Composition d'Informatique (2 heures), Filière MP
(XLCR)

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 556 (pour les candidats X-ENS). La note moyenne est de 15,19 avec un écart-type de 2,20. Les tableaux ci-dessous donnent la répartition détaillée des notes par série, ainsi que la synthèse calculée sur l'ensemble des copies corrigées. La note minimale est de 7/20 et la note maximale 19,5/20. Aucune copie n'a obtenu une note éliminatoire.

X - ENS

	Série 1		Série 2		Série 3		Série 4		**Synthèse**	
$0 \leq N < 4$	0	0,0%	0	0,0%	0	0,0%	0	0,0%	**0**	**0,0%**
$4 \leq N < 8$	2	1,4%	0	0,0%	0	0,0%	0	0,0%	**2**	**0,4%**
$8 \leq N < 12$	18	12,2%	10	7,2%	12	8,4%	11	8,7%	**51**	**9,2%**
$12 \leq N < 16$	69	49,9%	71	51,1%	78	54,5%	57	44,9%	**275**	**49,5%**
$16 \leq N \leq 20$	58	39,5%	58	41,7%	53	37,1%	59	46,5%	**228**	**41,0%**
Total	147	100,0%	139	100,0%	143	100,0%	127	100,0%	**556**	**100,0%**
Epreuve complète*	34	23,1%	37	26,6%	42	29,4%	38	29,9%	**151**	**27,2%**

* *Épreuve complète* signifie ici que le candidat a abordé toutes les questions de l'énoncé et obtenu une note non nulle à chacune des 18 questions.

	Série 1	Série 2	Série 3	Série 4	**Synthèse**
Nombre de copies	147	139	143	127	**556**
Note minimale	7,0	9,9	9,1	10,1	**7,0**
Note maximale	19,5	19,2	18,7	19,0	**19,5**
Note moyenne	15,02	15,36	15,10	15,32	**15,19**
Ecart-type	2,52	2,06	2,04	2,15	**2,20**

Les notes se répartissent selon le tableau suivant, avec une moyenne de 14,35 et un écart-type de 2,08 (pour les candidats français de l'École polytechnique) :

$0 \leqslant N < 4$	0	0,00 %
$4 \leqslant N < 8$	1	0,48 %
$8 \leqslant N < 12$	27	12,98 %
$12 \leqslant N < 16$	133	63,94 %
$16 \leqslant N \leqslant 20$	47	22,60 %
Total	208	100 %
Nombre de copies : 208		
Note moyenne : 14,35		
Écart-type : 2,08		

2. Commentaires

Cette année le sujet portait sur l'étude des réseaux sociaux, il était constitué de deux grandes parties totalement indépendantes. La première partie consistait en la programmation `Python`, et la seconde sur l'interrogation des bases de données à l'aide de requêtes `SQL`.

Pour de nombreuses questions, il est clairement demandé aux candidats d'expliciter la complexité de leur code, ou de proposer un code satisfaisant à une complexité donnée. Pour ces questions particulières, cette analyse de la complexité est essentielle et fait explicitement partie de la notation.

L'évaluation d'un code informatique repose sur plusieurs éléments essentiels, parmi lesquels

- évidemment le code doit être juste et donner le résultat correct,
- la clarté et la lisibilité du code sont également des éléments essentiels de l'évaluation : l'algorithme mis en œuvre doit être simple et clair, et un soin particulier doit être apporté dans la manière de présenter le code (indentation des boucles et tests, passages à la ligne, ...),
- les commentaires explicatifs ne sont pas strictement indispensables, ils peuvent néanmoins aider à comprendre la démarche mise en œuvre, en particulier lorsque la méthode utilisée n'est pas simple ou ne fonctionne pas,
- l'efficacité intervient également dans l'évaluation du code, même si cela n'apparaît pas explicitement dans l'énoncé.

Quelques remarques générales à la lecture des codes :

- La lisibilité des copies est très importante. En cas de doute ce n'est pas au correcteur d'évaluer ce que le candidat *aurait pu répondre* ; seul ce qui est objectivement inscrit et lisible sur chaque copie est réellement pris en compte.
- Cette lisibilité est d'autant plus importante pour les codes que le candidat doit écrire ; le langage informatique étant un langage structuré, la copie se doit de refléter cette structure et la logique du langage, ainsi que la logique de l'algorithme implémenté.

