UNIVERSITÉ DE
SHERBROOKE

Faculté des sciences appliquées

Département de génie électrique et de génie informatique

QUESTIONNAIRE DE LA FOLLE COURSE INFORMATIQUE

Novembre 1997

Introduction

Lisez ce préambule au questionnaire de la troisième 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 les spécifications. 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 conclu 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.

La première équipe qui soumet un programme fonctionnel pour un problème donné, se verra accorder un **bonus de 25 %** sur les points du problème en question. Un programme est fonctionnel s'il s'exécute correctement sur au moins un fichier test.

En huit heures, il est très peu probable 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 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

[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++.

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 en premier 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 fichers 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 1 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 de vérifications sur ces données.

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 pour 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 test et des correcteurs de programmes, en mettant beaucoup d'efforts et du temps. Même pendant la course nous faisons des vérifications afin d'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 satifaction 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.

Table des matières

Les Spécifications

Spécification 1

La stéganographie

`Fichier à produire contenant votre code source : P01_EQ##.*`
La stéganographie consiste à dissimuler de l'information que l'on désire transmettre à quelqu'un dans un flot de données, qui est en général diffusé au grand public. Par exemple, un message secret peut être caché dans les bits les moins significatifs d'un extrait sonore numérique. En l'écoutant, l'oreille humaine ne remarquerait pratiquement pas le léger bruit de fond introduit par certains bits du message, mais une personne connaissant son existence peut facilement l'extraire, à condition d'avoir le programme pour le faire.

Problème

Votre mission est de concevoir un programme qui servira à extraire un message caché dans un flot de bits, formé par le bit le moins significatif de chaque octet d'un fichier audio monophonique de 8 bits par échantillon. L'encodage a été effectué comme suit : pour chaque octet du fichier audio, le bit le moins significatif a été remplacé par un bit provenant du message secret. Le bit le plus significatif de chaque octet du message secret est placé en premier, suivi du deuxième bit le plus significatif, et ainsi de suite jusqu'au bit le moins significatif. De cette manière, un octet du message secret se retrouve dissimulé dans huit octets consécutifs du fichier audio. Chaque octet du message secret représente un caractère ASCII ÉtenduPC (0 à 255 en décimal). Le fichier audio est encodé dans un fichier texte, un octet du fichier audio est représenté par deux caractères représentant un nombre hexadécimal (les caractères `A` à `F` sont en majuscules). Par exemple `FE` sont deux caractères représentant 254 en décimal ou `11111110` en binaire. La figure 1.1 montre comment dans huit bits du flot encodé le caractère `A` est caché.

Fichier d'entrée

Le fichier que vous devez lire est "`P01.ENT`". La première ligne indique en décimal le nombre d'échantillons que vous devrez lire. Le minimum est 0 et le maximum d'échantillons est 10^4, il est toujours un multiple de 8. Les lignes suivantes contiennent chacune 16 échantillons (nombres) chacun étant représenté par deux chiffres hexadécimaux; entre deux échantillons il y a au moins un espace. Voilà un exemple de fichier :

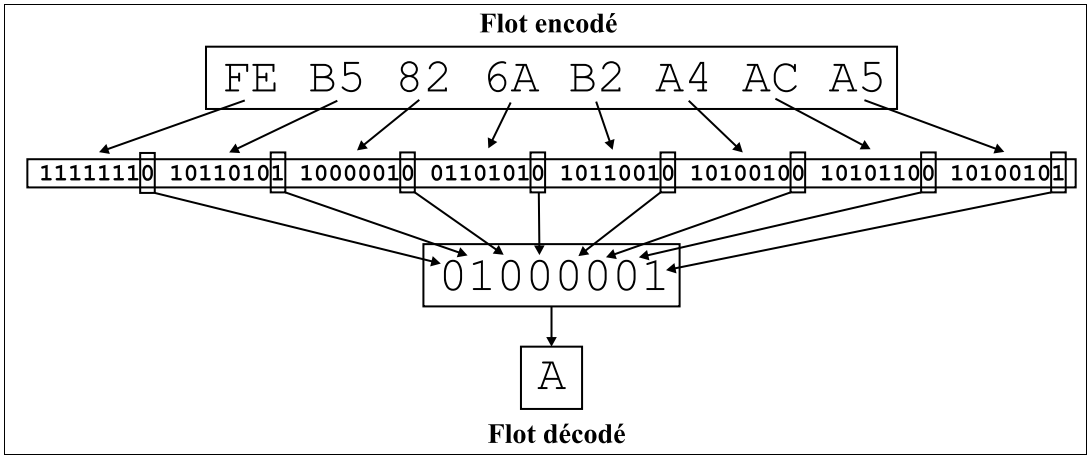


FIG. 1.1 – La stéganographie

```
32
FE B5 82 6A B2 A4 AC A5 AC AD 2A B4 6B 6B B4 B2
B4 6B B4 B2 B5 6B 6A B4 B4 B3 B4 B4 6B B3 6B B3
```

Dans le cas où il y a zéro ligne dans le fichier d'entrée, celui de sortie doit être vide.

Fichier de sortie

Vous devez écrire le fichier "`P01.SOR`" qui contiendra le message secret. Voilà celui qui correspond au fichier audio montré comme exemple :

```
ALLO
```

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	40
2	60
3	80
4	100

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

Spécification 2

Une table des matières HTML

`Fichier à produire contenant votre code source : P02_EQ##.*`
Quoi dire sur Internet, le Web, HTML et autres membres de la famille lorsque tout a déjà été dit. En bref, ce problème nous présente un petit défi de programmation très appliqué dans ce domaine.

Problème

Une compagnie de publication sur le Web vous demande d'automatiser la construction d'une page de table de matières d'un fichier HTML. Ce qu'elle désire, c'est de retrouver les textes marqués comme en-têtes dans leur fichier dans un autre fichier, toujours en format HTML. Les en-têtes en HTML ont la forme d'un texte contenu entre deux étiquettes. Par exemple :

```
<hn>Texte</hn>
```

où n varie entre 1 et 6 selon la grosseur de l'en-tête, la valeur 1 étant la plus grosse. *Texte* peut contenir n'importe quelle chaîne de caractères, y compris d'autres étiquettes HTML, à l'exception des étiquettes d'en-têtes.

La compagnie désire que l'ensemble des en-têtes soit formaté dans un nouveau fichier avec une en-tête par ligne et en effectuant l'élimination des blancs ou trimage.

Ce que l'on vous demande est de lire un ficher HTML, de retenir toutes les en-têtes et de faire compresser des caractères blancs.

Les caractères blancs sont les espaces, les tabs, les «retour chariot» et les «aller à la ligne».

La compression des caractères blancs s'effectue selon la règle suivante : lorsque plusieurs caractères blancs sont contigus, ils doivent être remplacés par un seul caractère espace. Exception à cette règle, le cas des caractères blancs qui figurent en préambule ou en postambule de *Texte*, ils sont tous éliminés.

