4. INFORMATIQUE

4.1. Informatique pour tous

Le sujet d'informatique commune portait cette année sur des techniques algorithmiques autour du thème des nombres premiers.

L'épreuve abordait un large spectre des notions vues durant les deux années de préparation des candidats.

Remarques générales

- La présentation générale de la très grande majorité des copies est satisfaisante.
- Le jury a été surpris de relever beaucoup d'erreurs de calcul parfaitement incongrues à ce niveau de formation. La simplification d'une somme géométrique comme $\sum_{i=0}^{N-1} 2^i$ pose des problèmes importants à plus d'un tiers des candidats. Bien qu'il ne s'agisse pas d'une épreuve de mathématique, on est en droit d'attendre des candidats une aisance calculatoire minimale.
- On a remarqué cette année dans beaucoup de copies un mélange entre des notations propres aux mathématiques (parties entières, racines...) et le code Python. Cela est généralement sanctionné : un algorithme en python n'est pas du pseudo-code, il faut utiliser les opérateurs du langage.
- Nous souhaitons insister sur le fait que la syntaxe **L=L+[a]** est beaucoup moins efficace pour ajouter un élément à la fin d'une liste que **L.append(a)**. Il serait souhaitable que les étudiants privilégient systématiquement cette deuxième solution au cours de leur formation.
- La gestion efficace des booléens n'est pas encore très assurée : on a par exemple croisé beaucoup plus de **if L[i]!=False** que de **if L[i]** … pourtant équivalents si **L[i]** est un booléen.
- Une remarque récurrente année après année : certains candidats semblent n'avoir aucune notion de la complexité algorithmique quand ils écrivent leurs programmes. Faire appel à une fonction de complexité significative comme **Pi** ou **erato_iter** au cœur d'une double boucle devrait pourtant leur poser un problème... surtout quand il existe une solution beaucoup plus simple et facilement accessible.
- Dans certaines copies, on peut parfois lire plusieurs solutions pour une question. Cette démarche pourrait être appréciée dans un autre contexte, mais dans le cadre d'une épreuve de concours en temps limité, le jury s'attend surtout à lire une solution qui fonctionne, si possible lisible et simple. Par ailleurs, le jury n'a pas à choisir entre une solution juste et une solution fausse : la notation se fera toujours sur la mauvaise.

Remarques particulières

Question 1. Généralement correcte, mais **numpy** et **matplotlib** n'ont rien à voir avec le module **math**...

Question 2. Beaucoup de confusions sur l'écriture des flottants… e-5, 10e-5, ou 10^-5 ne définissent pas 10^{-5} en python. Par ailleurs, l'affectation de 10^{-8} via rtol=0.00000001 n'est pas une solution élégante.

Question 3. Une application très simple de fonction récursive, généralement comprise.

Question 4. La moitié des candidats environ reconnaît la partie entière du logarithme en base b.

Question 5. Pour obtenir des points sur cette question qui abordait un point essentiel du programme d'informatique, il ne fallait pas se contenter de généralités creuses. il fallait reconnaître une manifestation des **erreurs d'arrondis** due à la **représentation des flottants sur un nombre limité de bits**. Beaucoup n'en ont pas été capables, mettant parfois en cause « l'ordinateur », « le langage python » ou prétendant que « l'addition est moins précise que la multiplication ».

Question 6. Trop de candidats sont incapables de répondre correctement à cette question, souvent à cause d'une mauvaise connaissance de la signification du préfixe giga.

Question 7. Une erreur fréquente : « un booléen doit être codé au minimum sur 2 bits, car il ne peut prendre que deux valeurs ». La notion de bit informatique est donc mal comprise de ces candidats.

Question 8 : Cette question de traduction d'un algorithme fourni en pseudo-code a été traitée de manière généralement décevante. Entre les initialisations fausses de la liste de booléens, les boucles qui ne commencent ou ne finissent pas avec la bonne valeur de l'incrément, des divisions euclidiennes sur les booléens (!) et, bien sûr, les erreurs d'indices lors de l'appel d'un élément de la liste (**erato_iter[i]** contenant l'information sur la primalité du nombre **i+1**...), on peut estimer à 10 % environ les candidats ayant proposé une solution parfaitement correcte de cet algorithme… On peut légitimement espérer mieux. Pour cela, une attention plus grande aux détails est nécessaire. Voir en annexe un exemple de bon code pour cette question.

Question 9. Pour traiter cette question correctement, il fallait bien entendu avoir proposé un parcours optimal à la question précédente.

Question 10. On a pu lire quelques très bonnes solutions pour cette question.

