Composition d'Informatique (2 heures), Filière PC
(XEC)

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 PC.

Cette année, le nombre total de candidats admissibles dans cette filière est 520. La note moyenne est 11,60 avec un écart-type de 2,60. 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 1,80 et la note maximale 19,20.

X - ENS

	Série 1		Série 2		Série 3		Série 4		**Synthèse**	
$0 \leq N < 4$	0	0%	1	1%	0	0%	4	3%	**5**	**1%**
$4 \leq N < 8$	14	11%	10	8%	9	8%	5	4%	**38**	**7%**
$8 \leq N < 12$	62	47%	61	48%	59	50%	71	51%	**253**	**49%**
$12 \leq N < 16$	50	38%	48	38%	43	36%	54	39%	**195**	**38%**
$20 \leq N \leq 20$	7	5%	8	6%	8	7%	6	4%	**29**	**6%**
Total	133	100,0%	128	100,0%	119	100,0%	140	100,0%	**520**	**100,0%**

	Série 1	Série 2	Série 3	Série 4	**Synthèse**
Nombre de copies	133	128	119	140	**520**
Note minimale	5,80	3,70	4,80	1,80	**1,80**
Note maximale	17,90	19,20	18,80	17,50	**19,20**
Note moyenne	11,45	11,73	11,68	11,47	**11,58**
Ecart-type	2,43	2,64	2,66	2,63	**2,59**

Les notes se répartissent selon le tableau suivant, avec une moyenne de 11,62 et un écart-type de 2,56 (pour les candidats français de l'École polytechnique) :

$0 \leqslant$ N $<$ 4	4	0,78 %
$4 \leqslant$ N $<$ 8	34	6,65 %
$8 \leqslant$ N $<$ 12	248	48,53 %
$12 \leqslant$ N $<$ 16	196	38,36 %
$16 \leqslant$ N $\leqslant$ 20	29	5,68 %
Total	511	100 %
Nombre de copies : 511		
Note moyenne 11,62		
Écart-type : 2,56		

2. Commentaires

L'épreuve de cette année comportait deux parties, à traiter respectivement en PYTHON et en SQL. La première partie portait sur l'étude des réseaux sociaux, en particulier le calcul d'une coupe optimale dans un graphe, c'est-à-dire le moyen de séparer un réseau en deux groupes de façon à minimiser le nombre de liens d'amitiés qui lient un membre du premier groupe à un membre du deuxième groupe. La deuxième partie, toujours sur le thème des réseaux sociaux, demandait d'écrire diverses requêtes sur deux tables représentant un réseau social : `INDIVIDUS` contient l'identifiant, le nom et le prénom de chaque utilisateur, tandis que `LIENS` contient les identifiants de chaque couple d'amis dans le réseau. D'une manière générale, les aspects suivants sont pris en compte (par ordre d'importance) dans la correction :

D'une manière générale, les aspects suivants sont pris en compte (par ordre d'importance) dans la correction :

1. la correction de l'*algorithme* proposé vis-à-vis de la spécification fournie ;
2. l'efficacité de la solution ;
3. la lisibilité du code (on favorise une solution simple devant une solution compliquée équivalente, on apprécie le code bien indenté et structuré,...) ;
4. la correction des détails d'*implémentation* ;
5. les explications accompagnant le code sous la forme de commentaires.

3. Commentaires détaillés

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

— 0 signifie qu'aucun point n'a été donné pour la question (question non traitée ou totalement fausse) ;
— $< 0,5$ signifie que moins de la moitié des points ont été donnés ;
— $\geq$ 0,5 signifie que plus de la moitié des points ont été donnés ;
— 1 signifie que la totalité des points ont été donnés.

Question 1 :

0		<0,5		≥0,5		1		Total	
0	0%	0	0%	22	4%	498	96%	520	100,0%

On demandait de donner la représentation sous forme de liste de deux réseaux. Cette question ne demandait que de lire le sujet attentivement, d'autant qu'un exemple était donné immédiatement avant la question.

Erreurs classiques :

— Certains candidats n'ont traité qu'un seul des deux réseaux ;
— d'autres ont utilisé 4 (l'identité du dernier nœud) au lieu de 5 (le nombre de nœuds) pour le premier élément de la liste.