Voici un exemple d'une partie de fichier HTML.

```
<h3>Introduction</h3>
    <h5> Remerciements</h5>
<h1>
   Titre             1
 </h1>
 <p> Patati


 patata
 </p>
        <h2>Sous-Titre  1.1</h2>
<h2>Sous-Titre          1.2</h2>
     <h1>Titre

     2</h1>
<h3>Section
         2.0.1</h3>
<h2>Sous-Titre2.1</h2>
```

Le résultat attendu est le suivant :

```
<h3>Introduction</h3>
<h5>Remerciements</h5>
<h1>Titre 1</h1>
<h2>Sous-Titre 1.1</h2>
<h2>Sous-Titre 1.2</h2>
<h1>Titre 2</h1>
<h3>Section 2.0.1</h3>
<h2>Sous-Titre2.1</h2>
```

Nous avons gardé juste les en-têtes. Par exemple, `<h1>Titre 1</h1>` car il comporte les étiquettes `<h1>` et `</h1>`. Les caractères blancs avant `Titre` ont été éliminés, les caractères blancs entre `Titre` et `1` ont été compressés et les caractères blancs après le `1` ont été éliminés.

Bien entendu il faut que les en-têtes soient dans le même ordre que dans le fichier original. Il faut aussi avoir une seule en-tête par ligne dans le fichier de sortie.

Votre fichier contiendra d'autres lignes HTML nécessaires pour afficher la nouvelle page sur un browser. Vous devez respecter obligatoirement ce format et mettre à la place de la ligne `[ Votre production rentrera ici... ]` les en-têtes telles que spécifiées précédemment.

```
<html>
<head>
<title>Index</title>
</head>
<body>
<h1>Table des Mati&egrave;res</h1>
[ Votre production rentrera ici... ]
</body>
</html>
```

Finalement, la compagnie vous garantit qu'aucun fichier n'aura de ligne plus longue que 100 caractères et que les titres, après trimage des espaces blancs, ne dépasseront pas 80 caractères. De plus, il est garanti que les seules étiquettes contenues dans le fichier sont : `<html>`, `</html>`, `<body>`, `<head>`, `</head>`, `</body>`, `<title>`, `</title>`, `<emph>`, `</emph>`, `<p>`, `</p>` et celles des en-têtes. Si le fichier ne contient pas d'en-têtes, vous devez insérer une ligne blanche à la place de la ligne `[ Votre production rentrera ici... ]`

Fichier d'entrée

Voici un exemple complet d'un fichier d'entrée "`P2.ENT`" :

```
<html>
Bla, bla, bla... information non importante
<body>
Possible bla, bla, bla]
  <h1>Introduction</h1>
Encore du bla bla bla
Bla, bla?
<h4>Tatati et Tatata</h4>
Par ci par la, du bla bla...
<h4>Toto et    Tati</h4>
<h1>Nouvelles personnalit&eacute;s
    <emph>c&eacute;l&egrave;bres!</emph></h1>
<h4>Bozo    le clown</h4>
<h4>    Jean Ouellette-Ren&eacute;-Bouchard</h4>
<h3>Personnalit&eacute; du monde animal</h3>
<h4>Bavar le   crocodile</h4>
La description de Bavar
<h4>Piqu&eacute; des vers,
le foie de veau</h4>
Conclusion... Pour avoir des points dans son travail
de fran&ccedil;ais.
</body>
</html>
```

Fichier de sortie

Et de son fichier de sortie "`P2.SOR`" correspondant :

```
<html>
<head>
<title>Index</title>
</head>
<body>
<h1>Table des Mati&egrave;res</h1>
<h1>Introduction</h1>
<h4>Tatati et Tatata</h4>
<h4>Toto et Tati</h4>
<h1>Nouvelles personnalit&eacute;s <emph>c&eacute;l&egrave;bres!</emph></h1>
<h4>Bozo le clown</h4>
<h4>Jean Ouellette-Ren&eacute;-Bouchard</h4>
<h3>Personnalit&eacute; du monde animal</h3>
<h4>Bavar le crocodile</h4>
<h4>Piqu&eacute; des vers, le foie de veau</h4>
</body>
</html>
```

Bonne mise en page!

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	40
2	60
3	80
4	100

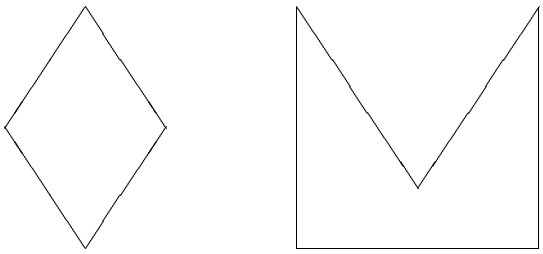
Temps maximal d'exécution de votre programme : 20 secondes.

Spécification 3

Le contour convexe

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

La géométrie comme nous la connaissons nous vient des anciens grecs. Beaucoup de problèmes du graphisme par ordinateur font appel à elle pour être résolus. Ici, nous vous proposons de faire la différence entre polygones convexes et concaves. Dans un polygone convexe tous les angles intérieurs sont inférieurs à 180^o. Dans un polygone concave au moins un des angles intérieurs est supérieur à 180^o. Voir figure 3.1



Polygone convexe Polygone concave

FIG. 3.1 – Polygone convexe et polygone concave

Pour ce problème nous allons utiliser un plan cartésien classique (Figure 3.2) et des coordonnées (x, y).

Problème

Soit une liste de n points ($3 \leq n \leq 10^3$) de coordonnées (x, y). Vous devez trouver un polygone convexe ayant pour sommets des points contenus dans la liste, la condition est qu'aucun point ne se trouve à l'extérieur du polygone.

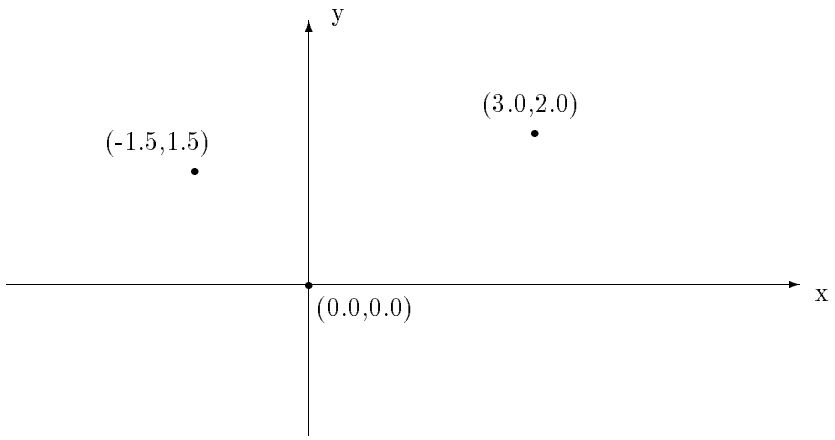


FIG. 3.2 – Plan cartésien (cadran positif en haut et à droite)

Fichier d'entrée

Le fichier que vous devez lire est "`P03.ENT`". La première ligne du fichier d'entrée contient le nombre de points du problème, ce nombre est bien entendu un entier, le nombre maximum de points est 10^3. Le nombre de points est aussi le nombre de lignes suivantes du fichier. Chaque ligne suivante contient un entier et deux nombres de type point flottant (séparés par au moins un espace). L'entier donne le numéro d'ordre du point dans la liste, le premier nombre point flottant est la coordonnée en x et le deuxième est la coordonnée en y. Les coordonnées réfèrent au plan cartésien classique.