- On voit souvent des solutions qui sont justes, mais totalement tordues avec un code très difficile à lire et à comprendre ; clairement une telle situation doit être évitée.
- Attention à certaines notations qui peuvent avoir un sens à l'écrit, mais pas dans un langage informatique, comme par exemple `t'`, ou encore les accents dans les noms de variables ou fonctions.
- Il faut faire attention de ne pas manipuler des variables qui ne sont pas initialisées, par exemple dans une fonction qui cherche à évaluer la valeur maximale d'un tableau.
- Les éléments de structuration du code (boucles ou tests) ont un début et une fin qui sont bien marqués par les règles syntaxiques du langage, il est impératif que les candidats veillent à spécifier ces mots-clés de manière précise et systématique ; dans le cas contraire, on a confusion totale sur ce qu'est supposé faire le code, et cette confusion n'est jamais à l'avantage du candidat.
  Dans le cas du langage `Python`, l'indentation de certaines portions de code est critique, puisque c'est elle qui détermine les instructions rattachées à une boucle ou à un test ; il est donc indispensable de bien respecter ces règles d'indentation.
- La syntaxe est parfois très approximative, avec un flou volontaire ou non ; rien ne doit être approximatif dans le domaine de l'informatique, il faut au contraire veiller à être très précis.
- De manière générale, on peut souvent considérer que les différentes questions sont à tiroir, il ne faut donc pas hésiter à utiliser les fonctions écrites dans les questions précédentes, plutôt que de tout refaire.
- Il est impératif d'écrire les codes de manière complète et précise ; on ne peut absolument pas accepter des écritures partielles, par exemple en omettant les paramètres que l'on transmet lors d'un appel de fonction, parce que **c'est peut-être justement là** que l'on pourra faire la différence entre un code qui est correct et un code qui ne l'est pas.
  A titre d'exemple, on voit assez souvent des appels de fonctions dans lesquels les parenthèses sont omises, voire un ou plusieurs de ses paramètres.
- Il faut éviter d'appeler la même fonction de manière itérative, par exemple dans une boucle ; il est préférable d'appeler la fonction une seule fois, stocker le résultat en mémoire, et ensuite réutiliser cette variable.
- Certains opérateurs logiques sont utilisés de manière un peu approximative, comme par exemple `><` à la place de `<>` ou `!=`, ou encore `=<` à la place de `<=`. Attention à la confusion entre `=` et `==`.
  Attention de bien faire la distinction entre `!=` et `=!`, la seconde option produisant soit une erreur de syntaxe, soit dans le meilleur des cas un comportement qui ne correspond pas à ce qui est attendu.
  On a trouvé quelques petites originalités très personnelles, telles que par exemple `=|=` ou `=/=` à la place de `!=`. Le signe `=` barré ne convient pas non plus.
  Les opérateurs && et `||` n'existent pas en `Python`, utiliser `and` et `or`.
- Il est important de bien distinguer les instructions `a=b` et `b=a`, l'opérateur `=` affecte à son opérande de gauche la valeur de son opérande de droite, et non l'inverse.
- De manière générale, il est très important, lorsqu'une fonction explore les valeurs d'un tableau, de vérifier que l'indice ne dépasse pas la taille du tableau ; il s'agit

là d'une erreur grave de programmation, les candidats qui n'ont pas été vigilants sur ce point ont systématiquement été sanctionnés. A titre de rappel, en langage `Python`, l'indexation des tableaux commence à 0 et non à 1.

- Pour les listes, on peut utiliser la fonction `append`, que l'on doit utiliser sous la forme `liste.append(valeur)`. Il faut en particulier éviter `liste=liste.append(valeur)` puisque `append` ne renvoie pas de résultat.
  Les fonctions `apend` ou `happend` n'existent pas.
  L'opérateur `+=` ne permet pas d'ajouter un élément à une liste, il faut utiliser `append`.

3. Commentaires détaillés

Pour chaque question, un tableau récapitule les taux de réussite avec les conventions suivantes :

- 0 signifie aucun point pour la question (question non traitée ou abordée mais totalement fausse),
- $<$0,5 signifie moins de la moitié des points de la question,
- $\geq$0,5 signifie plus de la moitié des points de la question,
- 1 signifie la totalité des points de la question.

