ÉCOLE POLYTECHNIQUE

ÉCOLE SUPÉRIEURE DE PHYSIQUE ET CHIMIE INDUSTRIELLES

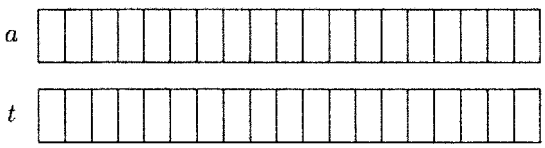
CONCOURS D'ADMISSION 2002 **FILIÈRES PSI ET PT**

ÉPREUVE FACULTATIVE D'INFORMATIQUE

(Durée : 2 heures)

Avertissement On attachera une grande importance à la clarté, à la précision, à la concision de la rédaction. On précisera aussi le langage de programmation utilisé. L'utilisation des calculatrices **n'est pas autorisé** pour cette épreuve.

Un capteur mesure pendant plusieurs années le rayonnement gamma émis par un lointain pulsar. Pour chaque gamma reçu, repéré par son rang i ($0 \leq i < n$), on mesure son énergie a_i et l'instant t_i de sa détection. L'unité d'énergie est le kilo électron-volt (keV), l'unité de temps est le dixième de seconde (1/10s), l'origine des temps correspond au début de la campagne de mesures. On supposera $n > 0$. Pour tout i, la quantité a_i est un entier naturel ($a_i \in \mathbf{N}$). Les valeurs a_i sont rangées dans un tableau a de n éléments de type entier. Ces mesures n'ont pas lieu à intervalles réguliers. On note les dates t_i (exprimées en 1/10s, $t_i \in \mathbf{N}$) dans un autre tableau t de n éléments de type entier. Pour tout i et j tels que $0 \leq i < j < n$, on a donc $t_i < t_j$.



Le temps d'exécution $T(f)$ d'une fonction f de la variable b est le nombre d'opérations élémentaires (addition, soustraction, multiplication, division, affectation, etc) nécessaires au calcul de $f(b)$. Lorsque ce temps d'exécution dépend d'un paramètre n, il sera noté $T_n(f)$. On dit que la fonction f s'exécute :

- en temps linéaire en n, s'il existe $K > 0$ tel que pour tout n, $T_n(f) \leq Kn$;
- en temps quadratique en n, s'il existe $K > 0$ tel que pour tout n, $T_n(f) \leq Kn^2$.

Question 1 Ecrire une fonction compte(x, a) qui retourne, en temps linéaire en n, le nombre de fois où la valeur x apparaît dans le tableau a.

Question 2 En déduire une fonction occurences(a) qui retourne, en temps quadratique en n, un tableau r tel que pour tout i ($0 \leq i < n$), l'élément r_i est le nombre de fois où a_i apparaît dans a.

Partie I

On cherche à calculer les périodes T de temps pendant lesquelles le rayonnement reste d'énergie constante.

$$T = t_j - t_i \quad \text{avec} \quad a_i = a_k = a_j \quad \text{pour tout } k \text{ tel que } i \leq k \leq j$$

Question 3 Ecrire une fonction maxconstant(a) qui retourne, en temps linéaire en n, la période la plus grande T pendant laquelle le rayonnement reste constant.

Partie II

Soit occ le tableau calculé à la question 2.

Question 4 Ecrire une fonction maxoccurences(a, occ) qui imprime, en temps linéaire en n, les indices i_1 et i_2 de deux rayonnements m_1 et m_2 qui apparaissent le plus grand nombre de fois dans le tableau de mesures a. (Si le tableau a contient des valeurs toutes identiques, on posera $i_2 = m_2 = -1$).

On veut maintenant réorganiser les tableaux de mesures a et de dates t pour mettre en tête toutes les mesures donnant m_1, puis celles valant m_2, puis toutes les autres.

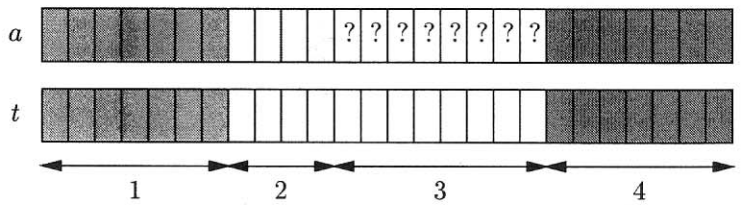
La réorganisation des tableaux a et t demandée est donc telle que

$$\begin{array}{lcl} 0 \leq k < b & \Rightarrow & a_k = m_1 \\ b \leq k < r & \Rightarrow & a_k = m_2 \\ r \leq k < n & \Rightarrow & a_k \notin \{m_1, m_2\} \end{array}$$

Après réorganisation, le tableau t vérifie toujours que t_i est la date à laquelle s'est produit le rayonnement a_i $(0 \leq i < n)$.