Voici un exemple de fichier d'entrée :

```
10
1 -3.69615 -2.73794
2 0.23905 3.37677
3 -2.04369 -3.5242
4 -2.04466 -4.84748
5 -1.14271 -0.734121
6 -1.04056 -2.54924
7 0.174816 4.90285
8 3.24814 2.47733
9 1.81646 -2.0120553
10 0.888974 2.22015
```

Fichier de sortie

Vous devez écrire le fichier "`P03.SOR`", il doit contenir le numéro d'ordre de tous les sommets du polygone en ordre dans le sens horaire ou dans le sens des aiguilles d'une montre. Le premier point de la liste appartenant au polygone doit être le point de départ pour construire le polygone. Le

premier numéro d'ordre doit aussi être le dernier numéro d'ordre de cette liste (pour boucler la boucle). Il doit avoir le numéro d'ordre d'un seul sommet par ligne.

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

```
1
7
8
9
4
1
```

Remarque finale pour ne pas avoir d'ambiguïté, il est garanti qu'un maximum de deux points peut se trouver sur une seule ligne appartenant au contour du polygone. Par exemple les trois points suivants $(1, 2.3)$, $(1, 5.6)$ et $(1, 7.9)$ ne peuvent pas se trouver sur le contour du polygone du fichier test car une ligne relie les trois points, leur coordonnée x étant la même. En réalité, le cas d'ambiguïté se présente lorsque la précision de calculs ne permet pas de déceler si les trois points font partie de la même ligne. Si nous prenons trois points non alignés, il vont former un triangle, le plus petit angle autorisé pour ce triangle doit être supérieur à 0.1^0.

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 4

Scrabble

`Fichier à produire contenant votre code source : P04_EQ##.*`
Le jeu de Scrabble consiste à tenter de former des mots à partir de lettres qu'un joueur a en main; en plus de former un mot, comme à chaque lettre est associée un nombre de points, les mots formés doivent faire un maximum de points pour que le joueur gagne. Parfois, pour déterminer l'ordre selon lequel les participants vont jouer, chacun tire sept lettres et tente de faire un mot avec ses lettres; le joueur qui réussit à faire le mot ayant le plus grand nombre de points sera le premier à jouer, le deuxième sera deuxième, etc.

Problème

Votre programme devra calculer le pointage de chacun des joueurs afin de déterminer l'ordre selon lequel les joueurs vont jouer. La valeur de chaque lettre du jeu de Scrabble est présentée dans le tableau suivant :

A :	1	G :	2	M :	2	S :	1	Y :	10
B :	3	H :	4	N :	1	T :	1	Z :	10
C :	3	I :	1	O :	1	U :	1	? :	0
D :	2	J :	8	P :	3	V :	4		
E :	1	K :	10	Q :	8	W :	10		
F :	4	L :	1	R :	1	X :	10		

Dans le jeu, les joueurs vont prendre des «tuiles». Une tuile est un carreau en bois (ou plastique) indiquant une lettre et sa valeur. Il existe deux tuiles qui n'ont pas de lettre et leur valeur est zéro; pour représenter ces deux tuiles nous avons choisi de les indiquer par le symbole `?`, oui, le symbole point d'interrogation.

Le fichier d'entrée

Le fichier d'entrée "`P04.ENT`" est composé de deux parties :

- La première partie est un dictionnaire de mots possibles correctement orthographiés, à raison d'un mot par ligne. Ces mots ont une longueur variant entre une et sept lettres. Aucun mot ne sera répété. La fin du dictionnaire est indiquée par une ligne vide et il ne contiendra pas plus de 100 mots.

- La deuxième partie est constituée de lignes qui présentent les lettres que chacun des joueurs a à sa disposition. Chaque ligne contient d'abord le nom du joueur, composé d'un maximum de trente caractères. Le nom du joueur est une chaîne contenant des caractères majuscules, sans espace. Après le nom il a un espace, puis une chaîne de sept caractères correspondant au jeu en main d'un joueur. Il est possible que cette chaîne contienne un ou plusieurs points d'interrogations (`?`). Ces derniers représentent des passe-partout qui peuvent jouer le rôle de n'importe quelle lettre et ont toujours la valeur zéro.

Il y aura toujours quatre (4) joueurs dans cette liste.

Notez que tous les caractères dans le fichier d'entrée sont des lettres majuscules et sans aucun accent sauf le caractère `?` qui est le passe-partout. Voici un exemple de fichier d'entrée :

```
CHEZ
MAISON
BERNE
MAIN
CAFE
SINON
TABLEAU
CAPSULE
COUTEAU
DORTOIR
BUFFLE
MALIN

FABIEN SONILEN
PATRICK RENIEBR
MICHEL YAL?EM?
BORNTOLOSE QWYZHCC
```

Le fichier de sortie

Le fichier de sortie "`P04.SOR`" devra contenir une ligne par joueur. Sur chacune de ces lignes, on devra retrouver le nom du joueur, un espace, le mot le plus payant que ce joueur peut faire, un espace, puis le pointage correspondant à ce mot. Si un passe-partout est utilisé, un point d'interrogation (`?`) devra être placé à la position de la lettre qu'il remplace.

Si un joueur ne peut faire aucun mot valide avec son jeu, une étoile (`*`) devra être placée à la place du mot, et le pointage zéro (`0`) devra être indiqué.

Les lignes du fichier de sortie doivent être ordonnées de façon à présenter les joueurs en ordre décroissant de pointage. Si deux joueurs ont le même pointage, vous devez les placer dans le même ordre dans lequel ils se trouvaient dans le fichier d'entrée.

Comme dans le fichier d'entrée, tous les mots doivent être strictement en majuscules ou contenir des `?`. Voici l'exemple de fichier de sortie correspondant à la solution du fichier d'entrée ci-haut :

```
PATRICK BERNE 7
FABIEN SINON 5
MICHEL MAL?? 4
BORNTOLOSE * 0
```

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 5

Marcel et le jeu de quilles (Bowling)

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

Marcel est un quilleur (joueur de bowling) plutôt maladroit, pas dans le jeu, mais dans la vie. Car peu importe l'attention qu'il porte à ses feuilles de pointage, il réussit presque invariablement à renverser un liquide dessus[1]. Ces malencontreux accidents ont tendance à rendre pratiquement illisible ses feuilles de pointage, ce qui empêche Marcel d'analyser le déroulement de ses parties pour tenter d'améliorer ses performances.

Il vous demande de réaliser un programme qui lui permettrait de reconstruire ses feuilles de pointage à partir des quelques informations récupérables à partir des feuilles endommagées.

Pour vous permettre de mener à bien cette tâche, voici les règles de calcul du pointage, telles qu'approuvées par l'organisation «American Bowling Comitee (ABC)». L'objet du jeu de quilles est de faire tomber le plus de quilles en faisant le moins de tirs possibles, au moyen vous l'aurez deviné, d'une boule de quilles.

