient.

Question 14 :

- Cette question est plus ouverte et les rares candidats qui l'ont abordée n'ont généralement proposé que des solutions partielles.