Composition d'Informatique (2 heures), Filières MP et PC

Rapport de MM. Samuel MIMRAM et Etienne LOZES, correcteurs.

1. Statistiques

Rappellons que cette épreuve n'est corrigée que pour les candidats admissibles. Cette épreuve est commune aux filières MP et PC, mais les copies étant indistinguables, les statistiques présentées intègrent ces deux filières.

$0 \leqslant \mathrm{N} < 4$	44	9 %
$4 \leqslant \mathrm{N} < 8$	157	31 %
$8 \leqslant \mathrm{N} < 12$	207	42 %
$12 \leqslant \mathrm{N} < 16$	80	16 %
$16 \leqslant \mathrm{N} \leqslant 20$	11	2 %
Total	499	100 %
Nombre de copies : 499		
Note moyenne : 8,77		
Écart-type : 3,58		
Note minimale : 0,3		
Note maximale : 19,3		

Les candidats ont majoritairement traité cette épreuve en répondant aux questions dans l'ordre, une minorité ayant omis les questions 9, 10 et 11 pour traiter directement les questions 12 et 13. Au final, 9% des candidats ont terminé cette épreuve, c'est-à-dire ont tenté de répondre aux 15 questions. Le choix du langage de programmation s'est porté sur Maple (88%), Python (10%), et plus rarement Mathematica, Pascal, ou C++.

2. Commentaires généraux

Le sujet portait sur la conception de fonctions de manipulation des permutations de $\{1, \ldots, n\} (n \in \mathbb{N})$ représentées par des tableaux d'entiers. Dans un premier temps, on étudiait leur structure de groupe (reconnaissance, composition, inversion), dans un second temps plusieurs grandeurs et propriétés propres à une permutation donnée (ordre, période d'un élément, transpositions, cycles), et dans un troisième temps on recherchait une algorithmique efficace pour les calculs de l'itérée k-ième d'une permutation et de son ordre.

Presque toutes les questions posées reposaient sur l'écriture d'une fonction dans un langage de programmation laissé au choix du candidat ; pour les questions 4 et 12, une

réponse mathématique était attendue, qui pouvait être complétée par un programme. Plusieurs primitives abstraites de constantes booléenes, de calcul de reste, et surtout de manipulation des tableaux étaient suggérées par l'énoncé. Ces dernières visaient à souligner les contraintes liées à la représentation mémoire des tableaux – notamment les problèmes d'allocation dynamique et de taille mémoire. Ces indications spécifiaient le type de manipulation attendu sur les tableaux. L'utilisation de primitives propres au langage de programmation choisi n'était pas explicitement interdite ; toutefois, le sujet proposant des primitives abstraites pour certaines opérations, il était attendu qu'elles soient utilisées, ou à défaut des primitives équivalentes dans le langage choisi. Les manipulations de tableaux plus avancées et spécifiques au langage de programmation choisi, notamment celles basées sur les correspondances entre listes et tableaux, n'ont pas été sanctionnées lorsqu'elles étaient maîtrisées, mais n'ont pas non plus été récompensées. Tenant compte du libre choix du langage de programmation à cette épreuve, il a été considéré qu'il convenait de valoriser la copie qui dépassait la seule connaissance d'un langage de programmation donné et concevait des solutions facilement transposables à d'autres langages de programmation. Dans cet esprit, les futurs candidats auront profit de s'entraîner à *utiliser les constructions élémentaires* (fonctions, boucles, tableaux...) de leur langage plutôt qu'à chercher à maîtriser les points les plus avancés.

En contrepartie, il est juste de constater qu'une majorité de candidats démontre une certaine familiarité avec le langage de programmation choisi. La minorité de candidats qui commet des erreurs grossières n'en apparaît que plus marginalisée. La correction syntaxique attendue est celle qui permet de reconstituer sans ambiguïté et avec peu d'efforts un code assimilable par un compilateur ou un interpréteur. La meilleure préparation à cette épreuve de programmation reste donc *une préparation pratique sur machine*. Plus concrètement, des erreurs moyennes telles que l'utilisation du `and` pour indiquer une composition séquentielle ont été considérées avec bienveillance cette année, de même que les erreurs minimes comme les oublis de fermeture de bloc d'instructions ou les confusions entre symbole d'affectation et d'égalité. Les erreurs graves, dépassant en réalité la simple erreur de syntaxe, comme le cas assez courant d'utilisation de boucles pour exprimer une conjonction

```
if for i from 1 to n t[i]=i then [..]
```

ont été, elles, sévèrement sanctionnées.