Question 1 :

	0		$<$0,5		$\geq$0,5		1		Total	
Question 1	2	0,4%	0	0,0%	18	3,2%	536	96,4%	556	100,0%

Cette question ne pose strictement aucune difficulté, elle permet simplement d'assurer que les candidats ont effectivement compris la structure de données utilisée pour représenter un réseau. Dans les deux cas on a 5 individus, les liens étant différents entre le réseau A et le réseau B.
Dans le cas du réseau A, il faut penser à prendre en compte l'individu 4, même s'il est isolé. Dans les deux cas, il faut veiller à prendre en compte de manière exhaustive tous les liens représentés sur les schémas respectifs. On observe parfois quelques oublis.

Question 2 :

	0		$<$0,5		$\geq$0,5		1		Total	
Question 2	2	0,4%	0	0,0%	5	0,9%	549	98,7%	556	100,0%

Là encore, cette question est très simple, il suffit de savoir que `[]` suffit à créer une liste de liens vide. Quasiment aucun souci avec cette question.

Question 3 :

	0		$<$0,5		$\geq$0,5		1		Total	
Question 3	27	4,9%	0	0,0%	30	5,4%	499	89,7%	556	100,0%

Cette question consiste à déterminer si deux individus `i` et `j` sont contenus dans une paire donnée. La seule subtilité est de bien voir que l'individu `i` peut indifféremment se trouver en première ou seconde position dans la paire, mais ceci a globalement été bien vu par la grande majorité des candidats.
Une première solution consiste à comparer `paire[0]` et `paire[1]` à `i` et `j` ou inversement. Une seconde approche, plus compacte, peut simplement comparer `paire` à `[i,j]` ou `[j,i]`.
A noter que dans la première solution, il est nécessaire de jouer avec les opérateurs logiques `and` et `or`, et qu'une utilisation des parenthèses, même si elles ne sont pas indispensables dans ce cas précis, rend la lecture et la compréhension plus aisées.

A ces quelques remarques près, quasiment aucun souci avec cette question.

Question 4 :

	0		<0,5		≥0,5		1		Total	
Question 4	0	0,0%	15	2,7%	116	20,9%	425	76,4%	556	100,0%

Cette question consiste à déterminer si deux individus `i` et `j` du réseau sont amis. Pour cela il suffit d'explorer la liste des liens (`reseau[1]`) et de faire appel à la fonction précédente.
En terme de complexité, dans le pire des cas on doit explorer l'ensemble des liens, ce qui nous donne une complexité en $O(m)$. A noter qu'on peut interrompre la boucle dès que l'on trouve un lien entre les deux individus ; les liens restants n'apportent rien, et en moyenne cela nous fait gagner un facteur deux dans le temps d'exécution.

Points importants qui concernent cette question, ainsi que les questions suivantes :
- dans un problème de ce genre, on a deux variables d'échelle qui sont `n` le nombre d'individus et `m` le nombre de liens, il est donc très important de veiller à ce qu'il n'y ait aucune confusion entre ces deux facteurs,
- lorsque l'on parle de complexité en $O(m)$, il ne faut pas répondre par $O(m+2)$ par exemple, le facteur correctif +2 étant ici non significatif.

Question 5 :

	0		<0,5		≥0,5		1		Total	
Question 5	3	0,5%	2	0,4%	43	7,7%	508	91,4%	556	100,0%

Cette question consiste à déclarer un lien d'amitié entre deux individus `i` et `j` du réseau. Il convient de noter qu'il suffit simplement d'ajouter un lien à la liste des liens existants (`reseau[1]`, et non `reseau[0]`). Il faut préalablement vérifier que ce lien n'existe pas déjà, ce qu'un simple appel de la fonction précédente permet de faire.
L'ajout du lien utilise simplement la méthode `append([i,j])` de la liste `reseau[1]`, puisqu'on n'a pas d'exigence particulière dans l'ordre d'apparition des liens.

Question 6 :

	0		<0,5		≥0,5		1		Total	
Question 6	0	0,0%	29	5,2%	205	36,9%	322	57,9%	556	100,0%

Cette question, un peu plus compliquée que les précédentes, vise à obtenir la liste des amis d'un individu `i` donné du réseau. Là il y a clairement deux approches différentes :

- la première consiste à faire une boucle sur les `n` individus, et appeler la fonction précédente `sontAmis()` : l'inconvénient de cette approche est qu'elle est de complexité $O(n \times m)$, que l'on ne peut pas ramener à $O(m)$ simplement parce que telle est la consigne de l'énoncé...
- la seconde consiste à faire une boucle sur les `m` liens, et à tester l'existence d'un lien d'amitié avec l'individu `i` donné : cette approche est effectivement de complexité $O(m)$, comme demandé.