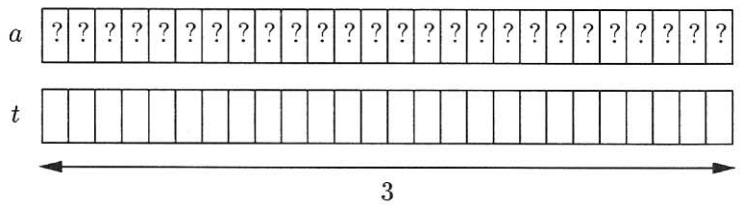
Question 5 Ecrire une fonction trier(a, t, m_1, m_2) qui réordonne, en temps linéaire en n, les tableaux a et t pour regrouper en tête les deux mesures les plus fréquentes, comme indiqué précédemment.

Indication : on parcourra les tableaux a et t (dans le sens des indices croissants) en maintenant une décomposition de la forme suivante

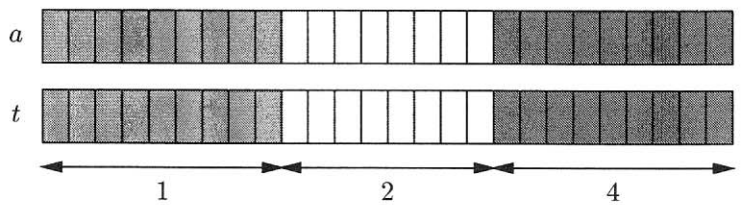


avec a_i valant respectivement m_1, m_2 et une valeur non prise dans $\{m_1, m_2\}$ dans les zones 1, 2 et 4.

Au début les tableaux a et t sont de la forme :



A la fin les mêmes tableaux a et t sont de la forme :



Question 6 La fonction précédente garde-t-elle la croissance des dates à l'intérieur de chaque zone, c'est-à-dire que $i < j$ implique $t_i < t_j$ pour i et j dans une même zone ? Justifier votre réponse.

Épreuve facultative d'Informatique, filières PSI et PT

Rapport de M. Marc Pouzet, correcteur.

De l'épreuve

C'était la première édition de cette épreuve spécifique à l'École polytechnique et commune aux filières PSI et PT. L'épreuve était facultative et seules les copies des candidats admissibles ont été corrigées. Le nombre de copies était faible (47 pour la filière PSI et 8 pour la filière PT).

Résultats

La moyenne des candidats est de 12.7 avec un écart type de 4.1. La répartition est la suivante :

$0 \leq N < 4$	$4 \leq N < 8$	$8 \leq N < 12$	$12 \leq N < 16$	$16 \leq N < 20$
1%	11%	33%	33%	22%

L'écart type s'explique par la proportion importante de très bonnes copies se détachant largement de la moyenne (12% de copies entre 18 et 20), avec un creux relatif entre la moyenne et ces copies. Ceci s'explique certainement par la présence de candidats ayant une grande pratique de la programmation.

Les coefficients des questions, identiques pour les filières PSI et PT sont les suivants :

Question	1	2	3	4	5	6
Coef.	3	4	5	6	5	3

Le langage Maple a été choisi par la majorité des candidats. Les autres langages utilisés étaient Java, Pascal, C et Caml.

Évaluation

Cette épreuve était composée de deux parties. La première partie consistait à écrire des fonctions simples de parcours de tableaux de valeurs entières.

La seconde aboutissait à l'écriture d'un programme de tri. Les structures de données et de contrôle nécessaires se résumaient aux tableaux, boucles `for`, conditionnelles, séquence, définition et utilisation de fonctions. L'utilisation de la récursivité n'était pas indispensable dans cette épreuve.

Un certain nombre de candidats ont choisi d'utiliser des opérations spécifiques à Maple (listes, opérateur `seq`). L'utilisation de ces primitives nécessite une bonne connaissance de Maple et elles ont été utilisées le plus souvent de manière approximative (problèmes de type, de bornes) ce qui a fait perdre des points à ces candidats. Dans ce type d'épreuve, il est conseillé de s'en tenir à des structures de données et de contrôle élémentaires (boucles `for`, tableaux). De plus, elles se prêtent mieux à un calcul de complexité.

Toutes les questions (sauf la question 6) demandaient d'écrire du code. Le code de chaque question étant court, la correction commence par la lecture du code. Il est donc essentiel que ce code soit bien présenté et les noms des variables choisis de manière judicieuse (un commentaire décrivant succintement le rôle de chaque variable peut être une aide précieuse). Si le code est juste, je ne tiens pas trop compte des explications (souvent approximatives). Sinon, je les lis attentivement pour comprendre l'idée du candidat. Toutefois, le candidat a déjà perdu une bonne partie de ses points. La situation peut être sauvée partiellement lorsque le candidat a eu l'idée de formuler un invariant de boucle (sous forme de suites récurrentes par exemple).