Chaque joueur a une grille de pointage contenant dix grands carreaux. La figure 5.1 montre une grille typique. Chaque carreau de la feuille de pointage correspond à une des dix séries de tirs qu'un joueur fera dans une partie. Chacune de ces séries de tirs est d'ailleurs dénommée «carreau».

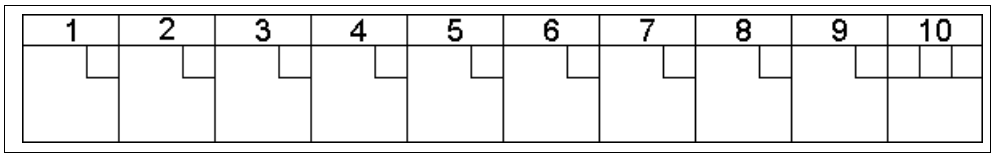


FIG. 5.1 – La grille de pointage

Pour les carreaux 1 à 9.

À chaque carreau, un joueur est placé devant 10 quilles, il doit lancer la boule au maximum deux fois pour faire tomber les 10 quilles.

[1] Pour rester politiquement correct nous avons écrit liquide à la place de bière.

Avant de détailler les modalités du calcul de pointage, voici quelques définitions essentielles :

Abat Un joueur marque un abat lorsqu'il fait tomber les dix quilles d'un carreau au premier de ses deux tirs. Dans ce cas, le deuxième tir est inutile et ne sera pas effectué. Un × désigne un abat sur la feuille de pointage et il est placé dans le petit carreau.

Réserve Un joueur marque une réserve lorsqu'il réussit à faire tomber lors de son deuxième tir le reste des quilles encore debout après le premier tir. Une barre oblique / désigne une réserve et elle est placée dans le petit carreau.

Carreau Ouvert On dit qu'un carreau est ouvert lorsque le joueur ne réussit pas à faire tomber les 10 quilles en deux tirs. On indique les quilles tombées avec le premier tir par un chiffre à gauche du petit carreau, tandis que dans le petit carreau on indique le nombre de quilles tombées par le deuxième tir.

Ainsi, le résultat de chacun des carreaux est inscrit dans la partie supérieure du carreau et le pointage total accumulé jusqu'à ce tir est inscrit dans la partie inférieure.

Vous aurez peut-être remarqué que le format du dixième carreau est légèrement différent des neuf autres. Au dixième carreau, il est en effet possible d'obtenir un ou deux tirs supplémentaires. Ainsi, un abat au premier tir donne droit à deux tirs supplémentaires, soit un total de trois tirs pour le dixième carreau. Une réserve au deuxième tir donne aussi droit à un troisième tir.

Voici maintenant les règles de calcul du pointage:

1. Sur un carreau ouvert, le total du carreau est égal au nombre de quilles abattues après les deux tirs.

2. Sur une réserve, le total du carreau est égal à dix plus le nombre de quilles abattues au prochain tir (le premier tir du carreau suivant), sauf dans le cas du dixième carreau comme on le verra plus loin. Dans l'exemple présenté à la figure 5.2, on peut voir que le total à ajouter au deuxième carreau est de 15, puisque le tir suivant la réserve (le premier tir du troisième carreau) a abattu 5 quilles. De la même manière, au cinquième carreau on indique 20 points à ajouter : 10 du cinquième carreau plus 10 de l'abat du sixième carreau.

3. Sur un abat, le total du carreau est égal à 10 (dix) plus le nombre de quilles abattues lors des deux prochains tirs, sauf dans le cas du dixième carreau comme on le verra plus loin. Toujours dans l'exemple de la figure 5.2, on voit que l'abat enregistré au premier carreau donne un pointage de 20 pour ce carreau, puisque les deux tirs suivants (les deux tirs du deuxième carreau) totalisent 10 quilles. De la même façon, le total des points à ajouter du sixième carreau est de 29, puisque les deux tirs suivants totalisent 19 points (un abat au septième carreau, donc 10 quilles, plus le premier tir du huitième carreau, c'est-à-dire 9 quilles).

4. Pour le dixième carreau, on ne considère que le total des quilles tombées. Ainsi, un abat ou une réserve valent chacun 10 points. Dans l'exemple de la figure 5.2, le total du dixième carreau est donc de 20 points, puisqu'un abat et une réserve ont été inscrits. Cependant, le nombre de quilles abattues à chacun des tirs est quand même reporté aux carreaux précédents si un abat ou une réserve y ont été inscrits. Ainsi, la réserve inscrite au neuvième carreau vaut 20 points, puisque le premier tir du dixième carreau (un abat) totalise 10 quilles.

Au dernier carreau le maximum de points qu'un joueur peut obtenir est 30. Il doit faire tomber 3 fois les 10 quilles, soit 3 abats consécutifs. Car pour le premier abat il a le droit de tirer deux autres boules et il peut faire encore deux abats.

Pour vérifier si vous avez bien compris les principes du calcul de pointage, nous vous suggérons de recalculer le pointage inscrit à la figure 5.2 pour vous assurer que vous arrivez au même résultat.

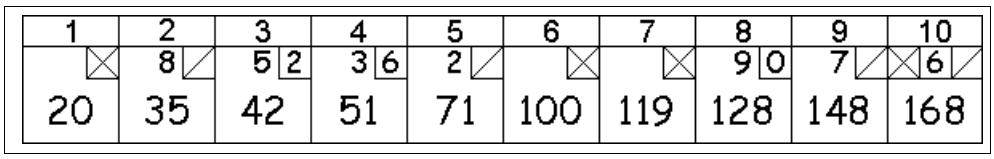


FIG. 5.2 – La grille de pointage

Le problème

Pour être en mesure de venir en aide à Marcel, votre programme devra être en mesure de déterminer la valeur des tirs manquants de la feuille de pointage passée en entrée, en fonction des tirs dont le résultat est connu et du pointage final. Marcel est conscient que plus d'une série de valeurs soit possible pour obtenir un pointage donné. Si le cas se présente, Marcel désire simplement n'importe laquelle des séries possibles, un certain degré d'incertitude étant tout-à-fait acceptable dans ses statistiques personnelles.

Le fichier d'entrée

Le fichier que votre programme doit lire se nomme "`P05.ENT`". Il comprend onze (11) lignes. Les 10 premières lignes représentent chacune un des dix carreaux de la feuille de pointage, en ordre croissant (première ligne → premier carreau, deuxième ligne → deuxième carreau, etc). Chaque ligne contient, en notation décimale, le nombre de quilles abattues à chaque tir, les totaux des différents tirs étant séparés par un espace. Lorsqu'un abat est inscrit, il est noté par un `X`. De la même façon, une réserve est notée par un `/` au tir où elle a été inscrite.

Chaque tir dont le résultat est inconnu est noté par un `?`. Pour les carreaux où les résultats de deux tirs sont des inconnus, à l'exception du dixième carreau, il est possible qu'un abat soit inscrit à ce carreau (les deux inconnus se résumant alors par le résultat obtenu par un seul tir). De la même façon, si le fichier d'entrée indique une inconnue au troisième tir du dixième carreau, rien ne garantit que ce tir ait été effectué. Il faut en effet qu'au moins une réserve ou un abat soit inscrit dans les deux premiers tirs du carreau (qui peuvent eux aussi être des inconnues).

