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

Rapport de M. Didier CASSEREAU, correcteur.

1. Bilan général

A titre de rappel, cette épreuve n'est corrigée que pour les candidats admissibles. Pour ma part, je n'ai corrigé que la filière MP.

Cette année le nombre total de candidats admissibles dans cette filière est de 331. La note moyenne est de 12,08 avec un écart-type de 3,75. 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 0,8/20 et la note maximale 19,5/20. Une copie a obtenu la note éliminatoire de 0,8/20. Quatre copies n'ont pas été notées, les candidat(e)s ne s'étant pas présenté à l'épreuve.

	Série 1		Série 2		Série 3		Série 4		**Synthèse**	
$0 \leq N < 4$	0	0,0%	3	3,3%	1	1,4%	0	0,0%	**4**	**1,2%**
$4 \leq N < 8$	13	12,5%	16	17,8%	12	16,2%	8	12,7%	**49**	**14,8%**
$8 \leq N < 12$	30	28,8%	32	35,6%	20	27,0%	19	30,2%	**101**	**30,5%**
$12 \leq N < 16$	47	45,2%	21	23,3%	28	37,8%	27	42,9%	**123**	**37,2%**
$20 \leq N \leq 20$	14	13,5%	18	20,0%	13	17,6%	9	14,3%	**54**	**16,3%**
Total	104	100,0%	90	100,0%	74	100,0%	63	100,0%	**331**	**100,0%**
Epreuve complète*	37	35,2%	29	32,2%	26	34,2%	27	42,2%	**119**	**35,5%**

* *Épreuve complète* signifie ici que le candidat a abordé toutes les questions de l'énoncé et obtenu une note non nulle à chacune des 12 questions.

	Série 1	Série 2	Série 3	Série 4	**Synthèse**
Nombre de copies	104	90	74	63	**331**
Note minimale	4,1	0,8	3,5	5,1	**0,8**
Note maximale	19,5	19,2	19,2	18,4	**19,5**
Note moyenne	12,35	11,53	12,08	12,43	**12,08**
Ecart-type	3,44	4,18	3,89	3,42	**3,75**

Le langage de programmation choisi par les candidats est dominé par Maple (qui est majoritairement bien orthographié, avec encore quelques candidat(e)s qui s'obstinent à composer en Mapple) avec 52,0% des copies, suivi ensuite par Caml (26,9% des copies)

et Python (12,7% des copies). On trouve enfin quelques copies en C/C++ (7,3%), Pascal (0,3%) et autres (0,9%, incluant par exemple Mathematica). Le tableau ci-dessous illustre cette répartition des langages choisis par les candidats.

Par rapport aux années antérieures, la tendance s'est inversée entre C/C++ et Python.

Maple	Caml	C/C++	Python	Pascal	Java	Autres	**Total**
172	89	24	42	1	0	3	**331**
52,0%	26,9%	7,3%	12,7%	0,3%	0,0%	0,9%	**100,0%**

2. Commentaires

Cette année le sujet porte sur la détermination de zones rectangulaires monochromatiques dans une image, les couleurs étant ici restreintes au noir et au blanc, représentées par les valeurs numériques 0 et 1. Le problème consiste donc à identifier les zones contigües de cases de valeur 0, dans un tableau à une ou deux dimensions. Le problème comporte 4 parties :

- la première partie permet d'étudier l'aspect 1D,
- la deuxième partie consiste à passer du problème 1D au problème 2D,
- la troisième partie consiste à mettre en place les bases d'un algorithme optimisé pour le problème 2D,
- enfin la quatrième partie permet de conclure.

Tout au long du problème, les candidats sont invités à écrire des codes optimisés en nombre d'opérations, il leur est également clairement demandé d'être en mesure d'évaluer la complexité de leurs codes.

Les candidats sont invités à passer outre certaines contraintes liées au langage de programmation (l'indexation des tableaux qui commence à 0 et non à 1 par exemple). Cette directive a été largement respectée par la grande majorité des candidats.

L'objectif de cette épreuve est d'évaluer la capacité des candidats à écrire un code informatique permettant de résoudre un problème algorithmique donné. Il est important de proscrire tout ce qui relève du calcul formel ; dans le cas contraire le candidat prend le risque de se voir sanctionné par rapport à d'autres copies dans lesquelles on peut trouver un code complet et juste.

J'invite en particulier les candidats composant dans des langages tels que Mathematica à être très vigilants sur ce point. J'ai déjà mentionné ce genre de réserve dans mes rapports des années précédentes, et je tiens à le rappeler une fois encore cette année.

