ÉPREUVE PRATIQUE D'ALGORITHMIQUE ET DE PROGRAMMATION
ENS : PARIS — LYON

Coefficient **: 4**

MEMBRES DE JURYS : L. BOUGÉ, Y. ROBERT

L'objectif de cette nouvelle épreuve est d'évaluer les capacités des candidats à mettre en œuvre de manière cohérente la chaîne *complète* de résolution d'un problème informatique : analyse de la spécification abstraite d'un problème, conception d'un algorithme, évaluation de son coût, programmation sur machine dans l'un des langages proposés, exécution sur un jeu de valeurs tests, discussion des résultats obtenus.

Il s'agit donc d'une épreuve transversale, complémentaire des épreuves écrites, plutôt centrées sur les aspects formels et mathématiques, et des épreuves orales, plutôt centrées sur les aspects algorithmiques. Cette nouvelle épreuve permet aux candidats de mettre en valeur leur maîtrise d'un environnement informatique et d'un langage de programmation au service d'une démarche scientifique de résolution de problème et d'évaluation des diverses stratégies possibles. L'étalement des notes obtenues montre que cette épreuve a bien rempli son rôle, en faisant émerger un lot de candidats particulièrement brillants.

Le jury a examiné 66 candidats (que des garçons !), répartis en 5 sessions de 4 heures chacune. Les candidats avaient le choix entre plusieurs plates-formes de programmation :
– PC sous Windows 98, avec Caml Light 0.74 et Maple 7 ;
– PC sous Linux/KDE, avec Caml Light et OCaml 3.04 (sous XEmacs/Tuareg), C (sous KDevelop 2.0 ou XEmacs, compilateur `gcc`) et Pascal (XEmacs, compilateur `gpc`).

Les environnements Caml utilisés sont disponibles librement sur le serveur INRIA `http://caml.inria.fr/`. Les compilateurs `gcc` et `gpc` sont disponibles librement à partir du serveur `http://www.gnu.org/`. L'éditeur XEmacs est disponible librement sur le server `http://www.xemacs.org/`.

Il était demandé aux candidats de choisir l'une des plates-formes et l'un des langages par avance. Les trois quarts des candidats ont choisi de programmer en Caml Light sous Windows. Seulement deux candidats ont utilisé Maple. Le reste des candidats ont choisi de composer sous Linux/KDE, et se sont répartis de manière équilibrée entre Caml Light, OCaml, Pascal et C.

Les sujets ont été préparés selon plusieurs exigences.
– L'épreuve devait être évaluable *uniquement* à partir des copies rendues par les candidats (réponses aux questions et résultats numériques finaux). Les disquettes rendues par les candidats n'ont été utilisées qu'à titre exceptionnel. Il n'a *pas été tenu compte* de la structuration des programmes, ou même de la possibilité de rejouer leur exécution après l'épreuve.
– Les problèmes posés ne devaient pas faire intervenir de notion conceptuelle complexe pour pouvoir être abordés par tous les candidats. Nous nous sommes volontairement restreints à des parcours de tableaux ou de matrices.
– Les solutions devaient être programmables dans l'ensemble des langages proposés. Ceci excluait en particulier tout sujet reposant explicitement sur des structures de données dynamiques comme des listes ou des arbres.
– Les résultats devaient être reproductibles sur l'ensemble des plates-formes disponibles. Ceci

excluait l'usage des générateurs aléatoires ou des bibliothèques mathématiques spécifiques aux environnements. Les calculs flottant ont aussi été utilisés avec la plus grande prudence pour ne pas dépendre des erreurs d'arrondi.
- Les problèmes posés devaient, si possible, être abordables par *plusieurs algorithmes*, avec des degrés de raffinement (et donc de coût) échelonnés. L'objectif est que l'algorithmique ne soit pas un obstacle majeur pour les candidats, mais que les meilleurs candidats puissent trouver matière à exprimer librement leur talent. Typiquement, il existait toujours un algorithme simple mais coûteux (par exemple, $O(n^4)$), mais aussi des algorithmes plus élaborés moins coûteux (par exemple, $O(n^2)$). La réponse à certaines questions par un algorithme simpliste (mais correct) pouvait pousser les plates-formes à leurs limites matérielles (taille des entiers, espace mémoire, etc.) ou conduire à des temps de calcul déraisonnables dans ce cadre (plusieurs minutes, voire des heures !)
- Des efforts ont été faits pour permettre aux candidats de *tester* leurs algorithmes et leurs implémentations sur des versions réduites des problèmes proposés, afin de pouvoir vérifier la correction de leurs résultats avant de passer à l'échelle. Par exemple, il était suggéré d'appliquer la méthode d'abord à une chaîne de longueur 10 avant de considérer une chaîne de longueur 10 000. Malheureusement, certains candidats n'ont pas compris l'intérêt de ces préliminaires : ils se sont lancés "tête baissée" dans des calculs gigantesques... mais faux.

On trouvera en annexe à ce rapport quelques sujets proposés (mais pas les solutions !)

