COMPÉTITION
INFORMATIQUE
http://www.gel.usherb.ca/grpetudiants/concours_FCI

QUESTIONNAIRE DE
LA FOLLE COURSE INFORMATIQUE

UNIVERSITÉ DE
SHERBROOKE

Faculté de génie

Département de génie électrique et de génie informatique

ÉCOLE
POLYTECHNIQUE
MONTRÉAL

Département de génie électrique et de génie informatique

Novembre 1998

Quelques mots sur la folle course

Nous sommes déjà à la quatrième édition de la Folle course informatique. En faisant une histoire courte, la première édition s'était déroulée uniquement à Sherbrooke (Organisée par le Département de génie électrique et de génie informatique). À la deuxième et troisième édition, l'École Polytechnique de Montréal (Département de génie électrique et de génie informatique) a organisé la course dans ses locaux, grâce à l'appui de Martine Bellaïche, mais tout le questionnaire a été fait à Sherbrooke.

Nous espérons avoir une forte participation légèrement supérieure aux autres années, car si la Folle course informatique existe, c'est grâce aux participants qui prennent un samedi pour travailler pour le plaisir.

Pour cette année voilà les nouveautés :

L'École Polytechnique a proposé quelques problèmes, donc le questionnaire porte les logos de l'Université de Sherbrooke et de l'École Polytechnique. Encore grâce à Martine Bellaïche.

Nous avons décidé de donner le crédit aux personnes ayant eu les idées de problèmes et qui ont rédigé les spécifications correspondantes. Les noms des auteurs sont associés à chaque problème. Cependant, nous pensons qu'il est aussi nécessaire de nommer toutes les personnes participant à la lecture du questionnaire, par ordre alphabétique : Martine Bellaïche, Alex Boisvert, Paule Bolduc, Jean-Denis Boyer, Jean-Marie Dirand, Ruben Gonzalez-Rubio, Louis Guindon, Jean-Yves Hervé, Sophie Mc Nicoll et Ginette Roy.

Cette année, Heenan Blaikie est le parrain de l'édition 98. Il offre un prix de 1 000$ à l'équipe gagnante de la Folle course informatique, au niveau de **tous** les participants.

Nous avons décidé de changer l'attribution de bonus pour qu'elle soit indépendante du site de la course, ainsi la répartition de bonus sera plus équitable et rendra la course plus intéressante.

Introduction

Lisez ce préambule au questionnaire de la quatrième édition de la *Folle Course Informatique*. Il contient un résumé du règlement de la Course et diverses instructions reliées à la soumission des programmes.

La Folle Course Informatique

La Folle Course Informatique est une compétition de programmation. Les équipes participantes doivent écrire des programmes selon des spécifications données. Chaque programme peut vous faire gagner des points. Pour gagner, une équipe doit accumuler plus de points que les autres en résolvant les problèmes qui sont présentés aux participants.

Pour chaque spécification, vous devez concevoir un programme C ou C++[1] qui respecte la spécification. Ce programme sera testé avec un ou plusieurs fichiers de test, un nombre de points vous sera accordé selon les fichiers correctement traités par votre programme. Le nombre de tests et de points correspondant est indiqué à la fin de chaque problème. Les mêmes jeux de fichiers de test seront utilisés pour évaluer les programmes de toutes les équipes.

Un temps maximum est alloué pour l'exécution d'un programme, il est clairement indiqué dans l'énoncé. Si un programme dépasse le temps alloué pour un fichier d'entrée donné, il sera assumé qu'il a échoué ce test. Il est garanti qu'il existe une solution produisant des résultats corrects à l'intérieur de la limite de temps proposée.

Nouveauté pour le bonus Les points bonus seront donnés aux équipes qui vont réussir **tous les tests du programme**, ceci en fonction de la difficulté et l'heure de dépôt.

Le tableau suivant indique les points bonus qui seront accordés :

Numéro de problème	Bonus à t_0	Bonus à la fin de la course
1, 2, 3	30 %	0 %
4, 5, 6, 7, 8	40 %	0 %
9, 10	50 %	0 %

où t_0 est à l'instant de démarrage de la compétition et t_f est la fin de la compétition.

[1] À Sherbrooke nous avons choisi ces deux langages, cependant, le correcteur permet d'utiliser d'autres langages compilés. Il faut voir si votre organisation locale a pris un langage différent, dans ce cas il faut consulter les procédures particulières pour la soumission de programmes. Dans le questionnaire, nous faisons référence uniquement aux langages C et C++.

Ceci veut dire que le bonus accordé diminue de manière linéaire, au fur et à mesure que le temps avance. Par exemple, si après quatre heures de compétition une équipe livre le programme de la question 10 et qu'il passe tous les tests, il aura un bonus de 25 %. Si une autre équipe livre le programme de la question 10 après six heures de compétition et qu'il passe tous les tests, il aura un bonus de 12,5 %.

Le bonus est un pourcentage calculé selon la formule suivante :

$$B = p \times \frac{t_r}{t_t}$$

où, B est le pourcentage de bonus à recevoir, p est le pourcentage de départ à t_0, t_r est le temps restant lorsque le programme est livré et t_t est le temps total de la course. La granularité du temps est celle du correcteur, elle est estimée à deux secondes.

En une phrase, plus vite est livré un programme, plus grand sera le bonus, bien entendu à condition que le programme passe tous les tests.

En huit heures, il est peu probable mais pas impossible qu'une équipe dispose de suffisamment de temps pour programmer tous les problèmes présentés. Vous devrez faire preuve de jugement en choisissant les problèmes auxquels vous vous attaquez.

Nous avons tenté de présenter les problèmes de façon uniforme et sans ambiguïté. Chaque énoncé comporte une description de spécification (non formelle) mais claire du problème à résoudre, ainsi que des exemples et des fichiers impliqués dans le traitement.

À la fin de la course, le classement des équipes sera établi selon le total des points accumulés pour chacun des programmes soumis. Dans l'éventualité où deux équipes obtiendraient un nombre égal de points, c'est l'équipe qui aura atteint ce pointage le plus tôt qui sera considérée gagnante (en d'autres mots, l'équipe qui aura soumis le plus tôt son dernier programme ayant obtenu des points).

La soumission des programmes

Vous devez soumettre un seul fichier source pour chaque problème. Ce fichier aura l'extension "`.C`" s'il doit être compilé en C, "`.CPP`" s'il s'agit de C++. La première partie du nom sera composée du numéro du problème suivi du numéro de votre équipe selon le format "`P##_EQ##`". Par exemple, le problème 3 de l'équipe 9, codé en C++, porterait le nom "`P03_EQ09.CPP`" et "`P03_EQ15.CPP`" pour le même problème de l'équipe 15. Le nom du fichier comporte exactement 8 caractères avant le point. **Les fichiers qui ne se conforment pas à ce format ne seront pas considérés pour la correction**.

Chaque fichier source soumis sera compilé afin de produire un programme exécutable. Ce programme sera exécuté plusieurs fois pour traiter les fichiers de test. Les fichiers de sortie seront analysés afin de vérifier s'ils sont conformes aux spécifications, des points seront accordés en conséquence. Un programme doit compiler sans aucune erreur (les avertissements (`warnings`) seront tolérés). Un programme qui ne compile pas ne rapportera donc aucun point. À la correction, les sorties sur la console (comme `printf` ou autres) sont aussi tolérées, mais le temps d'exécution augmente rapidement. Il est donc conseillé de les éviter afin de ne pas dépasser le temps alloué. Pour finir les conseils,

votre code source ne sera pas examiné, vous avez la liberté totale sur le style de programmation à utiliser.

Notez que la correction est automatisée et s'effectue pendant la course en temps réel. À moins d'un problème technique avec le système de correction, vous pourrez consulter vos résultats quelques instants après la soumission de votre programme sur un écran placé dans votre local.

Les entrées et sorties se font toujours via des fichiers textes ASCII. Ceux-ci seront nommés selon le numéro du problème avec le format "`P##.ENT`" pour les fichiers en entrée et "`P##.SOR`" pour les fichiers en sortie. Le `##` indique le numéro du problème, il varie entre `01` et `10`. Leur contenu et la façon de les utiliser sont clairement définis dans l'énoncé de chaque problème. De plus, un exemple est présenté pour chacun de ces fichiers.

Prenez pour acquis que les fichiers d'entrée qui seront utilisés pour tester vos programmes suivront rigoureusement le format qui est indiqué dans chaque problème. Il n'est pas nécessaire de programmer des vérifications sur ces données. Les exemples de fichiers montrés dans le texte vous seront fournis.

Il est crucial que les fichiers produits par vos programmes respectent rigoureusement les spécifications puisqu'ils seront corrigés automatiquement.

Lorsque vous allez soumettre votre programme pour son évaluation, vous devrez le copier dans une « boîte de dépôt » qui vous sera indiquée lors de la course ainsi que la procédure exacte. Vous ne pourrez déposer qu'une seule fois votre solution pour chaque problème. Une fois un programme soumis, aucune modification ne sera acceptée. Si vous déposez votre programme de nouveau, la nouvelle copie sera ignorée.

Le règlement

- Un seul ordinateur sera assigné à chaque équipe. Une équipe ne peut utiliser que l'ordinateur qui lui est assigné. En cas de panne, il faut attendre qu'un organisateur assigne un nouvel ordinateur.

- Une salle de réflexion à proximité de cette dernière sera également mise à la disposition des équipes.