La onzième et dernière ligne du fichier d'entrée contient, en notation décimale, le pointage final de la partie. Cette valeur est toujours connue.

Par exemple si nous avons le fichier suivant :

```
X
8 ?
5 2
? 6
2 ?
X
? ?
? 0
7 /
X 6 ?
168
```

Nous pouvons obtenir comme un résultat valide la grille de la figure 5.2
Voici le fichier d'entrée "`P05.ENT`" :

```
5 ?
X
9 /
? /
X
X
? ?
7 ?
8 /
X ? ?
181
```

Fichier de sortie

Vous devez écrire le fichier "`P05.SOR`". Il doit comporter 10 lignes contenant chacune le résultat de chacun des tirs du carreau correspondant. Le format est le même que pour le fichier d'entrée, soit chaque tir d'un même carreau séparé du suivant par un espace, un abat étant noté par un `X` et une réserve par un `/`. Notez qu'il ne faut pas écrire le total de points dans le fichier de sortie. Voici le fichier de sortie correspondant à une solution vérifiant le fichier d'entrée "`P05.ENT`" présenté plus haut :

```
5 3
X
9 /
7 /
X
X
3 /
7 2
8 /
X X 7
```

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 6

Passer d'une dimension à deux dimensions

`Fichier à produire contenant votre code source : P06_EQ##.*`
L'année dernière nous avons présenté le problème suivant : Transformer une forme à deux dimensions en un vecteur. Nous avons perdu nos fichiers à deux dimensions mais nous avons gardé des fichiers contenant des vecteurs (des millions de vecteurs) et nous voulons reconstituer des formes à deux dimensions[1].

Enoncé de l'ancien problème

Soit l'image bitmap suivante :

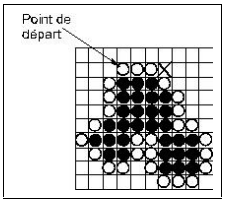


FIG. 6.1 – Le bitmap

Dans cette figure, les points noirs constituent le bitmap, ou l'image à vectoriser. Afin de vectoriser l'image, on en trace le contour, à partir d'un point de départ donné. Pour ce faire, on trouve les points de contour, les points blancs de la figure 6.1, et on note la direction prise pour passer d'un point de contour à un autre. Un point de contour est un point qui possède au moins un côté adjacent à la figure à vectoriser. Par exemple, la case X de la figure 6.1 ne peut pas faire partie du contour puisqu'elle ne possède pas de côté adjacent à la figure. Les huit directions possibles de mouvement

[1] Nous n'avons rien perdu mais il fallait un prétexte pour introduire ce problème.

sont données par la figure 6.2. Pour simplifier le problème, le point de départ est toujours le point juste au-dessus du point le plus à gauche de la ligne la plus haute de l'image, comme à la figure 6.1.

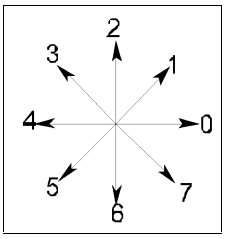


FIG. 6.2 – Les directions

Ainsi on obtient la représentation vectorielle du bitmap. Voir figure 6.3.

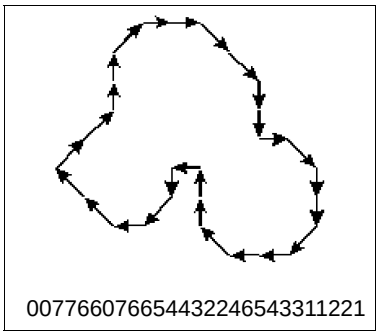


FIG. 6.3 – Représentation vectorielle

Notez bien que le tour de l'image est fait dans le sens des aiguilles d'une montre. De plus, le premier symbole de direction indiqué est celui pour passer du point de départ au deuxième. Le dernier symbole est celui pour passer de l'avant-dernier point au point de départ.

Le cul-de-sac

Tel qu'illustré à la figure 6.4, il peut être compliqué de revenir en arrière sur des cases déjà visitées lorsque l'image à vectoriser possède un cul-de-sac. En arrivant au point B, on se retrouve dans un cul-de-sac, la case précédente étant déjà visitée et les autres cases étant celles de l'image.

Une solution possible à ce problème est d'exploser l'image du premier bitmap en faisant correspondre à chaque point du bitmap original seize points dans le bitmap explosé. Chaque point est donc transformé en une matrice de 4×4 points. Notons que même après l'explosion de l'image, le point de départ reste le point juste au-dessus du point le plus à gauche de la ligne la plus haute de l'image.

La figure 6.5 illustre cette solution.

On pourrait obtenir le vecteur suivant par simple observation du bitmap original, c'est-à-dire la partie gauche de la figure 6.5:

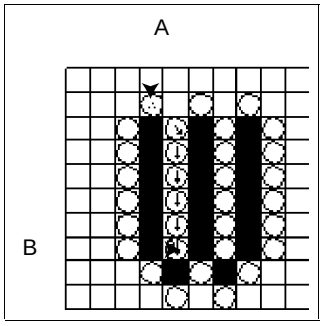


FIG. 6.4 – Le cul-de-sac

```
766662222176666222217666655353322221
```

Le vecteur qu'on obtient après l'agrandissement du bitmap est :

```
000 7 6666 6666 6666 6666 666 000 2222 2222 2222 2222 222 1
000 7 6666 6666 6666 6666 666 000 2222 2222 2222 2222 222 1
000 7 6666 6666 6666 6666 666 5 444 666 5 444 3 222 444 666 5
444 3 222 444 3 2222 2222 2222 2222 222 1
```

Il est possible de transformer cet énorme vecteur pour revenir à l'échelle du bitmap original, en appliquant les quatre règles suivantes :

C étant un symbole de direction entre 0 et 7,

- *CCCC* -> *C* Quatre symboles consécutifs égaux en produisent un seul ;
- *CCC* -> *{}* Trois symboles consécutifs ne produisent pas de symbole, *{}* veut dire vide ;
- *CC* -> *CC* Deux symboles consécutifs produisent deux fois le même symbole ;
- *C* -> *C* Un seul symbole produit le même symbole.

Ainsi nous retrouvons le même vecteur que celui trouvé par observation :

```
766662222176666222217666655353322221
```

Le saut en diagonale

Faites bien attention de ne pas oublier certains points en contournant une image. En effet, lors d'un déplacement en diagonale comme à la figure 6.6, on pourrait, à tort, omettre de contourner certaines parties d'une image.