Une remarque générale du jury est la suivante. La plupart des candidats ont été capables de proposer un algorithme pour résoudre les problèmes posés, et d'en évaluer grossièrement la complexité, ce qui est bien. Par contre, la mise en œuvre *concrète* des algorithmes proposés a souvent été maladroite, et source de nombreuses erreurs. L'évaluation de l'épreuve étant essentiellement fondée sur la correction numérique des résultats, ce point ne peut être négligé.

La plupart des erreurs sont dues à l'incapacité des candidats de *réfléchir* quelques minutes crayon en main avant de se précipiter sur le clavier de leur PC. Cette hâte est souvent fatale. L'un des problèmes demandait par exemple de parcourir des matrices binaires (dont les éléments sont des 0 ou des 1) pour détecter la présence d'un motif particulier. Tous les candidats (ou presque !) réussissent à compter correctement le nombre de 1 dans une matrice binaire carrée de taille $n \times n$. Par contre, beaucoup peinent à trouver le nombre de carrés de taille $p \times p$ composés uniquement de 1 dans une telle matrice, même par la méthode la plus simple : pour chaque position (i, j) de la matrice, tester si tous les élément du carré $[i..(i + p - 1), j..(j + p - 1)]$ sont des 1 (on peut évidemment faire beaucoup plus efficace). En fait, la plupart des candidats se trompent dans les indices des boucles externes sur i et j. Caml ou Maple détectant les débordements de tableaux, il y a peu d'erreurs par excès. Par contre, il y a de nombreuses erreurs par défaut, les carrés du bord de la matrice n'étant pas complètement explorés. Nous pourrions multiplier de tels exemples !

Il ne s'agit pas là d'un manque de préparation ou de talent. Il s'agit d'un *manque de méthode scientifique* pour la conception et la mise en œuvre des algorithmes. Nous sommes par exemple désolés du faible nombre de candidats qui s'aident de croquis, alors que cela aurait été d'une grande aide dans le cas ci-dessus. De même, quelques minutes de réflexion sur papier auraient souvent suffi à éviter des heures de débogage pénible et surtout bien incertain. Enfin, tester le programme sur une petite sous-matrice, par exemple avec $n = 10$ et $p = 3$, aurait certainement permis de détecter la plupart des erreurs d'indices...

Références

Certains exercices présentés ci-après ont été inspiré du livre suivant :

Alain Darte et Serge Vaudenay, *Algorithmique et optimisation, exercices corrigés*, Collection Sciences Sup, Dunod, 2001. ISBN : 2100056433. `http://www.dunod.com/`.

Il s'agit des exercices suivants :

Algorithmes d'ordonnancement : chapitre 2.5 ;

Voyageur de commerce : chapitre 2.6.

Remerciements

La mise en place de cette nouvelle épreuve n'aurait pas été possible sans le soutien de nombreuses personnes que nous tenons à remercier chaleureusement ici : Hervé Brunet, Vincent Danjean, Alain Darte, Louis Granboulan, Arnaud Legrand, Jean-Louis Moisy, Eric Patinaud, Loïs Taulelle.

Algorithmes d'ordonnancement

Épreuve pratique d'algorithmique et de programmation

Juillet 2002

On veut exécuter un ensemble $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ de $n = 200$ tâches. Les durées de ces tâches sont stockées dans un tableau W de taille n. Le tableau est initialisé comme suit. On définit la suite récurrente d'entiers $(x_i)_{0 \leq i \leq n}$ par :
– x_0 est égal au numéro inscrit sur votre table d'examen (on pourra prendre toute valeur entre 23 et 30, par exemple),
– $x_i = (x_{i-1} \times x_{i-1})$ modulo 3953 pour i compris entre 1 et n.
Puis, pour $1 \leq i \leq n$, on pose $W[i] = (x_i \text{ modulo } 10) + 1$.

Partie I. Préliminaires

Question I.1. Combien d'éléments du tableau W sont égaux à 5 ? S'il en existe, quel est l'indice de l'avant-dernier d'entre eux ?

Question I.2. Quelles sont la ou les valeurs qui apparaissent le plus souvent dans le tableau W ?

On dispose d'un ensemble $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$ de m machines identiques, capables d'exécuter n'importe quelle tâche. Une machine ne peut exécuter qu'une tâche à la fois. L'exécution débute au top $t = 0$. Si une machine commence l'exécution de la tâche T_i au top t, elle l'exécute sans interruption dans l'intervalle $[t, t + W[i][$; au top $t + W[i]$ l'exécution est terminée, et la machine devient disponible pour une autre tâche.

On veut construire un ordonnancement, c'est-à-dire associer à chaque tâche T_i un top de début d'exécution $\sigma(i)$ et un numéro de machine d'affectation $\mu(i)$, compris entre 1 et m, qui détermine la machine sur laquelle T_i est exécutée. On note D la durée totale de l'exécution : $D = \max_{1 \leq i \leq n} \sigma(i) + W[i]$.