- Vous avez le droit d'apporter et d'utiliser toute documentation pertinente, en autant que celle-ci soit imprimée. Tout support matériel (autre que documents) est interdit dans les locaux de la course, incluant disquettes et ordinateurs portatifs. **La présence de matériel non-autorisé sera un facteur jugé suffisant pour disqualifier une équipe**.

- Le réseau local sera coupé du monde extérieur. L'Internet sera inutilisable.

- Afin d'éviter les accidents fâcheux, toute nourriture ou boisson sera interdite dans les locaux des ordinateurs.

- L'honnêteté et la bonne foi des participants est de mise.

Remarques finales

La Folle Course Informatique est organisée par une équipe de bénévoles, cette équipe se renouvelle à chaque édition. Nous nous efforçons d'écrire des spécifications aussi claires que possible, ceci pour lever toute ambiguïté même au détriment d'un style épuré.

Nous avons écrit des programmes répondant aux spécifications, nous avons aussi écrit des jeux de fichiers de tests et des correcteurs de programmes, en mettant beaucoup d'efforts et du temps. Même pendant la course, nous faisons des vérifications afin de s'assurer que tout est correct et que tout se passe de manière équitable. Toutefois, nous ne prétendons pas être arrivés à la perfection. Pour cette raison, nous vous demandons de faire appel à votre esprit « sportif-informatique » afin d'accepter les classements « officiels » donnés à la fin de la course. En effet, il est pratiquement impossible de changer la répartition des prix si des changements dans le classement se produisaient. Nous pensons que la plus grande récompense associée à cette compétition est la satisfaction d'avoir fait un effort pour écrire des programmes et éventuellement d'avoir appris quelque chose. Cependant, nous sommes ouverts à des remarques qui pourraient améliorer les futures compétitions ou pour nous signaler une erreur.

Le prix de 1 000 $ sera attribué quelques jours après pour s'assurer que toutes les vérifications possibles ont été faites.

Conventions utilisées dans le questionnaire

Pour indiquer le début et la fin d'un fichier nous utilisons les symboles ▷ et ◁ respectivement. Bien entendu ces symboles ne font pas partie du fichier.

Exemple, le fichier suivant contient une seule ligne avec la chaîne `hello`.

```
▷
hello
◁
```

Table des matières

Les Spécifications

Spécification 1

Les étendues d'eau

Ruben Gonzalez-Rubio

`Fichier à produire contenant votre code source : P01_EQ##.*`

Ce problème[1] est un traitement possible des images numériques ou bitmap. Le traitement à effectuer servira à repérer des surfaces d'eau exploitables dans la mesure où leur taille dépasse une certaine dimension. Nous avons imposé quelques simplifications :

- Les images sont en noir et blanc. Le noir représente une surface d'eau et le blanc toute surface sauf celle de l'eau.
- À l'écran, les points ou pixels sont considérés comme des carrés.
- Les images ont été manipulées pour pouvoir appliquer des règles simples pour définir les surfaces d'eau exploitables.

Problème

D'après une grille (ou le bitmap) de $n \times n$ carrés où certains d'entre eux sont occupés, un exemple est à la figure 1.1. On désire identifier les carrés d'eau représentés par des carrés noirs.

Soit les règles suivantes :

- Une étendue d'eau exploitable est formée par au moins deux carrés noirs.
- Deux carrés appartiennent à la même étendue d'eau exploitable s'ils ont un côté commun.

[1] Ce problème a été inspiré d'un exercice proposé par M. A. Weiss.

Il faut calculer le nombre de surfaces d'eau exploitables dans la grille et la surface totale occupée par ces surfaces d'eau exploitables.

Par exemple, dans la figure 1.1, nous avons une grille de 10×10. Il existe, selon les règles données, cinq étendues d'eau exploitables et la surface totale occupée par ces étendues est de dix-sept. Notez qu'il y a sept carrés qui ne font pas partie d'aucune surface exploitable et qu'ils ne sont pas comptés dans la surface totale.

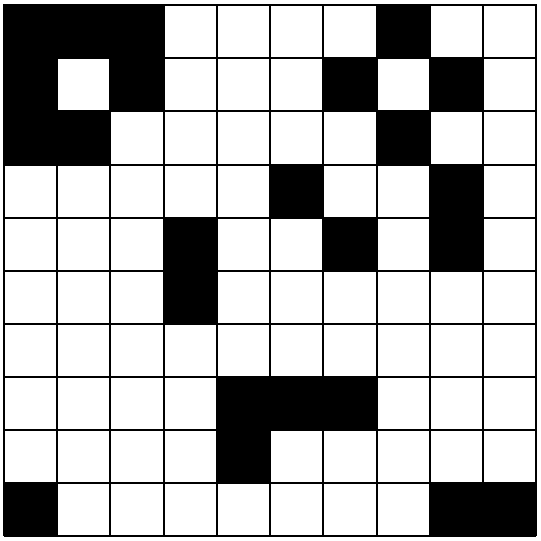

FIG. 1.1 – La surface totale

Fichier d'entrée

Sur la première ligne, on trouve la taille de la grille, comme les grilles sont toujours carrées, un seul entier suffit pour indiquer sa taille. Pour représenter la grille, nous utilisons la convention suivante : un zéro `0` représente une case blanche et un `1` une case noire. Le fichier que vous devez lire est "`P01.ENT`". Chaque ligne contient une suite de uns et de zéros représentant une ligne de la grille. Les `1`s et `0`s sont séparés par un espace. La taille d'une grille sera inférieure ou égale à 1000×1000.

Voici le fichier d'entrée correspondant à l'exemple montré sur la figure 1.1 :

```
▷
10
1 1 1 0 0 0 0 1 0 0
1 0 1 0 0 0 1 0 1 0
1 1 0 0 0 0 0 1 0 0
0 0 0 0 0 1 0 0 1 0
0 0 0 1 0 0 1 0 1 0
```

```
0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 0 0 0
0 0 0 0 1 0 0 0 0 0
1 0 0 0 0 0 0 0 1 1
```

◁

Fichier de sortie

Vous devez écrire le fichier "`P01.SOR`". Il doit contenir le nombre d'étendues d'eau exploitables sur une ligne et la surface totale occupée par ces étendues sur une autre ligne.

Voici le fichier de sortie obtenu à partir de la grille du fichier d'entrée :

▷

```
5
17
```

◁

Le pointage

Le problème sera corrigé en soumettant votre programme à quatre fichiers d'entrée différents . Un fichier de test sera considéré comme réussi, si les résultats sont conformes aux spécifications données.

Nombre de fichiers réussis	Points accordés
1	80
2	120
3	160
4	200

Temps maximal d'exécution de votre programme : 20 secondes.

Spécification 2

Le tri de palindromes

Ruben Gonzalez-Rubio

`Fichier à produire contenant votre code source : P02_EQ##.*`

Le mot « Palindrome » vient du Grec palindromos, «qui revient sur ses pas». Les palindromes sont des mots ou des suites de mots se lisant de gauche à droite ou de droite à gauche de la même manière, un exemple classique est le mot "*radar*". Il existe aussi des nombres palindromes présentant la même caractéristique de symétrie. Ce problème demande de trier des palindromes en prenant comme critère les lettres du milieu du palindrome.

Problème

Nous avons donné une définition non formelle du problème, maintenant soyons plus précis. Soit un alphabet α des symboles (les symboles imprimables du code ASCII) sauf le symbole ", ce symbole servira pour indiquer le début et la fin d'une chaîne. Une chaîne est formée par des symboles de l'alphabet. Une chaîne est un palindrome si $C[1..n] = C[n..1]$ où n est le nombre de symboles du palindrome. Les palindromes doivent être triés selon la lettre ou les lettres se situant au milieu du palindrome.

Il y a deux cas à distinguer : si n est impair et si n est pair. Dans le cas où n est impair, il y a un seul symbole au milieu du palindrome. Dans le cas où n est pair, il y a obligatoirement deux symboles égaux au milieu du palindrome. Il faut placer un palindrome avec un nombre impair de symboles avant un de nombre pair, si les deux ont les mêmes symboles au milieu. Par exemple `aba` précède `abba`, sinon on prend en compte l'ordre des symboles du code ASCII pour définir l'ordre alphabétique pour effectuer le tri, rappelez-vous les espaces et autres caractères précèdent les chiffres, les chiffres précèdent les majuscules, et les majuscules précèdent les minuscules.

Fichier d'entrée

Le fichier que vous devez lire est "`P02.ENT`". Le fichier contient plusieurs lignes, sur chaque ligne se trouve un palindrome. Afin de bien délimiter le palindrome, il est présenté entre des symboles " et ". Un exemple de fichier en entrée est :

▷

```
"abba"
"radar"
"ceci est un radar nu tse icec"
"La folle course informatique euqitamrofni esruoc ellof aL"
"aca"
"adda"
"addda"
"la folle course informatiqueeuqitamrofni esruoc ellof al"
"la folle course informatiqueuqitamrofni esruoc ellof al"
"zabaz"
"dabbad"
"aba"
```

◁

Il est garanti que le fichier contient au moins une ligne, que les lignes ne sont pas vides et qu'elles contiennent des palindromes. Les lignes auront au maximum 256 caractères en incluant les caractères de début et fin de chaîne. Il y aura au maximum 1024 palindromes.

