Concours canadien de mathématiques

Une activité du Centre d'éducation
en mathématiques et en informatique
Université de Waterloo, Waterloo (Ontario)

Concours canadien d'informatique

pour les bourses de SYBASE®

Le mercredi 5 mars 1997

Problème A

Des phrases

Fichier des données : sent.in Fichier des résultats: sent.out

Écrire un programme qui, étant donné une liste de sujets, une liste de verbes et une liste de compléments, produira l'ensemble de toutes les possibilités de phrases ayant respectivement un sujet, un verbe et un complément.

Modalités pour l'entrée des données:

La première ligne du fichier des données est un entier *n* correspondant au nombre d'ensembles de données dans les lignes subséquentes. Pour chacun de ces *n* ensembles de données, les trois premières lignes donnent chacune un nombre, inférieur ou égal à 20, correspondant au nombre de sujets, de verbes et de compléments. Les lignes qui suivent énumèrent respectivement, un par ligne et en ordre alphabétique, ces sujets, ces verbes et ces compléments. La longueur maximale de chacun de ces sujets, verbes et compléments est de 25 caractères.

Modalités pour la production des résultats:

Le fichier des résultats doit comprendre toutes les phrases qu'il est possible de former d'un sujet, d'un verbe et d'un complément pour chacun des *n* ensembles de données. Ces phrases seront énumérées en ordre alphabétique et se termineront chacune par un point. Les résultats pour chacun des ensembles de données seront séparés d'une ligne vide.

Exemple d'un fichier des données:

```
1
3
3
2
Il
Le chat
Le chien
botte
mord
voit
celui-ci
la souris
```

... 2

Exemple du fichier des résultats

Il botte celui-ci.
Il botte la souris.
Il mord celui-ci.
Il mord la souris.
Il voit celui-ci.
Il voit la souris.
Le chat botte celui-ci.
Le chat botte la souris.
Le chat mord celui-ci.
Le chat mord la souris.
Le chat voit celui-ci.
Le chat voit la souris.
Le chien botte celui-ci.
Le chien botte la souris.
Le chien mord celui-ci.
Le chien mord la souris.
Le chien voit celui-ci.
Le chien voit la souris.

Problème B

De vilains nombres

Fichier des données : nasty.in Fichier des résultats: nasty.out

Pour les fins de ce problème un nombre vilain est un nombre qui a au moins deux paires de facteurs telles que la somme des facteurs d'une paire équivaut à la différence des facteurs de l'autre.

Ainsi, par exemple, le nombre 6 est un nombre vilain puisque 6 □ 1 = 6 , 2 □ 3 = 6, et 6 - 1 = 2 + 3; aussi, 24 est un autre nombre vilain puisque 12 - 2 = 6 + 4.

Écris un programme qui, étant donné une liste d'entiers positifs, déterminera si ceux-ci sont des nombres vilains ou non.

Modalités pour l'entrée des données:

Le fichier des données est tout simplement une liste d'entiers positifs présentés un par ligne. Le premier de ces nombres représente la quantité de nombres à vérifier et est inférieur ou égal à 20. Les nombres à vérifier suivent et sont inférieurs à 32 001.

Modalités pour la production des résultats:

Le fichier des résultats présentera une ligne pour chacun des nombres à vérifier. Chaque ligne indique le nombre vérifié et s'il est vilain ou non.

Exemple d'un fichier des données

```
4
6
24
30420
10078
```

Exemple du fichier des résultats

```
4 est vilain
24 est vilain
30420 est vilain
10078 n'est pas vilain
```

Problème C

Le tournoi à double élimination

Fichier des données : dkc.in Fichier des résultats: dkc.out

Dans certaines compétitions sportives, l'équipe championne est déterminée par un tournoi à double élimination. Lors de ces tournois, toutes les équipes continuent à jouer jusqu'à ce qu'elles aient deux défaites. On procède par ronde d'élimination: lors de chacune de ces rondes on établit les équipes qui se rencontrent selon le critère suivant: les équipes qui n'ont pas perdu sont jumelées entre elles et ne rencontrent jamais une équipe qui a une défaite. Dans chacune des rondes d'élimination, on organise le plus grand nombre de matchs possible. Après un certain nombre de rondes, il ne reste que deux équipes: une invaincue et l'autre ayant perdu une seule fois. À ce point, ces deux équipes se rencontrent dans une ronde d'un seul match. Si au terme de ce match, aucune n'a perdu deux fois, les deux équipes jouent à nouveau dans une ronde finale. Pour les fins du présent programme, il est sous-entendu qu'une ronde finale s'avère toujours nécessaire.

a) Écrire un programme qui étant donné le nombre d'équipes inscrites à la compétition produira un bilan du tournoi.

Modalités pour l'entrée des données:

La première ligne du fichier des données indique un nombre positif n correspondant au nombre de tournois à vérifier dans les lignes subséquentes. Les prochaines n lignes contiennent chacune un nombre positif t inférieur à 32 768 correspondant au nombre d'équipes dans chacun des n tournois à vérifier.

Modalités pour la production des résultats:

Pour chacun des tournois à vérifier la première ligne du bilan sera:

Ronde 0: 2 invaincues, 0 une défaite, 0 éliminées

Le reste du bilan sera constitué d'une ligne dans ce même format par ronde d'élimination et d'une dernière ligne qui indiquera le nombre de rondes que le tournoi a nécessité. Les bilans des différents tournois seront séparés d'une ligne vide.