Partie II. Tâches indépendantes

On suppose dans cette partie que chaque tâche peut être exécutée à n'importe quel top. Le principe général des algorithmes étudiés est de parcourir la liste des tâches selon un certain ordre ; la tâche courante sera exécutée sur une machine donnée en fonction des décisions déjà prises par l'algorithme pour les tâches précédentes.

On commence par l'algorithme *glouton* suivant. On prend les tâches dans l'ordre initial, pour i variant de 1 à n. On affecte T_i à la machine qui permet de commencer son exécution au plus tôt, compte tenu des décisions déjà prises. En cas d'égalité, on affecte T_i à la machine d'indice le plus petit.

Par exemple, suivons le début de l'algorithme avec $m = 3$ machines. On aura toujours $\sigma(1) = \sigma(2) = \sigma(3) = 0$, $\mu(1) = 1$, $\mu(2) = 2$, et $\mu(3) = 3$. Si $W[1] = 3$ et $W[2] = W[3] = 1$, on aura $\sigma(4) = 1$ et $\mu(4) = 2$ (cas d'égalité au top 1). Ainsi de suite ...

Question II.1. On suppose $m = 2$ dans cette question. On cherche à calculer D.

1. Expliquer brièvement l'algorithme que vous mettez en oeuvre.
2. Définir parmi les opérations élémentaires effectuées par votre programme celles qui sont les plus significatives (justifier votre choix) ; indiquer (en ordre de grandeur) leur nombre en fonction de n.
3. Que valent D, $\sigma(45)$ et $\mu(127)$?

Question II.2. On suppose $m = 5$ dans cette question. Que valent D, $\sigma(45)$ et $\mu(127)$?

Question II.3. On décide de trier les tâches par durée décroissante : on définit une permutation p de $[1..n]$ telle que

$$p[i] < p[j] \Leftrightarrow (W[i] < W[j]) \text{ ou } (W[i] = W[j] \text{ et } i < j).$$

On exécute l'algorithme précédent en considérant les tâches dans l'ordre donné par p. Reprendre les deux questions précédentes, en expliquant brièvement l'algorithme que vous mettez en oeuvre.

Partie III. Tâches indépendantes, avec dates de terminaison imposées

Les tâches ont maintenant des dates de terminaison imposées : pour tout i, on cherche à terminer la tâche T_i au plus tard à la date $Z[i]$, c'est-à-dire obtenir l'inégalité $\sigma(i) + W[i] \leq Z[i]$. Le tableau Z est initialisé comme suit : $Z[i] = i$ si i est pair et $Z[i] = 2i$ si i est impair, où $1 \leq i \leq n$.

Au cours de l'exécution de l'algorithme d'ordonnancement, il se peut qu'une tâche ne puisse être terminée avant sa date de terminaison imposée. Dans ce cas, on supprime la tâche en question (on ne l'exécute pas), on paie une pénalité, et on continue avec la tâche suivante.

Question III.1. On suppose $m = 3$ dans cette question. Combien de pénalités paie l'algorithme glouton précédent qui examine les tâches dans l'ordre initial ? Que vaut alors D ? Expliquer brièvement l'algorithme que vous mettez en oeuvre.

Question III.2. On suppose toujours $m = 3$ dans cette question. Combien de pénalités paie l'algorithme qui examine les tâches "les plus urgentes" d'abord, c'est-à-dire dans l'ordre donné par la permutation q de $[1..n]$ telle que

$$q[i] < q[j] \Leftrightarrow (Z[i] < Z[j]) \text{ ou } (Z[i] = Z[j] \text{ et } i < j) \text{ ?}$$

Que vaut alors D ? Expliquer brièvement l'algorithme que vous mettez en oeuvre.

Partie IV. Tâches avec dépendances

On définit une matrice E de taille $n \times n$ dont les éléments valent 0 ou 1 comme suit :

$$\text{Pour } 1 \leq i, j, \leq n, \quad E[i,j] = 1 \Leftrightarrow i < j \text{ et } (x_{(i+j)\div 2} \text{ modulo } 13) \in \{1, 8\}$$

(où la suite (x_i) est celle définie au début de l'énoncé). Si $E[i,j] = 1$, on dit qu'il y a une relation de dépendance de T_i vers T_j : l'exécution de T_i devra être terminée avant que celle de T_j ne puisse commencer. L'ordonnancement doit respecter toutes ces contraintes :

$$E[i,j] = 1 \Rightarrow \sigma(i) + W[i] \leq \sigma(j).$$

Par contre les tâches peuvent toujours être exécutées sur n'importe quelle machine. Lorsque $E[i,j] = 1$ on dit que T_i est un prédécesseur de T_j ; de plus, lorsqu'il n'existe pas d'indice k tel que $E[i,k] = 1$ et $E[k,j] = 1$, on dit que T_i est un prédécesseur direct de T_j.

Question IV.1. Combien y-a-t-il de tâches sans aucun prédécesseur ?

Les tâches ont une priorité donnée par la valeur du tableau P défini plus bas ; T_i est d'autant plus prioritaire que $P[i]$ est grand.