Fichier de sortie

Vous devez écrire le fichier "`P02.SOR`" qui contiendra les lignes triées selon la spécification. Voilà celui qui correspond au fichier d'entrée présenté ci-haut :

▷

```
"La folle course informatique euqitamrofni esruoc ellof aL"
"aba"
"abba"
"dabbad"
"zabaz"
"aca"
"adda"
"ceci est un radar nu tse icec"
"radar"
"addda"
"la folle course informatiqueuqitamrofni esruoc ellof al"
"la folle course informatiqueeuqitamrofni esruoc ellof al"
```

◁

Le pointage

Le problème sera corrigé en soumettant votre programme à quatre fichiers différents d'entrée. Un fichier de test sera considéré comme réussi, si les résultats sont conformes aux spécifications données.

Nombre de fichiers réussis	Points accordés
1	80
2	120
3	160
4	200

Temps maximal d'exécution de votre programme : 20 secondes.

Spécification 3

La cache du fureteur

Alex Boisvert

`Fichier à produire contenant votre code source : P03_EQ##.*`

Comme beaucoup d'internautes, vous avez sûrement déjà utilisé un programme tel Netscape Navigator ou Microsoft Internet Explorer pour consulter des documents sur l'Internet. De tels programmes, appelés *fureteurs*, vous présentent visuellement des documents électroniques composés de plusieurs fichiers. En effet, un document HTML peut être composé de plusieurs images.

Afin de réduire le temps de chargement des pages web, les fureteurs conservent une copie des fichiers accédés fréquemment sur un disque local. Ainsi, lorsque vous reconsultez une même page (ou une partie de cette page), le fureteur accède directement aux fichiers disponibles sur le disque local plutôt que de les retransférer par le réseau, sauvant ainsi un temps appréciable lors de l'affichage d'une page web donnée. On appelle généralement le principe de conserver localement des données coûteuses à récupérer une *cache*.

Le concept utilisé pour déterminer quels fichiers sont à conserver sur le disque local est appelé MRU (pour Most Recently Used). C'est-à-dire que le programme conserve les fichiers les plus récemment consultés, jusqu'à concurrence d'un espace disque donné. Par exemple, on pourrait allouer 5 mégaoctets à des fins de cache sur disque.

Problème

Il vous faut développer un programme qui détermine quels sont les fichiers à conserver sur le disque local. Pour la réalisation, vous devez utiliser le concept MRU appliqué à des noms de fichiers qui seront fournis à votre programme.

Votre programme doit retenir les noms des fichiers les plus récemment utilisés, sans que la taille de ceux-ci ne dépasse jamais la taille maximale de la cache. Autrement dit, lorsque la somme des

tailles des fichiers disponibles localement dépasse la taille de l'espace disponible pour la cache, votre programme doit éliminer le ou les fichiers les plus anciennement accédés.

Fichier d'entrée

Le fichier que vous devez lire est "`P03.ENT`". Il est composé de deux éléments principaux: la taille de la cache et une liste d'accès à différents fichiers. La première ligne du fichier donne la taille de la cache en kilooctets.

Ensuite, les lignes suivantes représentent l'accès chronologique à différents fichiers. Chaque accès à un fichier est composé de deux données: le nom du fichier suivi de la taille du fichier en kilooctets. Les deux données sont séparées par un espace (caractère ASCII 32 en décimal).

```
▷
600
allo.gif 34
bonjour-fr.html 2
bonjour-en.html 2
mim98.ppt 218
point.jpg 39
readme.html 23
encore.doc 312
mim98.ppt 218
allo.gif 34
bonjour-fr.html 2
◁
```

Quelques notes:

- La liste de fichiers accédés peut contenir entre 1 et 1000 éléments.
- Si la taille d'un fichier est plus grande que la taille de la cache, on ignore simplement ce fichier.
- La taille de la cache et des fichiers varie entre 1 et 100000 kilooctets

Fichier de sortie

Vous devez écrire le fichier "`P03.SOR`" qui contiendra la liste des fichiers disponibles localement par le fureteur après avoir effectué les accès donnés en entrée et appliqué le concept MRU. Un seul nom de fichier doit apparaître par ligne. L'ordre d'apparition des fichiers est libre à vous.

```
▷
allo.gif
readme.html
encore.doc
mim98.ppt
bonjour-fr.html
◁
```

Le pointage

Le problème sera corrigé en soumettant votre programme à quatre fichiers différents d'entrée. Un fichier de test sera considéré comme réussi, si en appliquant le concept MRU à la liste chronologique des accès aux fichiers en entrée, on obtient une cache contenant les fichiers les plus récemment accédés, sans dépasser la taille maximale de la cache.

Nombre de fichiers réussis	Points accordés
1	80
2	120
3	160
4	200

Temps maximal d'exécution de votre programme : 20 secondes.

Spécification 4

Agence de rencontres

Ruben Gonzalez-Rubio

`Fichier à produire contenant votre code source : P04_EQ##.*`

Pour rencontrer la princesse ou le prince charmant, il ne faut pas aller le ou la chercher dans un jeu Nintendo. Il faut aller dans une agence spécialisée pour la ou le trouver. Les agences procèdent de plusieurs manières; une manière simple est de réunir toutes les personnes faisant une demande dans une salle et laisser au hasard la possibilité d'avoir des coups de foudre entre deux participants. Il y a bien sûr, la manière « scientifique ». On réunira autour d'une table, un couple présentant des affinités selon des réponses à des questions bien choisies. Afin que l'agence soit rentable, on doit avoir un grand nombre de tables à remplir.

Problème

Vous avez à programmer des rencontres de manière scientifique. Les rencontres se font dans une salle où il y a des tables pour deux personnes (les deux personnes qui vont se rencontrer). Afin de simplifier le problème nous allons considérer qu'un couple est composé de deux personnes de sexe différent. Nous allons former des couples à partir des listes de filles et de garçons. Le nombre de filles sera égal au nombre de garçons.

Chaque demandeur de rencontre va remplir un questionnaire de 10 questions. Les questions sont du type « aimez-vous la musique? » La réponse est quantifiée selon l'échelle suivante :

Sans réponse	Pas de tout	Très peu	Ni oui, ni non	Un peu	Beaucoup
0	1	2	3	4	5

Parfois, la personne répondant au questionnaire ne veut pas répondre à une question et elle indique aucune réponse, ceci est noté comme un zéro, tel que c'est indiqué dans l'échelle.

Pour évaluer si deux personnes ont des affinités nous allons utiliser la table suivante :

Personne 1	Personne 2	Points pour le couple par question
4 ou 5	4 ou 5	10
3	3	9
1 ou 2	1 ou 2	8
4 ou 5	3	7
3	4 ou 5	7
1 ou 2	3	6
3	1 ou 2	6
1 ou 2	4 ou 5	5
4 ou 5	1 ou 2	5
0	0	2
3	0	1
0	3	1
0	4 ou 5	0
4 ou 5	0	0
0	1 ou 2	0
1 ou 2	0	0

Afin d'identifier les filles, on utilise des entiers compris entre 1000 et 4999 et pour identifier les garçons, on utilise des entiers compris entre 5000 et 9999. Ceci n'est pas très romantique mais simplifie énormément les entrées et sorties.

Pour chaque fille et pour chaque garçon, nous avons gardé les dix réponses clés du questionnaire.

Le problème est de trouver la répartition optimale des couples, c'est-à-dire, qu'il faut trouver la répartition (un ensemble de couples) qui donnera le plus grand total des points de manière globale.

Fichier d'entrée

Le fichier que vous devez lire est "`P04.ENT`". Sur la première ligne on trouve le nombre de couples à former. Ensuite, sur chaque ligne il y a un identificateur (fille ou garçon) suivi de ses réponses aux dix questions. La première moitié du fichier contient des filles. Donc, sur chaque ligne se trouve un entier compris entre 1000 et 4999 identifiant une fille suivi des dix entiers donnant les réponses aux questions. Bien entendu, tous ces entiers sont séparés par un espace. La deuxième moitié du fichier contient des garçons. Donc sur chaque ligne se trouve un entier entre 5000 et 9999 identifiant un garçon suivi de dix entiers correspondant aux réponses du questionnaire. Voici un exemple de fichier :

```
▷
3
1002 5 2 3 4 5 5 2 3 0 5
1003 1 2 3 4 3 1 3 3 4 5
1005 1 2 3 4 5 1 2 3 4 2
```

```
5002 0 1 0 4 1 1 3 1 3 1
5005 3 5 2 5 5 2 5 2 5 5
5003 0 2 3 4 0 1 2 0 4 5
```

◁

Il est garanti que le nombre de filles et le nombre de garçons sont toujours égaux. Que les filles précèdent les garçons dans les fichier. Chaque fille est unique, c'est-à-dire qu'elle apparaît une seule fois dans l'ensemble de filles. Pour les garçons c'est la même chose, chacun est unique. Mais deux ou plusieurs filles (ou garçons) peuvent donner les mêmes réponses au questionnaire. Il y aura au maximum six couples à former.

Fichier de sortie

Vous devez écrire le fichier "`P04.SOR`" qui contiendra le grand total de points obtenus par une répartition optimal des couples ainsi que la liste des couples formés. Le format : sur la première ligne, l'entier donnant le total obtenu par l'ensemble; sur les lignes suivantes les couples, un couple par ligne, l'entier identifiant la fille suivi de l'entier identifiant le garçon suivi du nombre de points obtenus par le couple.

▷

```
186
1002 5005 64
1003 5003 63
1005 5002 59
```

◁

Le problème peut avoir plusieurs solutions, le test vérifie que votre solution correspond à une solution optimale.

Le pointage

Le problème sera corrigé en soumettant votre programme à quatre fichiers différents d'entrée. Un fichier de test sera considéré comme réussi, si les résultats sont conformes aux spécifications données.