Des confusions de différents ordres entre le monde mathématique et celui informatique ont pu être relevées, notamment le recours aux indices, aux x_i de E_n (bien que l'énoncé indiquât de prendre $x_i = i$ et $E_n = \{1, \ldots, n\}$), voire aux exposants $\mathtt{t}^i$ pour `t` une variable de programme. Ces confusions n'ont en général pas été sanctionnées cette année, sauf dans le dernier cas lorsqu'elles ne consistaient pas en des identifiants de variables étendues, ou des annotations de code, mais à des indications de calcul erronées. Plus rare, mais révélateur d'une confusion répandue par Maple, l'utilisation hasardeuse du mot-clé *infinity* dans

```
for i from 1 to infinity do [..] od;
if i=infinity then [..]
```

qui suppose pour les ordinateurs une capacité de calcul qu'ils n'ont pas.

Les spécificités du langage de programmation choisi, ainsi que les hypothèses implicites (et parfois, de manière bienvenue explicites) sur les conventions adoptées sur ce langage ont pu faire débat. Il est souhaitable que les candidats s'interrogent sur les conventions induites par le choix d'un langage de programmation, qu'ils commentent et explicitent ces conventions, et qu'ils s'autorisent à adopter d'autres conventions, tant qu'elles sont explicitées et effectives sur l'ensemble du sujet.

Entiers machines. Le sujet ne précisait pas si les entiers machines étaient signés ou positifs. En l'absence de précision sur la convention adoptée par le candidat, un test de la forme $x = 0$ à la question 1, au lieu du test tout aussi simple mais plus robuste $x \leq 0$, a été très faiblement pénalisé.

Tableaux : allocation. L'allocation et le test de taille de tableaux étaient définis par l'énoncé. Les copies n'ayant pas utilisé l'allocation, ou ayant émis des hypothèses sur le contenu des tableaux fraîchement alloués ont été sanctionnées.

Tableaux : copie. La copie de tableau a été mise en œuvre par la grande majorité des candidats comme une simple affectation. Elle n'a jamais été sanctionnée. Les très rares copies ayant invoqué une primitive `x :=copy(t)` ou ayant écrit un code effectuant la copie ont été faiblement récompensées.

Tableaux : égalité. L'égalité de tableau n'était pas précisée par l'énoncé ; les langages les plus populaires chez les candidats présentent une égalité de tableau « extensionnelle », et non au sens d'« adresse mémoire », que l'on trouve néanmoins dans de nombreux autres langages de programmations. On a ainsi récompensé les candidats ayant témoigné d'une connaissance de cette subtilité. Le recours à une égalité extensionnelle facilitait en effet la conception du code, notamment à la question 5. En cas d'erreur, (le plus souvent, une erreur d'indice), les candidats ayant eu recours à l'égalité extensionnelle sans émettre de commentaires à ce sujet étaient plus pénalisés que ceux ayant fait la même erreur mais ayant reprogrammé correctement l'égalité extensionnelle.

Tableaux : indiçage. L'indiçage des tableaux à partir de 1 était imposé par l'énoncé ; la très grande majorité des copies a fort justement suivi cette convention. Les quelques copies ayant fait le choix d'un indiçage à partir de 0 ont presque toujours échoué aux questions 2 et 3 (composition et inversion).

Déroutements et récursivité. La plupart des langages choisis offrent des primitives de déroutement (`return, break, continue`, etc) qui permettaient parfois de clarifier le code ou de le rendre plus efficace, notamment aux questions 5, 6, et 7. De manière similaire, l'emploi de fonctions récursives simplifiait le code de la question 12. Il paraît

souhaitable que les candidats aient une meilleure connaissance de ces possibilités, car les rares copies les ayant exploitées ont généralement mieux répondu à ces questions.

Primitives propres au langage choisi. Comme indiqué précédemment, un certain nombre de primitives de langages permettaient parfois d'écrire un code plus concis, notamment le test booléen `k in t` en Python. Ce type de programmation n'a été sanctionné qu'en cas d'erreur, mais seuls les candidats capables de convaincre de leur maîtrise de la programmation devraient se l'autoriser. A titre d'exemple, certains candidats ayant opté pour Maple ont été pénalisés pour avoir converti un tableau en une séquence afin de déterminer si deux tableaux avaient le même ensemble de valeurs.