Question 2 :

0		<0,5		≥0,5		1		Total	
22	4%	10	24%	199	38%	289	56%	520	100,0%

On demandait d'écrire une fonction `creerReseauVide(n)` qui renvoie un réseau vide à `n` éléments, c'est-à-dire `[n,[]]`. Les candidats dont le programme prenait plus d'une ligne n'ont pas reçu tous les points.

Erreurs classiques :

— Certains candidats n'ont renvoyé qu'une liste vide ;
— d'autres ont défini une fonction sans argument ;
— enfin, certains ont imprimé le réseau au lieu de le renvoyer.

Question 3 :

0		<0,5		≥0,5		1		Total	
17	3%	8	2%	306	6%	465	89%	520	100,0%

On demandait d'écrire une fonction `estUnLienEntre(paire,i,j)` qui renvoie `True` si `i` et `j` sont les deux éléments de `paire` et `False` sinon. Il s'agissait essentiellement de faire deux tests imbriqués ou un test dont la condition est `(i == paire[0] and j == paire[j]) or (j == paire[0] and i == paire[1])`.

Erreurs classiques :

— Certains candidats ont renvoyé `False` dès le premier test échoué ;
— d'autres n'ont pas renvoyé `False` lorsque la condition n'était pas vérifiée ;
— enfin, certains candidats se sont trompés de formule logique.

Question 4 :

0		<0,5		≥0,5		1		Total	
18	3%	9	2%	104	20%	389	75%	520	100,0%
40	8%	4	1%	103	20%	373	72%	520	100,0%

La première ligne correspond à l'implémentation, la deuxième à la complexité.

On demandait d'écrire une fonction `sontAmis(reseau,i,j)` qui cherche un lien d'amitié entre `i` et `j` dans la liste de paires `reseau`. On attendait une boucle `while` (ou `for`) avec un appel à la fonction précédente. Les candidats qui ont réécrit le code de `estUnLien` n'ont pas reçu tous les points, tout comme ceux qui n'ont pas interrompu la boucle dès que la paire a été trouvée. De nombreux candidats ont donné une complexité concrète (par exemple $2m + 3$) ou mal justifiée.

Erreurs classiques :

— Certains candidats ont renvoyé `False` dans le corps de la boucle ;
— d'autres ont oublié le `return False final` ;
— enfin, certains candidats ont utilisé la fonction `pop`, qui modifie le `reseau` donné en argument.

Question 5 :

0		<0,5		≥0,5		1		Total	
5	1%	12	2%	70	13%	433	83%	520	100,0%
12	2%	3	1%	94	18%	411	79%	520	100,0%

La première ligne correspond à l'implémentation, la deuxième à la complexité.

On demandait d'écrire une fonction `declareAmis(reseau,i,j)` qui modifie `reseau` pour y ajouter un lien d'amitié $[i, j]$ uniquement si ce lien n'existait pas déjà. Cette question a majoritairement été bien traitée. De façon surprenante, certains candidats ont utilisé `assert` (dans `assert` $[i, j]$ `notin reseau`). Encore une fois, des candidats ont donné des complexités concrètes ou non justifiées.

Question 6 :

0		<0,5		≥0,5		1		Total	
6	1%	22	4%	280	54%	212	41%	520	100,0%
82	16%	20	4%	107	21%	311	60%	520	100,0%

La première ligne correspond à l'implémentation, la deuxième à la complexité.

On demandait d'écrire une fonction `listeDesAmis(reseau,i)` qui renvoie la liste des amis de `i` dans `reseau`. La solution la plus simple était d'utiliser la fonction précédente pour tester tous les amis potentiels. Cette approche donnait une complexité en $O(n * m)$. Une meilleure solution consistait à faire une boucle sur tous les liens d'amitiés, ce qui donnait une complexité en $O(m)$. Les candidats qui ont justifié correctement une complexité en $O(n * m)$ ont reçu tous les points pour la partie "complexité" de cette question.

Erreurs classiques :

- Certains candidats ont renvoyé le nombre d'amis ;
- d'autres ont renvoyé une liste de liens d'amitié ;
- enfin, certains candidats ont trouvé une complexité quadratique pour un algorithme linéaire.