Lors de la lecture des programmes, les petites erreurs de syntaxe (oubli d'un `end`, d'une marque de fin de boucle) ne sont pas trop prises en compte dès lors qu'il n'y a pas d'ambiguïté possible et qu'il semble qu'elles seraient immédiatement corrigées lors d'une programmation sur machine. En revanche, les erreurs de type et une absence d'initialisation des variables sont systématiquement sanctionnées.

Pour chaque question, j'indique la moyenne obtenue (en ramenant la note de la question sur 5).

Question 1. [4.3] Presque tous les candidats ont répondu correctement à cette question en donnant un algorithme de complexité linéaire.

S'agissant d'une question facile, la solution devait être parfaite et l'utilisation de variables non initialisées a été sanctionnée.

Question 2. [4.1] Cette question a également été traitée par la majorité

des candidats. Il fallait clairement privilégier ici une solution modulaire effectuant un appel à la fonction `compte` définie précédemment. Une solution non modulaire ou trop compliquée n'a pas reçu tous les points.

Question 3. [2.6] Cette question était un peu plus difficile que les précédentes mais a souvent été traitée correctement. Il s'agissait de calculer progressivement la valeur `max` contenant la période constante maximale courante. Ce calcul pouvait être réalisé par une seule boucle.

Les raisons principales pour ne pas obtenir tous les points ont été une mauvaise initialisation des variables (ou plus souvent un oubli), une erreur dans les bornes de la boucle et une solution compliquée (allouant par exemple un premier tableau des périodes maximum courantes puis le calcul du maximum global) ou l'écriture d'un algorithme non linéaire.

Question 4. [1.65] La solution attendue consistait à effectuer une boucle (d'indice i) parcourant les tableaux `occ` et `a` en calculant progressivement les indices i_1 et i_2. Cette question s'est révélée la plus difficile de l'épreuve. Souvent, le candidat a fait des erreurs lors de la mise à jour de ces deux indices et il a perdu des points.

Les algorithmes quadratiques (faisant d'abord un calcul des occurrences suivi d'un calcul de l'indice du rayonnement maximum puis du même calcul en ôtant ce rayonnement maximum) ont été fortement sanctionnés.

Question 5. [2] Le programme demandé dans cette question correspond à l'algorithme de *tri du drapeau hollandais* de Dijkstra. Dans cet algorithme, il s'agit d'ordonner en effectuant des échanges de valeurs, un tableau rempli avec trois valeurs distinctes (ou trois couleurs) en trois parties homogènes.

L'énoncé suggère de décomposer le tableau en quatres zones en maintenant l'invariant suivant :

$$\begin{array}{ll} \text{zone 1 :} & \forall k, 0 \leq k < i_2, a_k = m_1 \\ \text{zone 2 :} & \forall k, i_2 \leq k < i_3, a_k = m_2 \\ \text{zone 4 :} & \forall k, i_4 < k < n, a_k \notin \{m_1, m_2\} \end{array}$$

i_2 désigne le début de la zone 2, i_3 désigne le début de la zone 3 et i_4 le début de la zone 4. La zone 3 est la zone complémentaire et correspond à la partie du tableau non encore parcourue. Le programme découle naturellement de l'invariant donné ci-dessus en faisant varier i_3 ($0 \leq i_3 < i_4$) et en modifiant progressivement i_2 et i_4 suivant la valeur courante de $a(i_3)$.

Peu de candidats ont l'idée d'écrire des invariants de boucle avant de se lancer dans la programmation. C'est dommage dans ce genre d'exercices où la correction du programme tient souvent à des conditions fines d'initialisation de variables ou de bornes de tableaux. L'écriture d'un invariant de boucle est une aide précieuse pour arriver à un programme correct.

Les solutions quadratiques ou utilisant un tableau auxiliaire pour ranger le tableau trié ont été très fortement pénalisées, ne recueillant pas ou peu de points.

Une bonne programmation de l'exercice supposait l'écriture d'une fonction d'échange de deux valeurs d'un tableau, fonction utilisée ensuite pour les tableaux `a` et `t`.

Question 6. [0.8] Cette question était la suite de la question précédente. Elle a cependant été relativement mal traitée par les candidats, même ceux ayant répondu correctement à la question précédente.

De nombreux candidats se sont lancés dans des explications compliquées sans penser à donner un contre-exemple. Une réponse recevant tous les points se résumait à l'énoncé d'un contre-exemple montrant que la croissance des dates était modifiée par l'algorithme. Une réponse non justifiée, elle, ne recevait aucun point.