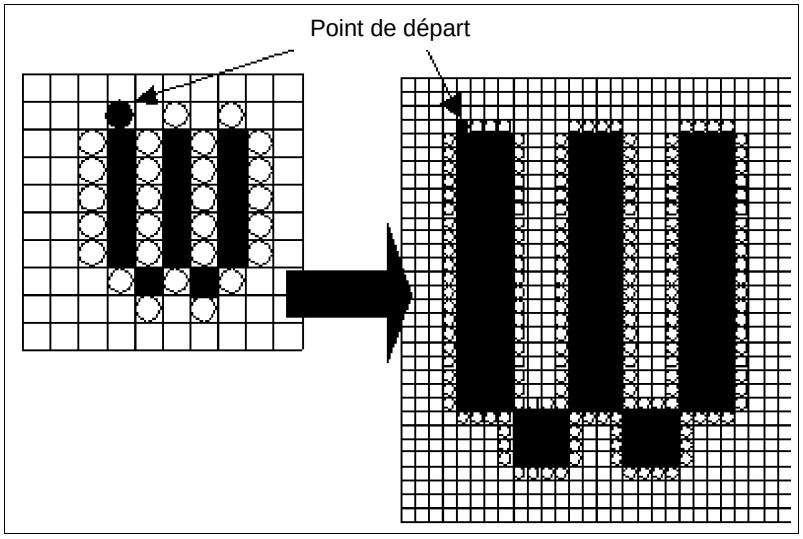


FIG. 6.5 – Une solution

L'entrée

L'entrée se trouve dans un fichier nommé P06.ent[2]. Il contient une matrice de points qui décrit le bitmap. Chacun de ces points peut être 1 ou 0. Si un point est 0, cela signifie qu'il ne fait pas partie de l'image à vectoriser. Un point égal à 1 est un point qui fait partie de l'image à vectoriser. La taille maximale de la matrice est de 16 × 16 éléments. Elle n'est pas toujours carrée.

Afin d'éviter des problèmes évidents, il est garanti que la figure à vectoriser n'est pas creuse, comme par exemple le A de la figure 6.7 :

Il est garanti que le tour du bitmap ne fait pas partie de la figure. Autrement dit, il y a au moins

[2] ancien problème

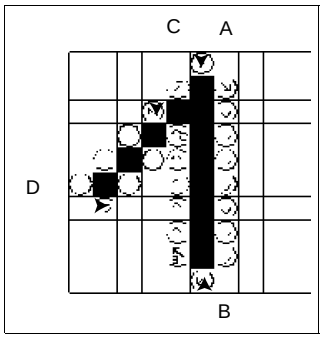


FIG. 6.6 – Le point D pourrait être oublié après un déplacement en diagonale

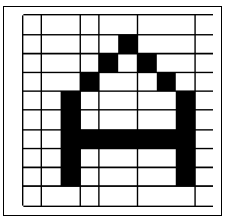


FIG. 6.7 – Les images ne contiennent pas de trou comme celui d'un A

un point blanc entre chacun des points noirs et les bords.

La sortie

La sortie sera constituée d'une seule ligne décrivant le vecteur obtenu en traçant le contour de la figure selon la méthode indiquée. Ce résultat devra être sauvegardé dans un fichier nommé P07.SOR [3].

Exemple

Fichier d'entrée

```
0000000000
0000000000
0001010100
0001010100
0001010100
0001010100
0001010100
0000101000
0000000000
0000000000
```

Fichier de sortie

```
7666622221766662222176666553533222 21
```

Le problème de cette année

Le problème est de faire la transformation inverse en partant d'un vecteur et trouver la forme en deux dimensions.

[3] ancien problème

Fichier d'entrée

Le fichier que vous devez lire est "`P06.ENT`". Il contient le vecteur sur une seule ligne.

```
7666622221766662222176666553533222 21
```

Fichier de sortie

Vous devez écrire le fichier "`P06.SOR`" qui contiendra la forme en deux dimensions. Il ne faut pas que le contour de la figure «touche» les bords du bitmap. On peut considérer que la forme est contenue dans un rectangle imaginaire, les points les plus externes du contour de la forme définissent la taille du rectangle. Dans le fichier de sortie, le rectangle doit être entouré des points blancs, de cette manière le contour de la figure ne touchera pas les bords du bitmap. Ceci est le fichier de sortie correspondant au fichier d'entrée.

```
0000000
0101010
0101010
0101010
0101010
0101010
0010100
0000000
```

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 7

Le tri renversé

`Fichier source à produire avec votre code : P07_EQ##.*`
Un des premiers programmes non numérique, codé par Von Neumann lui-même a été un programme de tri. Aujourd'hui, cette tâche est devenue une opération élémentaire. Qui n'a pas fait comme exercice de programmation au moins l'un des nombreux algorithmes de tri. Le sujet de ce problème est le tri, mais vous n'allez pas trouver l'algorithme pour le résoudre dans le chapitre «Algorithmes de tri» de votre livre préféré.

Problème

Soit une série de mots triée, en suivant l'ordre d'un alphabet de symboles α. Les mots sont composés par des symboles de cet alphabet. Comme un exemple vaut mille mots, voici par exemple un alphabet α composé par 5 symboles :

$\alpha = \{SY5MB\}$ où, $S < Y < 5 < M < B$

Les symboles sont placés dans un ordre spécial. Ce n'est pas l'ordre alphabétique, ni numérique, ni ASCII, c'est un ordre particulier, qui sera utilisé pour trier les 5 mots suivants :

```
B55Y
YMB5
YSM
5
SS
```

Ce qui donnera, vous l'aurez deviné :

```
SS
YSM
YMB5
5
B55Y
```

Le problème proposé est de trouver l'alphabet α bien entendu dans le bon ordre, à partir d'une liste de mots, triée selon cet alphabet.

Fichier d'entrée

Le fichier que vous devez lire est "`P07.ENT`". Tout ce qu'il y a dedans, c'est une série de mots de longueur variable, un mot par ligne, (chaîne de caractères sans espaces) rien de plus. Tous les symboles sont utilisés au moins une fois dans l'ensemble de mots.

Vous retrouvez entre 1 et 94 symboles différents. Ces symboles sont compris uniquement dans les caractères ASCII de valeur décimale de 33 à 126 inclusivement. Il y aura entre 1 et 200 mots, un par ligne. Chaque mot est composé de 1 à 50 caractères. Il y aura suffisamment d'information pour que l'ordre de tri soit déterminé sans ambiguïté.

Voici un exemple de fichier d'entrée :

```
ABB5
ABB6
5B
56
BA5
B5
```

Fichier de sortie

Le fichier que vous devez écrire en sortie est "`P07.SOR`". Vous devez donner, sur une ligne, tous les symboles utilisés, et dans l'ordre qu'ils étaient lors du tri des mots.

Suite à l'exemple donné en entrée, voici le fichier de sortie :

```
A5B6
```

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

Les maisons

`Fichier à produire contenant votre code source : P08_EQ##.*`
Un casse-tête logique demandent de l'analyse, de la patience, de la méthode et de la logique pour être résolu. Armez-vous de tous ces éléments pour écrire le programme correspondant à l'énoncé suivant.

Problème

Cinq maisons se trouvent une à côté de l'autre sur une rue, elles sont chacune d'une couleur différente, leurs occupants, un par maison, sont chacun d'une nationalité différente, ont une profession différente, une boisson préférée différente et un animal domestique différent.