Question 7 :

0		<0,5		≥0,5		1		Total	
1	0%	2	0%	157	30%	360	69%	520	100,0%

On demandait de donner les valeurs des tableaux parents d'après deux représentations filiales données dans le sujet, ainsi que la liste des représentants correspondante. Encore une fois, il s'agissait surtout de lire attentivement le sujet.

Erreurs classiques :

- Certains candidats n'ont traité qu'une des deux représentations ;
- d'autres ont oublié de donner la liste des représentants.

Question 8 :

0		<0,5		≥0,5		1		Total	
10	2%	20	4%	20	4%	470	90%	520	100,0%

On demandait d'écrire une fonction `creerPartitionEnSingletons(n)` qui renvoie un tableau `parent` représentant la partition de `n` éléments en `n` singletons. Les candidats qui ont simplement renvoyé `range(n)` n'ont pas reçu tous les points.

Erreurs classiques :

— Certains candidats ont tenté d'utiliser `[]*n` pour créer une liste à `n` éléments ;
— d'autres ont créé une liste avec `[[]]*n` et utilisé `append(i)` dans une boucle, renvoyant `[n,n,...,n]` au lieu de `[1,2,...,n]`.

Question 9 :

0		<0,5		≥0,5		1		Total	
15	3%	2	0%	27	5%	476	92%	520	100,0%
23	4%	4	1%	56	11%	437	84%	520	100,0%
83	16%	3	1%	89	17%	345	66%	520	100,0%

Les lignes correspondent respectivement à l'implémentation, à la complexité, et à l'exemple.

On demandait d'écrire une fonction `représentant(parent,i)` qui renvoie le représentant du second argument, sa complexité dans le pire des cas, et un exemple qui réalise cette complexité.

Erreurs classiques :

1. Code :
   — Certains candidats ont tout simplement retourné `parent(i)`, sans faire d'appel récursif ou de boucle ;
   — d'autres ont mal choisi la condition d'arrêt, créant des boucles qui ne commencent pas ou qui ne terminent jamais ;
   — enfin, certains ont fait des erreurs aux bornes des boucles, renvoyant par exemple le fils du représentant au lieu du représentant lui-même.
2. Complexité :
   — Certains candidats ont justifié correctement une complexité en $O(m)$;
   — d'autres ont trouvé la complexité correcte en $O(n)$ mais sans la justifier.
3. Exemple :
   — Certains candidats ont choisi un tableau `parent` correspondant à un cycle ;
   — d'autres ont donné le bon tableau, sans préciser pour quel argument la complexité maximale était atteinte.

Question 10 :

0		<0,5		≥0,5		1		Total	
15	3%	2	0%	36	7%	467	90%	520	100,0%

On demandait d'écrire une fonction `fusion(i,j)` qui fusionne les groupes contenant `i` et `j`, respectivement. Le sujet précisait la méthode à employer : calculer les représentants `p` et `q` des deux groupes, et assigner `parent[p] = q`.

Erreurs classiques :

- Certains candidats se sont contentés d'assigner `parent[i]=j` ;
- d'autres n'ont cherché qu'un seul représentant : `parent[représentant(parent,i)] = j` ;
- enfin, certains ont rendu un algorithme correct, mais ont renvoyé le tableau obtenu, ce qui n'était pas demandé.

Question 11 :

0		<0,5		≥0,5		1		Total	
364	70%	11	2%	41	8%	104	20%	520	100,0%

On demandait d'illustrer la mauvaise performance de l'algorithme en proposant une suite de $(n-1)$ fusions dont l'exécution à partir de la partition en n singletons prenait un temps quadratique en n.

Erreurs classiques :

- Certains candidats se sont contentés d'affirmer qu'appeler n fois un programme linéaire menait nécessairement à une complexité quadratique ;
- d'autres ont correctement expliqué la complexité quadratique comme $\sum_{i=1}^{n} i$, mais se sont trompés dans l'ordre des fusions à effectuer ;
- enfin, certains ont donné la bonne suite de fusions, mais sans justifier la complexité obtenue.