3. Commentaires par question

Un aperçu de la réussite par question est donné dans le tableau ci-dessous. Chaque question était notée sur 4 (avant pondération par le barème), la note maximale étant 5 (bonus). La première ligne indique le pourcentage de candidats ayant obtenu une note ≥ 4, la deuxième la moyenne, et la troisième le pourcentage de candidats ayant obtenu un zéro.

Question	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Réussite %	15	41	77	79	18	52	32	53	2	0,4	15	18	14	46	0,4
Moyenne	1,9	3,3	3,9	3,6	2,0	2,9	2,7	2,9	0,4	1,1	1,4	0,8	1,0	2,0	0,2
Zéro %	24	4	14	3	17	10	15	26	65	24	50	77	62	46	86

Question 1 : La solution juste la plus souvent rencontrée consistait à tester la surjectivité sur $\{1, \ldots, n\}$, comme suggéré par l'énoncé, l'autre solution correcte parfois rencontrée étant un test d'injectivité avec vérification que $\texttt{Im}(t) \subseteq \{1, \ldots, n\}$. De nombreux candidats n'ont testé qu'une seule condition dans la deuxième approche. Bien que cela fût inutile, certains candidats ont écrit, assez rarement avec succès, une procédure de tri visant à simplifier le test de surjectivité de la première approche. Un débordement de tableaux était possible pour certaines mises en œuvre de la première approche, comme dans le code érroné ci-dessous, ce qui coûtait au candidat un quart des points :

```
estPermutation := proc(t)
   local n,u ;
   n :=taille(t) ;
   u :=allouer(n) ;
   for i from 1 to n do u[i] :=0 od ;
   for i from 1 to n do u[t[i]] :=1 od ;
   for i from 1 to n do if u[i]=0 then return faux fi od ;
   return vrai
end ;
```

Question 2 : Bien que facile, cette question n'a pas rapporté le maximum de points à de nombreux candidats qui ont oublié d'allouer le tableau contenant le résultat, ou ont supposé que la taille des tableaux était contenue dans une variable globale. Quelques

copies ont effectué la composition $u \circ t$ au lieu de celle attendue.

Question 3 : Cette question a été généralement bien traitée, avec un nombre important de solutions efficaces en temps linéaire, qui ont été récompensées.

Question 4 : Cette question du domaine des mathématiques a généralement été bien traitée, les erreurs les plus courantes étant de donner une transposition comme exemple d'ordre 1 et la permutation « miroir » $i \mapsto n - i + 1$ comme exemple d'ordre n.

Question 5 : Cette question, comme la question 1, présentait de nombreux écueils, notamment sur le test d'arrêt de boucle, qui nécessitait de tester l'égalité de deux tableaux. Ce test a parfois été confondu avec la boucle d'itération principale, comme dans le code érroné ci-dessous :

```
ordre := proc(t)
   local i,k,u ;
   k :=1 ; u :=t ;
   for i from 1 to n do
     while t[i]<>i do k :=k+1 ; u :=composer(t,u) od
   od ;
   return k
end
```

Certains candidats ont choisi d'écrire une fonction auxiliaire pour la condition d'arrêt, et ont été récompensés pour leur code clair et modulaire.

Question 6 : Cette question a été généralement mieux traitée que la question 5, car la condition d'arrêt était plus simple. Les écueils communs aux questions 5 et 7 restaient cependant assez fréquents : le calcul des itérées de `t` dans une variable auxiliaire `u` était parfois mis en œuvre par `u :=composer(t,t)`, ou encore la valeur de retour différait de ± 1 de celle attendue soit sur toutes les entrées, soit seulement sur l'identité. Pour éviter ces erreurs, on ne saurait trop recommander aux candidats de tester mentalement leur code sur des exemples.

Question 7 : En plus des écueils communs aux questions 5 et 6, cette question présentait une difficulté supplémentaire sur la condition d'arrêt. Bien que la plupart des candidats aient choisi une condition très large (n, voire ordre(t) itérations), certains candidats ne faisaient que *periode(i)* -1 itérations, et ne parcouraient donc pas la totalité de l'orbite. Certains candidats ont proposé des solutions particulièrement efficaces - parcourant la suite des $t^k(i)$ sans faire appel à `composer`, et renvoyant le résultat aussitôt que celui-ci pouvait être déterminé - et ont dans ces conditions été récompensés pour leur code efficace.