Nombre de fichiers réussis	Points accordés
1	120
2	180
3	240
4	300

Temps maximal d'exécution de votre programme : 20 secondes.

Spécification 5

Les amis

Alex Boisvert

`Fichier à produire contenant votre code source : P05_EQ##.*`

Vous avez sûrement déjà entendu le dicton populaire suivant: "Les amis de mes amis sont mes amis". Dans la réalité, cependant, on remarque que souvent même nos meilleurs amis ont des amis qu'on n'aime pas personnellement. Ou l'inverse, c'est-à-dire que certains de nos amis peuvent détester des gens qu'on aime. C'est la vie ...

Comment alors se forment les cercles d'amis?

Problème

Pour un groupe de personnes donné, il existe certaines relations entre certains individus. À la base, il y a deux types de relation : "aime" ou "déteste". Par exemple, une relation pourrait être "Julien aime Amélie", ou bien "Julien déteste Igor". Chaque relation implique deux personnes et chaque relation est réciproque. En conséquence, si "Julien déteste Igor", Igor déteste aussi Julien.

Tel qu'illustré dans la figure 5.1, un graphe peut être créé pour représenter les relations existant dans un groupe. Dans cette illustration, les traits droits représentent les relations "aime" alors que les traits ondulés représentent les relations "déteste".

La relation "aime" peut être communiquée à plusieurs niveaux. C'est-à-dire que si "Julien aime Amélie" et que "Amélie aime Élise", on peut en conclure que Julien aime Élise.

La relation "déteste" peut aussi être communiquée à plusieurs niveaux, mais toujours en suivant une relation "aime". Ainsi donc, si "Julien déteste Igor" et que "Amélie aime Julien", on saura en conclure que Amélie déteste Igor.

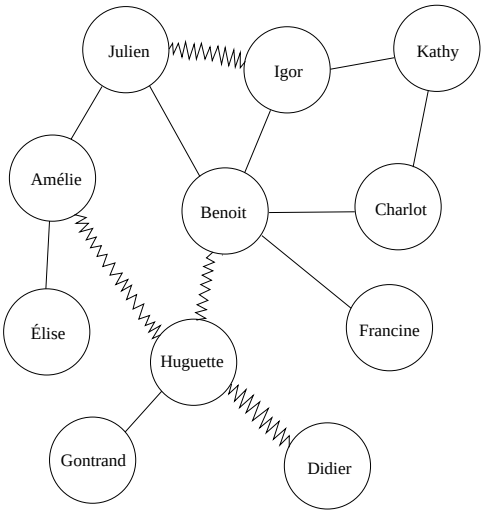

FIG. 5.1 – Relations dans un groupe de personnes

Ceci, hélas, complique les relations amoureuses. Comme vous pouvez le constater, une personne pourrait se retrouver dans la situation où elle aime quelqu'un et elle le déteste en même temps! C'est cruel mais c'est la vie.

Pour différencier le vrai amour d'un amour conflictuel, nous introduisons un autre concept. Une personne "aime vraiment" une autre personne si elle l'aime et qu'elle n'a aucune raison de la détester par le biais de ses amis. Ainsi, on ne peut pas "aimer vraiment" une personne qu'un de nos amis déteste.

Expliquons-nous en étant plus formel. Une personne, P1, "aime vraiment" une autre personne, P2, SI elle a une relation "aime" (directe ou indirecte) avec P2 ET qu'elle n'a pas de relation "déteste" (directe ou indirecte) avec P2. Il y a cependant un cas d'exception. Une personne, P1, ne peut pas détester une autre personne, P2, par relation indirecte si P2 se retrouve trois fois dans la relation indirecte. Une façon de raisonner ceci est qu'on ne peut détester quelqu'un parce qu'il est notre ami et qu'un de ses amis (à lui) le déteste. Cet exemple est illustré par le fait que Amélie "aime vraiment" Julien même si Amélie aime Igor (indirectement) et que Igor déteste Julien. Dans ce cas, Julien se retrouve trois fois dans la relation indirecte: Amélie aime Julien, Julien aime Benoît, Benoît aime Igor et Igor déteste Julien.

Finalement, définissons un "cercle d'amis" comme étant un groupe de personnes dans lequel toutes les personnes "s'aiment vraiment" mutuellement. Selon cette définition, on pourrait affirmer que Amélie, Élise et Julien forment un cercle d'amis. Cependant, Igor, Julien et Benoît ne forment pas un cercle d'amis car Julien et Igor se détestent et Benoît n'aime pas vraiment ni Igor, ni Julien.

Quelques remarques :

- Un groupe de personnes peut compter jusqu'à 100 individus, chacun ayant un prénom différent.

- Un cercle d'amis comprend nécessairement plus d'une personne.
- Les prénoms des personnes sont composés des lettres A jusqu'à Z en minuscules ou majuscules et peuvent contenir un trait d'union (ex: Jean-Marie). Les noms ne contiendront pas d'accents.

Votre programme doit déterminer si des groupes de personnes donnés sont des cercles d'amis.

Fichier d'entrée

Le fichier que vous devez lire est "`P05.ENT`". L'entrée est divisée en deux parties. La première partie consiste en une séquence de relations. Il peut y avoir entre 1 et 300 relations. Aucune relation ne peut être répétée, même dans la forme réciproque.

La deuxième partie représente des groupes de personnes. Chaque groupe est présenté sur une ligne, les prénoms des personnes sont séparés d'un espace.

Les deux parties du fichier d'entrée sont séparées par une ligne vide.

▷

```
Amelie aime Elise
Amelie aime Julien
Benoit aime Julien
Igor aime Benoit
Igor aime Kathy
Kathy aime Charlot
Charlot aime Benoit
Huguette aime Gontrand
Francine aime Benoit
Huguette deteste Amelie
Didier deteste Huguette
Benoit deteste Huguette
Igor deteste Julien

Amelie Elise Julien
Benoit Charlot Francine Kathy
Gontrand Huguette
Benoit Igor Julien
```

◁

Fichier de sortie

Vous devez écrire le fichier "`P05.SOR`" qui contiendra si oui ou non les groupes de personnes données forment des cercles d'amis.

L'ordre de sortie des cercles d'amis et des noms n'a pas d'importance.

▷

```
oui
oui
oui
non
```

◁

Le pointage

Le problème sera corrigé en soumettant votre programme à quatre fichiers d'entrée différents. Un fichier de test sera considéré comme réussi si vous réussissez à identifier correctement tous les cercles d'amis existant dans le groupe donné.

Nombre de fichiers réussis	Points accordés
1	120
2	180
3	240
4	300

Temps maximal d'exécution de votre programme : 20 secondes.

Spécification 6

Le billard

Louis Guindon

`Fichier à produire contenant votre code source : P06_EQ##.*`

Vous avez tous déjà joué au billard (ou avez au moins une idée de la façon dont on joue : une table avec tapis vert, des boules, une queue et des trous !). La variante du billard dont il est question dans ce problème est le jeu du 8. Ce jeu est assez simple, le principe de base : on frappe sur la boule blanche à l'aide de la queue. La boule blanche doit alors toucher une boule numérotée du groupe du joueur, c'est-à-dire soit les rayées, soit les pleines, pour ensuite l'empocher dans un trou.

Le billard se compose de :

- une table de billard;
- 6 trous disposés en périphérie de la table;
- une boule blanche (numérotée 0 pour les besoins du problème);
- 7 boules pleines numérotées de 1 à 7;
- une boule noire numérotée 8;
- 7 boules rayées numérotées de 9 à 15;
- une queue de billard.

Problème

Le problème consiste tout simplement à envoyer une seule boule dans un des trous de la table de billard à l'aide de la boule blanche tirée par la queue de billard, à déterminer l'angle selon lequel la

boule blanche frappe la boule ciblée de telle sorte que cette dernière soit "empochée" et à déterminer l'angle selon lequel la boule ciblée s'empoche. Aucune autre boule que la boule ciblée ne doit être "empochée". Pour ce faire, il faut interpréter le fichier d'entrée afin de déterminer la situation du jeu. Pour déterminer la situation du jeu, il faut identifier la boule que l'on doit frapper avec la blanche, ceci s'effectue en suivant les instructions suivantes :

1. Déterminer l'ensemble de boules sur lequel on joue, c'est-à-dire, l'ensemble des pleines (1 à 7) ou l'ensemble des rayées (9 à 15).

2. Identifier le plus petit numéro de boule de l'ensemble. Certaines boules de l'ensemble peuvent déjà avoir été envoyées dans un des trous et elles ne font plus partie du jeu. Si l'ensemble est vide, alors la boule à jouer est la noire.

Dans un deuxième temps, il faut déterminer avec quel angle il faut frapper la boule avec la queue afin d'empocher la boule identifiée. L'angle est en degrés et se situe par rapport à l'axe des X. Les trous sont situés à $(0,0)$, $(n,0)$, $(0,m)$, (n,m), $(n/2,0)$ et $(n/2,m)$.