Une erreur assez fréquente a consisté à renvoyer la liste des liens d'amitié, et non pas simplement la liste des amis, ce qui nécessitait quelques opérations supplémentaires.

Question 7 :

	0		<0,5		≥0,5		1		Total	
Question 7	9	1,6%	0	0,0%	101	18,2%	446	80,2%	556	100,0%

Cette question est très simple, elle consiste à expliciter le tableau des parents des individus correspondant à deux exemples de représentations filiales. Il n'y a là aucune difficulté, il faut juste bien comprendre ce que représente le tableau `parent`, et suivre minutieusement les hiérarchies illustrées par les deux schémas.
Cette question a très majoritairement été traitée correctement, les erreurs faites résultant probablement d'une inattention des candidats. Il ne faut naturellement pas oublier la seconde moitié de la question.

Question 8 :

	0		<0,5		≥0,5		1		Total	
Question 8	7	1,3%	0	0,0%	40	7,2%	509	91,5%	556	100,0%

Pour cette question, il est demandé de construire le tableau `parent` correspondant à la situation où chaque individu est son propre représentant. A noter une coquille dans l'énoncé, qui fait référence à `pere[i]=i` et non `parent[i]=i`, mais cette coquille n'a semble-t-il absolument pas perturbé les candidats.
`Python` met à notre disposition plusieurs solutions pour réaliser cette opération, qui consiste *in fine* simplement à retourner une liste de `0` à `n-1` inclus. La solution la plus compacte consiste à simplement utiliser `range(n)`, l'approche la plus "complexe" consiste par exemple à créer un tableau que l'on remplit par une boucle.

Une erreur observée à plusieurs reprises a consisté à retourner une liste de listes, et non pas simplement une liste. Ainsi, partant d'une liste vide, il faut utiliser `append(i)` dans la boucle de remplissage, et non `append([i])`.

Question 9 :

	0		<0,5		≥0,5		1		Total	
Question 9	0	0,0%	5	0,9%	67	12,1%	484	87,1%	556	100,0%

L'objectif de cette question est de remonter toute la hiérarchie des parents, à partir d'un individu `i` donné, jusqu'à ce que l'on rencontre le représentant du groupe, caractérisé par le fait qu'il est son propre parent.
Cette question a été globalement traitée correctement ; une boucle `while` ou une approche récursive convenaient parfaitement. A noter ici qu'une boucle `while` est plus pertinente qu'une boucle `for` dans la mesure où l'on ne sait pas par avance le nombre de passages nécessaires dans la boucle.
Dans le pire des cas, il est nécessaire de remonter toute la hiérarchie des `n` individus référencés par la liste `parent`, ce qui nous donne une complexité en $O(n)$. Par ailleurs la question demande explicitement une illustration de cette complexité, c'est donc une partie de la question à ne pas laisser de côté.

Question 10 :

	0		<0,5		≥0,5		1		Total	
Question 10	2	0,4%	5	0,9%	3	0,5%	546	98,2%	556	100,0%

Cette question consiste simplement à fusionner les deux groupes contenant les individus `i` et `j` donnés. L'énoncé présente une coquille, le représentant de l'individu `j` étant référencé par `question` et non `q`. Certains candidats l'ont explicitement noté sur leur copie, mais clairement cette coquille n'a pas été gênante.
Les différentes étapes à suivre sont explicitement dans l'énoncé de la question, qui du coup ne présente strictement aucune difficulté.

Question 11 :

	0		<0,5		≥0,5		1		Total	
Question 11	36	6,5%	16	2,9%	164	29,5%	340	61,2%	556	100,0%

Dans cette question il est demandé aux candidats de proposer une série de $(n-1)$ fusions à partir d'une partition de n singletons, avec une complexité de l'ordre de n^2. Cette question ne pose pas de difficulté particulière, il faut simplement veiller à être précis dans la description des différentes fusions, leur sens, et également dans le calcul et la justification de la complexité.