L'évaluation d'un code informatique repose sur plusieurs éléments à mon avis 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 :

- je sais bien que cette épreuve se déroule en temps limité, néanmoins j'attire l'attention des candidats sur l'importance de la présentation et de la lisibilité des copies ! en cas de doute ce n'est pas au correcteur d'évaluer ce que le candidat *aurait pu répondre* ; pour ma part, je juge exclusivement sur la base de ce qui est objectivement inscrit et lisible sur chaque copie !
- 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,
- certains candidats définissent des fonctions intermédiaires (c'est plutôt bien), mais quand elles s'appellent toutes aux, la lecture du code devient rapidement plus complexe ; cela ne coûte rien de choisir un nom plus représentatif de la tâche exécutée par la fonction ! cette remarque vise essentiellement les candidats composant en Caml,
- 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,
- à l'exception de Caml, certaines notations n'ont pas de sens dans un langage informatique, telles que 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,
- 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,
- 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,
- dans une boucle, par exemple indicée par une variable entière `i`, on ne peut pas manipuler une variable nommée `ti` en espérant que le nom de cette variable va suivre l'évolution de `i` ; cette erreur est assez fréquente et aboutit inévitablement à un code qui n'a pas grand sens et qui n'a surtout aucune chance de fonctionner,
- une erreur fréquente consiste à incrémenter le compteur d'une boucle `for` ; or le mécanisme de la boucle implique déjà cette incrémentation, de sorte que cela conduit à incrémenter deux fois, ce qui n'est en général pas ce que l'on cherche à faire ; inversement, les boucles `while` doivent explicitement contenir une instruction permettant d'incrémenter le compteur de la boucle, faute de quoi on risque de rentrer dans une boucle infinie,
- certains opérateurs logiques sont utilisés de manière un peu approximative, comme par exemple >< à la place de <> ou ! =, ou encore =< à la place de <=,
- 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.

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	10	3,0%	60	17,9%	58	17,3%	207	61,8%	335	100,0%

Cette question consiste à déterminer le nombre de 0 contigüs à partir d'un indice donné du tableau. Elle ne présente pas de difficulté particulière, elle peut être traitée par une simple boucle `while` ou `for`.

Globalement cette question a été plutôt bien traitée en moyenne. Une erreur courante consiste à mettre en place une boucle `while` sans protection contre le débordement de tableau.

Question 2 :

	0		$< 0,5$		$\geq 0,5$		1		Total	
Question 2	9	2,7%	86	25,7%	101	30,1%	139	41,5%	335	100,0%

Cette question n'est finalement qu'une application de la question précédente, dans la mesure où il suffit d'appeler cette fonction précédente autant de fois que nécessaire pour parcourir l'ensemble du tableau, et calculer la plus grande des valeurs ainsi collectées.

A nouveau cette question a été traitée plutôt correctement, avec parmi les erreurs les plus courantes :

- oubli ou erreur d'écriture de l'incrémentation du compteur de boucle (boucle `while`),
- pas de protection contre le débordement de tableau,
- cas assez fréquent de boucle infinie lorsque la fonction précédente retourne 0.

Question 3 :

	0		$< 0,5$		$\geq 0,5$		1		Total	
Question 3	43	12,8%	140	41,8%	128	38,2%	24	7,2%	335	100,0%

Cette question généralise le problème au cas 2D et consiste à calculer l'aire du plus grand rectangle de 0 partant d'un indice (`i,j`) donné. La complexité de cette question repose sur le fait qu'il ne suffit pas d'explorer la ligne `i` et la colonne `j`, il faut aussi tenir compte du fait que certaines lignes de 0 peuvent être plus ou moins longues à l'intérieur de la zone explorée.

La question n'était pas simple, elle a été moyennement réussie. Cette question illustre le fait qu'il est impératif d'écrire les choses de manière simple et claire, faute de quoi on aboutit à un code illisible et incompréhensible. Le soin dans la rédaction de la copie joue également un rôle très important.

Question 4 :

	0		$< 0,5$		$\geq 0,5$		1		Total	
Question 4	16	4,8%	19	5,7%	109	32,5%	191	57,0%	335	100,0%

Cette question amène le candidat à réfléchir à une approche naïve du problème de calcul d'un rectangle de 0 et d'aire maximale. L'idée consiste simplement à itérer la fonction précédente pour toutes les cases du tableau, et à évaluer la plus grande des aires ainsi collectées.

Cette question très naïve ne pose pas de difficulté particulière et a été plutôt bien traitée. Il convient cependant de rappeler aux candidats que les réponses doivent être présentées de manière claire et non ambigüe.

Question 5 :

	0		$< 0,5$		$\geq 0,5$		1		Total	
Question 5	13	3,9%	95	28,4%	118	35,2%	109	32,5%	335	100,0%

Cette question consiste à calculer un tableau intermédiaire permettant de stocker le nombre de 0 contigüs au-dessus de chaque case du tableau. Le point essentiel est ici d'écrire un code efficace, il faut donc remarquer que chaque ligne de ce tableau intermédiaire peut très facilement se déduire de la ligne précédente, sans calculs particuliers. A condition toutefois de traiter séparément la première ligne bien entendu. L'efficacité du code présenté par les candidats a été un critère très important d'évaluation pour cette question.