L'algorithme affecte les tâches aux machines en gérant un tableau d'occupation des machines. On dit qu'une tâche (pas encore affectée à une machine) est prête si tous ses prédécesseurs directs ont été exécutés. Au début, seules sont prêtes les tâches sans aucun prédécesseur. On procède comme suit :

- au début toutes les machines sont libres ;
- on considère le premier instant t où une ou plusieurs machines se libèrent (information à extraire du tableau d'occupation des machines) ;
- on met à jour l'ensemble des tâches prêtes ;
- à cet instant t, supposons que q machines se libèrent, d'indices $p_1 \leq p_2 \leq \ldots \leq p_q$. Soit r le nombre de tâches prêtes à l'instant t, et $s = \min(q, r)$;
- on débute au temps t l'exécution des s tâches prêtes de de plus grande priorité : celle de priorité maximale est affectée à la machine d'indice p_1, la seconde à la machine p_2, etc.
- on met à jour le tableau d'occupation des machines, et on recommence tant qu'il reste des tâches non exécutées.

Question IV.2. Pour cette question seulement, on considère le petit exemple illustratif de la Figure 1, où les valeurs de W, E, P n'ont rien à voir avec le reste de l'énoncé. On suppose $m = 3$. Donner les 6 valeurs manquantes de σ et de μ (vous devez trouver que $D = 4$).

Matrice des dépendances : tous les éléments de E sont égaux à 0, sauf les neuf éléments suivants :

$$E[1,2] = E[1,5] = E[3,9] = E[4,5] = E[5,7] = E[6,7] = E[6,8] = E[6,9] = E[7,8] = 1$$

Tâches	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
Durée W	1	1	1	1	1	1	1	1	1
Priorité P	5	3	2	6	7	9	8	4	1
Ordonnancement σ	0	1	1	0	1	0			
Machine μ	3	2	3	2	1	1			

FIG. 1 – Exemple pour les tâches avec dépendances

Question IV.3. On suppose $m = 3$ et que $P[i] = i$ pour $1 \leq i \leq n$. Que valent D, $\sigma(55)$ et $\mu(27)$? Expliquer brièvement l'algorithme que vous mettez en oeuvre.

Question IV.4. On suppose $m = 5$ et que $P[i] = n + 1 - i$. Que valent D, $\sigma(55)$ et $\mu(27)$?

Question IV.5. On suppose que $m = 4$ et que $P[i] = 2i + 1$ si i est pair, $P[i] = 2(n + 1 - i)$ si i est impair, $1 \leq i \leq n$. Que valent D, $\sigma(55)$ et $\mu(27)$?

Matrices 0-1

Épreuve pratique d'algorithmique et de programmation

Juillet 2002

Le sujet propose diverses manipulations d'une matrice carrée T de taille $n = 60$ dont tous les éléments valent 0 ou 1. On définit la suite récurrente d'entiers $(x_i)_{0 \leq i \leq n^2}$ par :
- x_0 est égal au numéro inscrit sur votre table d'examen (on pourra prendre toute valeur entre 23 et 30, par exemple),
- $x_i = (x_{i-1} \times x_{i-1})$ modulo 3953 pour i compris entre 1 et n^2.

La matrice est initialisée comme suit :

$$\text{Pour } 1 \leq i, j \leq n, i \neq j, \text{ si } x_{(i-1)n+j} = 1 \text{ modulo } 4 \text{ alors } T[i,j] = 1 \text{ sinon } T[i,j] = 0.$$

On pose enfin $T[i,i] = 1$ pour $1 \leq i \leq n$.

Partie I. Préliminaires

Question I.1. Combien d'éléments de la matrice T sont-ils égaux à 0 ?

Question I.2. Combien de colonnes de la matrice T contiennent-elles strictement plus de 0 que de 1 ? quel est le plus petit indice (s'il existe) d'une telle colonne ?

Partie II. Éléments booléens

On considère ici les éléments de la matrice comme des booléens de valeur "vrai" ou "faux", c'est-à-dire avec l'arithmétique $0 + 0 = 0$, $0 + 1 = 1 + 0 = 1 + 1 = 1$, $0 \times 0 = 0 \times 1 = 1 \times 0 = 0$, et $1 \times 1 = 1$. Cette arithmétique s'étend aux matrices de façon usuelle.

Question II.1. Combien d'éléments de la matrice $T^2 = T \times T$ sont égaux à 1 ?

Question II.2.

1. Montrer (faire une preuve écrite) qu'il existe un indice $p \geq 2$ tel que $T^p = T^{p-1}$.
2. On cherche à déterminer $p_{\min}$, le plus petit p qui répond à la question. Expliquer brièvement l'algorithme que vous mettez en oeuvre.
3. Définir parmi les opérations élémentaires effectuées par votre programme celles qui sont les plus significatives ; indiquer (en ordre de grandeur) leur nombre en fonction de n.
4. Quelle est la valeur de $p_{\min}$ et des deux éléments de $T^{p_{\min}}$ d'indices $(i,j) = (24, 25)$ et $(16, 29)$?

