Banque MP - ENS/X- Session 2017

Rapport sur l'épreuve écrite d'informatique (2h)

Coefficients (en pourcentage du total d'admission) :
Lyon : 6,8%
Paris-Saclay et Rennes 5,8%
Ecole Polytechnique : 2,9%

Membres du jury : Pierre Bourhis, Emmanuel Caruyer, Samuel Thibault

1 Bilan général

À titre de rappel, cette épreuve n'est corrigée que pour les candidats admissibles. Le présent rapport ne concerne que la filière MP. Cette année le nombre total de candidats admissibles dans cette filière est de 557. La note moyenne est de 13,0 avec un écart-type de 3,1. La note minimale est de 2,6/20 et la note maximale 20/20. Aucune copie n'a obtenu une note éliminatoire. Au total, 59 candidats (10, 6%) ont traité le sujet dans son intégralité, c'est-à-dire ont obtenu une note strictement positive à chacune des 16 questions. La répartition des notes est résumée dans le tableau qui suit.

	[0; 4[	[4; 8[	[8; 12[	[12; 16[	[16; 20]
Effectif	1 (0,2%)	28 (5,0%)	181 (32,5%)	248 (44,5%)	99 (17,8%)

2 Commentaires

Le sujet portait cette année sur le calcul de l'intersection de deux ensembles de points à coordonnées entières. La première partie abordait l'implémentation d'une solution naïve en Python ; la seconde partie demandait d'écrire des requêtes SQL liées à ce calcul d'intersection ; les trois dernières parties introduisaient une structure de données pour les points (codage de Lebesgue) et les ensembles de points (AQL) pour aboutir à un calcul plus efficace de l'intersection de deux ensembles de points.

Quelques remarques viennent à la première lecture des copies ; l'une des spécificités du sujet était d'imposer aux candidats l'utilisation d'un sous-ensemble des opérations sur les listes du langage Python, en particulier pour des raisons d'efficacité et de calcul de complexité. Le non respect de cette consigne a été à l'origine de beaucoup d'erreurs. Il est rappelé l'importance de lire le sujet dans son intégralité avant de traiter les questions. On peut citer entre autres l'utilisation non autorisée :

- des tranches de liste (syntaxe de type `l[i:j]`), ce qui par ailleurs peut changer la complexité d'une solution – au besoin, il était plus judicieux d'utiliser une fonction auxiliaire pour comparer des portions de listes plutôt que de les construire ;
- de l'ajout en tête de liste (là aussi la question de la complexité se pose) ;
- de la génération de liste utilisant une syntaxe de type `[f(x) for x in ... if ...]`, par ailleurs parfois mal maitrisée par les candidats.

Attention aux algorithmes récursifs : s'ils ne sont pas terminaux, ou fabriquent des tranches de listes (ce qui n'était de toute façon pas autorisé), la complexité est bien plus grande que prévue a priori. Et ils ne sont pas forcément plus facilement justes.

Il est en général inutile de traiter à part le cas des listes vides, les boucles fonctionnent correctement sur les listes vides.

On rappelle qu'il est possible d'itérer sur les élements d'une liste (syntaxe de type `for x in l: ...`) ; quand il n'est pas nécessaire d'utiliser un indice, cela permet d'écrire un code plus court, tout aussi lisible, et évite des erreurs de décalage d'indice.

De façon plus anecdotique, on voit encore souvent dans les copies des tests de type `if test == True: ...`, où `test` est une variable booléenne ; bien que cela ne constitue pas une faute en soi, c'est assez peu élégant et devrait être évité.

Enfin, un certain nombre de candidats ont négligé voire complètement fait l'impasse sur les questions relatives à SQL ; cela est déconseillé et a été relativement pénalisant. Parmi les erreurs fréquemment rencontrées pour la partie concernant SQL, un certain nombre de requêtes proposées présentaient des

ambiguïtés. Penser à nommer les tables ou sous-requêtes (en utilisant le mot-clé `AS`) quand cela est nécessaire.

3 Commentaires détaillés

Pour chaque question, on présente la répartition des résultats suivant les quatre catégories suivantes :
— $N = 0$ si la question est non traitée ou la réponse est complètement fausse,
— $N \in]0; 0,5[$ si la question est partiellement traitée ou la réponse comporte de nombreuses erreurs,
— $N \in [0,5; 1[$ si la question est traitée correctement avec quelques erreurs,
— $N = 1$ si la question est traitée correctement.

Question 1

	0	$]0; 0,5[$	$[0,5; 1[$	1
Question 1	28 (5,0%)	2 (0,4%)	30 (5,4%)	497 (89,2%)

Pour cette question, on ne pouvait se contenter d'utiliser l'opérateur `in`, qui n'était pas autorisé par le sujet. Par ailleurs, on préfère les solutions dont la complexité moyenne est de $l(q)/2$, où $l(q)$ est la longueur de la liste en entrée. Pour les candidats ayant utilisé une boucle `while`, ne pas oublier d'incrémenter l'indice dans la boucle.