Question 6 :

	0		$< 0,5$		$\geq 0,5$		1		Total	
Question 6	122	36,4%	0	0,0%	10	3,0%	203	60,6%	335	100,0%

Dans la grande majorité des cas, le code rédigé à la question précédente aboutit à une complexité en puissance 2 ou 3 de la taille du tableau traité. Cette question requiert de la part du candidat un certain recul par rapport au code présenté à la question précédente, puisqu'il doit évaluer la complexité de son code.

A noter que les copies qui n'ont pas satisfait la condition d'efficacité à la question 5, mais qui ont su conclure objectivement et honnêtement sur leur complexité effective, n'ont pas été sanctionnées pour cette question.

Question 7 :

	0		$< 0,5$		$\geq 0,5$		1		Total	
Question 7	24	7,2%	28	8,4%	170	50,7%	113	33,7%	335	100,0%

Cette question présente un algorithme de calcul d'un histogramme des hauteurs, à partir du tableau initial de 0 et de 1. Le candidat doit alors justifier que l'algorithme donne effectivement les valeurs attendues, avec une complexité linéaire illustrée sur un exemple particulier.

Cette question ne présente pas de réelle difficulté, il convient cependant d'insister sur le fait que la rédaction ne peut souffrir d'à peu près, il est vraiment indispensable d'être très précis, et de décortiquer étape par étape ce qui se passe réellement. Cette remarque vaut principalement pour l'analyse de l'exemple présenté, pour lequel il faut bien comprendre la spécificité de l'algorithme, en particulier dans le cas de la deuxième moitié du tableau.

A titre de rappel, un algorithme qui requiert n opérations pour aboutir au résultat a une complexité finale de type quadratique et non linéaire, dès lors que l'opération effectuée à l'indice de boucle `i` effectue `i` sous-opérations. Certains candidats ont indiqué qu'il y avait une erreur dans l'énoncé de l'algorithme. Ces candidats n'ont pas compris en quoi l'algorithme était effectivement très efficace en terme de nombre d'opérations.

Question 8 :

	0		$< 0,5$		$\geq 0,5$		1		Total	
Question 8	25	7,5%	4	1,2%	5	1,5%	301	89,9%	335	100,0%

Cette question consiste à justifier l'inclusion d'un rectangle de 0 caractérisé par ses `L[i], R[i]` et hauteur, dans l'histogramme introduit en début de partie III. Cette question ne présente vraiment aucune difficulté, et a été très majoritairement correctement traitée.

Question 9 :

	0		$< 0,5$		$\geq 0,5$		1		Total	
Question 9	42	12,5%	60	17,9%	49	14,6%	184	54,9%	335	100,0%

La réponse à cette question repose sur le critère de maximalité de l'aire du rectangle considéré. Ce critère doit en définitive être utilisé pour chacun des 3 aspects à justifier. Une majorité de candidats a traité cette question de manière correcte, il convenait cependant là encore d'être précis et rigoureux.

Question 10 :

	0		$< 0,5$		$\geq 0,5$		1		Total	
Question 10	69	20,6%	36	10,7%	156	46,6%	74	22,1%	335	100,0%

Cette question découle des remarques précédentes. Les points importants sont que le code doit impérativement avoir un comportement linéaire, ce qui exclut certains appels en boucle (par ailleurs inutiles) de fonctions telles que `calculeL()` ou `calculeR()`. Il faut également explicitement faire appel à ces deux fonctions au début du code, ce sont elles qui fournissent les tableaux `L` et `R` nécessaires.

Petit détail : la longueur d'un intervalle situé entre les indices `l` et `r` est `r-l+1` et non `r-l`. A noter également que certains candidats ne savent pas calculer l'aire d'un rectangle connaissant sa longueur et sa hauteur.

Question 11 :

	0		$< 0,5$		$\geq 0,5$		1		Total	
Question 11	105	31,3%	38	11,3%	76	22,7%	116	34,6%	335	100,0%

Cette question ne pose normalement pas de difficulté particulière, elle consiste à mettre ensemble les fonctions écrites dans les questions 5 et 10. A noter ici que la fonction de la question 5 doit être appelée avec comme paramètre l'histogramme et non le tableau `tab2` d'origine.

Cette question arrivant en fin d'épreuve, elle a été soit pas du tout traitée, soit majoritairement traitée correctement.

Question 12 :

	0		$< 0,5$		$\geq 0,5$		1		Total	
Question 12	126	37,6%	48	14,3%	33	9,9%	128	38,2%	335	100,0%

Cette dernière question consiste à prendre un peu de recul par rapport à tout ce qui précède et à évaluer la complexité du code qui en résulte. Là encore les candidats qui ont pu aborder cette question l'ont majoritairement fait de manière correcte.