Question II.3. Le *bon compte* d'un triplet $(i1, i2, i3)$ indiçant trois lignes distinctes de T est défini comme le nombre de composantes j telles que $T[i1, j] + T[i2, j] = T[i3, j]$. On cherche quel est le bon compte maximal qu'il est possible d'obtenir.

1. Expliquer brièvement l'algorithme que vous mettez en oeuvre, et préciser (en ordre de grandeur) le nombre d'opérations effectuées.
2. Donner la valeur du bon compte maximal, et les indices d'un triplet réalisant ce bon compte.

Partie III. Composantes connexes

On interprète la matrice T comme une image binaire, les points de l'image étant les éléments de T égaux à 1. Les quatre voisins de l'élément $T[i, j]$ sont les éléments $T[i \pm 1, j]$ et $T[i, j \pm 1]$ (s'ils existent). On définit la relation binaire d'adjacence comme suit : deux points sont adjacents s'ils sont dans l'image (égaux à 1) et s'ils sont voisins. Une composante connexe est une classe d'équivalence de la fermeture réflexive et transitive de la relation d'adjacence : deux points de l'image sont dans la même composante connexe s'il existe un chemin de points adjacents les reliant.

Question III.1. Chaque point (i, j) de l'image (tel que $T[i, j] = 1$) reçoit l'étiquette $(i - 1)n + j$. Chaque point (i, j) tel que $T[i, j] = 0$ reçoit l'étiquette 0. Construire le tableau U tel que :
– si $T[i, j] = 1$, alors $U[i, j]$ est le maximum de l'étiquette du point (i, j) et de celle de ses voisins (quatre en général, attention aux bords) ;
– si $T[i, j] = 0$, alors $U[i, j] = 0$.
Combien de valeurs non-nulles différentes contient le tableau U ?

Question III.2. Quel est le nombre de composantes connexes dans le tableau T ? Expliquer brièvement l'algorithme que vous mettez en oeuvre, et préciser (en ordre de grandeur) le nombre d'opérations effectuées.

Question III.3. Quel est le cardinal maximal d'une composante connexe ? Combien de composantes connexes sont-elles de cardinal maximal ?

Question III.4. La distance absolue de deux éléments $T[i, j]$ et $T[i', j']$ est définie comme $|i' - i| + |j' - j|$. Quelle est la distance absolue maximale entre deux éléments du tableau qui appartiennent à la même composante connexe ?

Question III.5. La distance dans l'image de deux éléments appartenant à une même composante connexe est la longueur du plus court chemin de points adjacents les reliant. Quelle est la distance dans l'image entre deux éléments du tableau qui appartiennent à la même composante connexe ? Expliquer brièvement l'algorithme que vous mettez en oeuvre, et préciser (en ordre de grandeur) le nombre d'opérations effectuées.

Partie IV. Découpage

On découpe récursivement la matrice T en régions rectangulaires comportant au plus trois points égaux à 1. On utilise le processus suivant, où le rectangle initial est toute la matrice :
– tant qu'un rectangle contient au moins quatre points égaux à 1, on le découpe ;
– le processus s'arrête quand il n'y a plus de rectangle à découper.

Soit un rectangle R à découper, comprenant $m \geq 4$ points de l'image, avec L lignes et C colonnes. On cherche à découper R en deux rectangles comprenant approximativement le même nombre de points égaux à 1.

Formellement, on compare deux solutions :

Découpe horizontale : R_1 est composé des h premières lignes de R et contient m_1 points égaux à 1, et R_2 est composé des $L - h$ dernières lignes de R et contient m_2 points égaux à 1 ($m_1 + m_2 = m$). On choisit h pour minimiser la différence $|m_2 - m_1|$ (et on prend la plus petite valeur de h s'il y en a plusieurs qui réalisent le minimum).

Découpe verticale : même principe avec les colonnes.

On retient la solution qui minimise la différence du nombre de 1 entre les deux rectangles (en cas d'égalité, on prend la solution obtenue par découpe horizontale).

Question IV.1.

1. Expliquer brièvement l'algorithme que vous mettez en oeuvre, et préciser (en ordre de grandeur) le nombre d'opérations effectuées.
2. En combien de rectangles le tableau T est-il découpé par le processus ?

Motifs et sous-chaînes

Épreuve pratique d'algorithmique et de programmation

Juillet 2002

Le sujet étudie des algorithmes de recherche de motifs dans une chaîne de caractères. Les questions sont (très librement !) inspirés de problèmes rencontrés en génomique.

On s'intéresse à une longue chaîne C de caractères représentés par les nombres 0, 1, 2 et 3. La chaîne de caractères $C = C[1] \dots C[N]$ est de longueur $N = 10000$.

La chaîne C est initialisée comme suit. On définit la suite récurrente d'entiers $(x_i)_{0 \leq i \leq N}$ par :
– x_0 est égal au numéro inscrit sur votre table d'examen,
– $x_i = ((2077 \times x_{i-1}) + 12345) \bmod 2^{16}$, pour i compris entre 1 et n.
Puis on remplace chaque x_i par la partie entière de $x_i/17$. Puis pour $1 \leq i \leq n$, on définit le i-ième caractère de la chaîne par $C[i] = x_i \bmod 4$.