Question 2

	0	$]0; 0,5[$	$[0,5; 1[$	1
Question 2	9 (1,6%)	1 (0,2%)	24 (4,3%)	523 (93,9%)

Certains candidats ont oublié de retourner la liste construite. Par ailleurs, l'énoncé indiquait que l'ensemble de points était représenté *« par une liste de points sans répétition »*, il n'était donc pas nécessaire de chercher à retirer les doublons.

Question 3

	0	$]0; 0,5[$	$[0,5; 1[$	1
Question 3	8 (1,4%)	0 (0,0%)	28 (5,0%)	521 (93,5%)

Il est demandé de justifier un minimum la complexité proposée.

Question 4

	0	$]0; 0,5[$	$[0,5; 1[$	1
Question 4	34 (6,1%)	3 (0,5%)	59 (10,6%)	461 (82,8%)

Attention à la syntaxe, il faut écrire `POINTS.x`, et non `x.POINTS`. Dans la clause `WHERE`, il faut utiliser l'opérateur `IN` et non `=` lorque l'on souhaite filter les entrées membres d'un ensemble. Certaines copies présentent les requêtes avec passages à la ligne et indentation judicieusement choisis, ce qui rend la solution plus lisible.

Question 5

	0	$]0; 0,5[$	$[0,5; 1[$	1
Question 5	247 (44,3%)	10 (1,8%)	70 (12,6%)	230 (41,3%)

Dans un certain nombre de cas, il manquait une jointure, ce qui revenait à écrire une requête (absurde) cherchant les enregistrements de la table `MEMBRE` pour lesquels `MEMBRE.idensemble` était à la fois égal à i et à j.

Une autre erreur couramment rencontrée a été de calculer l'union plutôt que l'intersection. La stratégie qui consiste à tester que `MEMBRE.idensemble` soit égal à i ou à j pouvait fonctionner, à condition de regrouper par `MEMBRE.idpoint` et de compter le nombre d'éléments retournés.

Question 6

	0	$]0;0,5[$	$[0,5;1[$	1
Question 6	173 (31,1%)	5 (0,9%)	75 (13,5%)	304 (54,6%)

Il était possible de réutiliser la requête de la question 4, en l'imbriquant dans la requête pour cette question. Attention là aussi à bien faire les jointures nécessaires, certaines solutions proposées ne renvoient (au mieux) que l'identifiant du point de coordonnées (a, b).

Question 7

	0	$]0;0,5[$	$[0,5;1[$	1
Question 7	40 (7,2%)	6 (1,1%)	19 (3,4%)	492 (88,3%)

Il est conseillé aux candidats de détailler un minimum le calcul de la solution proposée, de façon à ce qu'en cas de réponse incorrecte on puisse évaluer la gravité de l'erreur. À défaut, le correcteur est obligé de sanctionner lourdement. Parmi les erreurs récurrentes, certains candidats ont inversé le rôle de l'abscisse et de l'ordonnée.

Question 8

	0	$]0;0,5[$	$[0,5;1[$	1
Question 8	33 (5,9%)	57 (10,2%)	222 (39,9%)	245 (44,0%)

Un certain nombre de candidats ont présenté des solutions très « verbeuses » pour la conversion en base 4 des couples de bits ; il suffisait simplement d'écrire `2*bits(p[0], k) + bits(p[1], k)`. Par ailleurs, il était possible de commencer à traiter les bits de poids faible ou les bits de poids fort. On rappelle que la syntaxe en Python pour itérer à rebours de `n-1` à `0` (inclus) est : `for k in range(n-1, -1, -1): ...`. Si cette syntaxe (peu intuitive) est mal maitrisée, il est préférable d'écrire un code équivalent utilisant un parcours croissant des indices. Dans tous les cas, il fallait faire attention au sens de lecture pour la construction de la solution. Enfin, ne pas oublier de renvoyer le résultat à la fin de la fonction `code`.

Question 9

	0	$]0;0,5[$	$[0,5;1[$	1
Question 9	2 (0,4%)	1 (0,2%)	21 (3,8%)	533 (95,7%)

Cette question n'a pas posé de difficulté particulière ; elle a par ailleurs permis d'éviter de sur-sanctionner un candidat ayant compris la définition de comparaison à l'envers.