Question 12 :

	0		<0,5		≥0,5		1		Total	
Question 12	9	1,6%	11	2,0%	115	20,7%	421	75,7%	556	100,0%

Cette question est une amélioration de la fonction écrite à la question 9 pour identifier le représentant d'un individu `i` donné. L'idée ici peut par exemple reposer sur une première boucle permettant de trouver le représentant, suivie d'une seconde boucle qui remonte une seconde fois toute la hiérarchie, et modifie tous les représentants des individus rencontrés lors de cette seconde exploration.
Une approche récursive convient également très bien.
Il faut bien veiller à modifier les représentants de tous les individus rencontrés lors de l'exploration de la hiérarchie. C'est là une des causes les plus fréquentes d'erreur, avec des codes laissant inchangés le parent de certains individus.

Comme pour toute question comportant un aspect complexité, les candidats doivent clairement expliquer en quoi cette modification est neutre en terme de complexité.

Certains candidats ont manifestement été gêné par l'intitulé de cette question, qui mentionne une "procédure" là où la question 9 mentionnait une "fonction". Ce point un peu confus n'a pas donné lieu à une sanction dans la notation.

Question 13 :

	0		<0,5		≥0,5		1		Total	
Question 13	41	7,4%	141	25,4%	277	49,8%	97	17,4%	556	100,0%

L'objectif de cette question est de déterminer la liste des groupes d'individus, chaque groupe étant défini par un même représentant. L'algorithme à écrire est plus complexe, et cette question a clairement posé problème à nombre de candidats. Une erreur très classique, observée un grand nombre de fois, consiste à créer une liste des groupes initialement définie par `[[]]*n`. On comprend que l'objectif de cette instruction est de créer une liste de `n` sous-listes toutes vides, sauf que procéder ainsi signifie que les `n` sous-listes sont non-différenciées. Ainsi, le fait de modifier l'un quelconque des groupes revient à modifier tous les autres groupes.
Une solution pouvait par exemple s'écrire sous la forme `[[] for i in range(n)]`.
Une autre erreur fréquente consiste à partir d'une liste de groupes vides, d'explorer tous les individus et ainsi reconstituer tous les groupes, et finalement retourner le résultat obtenu. En procédant de cette manière, on prend le risque de retourner des groupes vides. Il est donc nécessaire, si on procède de cette façon, d'éliminer tous les groupes vides avant de quitter la fonction.
Une autre variante consiste à obtenir plusieurs fois le même groupe, situation à éviter à nouveau.

De nombreux candidats utilisent une imbrication de boucles menant à une complexité en $O(n * \alpha(n) + n * g)$, g étant le nombre de groupes. C'est dommage, car la complexité $O(n * \alpha(n))$ était atteignable de manière simple.

Compte tenu de la complexité algorithmique un peu plus importante, cette question a globalement été moins bien traitée que les questions précédentes.

Question 14 :

	0		<0,5		≥0,5		1		Total	
Question 14	72	12,9%	235	42,3%	246	44,2%	3	0,5%	556	100,0%

Cette question est la plus complexe, elle nécessite de suivre attentivement les différentes étapes de l'algorithme données dans l'énoncé, et de traiter les choses de manière appropriée. Le but ici est de constituer deux groupes d'individus, en minimisant le nombre de liens d'amitié entre ces deux groupes. Les différentes étapes de l'algorithme utilisé sont décrites précisément dans l'énoncé.

Pour cette question, on trouve de manière récurrente les mêmes erreurs, parmi lesquelles :

- le lien pris en compte dans la boucle est choisi au hasard, ce n'est donc ni le premier ni le dernier lien,
- il ne faut surtout pas faire appel à la fonction de tirage aléatoire tant qu'on tombe sur un lien précédemment traité : cette méthode est rapidement prohibitive en temps de calcul, en particulier lorsque le nombre de liens restants devient petit par rapport au nombre total de liens initiaux,
- le tirage aléatoire porte sur un des liens d'amitié, et non sur les deux individus d'un lien,
- il ne faut surtout pas appeler en boucle la fonction `listeDesGroupes()` ou toute autre fonction de complexité équivalente, c'est globalement beaucoup trop onéreux en temps de calcul,
- la boucle d'itération porte d'une part sur le nombre de groupes, d'autre part sur le nombre de liens non traités ; il faut donc veiller à bien tester ces deux conditions,
- de même, une série de dernières fusions doit être faite lorsque la boucle d'itération se termine avec plus de trois groupes, une proportion assez importante de candidats omettent cette dernière étape de l'algorithme,
- certains candidats utilisent `len(partition)` pour connaître le nombre de groupes ; ce n'est pas correct, le tableau `partition` ayant toujours la même longueur,
- la concaténation de listes ou la suppression d'un élément au sein d'une liste sont très coûteuses ; pour mettre le lien à part, il faut échanger simplement les cases,
- attention, un série d'instructions du genre
  ```
  lien = reseau[1][i]
  lien,reseau[1][n] = reseau[1][n],lien
  ```
  ne fonctionne pas,