(Remarque : notez bien que dans tout ce sujet les chaînes sont indicées à partir de 1, même si vous utilisez en machine des vecteurs ou des tableaux dont le premier élément est en position 0.)

(NB : Les numéros de table indiqués étaient 17, 19, 23, 29.)

Partie I. Préliminaires

Question I.1. Listez les 10 premières valeurs de la chaîne C : $C[1] \dots C[10]$.

Question I.2. Combien de caractères de C sont égaux à 0, 1, 2, 3 ?

Un *motif* M de longueur m est une chaîne de caractères $M[1] \dots M[m]$. C'est une *sous-chaîne* de C en position i si pour tout $k = 1, \dots, m$ on a $M[k] = C[i + k - 1]$.

La *fréquence* d'un motif M dans une chaîne C est le nombre de valeurs distinctes i telles que M soit une sous-chaîne de C en position i.

Question I.3. Quelle est la fréquence dans C du motif 0123 ?

Partie II. Notion de m-fréquence d'une chaîne

La m-*fréquence* d'une chaîne c de longueur n est égale à la fréquence du motif de longueur m le plus fréquent dans C. On cherche à déterminer la m-fréquence.

Question II.1. Décrivez brièvement un algorithme qui prend en entrée une chaîne c de longueur n et une longueur m et qui détermine la m-fréquence de c. Définissez parmi les opérations élémentaires effectuées par le programme celles qui sont les plus significatives (justifiez votre proposition !), et indiquez leur nombre en fonction de n et m.

(Aide : il pourra être utile d'encoder les différents motifs rencontrés, par exemple par des entiers, pour pouvoir mémoriser dans une table le nombre de leurs occurrences dans la chaîne.)

Question II.2. Quelle est la 1-fréquence de la chaîne C ?

Question II.3. Soit CC la chaîne formée des 10 premiers éléments de C. Quelle est sa 2-fréquence ? Sa 3-fréquence ? Sa 10-fréquence ? Dans chaque cas, précisez le(s) motif(s) le plus fréquent de la longueur considérée.

Question II.4. Quelle est 3-fréquence de C ? Sa 7-fréquence ? Dans chaque cas, précisez le(s) motif(s) le plus fréquent de la longueur considérée.

Question II.5. Pour quelle valeur maximale de m pouvez-vous calculer la m-fréquence de C en moins d'une minute ? Quelle est par exemple la 10-fréquence de C ? Sa 15-fréquence ? Sa 20-fréquence ?

À partir d'une certaine longueur tout motif est de fréquence 0 ou 1. La *longueur maximale de répétition* de la chaîne est la valeur maximale m pour laquelle il existe un motif de longueur m avec une fréquence d'au moins 2.

Question II.6. Décrivez brièvement un algorithme qui prend en entrée une chaîne c de longueur n et qui détermine la longueur maximale de répétition de c. Définissez parmi les opérations élémentaires effectuées par le programme celles qui sont les plus significatives (justifiez votre proposition !), et indiquez leur nombre en fonction de n.

Question II.7. Quelle est la longueur maximale de répétition de la chaîne CC de longueur 10 ? Quelle est celle de C ?

Partie III. Sous-chaînes sans ℓ-répétitions

Une sous-chaîne de longueur m d'une chaîne c est sans ℓ-répétition si sa longueur maximale de répétition est strictement inférieure à ℓ. Elle ne contient donc aucun motif de longueur ℓ répété.

Étant donné une chaîne c, on cherche à extraire des sous-chaînes sans ℓ-répétition de longueur maximale.

Question III.1. Décrivez brièvement un algorithme qui prend en entrée une chaîne c de longueur n et une longueur ℓ et qui détermine une sous-chaîne sans ℓ-répétition de longueur maximale. Définissez parmi les opérations élémentaires effectuées par le programme celles qui sont les plus significatives (justifiez votre proposition !), et indiquez leur nombre en fonction de n et de ℓ.

Question III.2. Donnez la longueur maximale d'une sous-chaîne de CC sans 2-répétition, et la décrire explicitement.

Question III.3. Donnez la longueur maximale d'une sous-chaîne de C sans 3-répétition.

Partie IV. Mariages de motifs

Deux motifs M et M' de même longueur m peuvent être *mariés* si pour tout $k = 1, \ldots .m$ on a $M[k] - M'[k] = 2 \bmod 4$.

Une chaîne contient un *mariage* de longueur m si elle contient deux sous-chaînes *disjointes*, chacune de longueur m et pouvant être mariées.

On pourra remarquer que ce problème est analogue à des questions pouvant survenir en génomique, car les mariages de motifs sont similaires à l'appariement de chaînes de nucléotides.

Question IV.1. Mêmes questions que précédemment, pour le calcul de la longueur maximale des mariages contenu dans C, en exprimant le nombre d'opérations en fonction de n.