Afin de vous simplifier la vie, nous avons donné un numéro (entre 1 et 5) à chaque valeur possible de chaque caractéristique. Dans tous les fichiers de travail, les caractéristiques seront remplacées par leur numéro correspondant.

- La couleur des maisons : rouge(1), verte(2), jaune(3), bleue(4), blanche(5).
- La nationalité : norvégienne(1), japonaise(2), espagnole(3), anglaise(4), canadienne(5).
- L'animal domestique : chien(1), escargot(2), zèbre(3), chat(4), cheval(5).
- La boisson : café(1), thé(2), lait(3), jus(4), eau(5).
- La profession : sculpteur(1), ingénieur(2), violoniste(3), acrobate(4), médecin(5).

La position des maisons est aussi importante, on suppose que les maisons sont sur une rue et qu'on est face aux maisons, ainsi la première maison est la plus à gauche, la troisième celle du centre et la dernière est la plus à droite.

Par définition, il ne peut pas y avoir plus d'une maison avec la même couleur, plus d'une personne avec la même nationalité, etc.

Vous devez trouver les associations entre la position de la maison, la couleur de la maison, la nationalité de l'occupant, sa profession, sa boisson préférée et son animal domestique, en fonction des indices suivants et des valeurs indiquées dans le fichier d'entrée.

Voici les indices :

1. La maison de gauche est occupée par une personne de nationalité _____.
2. L'occupant de la maison du centre boit _____.
3. L'anglais habite dans une maison de couleur _____.
4. Il y a un chien chez la personne de nationalité _____.
5. On boit du café dans la maison de couleur _____.
6. Le canadien boit _____.
7. La maison verte se trouve à droite de la maison de couleur _____.
8. Le sculpteur a pour animal domestique _____.
9. L'ingénieur habite dans une maison de couleur _____.
10. Le médecin habite à côté de celui dont l'animal domestique est _____.
11. L'ingénieur habite à côté de celui dont l'animal domestique est _____.
12. Le violoniste boit _____.
13. Le japonais exerce la profession de _____.
14. Le norvégien habite à côté de la maison de couleur _____.

Fichier d'entrée

Les attributs remplaçant les blancs se trouvent dans le fichier d'entrée "`P08.ENT`". Voici donc un exemple de fichier d'entrée:

```
1 3 1 3 2 2 5 2 3 4 5 4 4 4
```

Le premier chiffre correspond au premier indice. Le premier indice se lirait : la maison de gauche est occupée par une personne de nationalité norvégienne[1]. Le deuxième chiffre correspond au deuxième indice et ainsi de suite. Le fichier d'entrée contient une ligne avec 14 entiers séparés par au moins un espace.

Fichier de sortie

Vous devez écrire le fichier "`P08.SOR`" Le fichier de sortie doit contenir six lignes et cinq colonnes : une colonne pour chaque maison. La première ligne indique la position des maisons, la plus à gauche porte le numéro 1 et la plus à droite porte le numéro 5. Vous allez avoir au moins une ligne correcte dans ce programme, la première car elle est toujours `1 2 3 4 5`. La deuxième ligne indique la couleur de la maison. La troisième indique la nationalité de l'occupant, la quatrième l'animal de l'occupant, la cinquième la boisson de l'occupant et la sixième indique la profession de l'occupant.

[1] Il faut bien entenu faire les accords nécessaires lorsque l'on traduit l'indice en français.

Vous devez utiliser les codes indiqués au début de la spécification, pour les attributs; de plus, il faut séparer les chiffres par un espace.

Voici maintenant le fichier de sortie correspondant à une solution du fichier d'entrée ci-haut:

```
1 2 3 4 5
3 4 1 5 2
1 5 4 3 2
4 5 2 1 3
5 2 3 4 1
2 5 1 3 4
```

Exemple à exécuter :

On peut lire que la première maison est de couleur jaune, l'occupant est de nationalité norvégiene, il a un chat, il boit de l'eau et sa profession est ingénieur.

Note : Il est possible qu'il existe plusieurs bonnes réponses pour un fichier d'entrée. Nous allons vérifier que votre solution est possible selon les règles établies.

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.

Spécification 9

Sire Georges, le chevalier

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

Bien que l'on ait tenté de vous l'entrer dans la tête des milliers de fois lorsque vous étiez petit, vous l'avez toujours su : le chemin le plus court n'est pas nécessairement la ligne droite; il passe plutôt par le marchand de bonbons! Vous allez enfin avoir l'occasion de prouver une partie de ces dires dans ce problème.

Problème

Nous vous présentons Georges, le chevalier. Georges n'a qu'un seul but dans la vie : sauver des princesses. Pour cela, il parcourt le monde à la quête de nouvelles princesses à sauver! Mais Georges est un mauvais explorateur qui n'a pas idée du comment optimiser ses recherches[1]. Il vous demande de lui fabriquer un programme capable de lui montrer le chemin le plus rapide pour accomplir ces recherches. Tout ce qu'il peut vous fournir c'est une carte contenant diverses informations dont voici un exemple :

```
  4   4   4   3   3   3
4   0   3   4   3   2   2
  3   3   3   3   2   1   1
3   2   2   3   2   1   3
  2   2   3   3  -2   1   1
2  -2   3   4   4
  3   3   5   3   2   1
```

Cette carte, au premier abord aberrante, contient en fait les informations géographiques et touristiques du royaume que visite Georges.

- Le zéro est le château du Roi; c'est là qu'il part notre preux chevalier.
- Les chiffres positifs représentent le nombre de jours que prendra Georges pour quitter la région indiquée.

[1] Occuppé à penser comment sauver des princesses, il ne prêtait pas attention à ses cours de programmation en C++.

- Les chiffres négatifs, indiquent l'endroit où une princesse se trouve emprisonnée. Leur valeur absolue indique combien de jours il faut à Georges pour traverser la case.

La valeur absolue de chiffres est supérieure ou égale à un et inférieure ou égale à 9.

Georges devra aller délivrer toutes les princesses de la carte; le temps requis pour libérer une princesse est négligeable par rapport au temps de parcours indiqué sur la case. Puis il devra retourner au château recevoir sa récompense.

Lorsque Georges traverse la case où se trouve une princesse, il la libère et la valeur absolue de la case indique combien de jours prend Georges pour traverser cette case. Si Georges a besoin de passer une nouvelle fois par cette case, il ne libère plus une princesse (car c'est déjà fait), mais le nombre de jours pour la traverser est indiqué par la valeur absolue de la case.

Un petit détail, mais très important, la carte est constituée de zones hexagonales, ce qui signifie qu'en partant d'une case, Georges pourra se rendre sur l'une des 6 cases indiquées par les chiffres 1 à 6 dans la figure suivante :

```
X   X   X   X   X
  X   6   1   X
X   5   G   2   X
  X   4   3   X
X   X   X   X   X
```

où, bien sûr, Georges est représenté par la lettre `G` et les 6 chiffres représentent les directions que Georges peut suivre et les `X` les cases où il ne peut se rendre directement.

Pour savoir si une case située sur une ligne adjacente est accessible par Georges, sachez qu'il y a deux types de lignes : celles qui commencent par un espace et celles qui commencent par une valeur. Il ne peut pas y avoir deux lignes contiguës du même type.