L'énoncé comportait une erreur : il était demandé d'effectuer $k-1$ fusions à partir des k groupes pour obtenir 2 groupes, il fallait effectuer $k-2$ fusions seulement.

Question 15 :

	0		<0,5		≥0,5		1		Total	
Question 15	220	39,6%	70	12,6%	125	22,5%	141	25,4%	556	100,0%

Cette fonction vise à calculer le nombre de liens d'amitié entre les divers individus des

deux groupes issus de la question précédente. Dans l'absolu cette question ne pose pas de difficulté particulière, mais il faut veiller à éviter toute formulation excessive en terme de complexité.
La bonne approche consiste à explorer tous les liens d'amitié, et d'incrémenter un compteur à chaque fois que les deux individus du lien ont des représentants différents. A noter que certains candidats ont ici inversé le test. Cette approche permet d'atteindre une complexité linéaire en m, nombre de liens.

De nombreux candidats ont imbriqué de multiples boucles sur les individus, le pire des cas rencontrés consistant en 4 niveaux de boucles sur les individus, finissant sur un appel de la fonction `sontAmis()` qui représente donc un cinquième niveau de boucle. Une telle approche, même si elle peut donner un résultat correct, est totalement prohibitive en temps de calcul. Ce genre d'approche est donc à éviter, même si le critère de complexité n'apparaît pas explicitement dans la formulation de la question.

Les trois dernières questions portent sur l'interrogation de tables SQL, une première table référençant les individus, et une seconde table les liens d'amitié entre individus.
On peut mentionner quelques remarques générales, communes aux trois questions :

- on voit très souvent des requêtes SQL excessivement compliquées, qui en final ne peuvent pas donner le résultat (résultat erroné ou erreur de syntaxe),
- la condition indiquée après la clause `SELECT` est souvent écrite entre parenthèses, ce qui conduit à une erreur de syntaxe,
- le test d'égalité s'écrit `=` et non `==`,
- dans nombre de cas, les différents éléments de la requête sont bien présents, mais dans un ordre totalement incohérent.

Question 16 :

	0		<0,5		≥0,5		1		Total	
Question 16	107	19,2%	0	0,0%	0	0,0%	449	80,8%	556	100,0%

Dans cette question, il est demandé de donner la liste des amis d'un individu `x`. Pour cela il n'est pas utile de consulter la table `INDIVIDUS`, la seule table `LIENS` suffit.

Question 17 :

	0		<0,5		≥0,5		1		Total	
Question 17	145	26,1%	0	0,0%	93	16,7%	318	57,2%	556	100,0%

Cette question peut être vue comme une extension de la question précédente, la liste des amis devant ressortir sous la forme des nom et prénom, et non plus des identifiants dans la table `INDIVIDUS`.
Une possibilité consiste à utiliser la clause `JOIN`, à condition de l'écrire correctement au niveau de sa syntaxe SQL.

Question 18 :

	0		<0,5		≥0,5		1		Total	
Question 18	287	51,6%	0	0,0%	73	13,1%	196	35,3%	556	100,0%

Cette dernière question consiste à énumérer les amis des amis de l'individu `x` donné. Cette question a clairement donné lieu à des syntaxes totalement farfelues ou excessivement compliquées. Là encore l'utilisation de la clause `JOIN` permettait d'adresser assez simplement cette question.
A noter que des amis d'amis d'un individu donné peuvent l'être par plusieurs chemins ; ainsi un même individu peut apparaître plusieurs fois dans le résultat de la requête SQL. L'utilisation du mot-clé `DISTINCT` permet de résoudre ce problème. Aucune sanction n'a été prise à l'encontre des candidats qui n'ont pas fait appel à cette clause particulière.
Là encore il est inutile de consulter la table `INDIVIDUS`, la table `LIENS` suffit.