Pour toute valeur ℓ on peut découper la chaîne en sous-chaînes disjointes de longueur au moins ℓ et réparties en un ensemble de paires mariées et un ensemble de motifs célibataires.

La *ℓ-auto-affinité* d'une chaîne est la valeur maximale possible du nombre de caractères appartenant aux sous-chaînes mariées.

Question IV.2. Mêmes questions que précédemment, pour le calcul de la ℓ-auto-affinité en exprimant le nombre d'opérations en fonction de n et ℓ. Donnez un encadrement de la 3-auto-affinité de C.

Voyageur de commerce

Épreuve pratique d'algorithmique et de programmation

Juillet 2002

Ce problème propose d'étudier quelques heuristiques pour la résolution du problème du voyageur de commerce. On se donne un ensemble de *villes*, représentées par des points du plan euclidien. La distance entre les villes est la distance euclidienne. La longueur d'un parcours de ville en ville est la somme des distances entre les villes. Un *voyageur de commerce* doit partir d'un ville de départ, visiter chacune de ces villes, une fois et une seule, et finalement revenir à sa ville de départ. L'objectif est de trouver la permutation des villes qui engendre le parcours de longueur minimale. Cette permutation est appelé *parcours ou chemin minimal*.

Ce problème est bien connu dans la littérature pour être de difficulté *non-polynomiale* : On ne connaît aujourd'hui aucun algorithme permettant de déterminer, en un temps polynomial dans le nombre de villes, le parcours de longueur minimale. (La découverte d'un tel algorithme serait un événement scientifique de première grandeur, avec un impact majeur sur le domaine !)

On s'intéresse ici à l'expérimentation de quelques approches *heuristiques* qui permettent de trouver une "bonne" solution au problème en un temps polynomial.

On considère donc à un ensemble de de points du plan, les *villes*, de cardinal N, avec N grand. On choisit ici $N = 20$. Les points sont définis à partir de la suite récurrente d'entiers suivante :
– u_0 est égal au nombre inscrit sur votre table d'examen ;
– $u_i = f(u_{i-1})$ pour $1 \leq i < 2.N$.
On pose $v_i = u_i$ modulo 100, $0 \leq i < 2.N$.

Pour $0 \leq n < N$, le point P_n a pour abscisse $x_n = v_{2i}$ et $y_n = v_{2i+1}$.

La fonction $f(u)$ est définie comme suit :

$$f(u) = u^2 \text{ modulo } 3953$$

(NB : Les numéros de table indiqués étaient 3, 4, 5, etc.)

Partie I. Préliminaires

Un *chemin* entre les n villes $P_0, \ldots, P_{n-1}$ est un parcours $(P_{i_0}, P_{i_2}, \ldots, P_{i_{n-1}})$ du voyageur de commerce parmi ces villes qui passe par chaque ville une fois et une seule et qui revient à son point de départ. La longueur du chemin est la somme des distances $(P_{i_k}, P_{i_{k+1}})$ pour $k = 0, \ldots, n-1$ avec $i_n = i_0$.

Un chemin entre les villes $P_0, \ldots, P_{n-1}$ est donc défini par une permutation de $(0, \ldots, n-1)$.

Question I.1. Quelles sont les coordonnées des villes P_0, P_5, P_{10} ?

Question I.2. Créez un tableau bidimensionnel de taille $N \times N$ contenant les distances euclidiennes entre les N villes. Quelle est la longueur du chemin *fermé* $(0, \ldots, N-1)$ passant successivement par chacune de villes et revenant à son point de départ ?

Question I.3. Combien y a-t-il de chemins entre les $n = 9$ premières villes $P_0, \ldots, P_{n-1}$? Décrivez explicitement le chemin de longueur minimale et donnez sa longueur.

Question I.4. Pour quelle valeur de n le calcul de la longueur de tous les chemins possibles entre les villes $P_0, \ldots, P_{n-1}$ par leur énumération exhaustive prend-il plus de une minute sur votre machine ? Donnez une estimation du temps qui serait nécessaire pour $n = 20$ dans une unité adaptée : jours, mois, années, etc.

Partie II. Une méthode gloutonne par insertion

Une première méthode pour trouver rapidement une solution approchée est la suivante. On part du chemin entre les villes 0 et 1. On cherche à insérer la ville 2 dans ce chemin de manière à minimiser la longueur. Si plusieurs positions sont possibles, on choisi la dernière dans une exploration gauche-droite de la liste courante des villes. On procède ainsi successivement pour chaque ville P_i, $i = 2, \ldots, N-1$.

Question II.1. Décrivez brièvement la mise en oeuvre de cette idée. Précisez en particulier les structures de données utilisées. Quel est l'ordre de grandeur du coût en temps de votre approche ?

Question II.2. Considérez le chemin $(P_0, \ldots, P_{n-1})$ de longueur $n = 9$. Quel est la meilleure insertion pour la ville P_n dans ce chemin ? Décrire le chemin obtenu et donner sa longueur.

Question II.3. Appliquez la méthode à la liste ordonnée des villes $(P_0, \ldots, P_{n-1})$ de taille $n = 9$. Donnez la suite des insertions réalisées. Décrire le chemin obtenu et donner sa longueur.