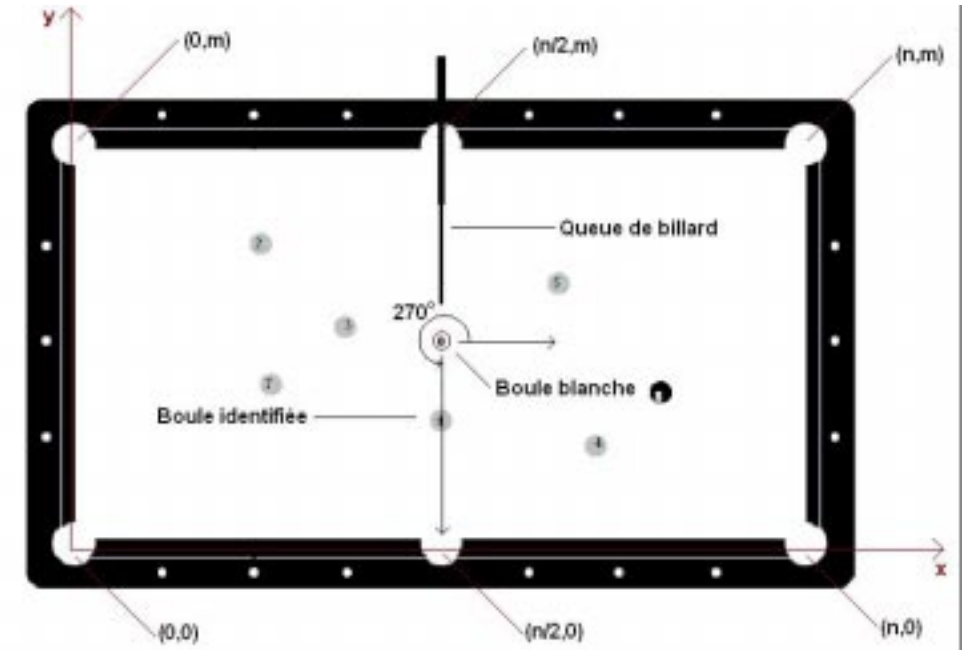

FIG. 6.1 – La table de billard

Comment déterminer l'angle avec lequel il faut frapper la boule blanche? Suivant les règles de conservation de la quantité de mouvement, nous sommes en mesure de déterminer l'angle avec lequel on peut frapper la boule blanche à l'aide de la queue. Dans ce problème, trois cas se présentent :

- Rebond sur la bande de la table de billard. Figure 6.2. Considérons une boule A entrant en collision avec la bande de la table de billard avec un angle θ et une vitesse Va. Après l'impact, la vitesse Va' de la boule est conservée en amplitude mais pas en direction. Toutefois, l'angle après l'impact démontre une symétrie parfaite par rapport à la ligne d'impact, $\theta = \theta'$.

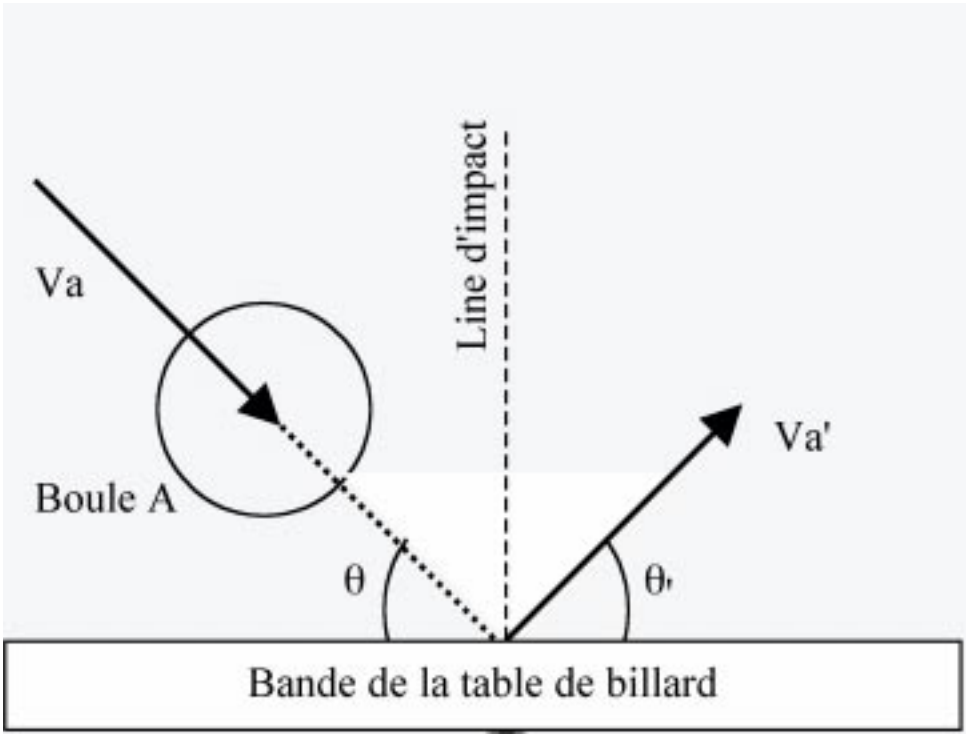

FIG. 6.2 – Rebond sur la bande de la table de billard

- Collision suivant la ligne d'impact. Figure 6.3. Considérons deux boules A et B entrant en collision l'une avec l'autre à une vitesse Va et Vb suivant la direction de la ligne d'impact. Dans ce problème-ci, la boule B a toujours une vitesse initiale nulle. Toujours en appliquant les règles de conservation de la quantité de mouvement, on constate que la vitesse finale Vb' de la boule B est égale (en amplitude et en direction) à la vitesse initiale de la boule A et que la vitesse finale Va' de la boule A est nulle. Autrement dit, $Vb' = Va$ et $Vb = Va' = 0$. En fait, la quantité de mouvement initiale de la boule A suivant l'axe n est transférée à la boule B après l'impact.

- Impact oblique par rapport à la ligne d'impact. Figure 6.4. Considérons deux boules A et B entrant en collision l'une avec l'autre à une vitesse Va et Vb suivant un angle θ par rapport à la ligne d'impact. Dans ce problème-ci, la boule B a toujours une vitesse initiale nulle. En appliquant les règles de conservation de la quantité de mouvement, on constate que la vitesse

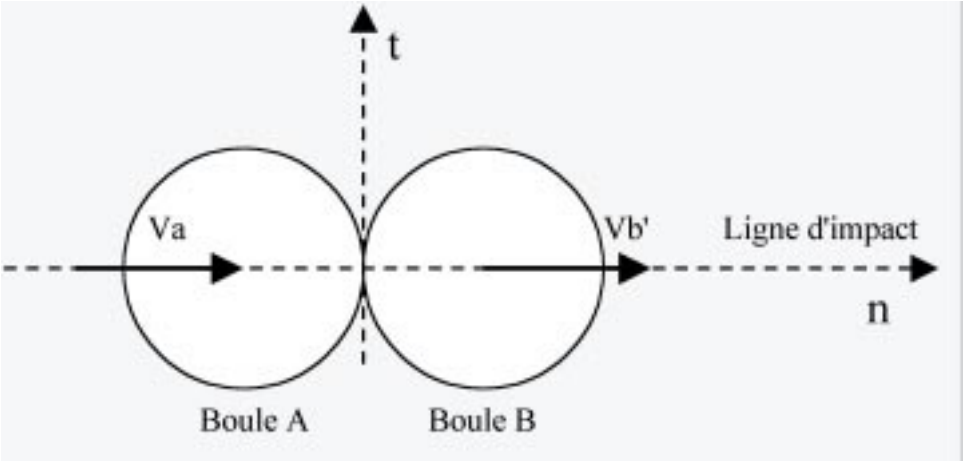

FIG. 6.3 – Collision suivant la ligne d'impact

finale des boules A et B, Va' et Vb' respectivement, sont fonction de l'angle d'impact. De plus, les deux vitesses se retrouvent perpendiculaires l'une par rapport à l'autre. Encore une fois, la quantité de mouvement initiale de la boule A suivant l'axe n est transférée à la boule B après l'impact. Autrement dit, $Vb = 0$, $Va' = VaSIN(\theta)$ en direction de l'axe t et $Vb' = VaCOS(\theta)$ en direction de l'axe n.

Pour arriver à déterminer le comportement des boules, plusieurs règles simplificatrices ont été prises en compte.

- Les bandes de la table de billard sont parfaitement élastiques;
- Les boules ont toutes un diamètre de 10 unités, n'oubliez pas de vérifier les collisions potentielles;
- Les boules restent en contact permanent avec la surface du tapis;
- Les dimensions maximales de la table sont (X_{max}, Y_{max}) = (1500,1000);
- Les collisions sont parfaitement élastiques;
- Le frottement est absent;
- Les boules ont toutes la même masse.