Question 10

	0	$]0;0,5[$	$[0,5;1[$	1
Question 10	47 (8,4%)	15 (2,7%)	73 (13,1%)	422 (75,8%)

Cette question demandait d'implémenter une relation d'ordre lexicographique. Il fallait naturellement commencer par comparer les bits de poids fort, c'est-à-dire faire une lecture de gauche à droite. La mise en place de la boucle `while` a posé un certain nombre de problèmes : test d'indice maladroit voire pas de test de dépassement, pas d'incrémentation de l'indice dans la boucle. Enfin, le cas d'égalité ne peut se détecter qu'après un parcours complet des deux codages de Lebesgue, ce n'est donc pas correct de renvoyer un `0` prématurément, c'est-à-dire à l'intérieur de la boucle.

Question 11

	0	$]0;0,5[$	$[0,5;1[$	1
Question 11	23 (4,1%)	0 (0,0%)	38 (6,8%)	496 (89,0%)

L'énoncé demandait (à tort) un codage de Lebesgue *« compacté »*, certains candidats ont écrit « 3 », au lieu de « 34 » qui aurait été conforme à la notation introduite à la question suivante. Bien entendu, cela n'a pas été sanctionné.

Question 12

	0	$]0;0,5[$	$[0,5;1[$	1
Question 12	38 (6,8%)	1 (0,2%)	58 (10,4%)	460 (82,6%)

Cette question n'a pas posé de difficulté particulière ; à l'instar de la question 11, elle était prioritairement destinée à inviter les candidats à travailler sur un exemple afin d'assimiler les concepts de codage de Lebesgue et d'AQL.

Question 13

	0	$]0;0,5[$	$[0,5;1[$	1
Question 13	33 (5,9%)	73 (13,1%)	269 (48,3%)	182 (32,7%)

Le sujet précisait que la liste `q` en entrée *« représenta[it] le codage de Lebesgue compacté d'un quadrant »* ; il était donc possible de ne pas tester toute la fin de la liste. Il était demandé de créer une *nouvelle* liste pour éviter de modifier la liste passée en paramètre ; à noter que la syntaxe `r = q` ne crée pas de nouvelle liste mais copie simplement la référence de la liste `q` dans une nouvelle variable, il faut écrire `r = list(q)` – syntaxe rappelée en début d'énoncé.

Question 14

	0	$]0;0,5[$	$[0,5;1[$	1
Question 14	290 (52,2%)	83 (14,9%)	128 (23,0%)	55 (9,9%)

Cette question complexe comportait plusieurs difficultés, la première consistait à identifier qu'un groupe de 4 AQL consécutifs formait un quadrant. La solution la plus élégante consistait probablement à tester l'égalité des préfixes des premier et quatrième AQL, ce que l'on pouvait faire en s'aidant de `ksuffixe`, cette fonction ne modifiant pas la liste passée en paramètre. On pouvait aussi écrire une fonction de comparaison *ad hoc*.

De manière générale, il fallait faire attention à la complexité… l'objectif de cette partie est de faire mieux qu'une complexité quadratique.

Attention si l'on utilise une boucle `while i < len(s)-3`, il faut penser à ajouter tout de même les 3 derniers éléments.

On rappelle que « supprimer » des éléments dans une liste, cela coûte très cher ! Il vaut mieux remplir une autre liste à mesure. De même, supprimer les doublons après modification peut coûter très cher selon la méthode utilisée.

Il ne faut pas se contenter de supprimer les doublons après toutes les itérations sur `k` : il est nécessaire de les enlever à mesure pour que les ensembles de points se retrouvent voisins dans la liste, pour pouvoir continuer à compacter.

Question 15

	0	$]0;0,5[$	$[0,5;1[$	1
Question 15	204 (36,6%)	12 (2,2%)	54 (9,7%)	287 (51,5%)

La structure de `compare_ccodes` est similaire à celle de `compare_pcodes` (question 10). Les difficultés rencontrées sont elles aussi comparables ; par ailleurs, certains candidats ont cherché à réutiliser `compare_pcodes`, le plus souvent de façon maladroite.

Question 16

	0	$]0;0,5[$	$[0,5;1[$	1
Question 16	355 (63,7%)	25 (4,5%)	76 (13,6%)	101 (18,1%)

La structure naturelle de la fonction `intersection2` est proche de celle d'un tri fusion, cependant contrairement à ce dernier c'est une intersection que l'on calcule, il ne faut donc pas terminer par inclure la dernière liste non vide dans le résultat final. Certains candidats ont d'ailleurs calculé l'union plutôt que l'intersection.

Attention de ne pas appeler inutilement `compare_ccodes` plusieurs fois.

Certains candidats n'ont pas exploité le fait que les listes en entrée étaient triées, la complexité de la solution proposée dans ce cas est $l(p) \times l(q)$, et non $l(p) + l(q)$.

À noter que l'appel à la fonction `compacte` était inutile ici : en effet, aucun « nouveau » quadrant ne peut apparaitre dans l'intersection.