Exemple d'un fichier des données:

```
1
2
```

... 2

Exemple du fichier des résultats:

Ronde 0: 2 invaincues, 0 une défaite, 0 éliminées
Ronde 1: 1 invaincues, 1 une défaite, 0 éliminées
Ronde 2: 0 invaincues, 2 une défaite, 0 éliminées
Ronde 3: 0 invaincues, 1 une défaite, 1 éliminées
Le tournoi a nécessité 3 rondes.

b) Si $t = 2^{(2^k)}$ équipes participent au tournoi, ou k est un entier, combien de rondes nécessitera le tournoi ?

c) Combien de matchs sont nécessaires pour un tournoi si t équipes sont inscrites ?

Problème D

Codage par répertoire dynamique

Fichier des données: ddc.in Fichier des résultats: ddc.out

Le codage par répertoire est une approche répandue de compression de textes qui consiste à remplacer chacun des mots d'un texte par un nombre qui représente sa position dans un répertoire. Un tel codage est dit statique si le répertoire de mots est connu à l'avance. Le principal inconvénient de cette méthode réside dans le fait que ce même répertoire doit aussi être connu pour décomprimer le texte. D'autre part, un codage par répertoire dynamique contourne ce problème en dérivant le contenu du répertoire de mots à partir du texte à être comprimé. Au début du procédé le répertoire est vide. En considérant le texte à partir du début, lorsqu'un mot figure dans le répertoire il est remplacé par le numéro de sa position et s'il n'est pas dans le répertoire il est ajouté à la fin et est laissé tel quel dans le texte.

a) Écrire un programme qui étant donné un texte, le comprime dans un code par répertoire dynamique.

Modalités pour le fichier des données:

La première ligne du fichier des données est un entier représentant le nombre de textes à comprimer. Chacun des textes subséquents consiste de plusieurs lignes, chacune strictement constituée de mots en lettres minuscules et d'espaces (il n'y aura aucun mot accentué dans cet exercice). Les différents textes sont séparés d'une ligne vide, chacun comprenant au plus 100 lignes d'une longueur maximale de 80 caractères formant des mots de 20 lettres ou moins.

Modalités pour le fichier des résultats:

Le fichier des résultats contiendra l'ensemble des textes comprimés par codage dynamique tel que décrit ci-dessus espacés d'une ligne vide un de l'autre. Les changements de lignes et l'espacement sont maintenus comme dans les textes de départ.

Exemple d'un fichier des données:

```
1
le chat pourchassait le rat tandis que
le chien pourchassait le chat chez le rat
```

Exemple du fichier des résultats:

le chat pourchassait 1 rat tandis que
1 chien 3 1 2 chez 1 4

b) Pourquoi, lorsqu'un mot est utilisé pour la première fois, est-il ajouté au répertoire mais son code n'est pas utilisé dans le texte ?

c) Quelle difficulté entraînerait l'utilisation de nombres dans les textes de départ ? Comment pourrait-on modifier la méthode pour contourner ce problème ?

Problème E

La grande division

Fichier des données: div.in Fichier des résultats: div.out

Jadis (1965), la calculatrice mécanique effectuait les divisions par soustractions répétées et par décalages. Par exemple, pour diviser 987 654 321 par 3 456 789, on alignait les nombres à partir des chiffres de gauche (voir [1] plus bas), et le diviseur était soustrait du dividende le plus grand nombre de fois sans obtenir un reste négatif. Le nombre de soustractions ainsi obtenu devient le premier chiffre du quotient. Le diviseur est ensuite décalé d'une position vers la droite (voir [2] plus bas) et à nouveau soustrait successivement du reste pour obtenir le prochain chiffre du quotient. Ce procédé est répété jusqu'à ce que le reste soit plus petit que le diviseur.

```
    987654321
-   3456789          ← première soustraction valable      [1]
    ========
    641975421
-   3456789          ← seconde soustraction valable
    ========
    296296521        ← reste
-   3456789          ← soustraction rejetée
    ========
    négatif

    296296521
-    3456789                                              [2]
    ========
    261728631
          etc.
```

a) Écris un programme qui effectue la grande division.

Modalités pour le fichier des données:

La première ligne du fichier des données est un entier positif n , $n < 20$, indiquant le nombre de divisions à effectuer. Chaque paire de lignes subséquente représente respectivement le dividende et le diviseur d'une division. Les nombres fournis seront positifs et d'une longueur maximum de 80 chiffres.

. . . 2

Modalités pour le fichier des résultats:

Pour chaque dividende et diviseur fournis, le fichier des résultats contiendra une paire de lignes indiquant respectivement le quotient et le reste. Les réponses des différentes divisions seront séparées d'une ligne vide et ne seront précédées d'aucun zéro.

Exemple d'un fichier des données:

```
3
987654321
3456789
33
11
11
33
```

Exemple du fichier des résultats:

```
285
2469456

3
0

0
11
```

b) Si le dividende contient n chiffres et le diviseur m chiffres, donne une formule en termes de m et n qui fournit un nombre approximatif de soustractions simples d'un chiffre effectuées par votre programme.