Au risque de se répéter, il y a plusieurs choses à faire attention. N'oubliez pas de considérer les points suivants :

- Ne pas envoyer aucune autre boule que la boule identifiée dans un des trous;

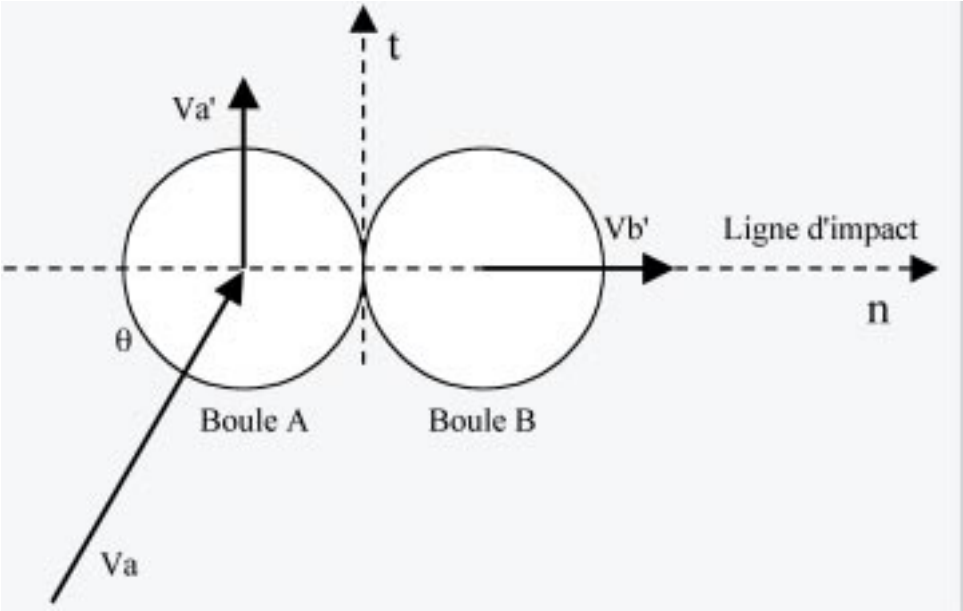

FIG. 6.4 – Impact oblique par rapport à la ligne d'impact

- Ne pas déplacer aucune autre boule que la boule identifiée et la boule blanche;
- Ne pas envoyer dans un trou la blanche après le rebond sur la boule identifiée. Seulement les deux premiers rebonds de la blanche sont considérés après l'impact avec la boule identifiée;
- Il est garanti qu'il existe une solution permettant d'empocher la boule identifiée au plus avec un rebond. Autrement dit, la boule identifiée peut s'empocher directement ou par une seule bande. Toutefois, si vous soumettez une solution qui implique plus qu'un rebond, vous serez pénalisé en perdant les points associés au jeu de test correspondant.
- Il est garantie qu'il existe une solution permettant la boule blanche d'atteindre la boule identifiée au plus avec un rebond. Autrement dit, la boule blanche peut atteindre directement ou par une seule bande la boule identifiée. Toutefois, si vous soumettez une solution qui implique plus qu'un rebond, vous serez pénalisé en perdant les points associés au jeu de test correspondant.

Comme le problème demande que l'on donne un angle qui soit par rapport à l'axe des x, il faut convertir les résultats obtenus, afin de respecter les spécifications du fichier de sortie.

Fichier d'entrée

Le fichier que vous devez lire est "`P06.ENT`". Sur la première ligne, on retrouve la dimension de la table de jeux. Ensuite, on retrouve sur la deuxième ligne soit `RAYEES` ou `PLEINES` qui indique sur

quel ensemble de boules le coup doit porter. Sur les lignes subséquentes, on retrouve un triplet de la forme n, x, y où n représente le numéro de la boule, x la position en abscisse sur le tapis de la table et y la position en ordonnée sur la table de billard.

▷

```
400 300
PLEINES
0 200 100
2 100 200
3 200 5
4 395 5
5 395 295
6 200 295
8 300 250
12 5 5
```

◁

Fichier de sortie

Vous devez écrire le fichier "`P06.SOR`". Il doit contenir sur la même ligne la boule visée, l'angle en degrés par rapport à l'abscisse avec lequel on doit frapper la blanche afin de toucher la boule visé et l'angle en degrés par rapport à l'abscisse avec lequel la boule visée s'enfonce dans un des trous.

▷

```
2 135.00 135.00
```

◁

Il est nécessaire de travailler avec des `double`s afin de conserver la plus grande précision. À la fin, pour le fichier de sortie, il faut afficher l'angle avec deux chiffres après le point.

Le problème peut avoir plusieurs solutions, le test vérifie que votre solution est acceptable.

Le pointage

Le problème sera corrigé en soumettant votre programme à quatre fichiers d'entrée différents. Un fichier de test sera considéré comme réussi si les résultats sont conformes aux spécifications données.

Nombre de fichiers réussis	Points accordés
1	120
2	180
3	240
4	300

Temps maximal d'exécution de votre programme : 20 secondes.

Spécification 7

Recherche de contours emboîtés

Martine Bellaïche, Jean-Yves Hervé

`Fichier à produire contenant votre code source : P07_EQ##.*`

Un problème classique en imagerie médicale est la détection des éléments cellulaires. Certains éléments peuvent être emboîtés (par exemple, une cellule peut contenir un noyau).

Problème

Le problème est de détecter, dans une image bitmap en format binaire, tous les contours qui la composent. Les contours sont des polygones fermés à angles droits et un contour peut en contenir d'autres. Les régions entre les différents contours sont d'épaisseur non nulle. Par conséquent, les contours ne se touchent pas. Chaque contour est identifié par son degré d'emboîtement. Pour déterminer le degré d'emboîtement d'un contour, on a les règles suivantes:

- Un contour qui n'est pas contenu dans aucun autre contour est de degré 1.
- Un contour directement contenu dans un contour de degré n, est de degré $n + 1$.

Par exemple selon la figure 7.1, C_1 et C_2 sont des contours de degré d'emboîtement 1; C_{11}, C_{12}, C_{13}, C_{21} et C_{22}, des contours de degré d'emboîtement 2; C_{131} et C_{132}, des contours de degré d'emboîtement 3; C_{1311} un contour de degré d'emboîtement 4.

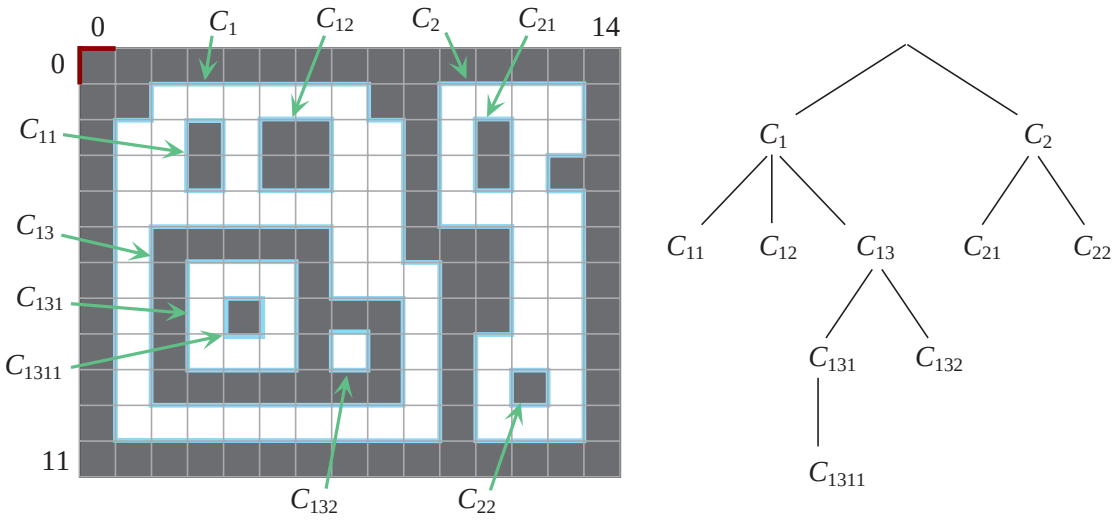

FIG. 7.1 – Exemple d'image bitmap avec quatre degrés de contours emboîtés.

Fichier d'entrée

Le fichier que vous devez lire est "`P07.ENT`". Sur la première ligne de ce fichier, on trouve le nombre de lignes et le nombre de colonnes de l'image bitmap. Le reste du fichier d'entrée contient les lignes de l'image bitmap composée de 0 et de 1 et en format texte. Le nombre maximal de lignes et de colonnes est 255.

Voici un exemple de fichier d'entrée d'après l'exemple du problème.

▷

```
12 15
000000000000000
001111110011110
011010011010110
011010011010100
011111111011110
010000011000110
010111011100110
010101000100110
010111010101110
010000000101010
011111111101110
000000000000000
```

◁

Fichier de sortie

Le fichier de sortie "`P07.SOR`" devra contenir :

- Sur la première ligne, le degré maximal d'emboîtement pour l'image;
- Sur la deuxième ligne, le nombre de contours de degré 1;
- Sur la troisième ligne, le nombre de contours de degré 2;
- Sur la quatrième ligne, le nombre de contours de degré 3;
- ...
- Sur la $n+1$-ième ligne, le nombre de contours de degré n.

Voici un exemple de fichier de sortie correspondant à la solution du fichier d'entrée ci-haut :

```
▷
4
2
5
2
1
◁
```

Le pointage

Le problème sera corrigé en soumettant votre programme à quatre fichiers d'entrée différents. Un fichier de test sera considéré comme réussi si les résultats sont conformes aux spécifications données.

Nombre de fichiers réussis	Points accordés
1	160
2	240
3	320
4	400

Temps maximal d'exécution de votre programme : 20 secondes.

Spécification 8

Spirale d'Ulam

Martine Bellaïche, Jean-Yves Hervé

`Fichier à produire contenant votre code source : P08_EQ##.*`

Un des problèmes les plus classiques des mathématiques est celui de la détermination du n-ième nombre premier sans calcul préalable des $n-1$ précédents. Le grand mathématicien américain d'origine polonaise Stanislaw Marcin Ulam a proposé une représentation graphique par laquelle on peut chercher à identifier des régularités dans la séquence des nombres premiers : la spirale d'Ulam.

Problème

Nous allons considérer ici une version générale de la spirale proposée par Ulam. Notre spirale sera définie par :

- un nombre de départ : le **germe** de la spirale;
- un nombre entier ≥ 1 : le **pas** initial;
- une **direction** de départ (Nord, Sud, Est, Ouest);
- un **sens** de rotation (le sens contraire aux aiguilles d'une montre ou le sens des aiguilles d'une montre).

On construit la spirale sur une grille régulière infinie en "enroulant" la séquence des nombres entiers autour du germe de la manière suivante :

- Le premier élément de la séquence est le **germe**, qui est placé à une position arbitraire sur la grille.