Question 8 : Cette question admettait une solution simple qui consistait à comparer simplement à 2 le nombre de $i \in \{1, \ldots, n\}$ tels que $t(i) \neq i$. De nombreux candidats ont

trouvé cette solution efficace et ont été récompensés.

Question 9 : Cette question n'a que très rarement été traitée correctement. La plupart des candidats ont pensé qu'un cycle était une permutation dont exactement un élément serait de période non unitaire. Les quelques candidats qui ont noté qu'une telle définition n'admettrait aucun exemple de permutation cyclique ont été moins pénalisés que les autres. Les candidats ayant produit un code faux, mais ayant adopté la définition attendue de permutation cyclique, ont été encore moins pénalisés.

Question 10 : Il s'agissait sans doute de la question la plus difficile du sujet. La plupart des candidats ont donné une solution simple en temps quadratique, voire cubique. Ceux ayant relevé le problème ont été un peu moins pénalisés que ceux qui affirmaient avoir une solution en temps linéaire. Ceux ayant envisagé une solution plus complexe, mais néanmoins en temps non linéaire - par exemple en $O(\max(periode) \times n)$ - ont été encore un peu moins pénalisés. La solution attendue, consistant à parcourir chaque orbite deux fois (une pour calculer sa taille, l'autre pour inscrire cette taille dans le tableau des résultats) a été trouvée par de rares excellents candidats. Les candidats qui ont produit un code effectuant un parcours par orbite, mais faux, ont été moins pénalisés que ceux cités précédemment.

Question 11 : Le code à produire devait exploiter l'identité $t^k(i) = t^r(i)$ pour r le reste de k modulo $periode(i)$, et faire appel au tableau des périodes de la question 10. La question présentait de nombreux écueils sur le calcul de $t^r(i)$. La plupart des candidats ont cherché à calculer t^r par appel à `composer`, mais ont soit oublié de réinitialiser le tableau contenant t^r à chaque nouveau r, soit mal géré le cas $r = 0$, ce qui a conduit à un faible taux de réussite. Les copies ayant calculé $t^r(i)$ sans appeler `composer` ont été récompensées pour leur code efficace.

Question 12 : Pour de nombreux candidats, l'ordre d'une permutation ne peut excéder sa taille - ce qu'ils ont voulu parfois exploiter à la question 5. Un sixième des candidats a répondu correctement à cette question, qui relevait du domaine des mathématiques.

Question 13 : Cette question, bien que classique, présentait de nombreux écueils - soit sur l'utilisation de variables auxiliaires pour réaliser une double affectation, soit sur la condition d'arrêt, soit sur le résultat retourné. Les rares copies ayant utilisé la récursivité - ce que n'interdisait pas le sujet - ont presque toutes produit un code correct. Une minorité des copies ayant traité cette question a été pénalisée pour ne pas avoir suivi la consigne de l'énoncé qui imposait un calcul par l'algorithme d'Euclide.

Question 14 : Question très facile pour qui l'abordait. Dans leur précipitation, certains candidats ont néanmoins écrit un code approximatif, comme $\frac{a*b}{pgcd(a;b)}$ au lieu de $a * b/pgcd(a, b)$.

Question 15 : Question ouverte abordée par relativement peu de candidats, parmi lesquels environ la moitié n'a pas donné un code calculant l'ordre d'une permutation. La

solution correcte le plus souvent rencontrée, consistant à calculer le ppcm de *toutes* les périodes par accumulation, a été modérément récompensée. Un calcul basé sur une approche diviser pour régner, bien que coûtant lui aussi $n-1$ appels à `ppcm`, a été un peu mieux récompensé. La méthode la plus efficace proposée par certains candidats consistait à remplacer la boucle

`for i from 2 to n do o :=ppcm(o,per[i]) od` par

```
for i from 2 to n do if reste(o,per[i])<>0 then o :=ppcm(o,per[i]) fi od
```

Cette solution donnait le maximum de points, bien qu'on aurait pu encore réduire le nombre d'appels à `ppcm`, par exemple en triant le tableau des périodes par ordre décroissant.