Question 11. Une grande majorité de candidats comprennent que $A=\sum_{i=0}^{N-1} 2^i$. Calculer correctement cette somme a par contre posé de nombreuses difficultés… cependant, on peut remarquer qu'en maîtrisant le principe du codage binaire, calculer la somme était inutile.

Question 12. On a relevé des confusions entre les opérateurs **%** et **//** . A ce propos, on a lu beaucoup trop de **if a/i==int(a/i)**, ce qui est une solution fausse pour vérifier que **a** est divisible par **i**.

Question 13. Très peu de candidats ont bien compris la valeur que devait prendre l'argument de **bbs**. Bravo à eux, ce point était subtil ! Par ailleurs, la condition de pseudo-primalité a

donné lieu à nombre de solutions fausses, dans lesquelles l'algorithme renvoyait un nombre dès que l'une des quatre conditions de divisibilité était satisfaite.

Question 14. Un seul appel à **erato_iter** judicieux évitait de reprogrammer des tests de divisibilité pour tous les nombres testés.

Question 15. La solution la plus simple consistait à faire appel une seule fois à **erato_iter**. De nombreux candidats se sont lancés dans des constructions trop complexes et coûteuses, qui ne respectaient pas la contrainte imposée par le sujet (complexité linéaire).

Question 16. On demandait de tester l'inéquation fournie à partir d'un nombre particulier, pas à partir de 0. L'appel à **Pi** dans la boucle a été sanctionné, à plus forte raison quand la liste de listes renvoyée par **Pi** était comparée à un flottant...

Question 17. Question en général bien réussie.

Question 18. Certains candidats croient que la méthode des trapèzes possède une complexité différente de celle des rectangles.

Question 19. Attention à l'utilisation de **range**. Trop de candidats ne semblent pas du tout gênés d'écrire **range(a,b,pas)** avec **a** et **b** et **pas** des flottants. De plus on a lu des solutions de la méthode des rectangles à gauche (certes celle exposée en cours le plus souvent, mais il faut savoir s'adapter).

Question 20. Les candidats ont souvent bien traité deux cas sur les trois selon la valeur de x, mais ont oublié le troisième.

Question 21. Question très peu traitée, et souvent incomprise.

Question 22. Question difficile, très rarement traitée.

Question 23. Question très rarement traitée.

Question 24. Attention à la solution tentante qui consistait à faire appel à une fonction de calcul de factorielle à chaque étape de la boucle... cela fait exploser la complexité.

Question 25. L'erreur la plus courante a été de dire que les deux tables possédant un attribut de même nom, celui-ci ne pouvait servir de clé primaire pour l'une des tables : cela n'a rien à voir avec la définition d'une clé primaire.

Question 26 : Peu de candidats ont perçu la nécessité d'une sous-requête (ou d'un **EXCEPT**) pour la deuxième requête. En revanche, la syntaxe d'une jointure semble mieux maîtrisée que les années précédentes.

Perles diverses

Comme chaque année, le jury a été ému de trouver dans les copies quelques perles inattendues, auxquelles nous avons décerné les prix suivants :

- Prix « perdu dans la ville » : « Le problème est que les erreurs d'arrondissement s'accumulent » ;
- Prix « Poésie et informatique » : « Le plus petit espace mémoire possible est le facteur spatial » ;
- Prix « le SQL Low-Cost » : « SELECT DISCOUNT FROM ordinateurs ».

ANNEXE : Proposition de code correct pour Q8

```
def erato_iter(N):
     liste_bool = N * [True]
     liste_bool[0] = False
     i = 2
     while i**2 <= N:
         if liste_bool[i - 1]:
             for k in range(2, N//i + 1):
                 liste_bool[k*i - 1] = False
         i += 1
     return liste_bool
```

4.2. Informatique option — filière MP

Généralités.

Le sujet traite d'une méthode de réduction des automates.

Il fait appel, d'une part, à la notion formelle d'automate et, d'autre part, à des structures informatiques complexes que le candidat doit manipuler. L'ensemble permet de bien évaluer l'acquisition du programme des deux années de classe préparatoire.

Les candidats abordent l'ensemble des questions dans leur grande majorité.

La présentation des copies est globalement satisfaisante.

Nous avons pu constater peu d'efforts de rédaction des quelques questions théoriques.

Beaucoup de candidats ne donnent pas d'arguments, se contentent d'arguments superficiels ou ne citent pas les résultats précédemment montrés.

Première partie : Premiers exemples.

Questions 1-4. Il s'agit ici de décrire le langage accepté par un automate sous forme qualitative ou rationnelle. Ces notions sont globalement comprises.

Question 5. Les candidats montrent qu'ils ont compris la représentation informatique d'un automate choisie ici.