Chacune des cases est représentée par sa valeur et séparées entre elles par un ou plusieurs espaces. En observant le tableau schéma précédent, on peut alors constater que lorsque Georges se trouve sur une ligne qui commence par une valeur (comme dans l'exemple, sur la troisième case d'une ligne commençant par une valeur), il pourra atteindre la case ayant le même indice de position ou la case avec l'indice de position *précédente* des lignes adjacentes. Toujours en suivant l'exemple, comme Georges se trouve dans la position indice 3 il peut y aller à la deuxième case et la troisième case des lignes adjacentes. Si Georges se trouve sur une ligne commençant par un espace, c'est la case ayant le même indice de position et case avec l'indice de position *suivante* sur les lignes adjacentes (Dans l'exemple, si Georges est sur la case marquée d'un 6, il pourra ainsi se rendre sur la case marquée 5 ou celle marquée G). Bien entendu, il pourra toujours se rendre dans les cases adjacentes sur la même ligne que lui. Vous ne devrez jamais prendre en compte le nombre d'espaces pour déterminer la position d'une case : toutes les cases sur la même ligne se suivent l'une après l'autre, successivement.

Attention toutefois, toutes les lignes n'ont pas nécessairement le même nombre de cases! Georges ne pourra jamais aller sur une case qui n'apparaît pas sur la carte mais il y a toujours au moins un chemin pour se rendre d'une case à une autre représentée sur la carte sans passer par le château. De plus, comme Georges ne s'intéresse présentement qu'aux petits royaumes, ces derniers n'auront pas plus de 10 lignes d'un maximum de 10 cases chacune et, pour aider son cheval, avec pas plus de 9 princesses à sauver par royaume.

Vous devrez trouver la meilleure solution, soit celle où Georges sauve toutes les princesses en le moins de jours possibles. Si plusieurs solutions existent, Georges vous demande de ne lui en rapporter qu'une seule. Georges déteste prendre des décisions.

Fichier d'entrée

Voici un exemple de fichier "`P09.ENT`", il contient la périlleuse aventure entre Georges et deux bandes de Trolls rivales qui avaient enlevé les filles du roi Baltazar :

```
-1  4   4   4   4
  1   1   4   4
4   0   1   4
  4   2   4   4
4   4   1   4
  4   4  -1
```

Il est garanti qu'il y a au moins une princesse à sauver.

Fichier de sortie

Ce que Georges désire par la suite est une marche à suivre très claire. Sur la première ligne du fichier il faut indiquer le nombre de jours nécessaires pour accomplir la tâche. Sur la deuxième ligne du fichier il faut indiquer le nombre de princesses que Georges a sauvées. Sur les lignes suivantes il faut indiquer la direction que Georges doit prendre selon le code de déplacements pour effectuer son parcours. Une direction par ligne. Ces détails, vont aider Georges à mieux négocier ses prix avec ses clients royaux. Voici un exemple du fichier"`P09.SOR`" montrant comment Georges doit exécuter sa noble tâche. N'oubliez pas que ce serait un déshonneur pour Georges de rentrer au château du Roi à moitié bredouille, c'est pourquoi il préfère contourner le château plutôt que de prendre un raccourci au travers.

```
12
2
3
3
3
6
6
1
6
5
6
3
3
```

Le fichier indique à Georges qu'il faut aller chercher la princesse en bas à droite, retourner chercher la princesse en haut à gauche en contournant le château par le haut, puis redescendre victorieux au château, tout ceci en douze jours. Facile, non?

Comme il peut y avoir plusieurs chemins ayant la même durée optimale, nous allons vérifier que celui trouvé par votre programme est un chemin possible et qu'il est optimal.

Alors, pensez-vous pouvoir aider Georges dans ses recherches?

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.

Spécification 10

Métamorphose

`Fichier à produire contenant votre code source : P10_EQ##.*`
La métamorphose est une transformation d'une chose à une autre. Kafka a écrit un livre où il décrit la transformation d'une personne en insecte. Rassurez-vous on ne va pas vous demander d'effectuer cette métamorphose.

Nous allons vous demander de transformer un mot en un autre ainsi le «curé» devient «porc». Pas de panique, c'est le mot `curé` qui devient le mot `porc`[1].

Le problème

Les règles de transformation sont simples :

- Une transformation est dite simple s'il est possible de changer une lettre d'un mot pour en composer un autre. Par exemple, `père` devient `mère`, la lettre `p` est changée en `m`. Une transformation est valide si les deux mots existent dans la langue de travail. Notez que les lettres du mot du départ peuvent occuper ou pas la même position dans le mot d'arrivée. Par exemple, `pour` devient `porc`, la lettre `u` est changée en `c`. Par contre, il n'est pas possible de changer la taille d'un mot par ajout ou par élimination d'une lettre. En d'autres mots, on peut changer une seule lettre et on peut permuter les lettres du mot résultant pour former un mot.
- Compétition française oblige, le dictionnaire est en français. Une règle de plus est nécessaire : le passage par une lettre accentuée compte comme une transformation simple. Exemple : passer de `cire` à `ciré` est une transformation simple.
- Une chaîne de transformations contient plusieurs transformations simples et valides. Par exemple, la chaîne :

  `curé` → `cure` → `dure` → `pure` → `pour` → `porc`

  nous fait passer du mot `curé` au mot `porc`.

Nous allons vous demander de nous indiquer une chaîne de transformations (incluant le point de départ et le point d'arrivée) pour passer d'un premier mot à l'autre mot.

[1] Nous voulons remercier Aude Couderc et Florence Gonzalez Rubio pour leur aide à la conception de ce problème.

Le fichier d'entrée

Le fichier en entrée est "`P10.ENT`" et comporte deux mots : deux chaînes de caractères sur deux lignes différentes.

```
curé
porc
```

Le premier mot est le point de départ de la méthamorphose et le deuxième est le point d'arrivée.

Vous devez consulter le fichier `DICO.TXT` pour déterminer si un mot est valide ou pas. Si le mot est dans le dictionnaire, il est valide sinon, il ne l'est pas. Ce fichier contient un mot par ligne. Un mot est une chaîne de caractères. Le fichier `DICO.TXT` est un dictionnaire français contenant près de 200 000 mots, en principe complet.

Le fichier de sortie

Pour le fichier de données en entrée vous devez produire le fichier "`P10.SOR`" contenant la chaîne de transformations trouvée incluant le mot du départ et le mot d'arrivée. Le fichier doit contenir un mot par ligne :

```
curé
cure
dure
pure
pour
porc
```

Dans ce programme il est garanti que la chaîne de transformations existe. Comme une transformation peut se faire de plusieurs manières, nous allons vérifier que votre transformation est possible selon les règles indiquées. Il est aussi garanti que le nombre maximum de caractères d'un mot en entrée est 8.

Autre exemple

Pour ce fichier en entrée :

```
père
mari
```

Voici un fichier possible en sortie :

```
père
mère
mare
mari
%mami
```

Bien entenu la taille des mots dans le fichier d'entrée peut être différent de quatre caractères par mot.

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 : 60 secondes.