Question II.4. Pouvez-vous donner une valeur de $n > 2$ pour laquelle le chemin ainsi obtenu par cette méthode est de longueur minimale parmi tous les chemins possibles ? Pouvez-vous en donner une autre pour laquelle il ne l'est pas ? Décrivez dans chaque cas le chemin obtenu et sa longueur.

Question II.5. Quelle est la longueur du chemin obtenu par cette méthode pour les villes $P_0, \ldots, P_{N-1}$? Décrivez le chemin obtenu et donner sa longueur. Combien de temps votre programme met-il pour le trouver ?

Partie III. Une méthode itérative par optimisation locale

On considère un parcours orienté du chemin : $(P_{i_0}, P_{i_2}, \ldots, P_{i_{N-1}})$. Des villes *voisines* au rang k sur ce chemin sont deux villes de la forme $(P_{i_k}, P_{i_{k+1}})$, $k = 0, \ldots, N-1$ et $i_N = i_0$.

Considérons deux couples de villes voisines sur un chemin orienté $(P_{i_0}, \ldots, P_{i_n})$, par exemple les villes aux rangs $(k, k+1)$ et $(l, l+1)$, avec $0 \leq k < k+1 < l < l+1 \leq N$ et la convention que la ville de rang n est celle de rang 0. Soient $(P_{i_k}, P_{i_{k+1}})$ et $(P_{i_l}, P_{i_{l+1}})$ ces villes. On dit que l'on *échange* les deux segments de rangs k et l si l'on transforme ce chemin en allant de P_{i_k} à P_{i_l} et de $P_{i_{k+1}}$ à

$P_{i_{l+1}}$, et en renversant le sens du parcours initial entre P_{i_l} et $P_{i_{k+1}}$. (Il est vivement recommandé de faire un dessin !).

On appelle *signature* de cet échange les noms des villes échangées, c'est-à-dire les deux couples (i_k, i_{k+1}) et (i_l, i_{l+1}).

Question III.1. Quelle est la longueur du nouveau chemin ainsi obtenu en fonction de celle du chemin initial ? Montrer que si les deux segments initiaux se croisaient (au sens strict, c'est-à-dire que leur intersection est un point unique, différent de leurs extrémités), alors leur échange conduit à un chemin strictement plus court. En déduire qu'un chemin de longueur minimal ne peut avoir de tels croisements.

Un échange qui fait décroître la longueur du chemin est dit *intéressant*.

L'idée est donc de partir d'un chemin et d'appliquer successivement des échanges intéressants jusqu'à ne plus en trouver.

Question III.2. Montrez sur un petit exemple que le chemin résultant dépend en général de l'ordre des échanges. Est-il de longueur minimale parmi tous les chemins possibles ?

Question III.3. Décrivez brièvement une procédure qui prend en paramètre un chemin et qui cherche l'échange intéressant qui fait décroître la longueur de la plus grande quantité. Si plusieurs tels échanges sont possibles, choisissez celui qui se trouve le plus à droite : l'échange des segments de rangs i et j tels que i est maximal, et, pour i fixé, celui où j est maximal. Précisez les structures de données que vous utilisez.

Question III.4. Considérez le chemin $(P_0, \ldots, P_{n-1})$ de longueur $n = 9$. Comporte-t-il un échange intéressant ? Si oui, quel est la signature de celui qui est sélectionné dans la procédure ci-dessus ? Décrivez le nouveau chemin ainsi construit et sa longueur.

Question III.5. Appliquez répétitivement votre procédure d'échange au chemin $(P_0, \ldots, P_{n-1})$ de longueur $n = 9$ jusqu'à ce qu'il n'y ait plus aucun échange intéressant possible. Combien avez-vous fait d'échanges ? Donnez la liste de leurs signatures. Décrivez le chemin final obtenu et sa longueur.

Question III.6. Pouvez-vous donner une valeur de n pour laquelle le chemin ainsi obtenu est minimal ? Pouvez-vous en donner une autre pour laquelle il ne l'est pas ? Décrivez dans chaque cas le chemin obtenu et sa longueur.

Question III.7. Quelle est la longueur du chemin obtenu par cette méthode pour les villes $P_0, \ldots, P_{N-1}$? Donnez la suite des signatures des échanges effectués. Combien de temps votre programme met-il pour le trouver ?

Partie IV. Méthodes hybrides

Vous avez exploré ci-dessus plusieurs méthodes d'approximation fondées sur des principes complètement différents. On peut bien sûr imaginer d'*hybrider* ces méthodes entre elles, par exemple en appliquant une méthode A au chemin trivial, puis en appliquant la méthode B au chemin obtenu, etc.

Question IV.1. Proposez une hybridation des deux méthodes ci-dessus qui vous semble intéressante et justifiez soigneusement votre proposition. Appliquez-la à l'ensemble des villes $(P_0, \ldots, P_{N-1})$. Discutez des résultats. Que se passerait-il pour N encore plus grand ?