Question 12 :

0		<0,5		≥0,5		1		Total	
48	9%	42	8%	53	10%	377	73%	520	100,0%
166	32%	14	3%	38	7%	302	58%	520	100,0%

La première ligne correspond à l'implémentation, la deuxième à la complexité.

On demandait d'écrire une nouvelle fonction `representant(parent,i)` qui modifie le tableau `parent` pour que tous les ancêtres de `i` aient le représentant de `i` comme parent direct. Il était aussi demandé de justifier en quoi cette modification est "gratuite" du point de vue de la complexité.

Erreurs classiques :

- Certains candidats n'ont modifié que `parent[i]` ;
- d'autres n'ont remonté qu'une génération (avec `parent[i] = parent[parent[i]]`) ;
- enfin, certains ont choisi des versions récursives qui ne modifiaient `parent` qu'après le `return`.

Question 13 :

0		<0,5		≥0,5		1		Total	
158	30%	96	18%	233	45%	33	6%	520	100,0%

On demandait d'écrire une fonction `listeDesGroupes(parent)` qui renvoie les différents groupes sous la forme d'une liste de liste. Il y avait peu d'indications sur la manière de procéder, ce qui a mené à une grande diversité dans les solutions proposées, au niveau de la complexité, de la simplicité, et de l'élégance du code.

Erreurs classiques :

— Certains candidats ont (mal) utilisé des dictionnaires ;

— d'autres ont obtenu des complexités déraisonnables.

Question 14 :

0		<0,5		≥0,5		1		Total	
289	56%	102	20%	98	19%	31	6%	520	100,0%
470	90%	15	3%	23	4%	12	2%	520	100,0%

La première ligne correspond à l'implémentation, la deuxième à la complexité.

On demandait d'écrire une fonction `coupeMinimumRandomisee(reseau)`, qui implémente l'algorithme décrit dans le sujet. Cet algorithme était complexe et peu de candidats ont rendu des fonctions correctes. Ceux qui ont su faire ressortir la structure de leur programme ont été récompensés, même si leur code comportait des erreurs mineures. Très peu de candidats ont compris l'analyse de la complexité amortie que permettait la question 12.

Question 15 :

0		<0,5		≥0,5		1		Total	
424	82%	21	4%	38	7%	37	7%	520	100,0%

On demandait d'écrire une fonction `tailleCoupe(reseau, parent)` qui calcule le nombre de liens coupés par la partition représentée par `parent`.

Erreurs classiques :

— Certains candidats ont compté deux fois (voire quatre fois) chaque lien coupé ;

— d'autres ont proposé des algorithmes avec des complexités cubiques ou d'ordre 4.

Question 16 :

0		<0,5		≥0,5		1		Total	
90	17%	15	3%	56	11%	359	69%	520	100,0%

On demandait d'écrire une requête qui renvoie les identifiants des amis de l'individu dont l'identifiant est `x`. La plupart des candidats ont répondu correctement. Les requêtes inutilement complexes, utilisant `HAVING` et `GROUP BY` sans raison, ou multipliant les `JOIN` inutiles, n'ont pas reçu tous les points.

Erreurs classiques :

— Certains candidats ont renvoyé chaque individu deux fois, par exemple avec `SELECT * FROM LIENS WHERE (id1=x OR id2=x)`.

Question 17 :

0		<0,5		≥0,5		1		Total	
144	28%	25	5%	71	14%	280	54%	520	100,0%

On demandait d'écrire une requête qui renvoie les noms et prénoms des amis de l'individu dont l'identifiant est `x`.

Question 18 :

0		<0,5		≥0,5		1		Total	
363	70%	19	4%	64	12%	74	14%	520	100,0%

On demandait d'écrire une requête qui renvoie les identifiants des individus qui sont amis avec au moins un ami de l'individu dont l'identifiant est `x`. Les requêtes inutilement complexes, utilisant `HAVING` et `GROUP BY` sans raison, ou multipliant les `JOIN` inutiles, n'ont pas reçu tous les points.

Erreurs classiques :

— Certains candidats ont utilisé le test `id2=id2` au lieu de `L1.id2=L2.id2`.