- Un nouvel élément de la séquence est obtenu en incrémentant de 1 son prédécesseur.
- On range les premiers éléments de la séquence en se déplaçant à chaque fois d'une case sur la grille dans la **direction** de départ, jusqu'à ce que la longueur du **pas** initial ait été complétée.
- À partir de cette position, on "enroule" la spirale dans le **sens** de rotation spécifié.

Dans l'exemple ci-dessous (figure 8.1), le germe de la spirale est 5, le pas initial est 3, la direction initiale est Est, et le sens de rotation est le sens des aiguilles d'une montre. On voit donc qu'à partir du germe 5, on entre 3 valeurs (6, 7, 8) en se déplaçant vers l'Est avant de commencer à enrouler la spirale dans le sens des aiguilles d'une montre.

93	94	95	96	97	98	99	100	101	102	103	104
92	59	60	61	62	63	64	65	66	67	68	105
91	58	33	34	35	36	37	38	39	40	69	106
90	57	32	15	16	**17**	18	19	20	**41**	70	107
89	56	31	14	**5**	6	7	8	21	42	**71**	108
88	55	30	**13**	12	11	10	9	22	43	72	**109**
87	54	29	28	27	26	25	24	23	44	73	110
86	53	52	51	50	49	48	47	46	45	74	111
85	84	83	82	81	80	79	78	77	76	75	112
				120	119	118	117	116	115	114	113

FIG. 8.1 – Enroulement de la spirale.

Sur la spirale, on marque les cases de la grille correspondant à des nombres premiers (on rappelle que 0 et 1 **ne sont pas** premiers). Sur la spirale de la figure 8.1, nous avons ainsi marqué les cases des nombres 5, 7, 11, 13, 17, 19, etc.

En observant les cases marquées sur la grille par la spirale que l'on enroule, on voit apparaître des alignements de nombres premiers selon les directions diagonales NO-SE et SO-NE. Il vous est demandé de détecter les premiers alignements diagonaux (NO-SE et SO-NE) de longueur déterminée

(2, 3, 4, etc.). Sur la spirale de la figure 8.1, nous avons ainsi marqué les deux premières diagonales de longueur 3 observées. La première, dans la direction SO-NE, passe par les cases 5, 13 et 17. La diagonale dans la direction NO-SE passe par les cases 41, 71, 109. Si on avait demandé des disgonales de longueur 6, on aurait obtenu 5, 13, 17, 29, 37 et 53 pour la diagonale SO-NE et aucune diagonale NO-SE car la séquence est arrêtée à 120.

Fichier d'entrée

Le fichier que vous devez lire est "`P08.ENT`", il est en format texte.

- La première ligne de ce fichier donne la valeur du germe (un nombre entier compris entre 0 et 65535);
- La deuxième ligne donne la longueur du pas (un nombre entier compris entre 1 et 65535);
- La troisième ligne donne la direction de départ (un caractère : `N`, `S`, `E` ou `O`);
- La quatrième ligne donne le sens de l'orientation (1 pour le sens contraire aux aiguilles d'une montre, -1 pour le sens des aiguilles d'une montre);
- La cinquième ligne donne la longueur des alignements diagonaux de nombres premiers recherchés;
- La sixième ligne donne le point d'arrêt de la spirale (un nombre inférieur ou égal à 65535).

Pour rechercher les diagonales de longueur 5 sur la spirale de la figure 8.1, on donnerait donc le fichier d'entrée suivant :

```
▷
5
3
E
-1
5
120
◁
```

Fichier de sortie

Le fichier de sortie "`P08.SOR`" doit contenir :

- sur la première ligne, tous les nombres premiers, classés dans l'ordre croissant, comprenant la première diagonale NO-SE de la longueur désirée si une telle diagonale existe, et 0 sinon.

- sur la deuxième ligne, tous les nombres premiers, classés dans l'ordre croissant, comprenant la première diagonale SO-NE de la longueur désirée si une telle diagonale existe, et 0 sinon.

Le fichier de sortie correspondant à l'entrée donnée plus haut serait donc le suivant :

```
▷
0
5 13 17 29 37
◁
```

Le pointage

Le problème sera corrigé en soumettant votre programme à quatre fichiers d'entrée différents. Un fichier de test sera considéré comme réussi, si les résultats sont conformes aux spécifications données.

Nombre de fichiers réussis	Points accordés
1	160
2	240
3	320
4	400

Temps maximal d'exécution de votre programme : 20 secondes.

Spécification 9

Mauvais numéro de téléphone...

Alex Boisvert

`Fichier à produire contenant votre code source : P09_EQ##.*`

Un test psychologique est présenté à plusieurs personnes d'un groupe cible. Un même numéro de téléphone leur est donné pour qu'ils le mémorisent. Trois semaines plus tard, les personnes sont réunies toutes ensembles pour vérifier qui se souvient du numéro original.

Les statistiques démontrent que très souvent (disons environ 88.2%), les répondants oublient SOIT un ou plusieurs chiffres, SOIT qu'ils changent l'ordre des chiffres par rapport au numéro original. Évidemment, les répondants pensent tous qu'ils ont le bon numéro...

Par exemple,

Numéro original	5147389120
Répondant 1	1478912
Répondant 2	51120
Répondant 3	0358719412
Répondant 4	5173920
Répondant 5	73890
Répondant 6	0219387515
Répondant 7	51789120
Répondant 8	5147389120
Répondant 9	1812

On voit plus haut que seul le répondant 8 s'est souvenu correctement du numéro original. Les répondants 3 et 6 ont mélangé la position de certains chiffres alors que les répondants 1,2,4,5,7 et 9 ont oublié certains chiffres.

Problème

En tant qu'expert en psychologie (et peut-être en informatique ?), il vous faut trouver un moyen de retrouver un numéro de téléphone, sous sa forme originale, à partir d'un groupe de répondants. Dit autrement, il vous faut identifier tous les chiffres apparaissant dans le numéro original ainsi que leur ordre original d'apparition.

Quelques notes:

- Les numéros à mémoriser ont toujours 10 chiffres (aucun autre caractère n'apparaît dans le numéro de téléphone).
- Il peut y avoir entre 1 et 25 répondants.
- Il n'est pas garanti que le groupe-cible de répondants que vous allez rencontrer pourra bel et bien vous aider à retrouver le numéro original.
- Les répondants se rapellent toujours d'au moins un chiffre.

Un exemple où il serait impossible de reconstituer fidèlement le numéro d'origine est celui-ci:

Numéro original	8198217000
Répondant 1	819
Répondant 2	821
Répondant 3	7000
Répondant 4	0007128918
Répondant 5	8217000

Dans ce cas, selon les répondants, il est impossible d'établir si la partie "819" précède la partie "8217000".

Fichier d'entrée

Le fichier que vous devez lire est "`P09.ENT`". La première ligne donne le nombre de répondants. Les lignes subséquentes fournissent le numéro retenu pour chacun des répondants. Voici un exemple de fichier d'entrée :

```
▷
7
8917256633
81963632
6363
81966327
1953637
13
8956627
◁
```

Fichier de sortie

Vous devez écrire le fichier "`P09.SOR`" qui contiendra sur une seule ligne le numéro de téléphone original. S'il est impossible de recomposer le numéro original à partir des réponses du groupe-cible, vous devez écrire sur une seule ligne la chaîne `impossible` textuellement.

▷

```
8195636327
```

◁

Le pointage

Le problème sera corrigé en soumettant votre programme à quatre fichiers d'entrée différents. Un fichier de test sera considéré comme réussi si vous réussissez à trouver le numéro original ou si vous indiquez qu'il est impossible de résoudre le problème avec le groupe-cible donné et que c'est effectivement le cas.

Si vous indiquez qu'un numéro est impossible à retrouver, mais qu'il ne l'est pas réellement, vous allez perdre tous les points associés au problème.

Nombre de fichiers réussis	Points accordés
1	320
2	480
3	640
4	800

Temps maximal d'exécution de votre programme : 20 secondes.

Spécification 10

Reconnaissance des formes

Martine Bellaïche, Jean-Yves Hervé

`Fichier à produire contenant votre code source : P10_EQ##.*`

Il arrive fréquemment qu'il faille reconnaître des éléments qui se trouvent dans une image bitmap. Si la forme des éléments est connue, il s'agit d'identifier ceux-ci dans l'image en tenant compte des transformations que ces éléments ont pu subir dans cette image.

Problème

Votre problème est de reconnaître toutes les formes se trouvant dans une image couleur (RGB : Red, Green, Blue). Au départ, on possède une liste de formes de base où chaque forme a un patron de 11 pixels par 11 pixels dont l'origine est le point en haut à gauche. Les patrons de base sont composés de 0 et 1. Par exemple, le patron de la forme `B` est :

```
00111110000
00100001000
00100000100
00100000100
00100001000
00111110000
00100001000
00100000100
00100000100
00100001000
00111110000
```

Dans l'image, les valeurs 1 du patron de base de la forme sont remplacées par la valeur de la couleur codée sur un octet de la façon suivante :

- Les différentes possibilités de valeur de la couleur rouge en base 2 sont $(00110000)_2$, ou $(00100000)_2$, ou $(00010000)_2$; en base 10 ces valeurs sont 48, ou 32, ou 16;
- Les différentes possibilités de valeur de la couleur verte en base 2 sont $(00001100)_2$, ou $(00001000)_2$, ou $(00000100)_2$; en base 10 ces valeurs sont 12, ou 8, ou 4;
- Les différentes possibilités de valeur de la couleur bleu en base 2 sont $(00000011)_2$, ou $(00000010)_2$, ou $(00000001)_2$; en base 10 ces valeurs sont 3, ou 2, ou 1;

Les formes dans l'image sont de couleur pure c'est-à-dire, soit toute rouge, soit toute verte, ou soit toute bleue. Par exemple, si le patron `B` a la couleur rouge $(00010000)_2 = 16_{10}$ alors toutes les valeurs 1 du patron de base seront remplacées par $16_{10} = (00010000)_2$ dans l'image.

De plus, on ajoute la condition que les formes dans l'image peuvent se superposer. Par exemple, si dans l'image un point d'une forme a la couleur rouge $(00110000)_2 = 48_{10}$ et un point d'une autre forme a la couleur verte $(00001000)_2 = 8_{10}$ et que ces 2 points sont superposés, alors ce point sera représenté dans l'image par la couleur $56_{10} = (00111000)_2$. Mais, 2 formes de couleur rouge (respectivement verte ou bleue) ne peuvent jamais se superposer.

On suppose que les formes de base sont complètes dans l'image.

D'autre part, pour compliquer la recherche des formes dans l'image, celles-ci auront subi des transformations. Les transformations possibles sont :

- un agrandissement par 2 ou par 4;
- une rotation dans le sens des aiguilles d'une montre de 90 degrés, ou de 180 degrés ou de 270 degrés;
- une translation: déplacement vertical et horizontal de l'origine de la forme par rapport à l'origine de l'image (le point le plus haut et à gauche).

D'après le patron de base de la forme `B` :

```
00111110000
00100001000
00100000100
00100000100
00100001000
00111110000
00100001000
00100000100
00100000100
00100001000
00111110000
```

Voici l'exemple d'agrandissement par 2 du patron `B` :

```
00001111111111100000000
00001111111111100000000
00001100000000011000000
00001100000000011000000
00001100000000000110000
00001100000000000110000
00001100000000000110000
00001100000000000110000
00001100000000011000000
00001100000000011000000
00001111111111100000000
00001111111111100000000
00001100000000011000000
00001100000000011000000
00001100000000000110000
00001100000000000110000
00001100000000000110000
00001100000000000110000
00001100000000011000000
00001100000000011000000
00001111111111100000000
00001111111111100000000
```

Voici l'exemple de rotation de 90 degrés du patron `B` :

```
00000000000
00000000000
11111111111
10000100001
10000100001
10000100001
10000100001
01001010010
00110001100
00000000000
00000000000
```

Voici l'exemple de rotation de 180 degrés du patron `B` :

```
00001111100
00010000100
00100000100
00100000100
00010000100
```

```
00001111100
00010000100
00100000100
00100000100
00010000100
00001111100
```

Voici l'exemple de rotation de 270 degrés du patron `B` :

```
00000000000
00000000000
00110001100
01001010010
10000100001
10000100001
10000100001
10000100001
11111111111
00000000000
00000000000
```

Les patrons de base donnés dans le fichier respectent les règles suivantes.

Si on pose que l'ensemble des formes possibles contient les patrons de base des formes et les patrons de base transformés alors :

1. Toutes les formes de cet ensemble sont distinctes.
2. Une forme possible ne peut pas être une partie d'une autre forme.

Fichier d'entrée

Le fichier que vous devez lire est "`P10.ENT`". Sur la première ligne, on trouve le nombre de formes de base, ainsi que le nombre de lignes et de colonnes de l'image bitmap. Ensuite, sur les autres lignes, on donne les patrons des formes de base et, par la suite, l'image bitmap. Dans l'exemple il y en a trois formes de base, elles sont `B`, `?` et `4`. Dans l'image bitmap, la couleur d'un point est indiquée en base 10. Les points sont séparés par un espace. Voici un exemple de fichier d'entrée :

```
▷
3 34 34
0 0 1 1 1 1 1 0 0 0 0
0 0 1 0 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 1 0 0 0
0 0 1 1 1 1 1 0 0 0 0
```

```
0 0 1 0 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 1 0 0 0
0 0 1 1 1 1 1 0 0 0 0
0 0 0 0 0 1 1 0 0 0 0
0 0 0 1 0 0 0 0 1 0 0
0 0 1 0 0 0 0 1 0 0 0
0 0 1 0 0 0 0 1 0 0 0
0 0 0 0 0 1 1 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 1 1 0 0 0 0
0 0 0 0 1 0 1 0 0 0 0
0 0 0 1 0 0 1 0 0 0 0
0 0 1 0 0 0 1 0 0 0 0
0 1 1 1 1 1 1 1 1 0 0
0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
48 48 48 48 48 48 48 48 48 48 48 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
48 0 0 0 0 48 0 0 0 0 48 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
48 0 0 0 0 48 0 0 0 0 48 0 3 3 3 3 3 3 3 3 3 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
48 0 0 0 0 48 0 0 0 0 48 0 3 3 3 3 3 3 3 3 3 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
48 0 0 0 0 48 0 0 0 0 51 3 0 0 0 0 0 0 0 0 3 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 48 0 0 48 0 48 0 0 48 3 3 0 0 0 0 0 0 0 0 3 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 48 48 0 0 0 48 51 3 0 0 0 0 0 0 0 0 0 0 3 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 3 3 0 0 0 0 0 0 0 0 0 0 3 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 3 3 0 0 0 0 0 0 0 0 0 0 3 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 3 3 0 0 0 0 0 0 0 0 0 0 3 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 3 3 0 0 0 0 0 0 0 0 4 4 3 3 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 3 3 0 0 0 0 0 0 0 0 4 4 3 3 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 3 3 7 7 3 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 3 3 7 7 3 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 3 3 0 0 0 0 0 0 0 0 4 4 3 3 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 3 3 0 0 0 0 0 0 0 0 4 4 3 3 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 3 3 0 0 0 0 0 0 0 0 0 0 4 4 3 3 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0 3 3 0 0 0 0 0 0 0 0 4 4 3 3 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 32 32 0 3 3 0 0 0 0 4 4 4 4 4 4 4 7 7 4 4 4 4 4 4 4 4 4 0 0 0 0
0 0 0 32 0 0 0 0 35 3 0 0 0 0 4 4 4 4 4 4 4 7 7 4 4 4 4 4 4 4 4 4 0 0 0 0
0 0 32 0 0 0 0 32 0 0 3 3 0 0 0 0 0 0 4 4 3 3 0 0 0 0 4 4 0 0 0 0 0 0
0 0 32 0 0 0 0 32 0 0 3 3 0 0 0 0 0 0 4 4 3 3 0 0 0 0 4 4 0 0 0 0 0 0
0 0 0 0 0 32 32 0 0 0 0 0 0 3 3 3 3 3 3 7 7 3 3 0 0 4 4 0 0 0 0 0 0 0 0
0 0 0 0 0 32 0 0 0 0 0 0 0 3 3 3 3 3 3 7 7 3 3 0 0 4 4 0 0 0 0 0 0 0 0
0 0 0 0 0 32 0 0 0 0 0 0 0 0 0 0 0 0 0 4 4 0 0 4 4 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 32 0 0 0 0 0 0 0 0 0 0 0 0 0 4 4 0 0 4 4 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 4 4 4 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 32 0 0 0 0 0 0 0 0 0 0 0 0 0 4 4 4 4 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

◁

Il est garanti que le fichier contient au moins une forme et au moins une transformation. Au maximum, il y aura dix formes de base; le nombre maximum de formes trouvées dans l'image est de 20 et la taille maximum du bitmap est de 256×256 points.

Fichier de sortie

Vous devez écrire le fichier "`P10.SOR`"qui contiendra, pour chaque forme trouvée dans l'image, le numéro de la forme de base, les quatre lettres `T`, `R`, `A` et `C` (obligatoire) avec leurs arguments :

- sur une ligne du fichier le numéro de la forme de base (la première forme de base porte le numéro 1.
- sur une ligne du fichier, la lettre `T` pour la translation, le numéro de la ligne et le numéro de la colonne de la translation, sachant que le point en haut à gauche de l'image, est à la ligne `0` et à la colonne `0`;
- sur la ligne suivante du fichier, la lettre `R` pour la rotation, ensuite le degré de rotation;
- sur la ligne suivante du fichier, la lettre `A` pour l'agrandissement, ensuite la valeur `2` ou `4`;
- sur la ligne suivante du fichier, la lettre `C` suivie de la valeur de la couleur, en décimale.

Et, si par hasard, la forme trouvée dans l'image n'a subi aucune translation, alors la ligne du fichier de sortie sur la translation aura le format suivant :

```
T 0 0
```

Si la forme trouvée n'a aucune rotation, alors la ligne du fichier sur la rotation sera :

```
R 0
```

Et si la forme trouvée n'a aucun agrandissement, alors la ligne du fichier sur l'agrandissement sera :

```
A 1
```

Voici le fichier de sortie d'après l'exemple du fichier d'entrée :

```
▷
1
T 0 0
R 90
A 1
C 48
1
T 4 4
R 180
A 2
C 3
3
T 10 10
R 180
A 2
C 4
2
T 20 0
R 0
A 1
C 32


◁
```

Le pointage

Le problème sera corrigé en soumettant votre programme à quatre fichiers d'entrée différents. Un fichier de test sera considéré comme réussi, si les résultats sont conformes aux spécifications données.

Nombre de fichiers réussis	Points accordés
1	320
2	480
3	640
4	800

Temps maximal d'exécution de votre programme : 20 secondes.