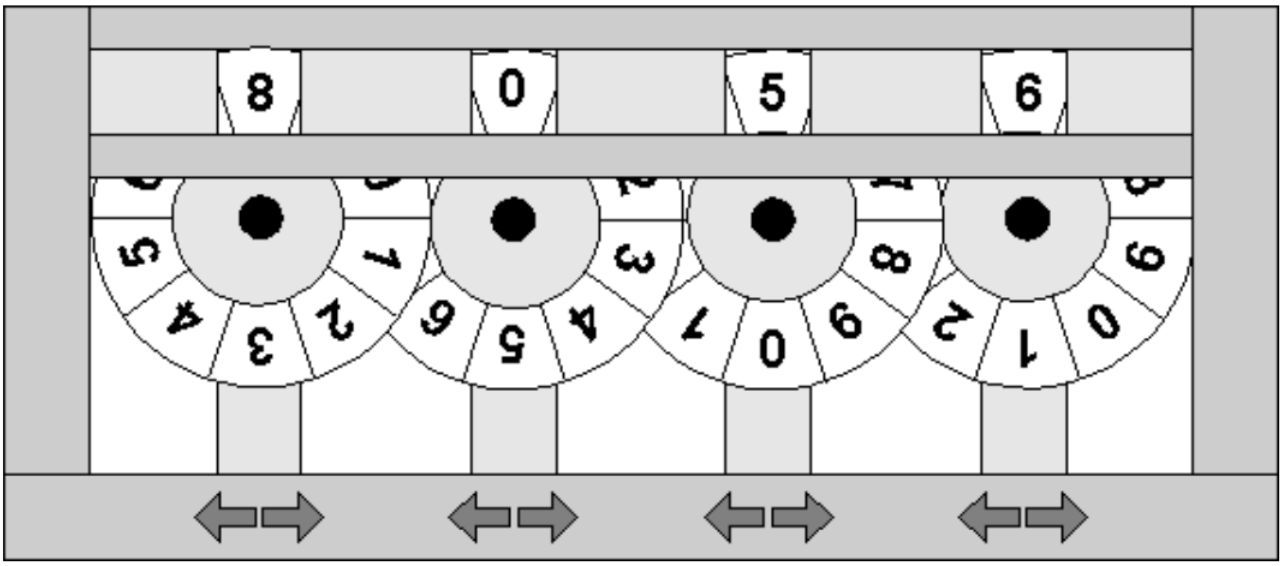
13ème édition

Le 21 avril 2017



Les roues tournent

Règlement

Date et Lieu : La 13ème édition du concours de programmation de l'EPFC se déroulera le **vendredi 21 avril 2017** dans les locaux de l'***EPFC*** à Woluwe-Saint-Pierre (2, av. Charles Thielemans, 1150 Bruxelles) de 13h25 à 21h30.

Candidats : Le concours est ouvert aux étudiants en deuxième et troisième année du bachelier en informatique de l'***EPFC***, ainsi qu'aux anciens étudiants. Des dérogations pour d'autres étudiants peuvent être accordées. Afin de faciliter l'organisation, des préinscriptions auprès de A. SILOVY (asilovy@ulb.ac.be) sont souhaitées. Les étudiants peuvent concourir en équipe de 2 ou 3 à condition que ces équipes soient clairement identifiées, auprès des responsables, au départ de l'épreuve. Bien entendu, dans ce cas, un éventuel prix sera partagé par les membres de l'équipe.

Epreuve : Programmation d'une (ou plusieurs) application(s) console. Les sujets des problèmes posés ne privilégient aucun thème particulier. Il s'agit avant tout de problèmes purement algorithmiques qui ne nécessitent pas de connaissances spécifiques sur des domaines précis d'informatique (programmation graphique, réseau, etc.) ni d'autre domaine scientifique (mathématiques, physique, etc.). Tout langage de programmation approprié à l'écriture d'une telle application est accepté. En particulier, l'infrastructure de l'EPFC prévoit l'usage des langages *Java, C/C++, C#, Python, Scala* et *Haskell*. Les participants désireux d'utiliser un autre langage de programmation doivent s'adresser préalablement à l'un des responsables. A l'issue du concours, les candidats devront remettre le code source de leur solution (ainsi que l'exécutable) sur le serveur du réseau informatique de l'***EPFC***.

Conditions : Les participants peuvent apporter toute documentation écrite ou électronique qu'ils peuvent juger utile. Ils peuvent donc employer des librairies de code sur clé *USB*… Cependant, durant la durée de l'épreuve, ils ne peuvent pas communiquer avec l'"extérieur". L'usage d'*Internet* et du *GSM* (à l'exception évidente d'une impérieuse exigence) est proscrit.

Classement : Contrairement aux habitudes pédagogiques, le premier critère de sélection sera le fonctionnement des exécutables répondants aux demandes formulées dans l'énoncé. Il correspond donc à **la correction du programme**. Pour départager les candidats, le classement se fera ensuite sur base des critères suivants :

- L'algorithme choisi (et sa complexité)
- Le choix des structures de données
- L'analyse du problème
- L'élégance du code source
- L'efficacité de la solution

Prix : **Des prix sont prévus, soit sous forme de matériel, soit sous forme de bons d'achat.**

Une proclamation des résultats sera organisée avec remise des prix.
Elle sera suivie d'un drink offert à tous les participants

Le problème : Chacune des 4 roues présentées sur la page de garde contient les chiffres de 0 à 9. Le haut des roues présente ainsi un nombre de 4 chiffres (8056 sur l'exemple). A chaque roue sont associés deux boutons symbolisés par les flèches qui permettent de faire tourner les roues d'une position, dans un sens ou dans l'autre. Etant donnée une configuration de départ des roues, nous voudrions atteindre une configuration d'arrivée. Malheureusement, le mécanisme est défectueux et certaines configurations données à l'avance sont impossibles. Quel chemin suivre pour atteindre la configuration désirée ?

Par exemple, partant de la configuration montrée (qui affiche donc 8056), nous voudrions atteindre la configuration qui affiche le nombre 0056. Nous pourrions simplement tourner la première roue (en partant de la gauche) deux fois dans le sens contraire des aiguilles d'une montre, et y arriver donc en 2 étapes (8056 → 9056 → 0056), mais malheureusement le mécanisme se bloque et ne peut passer par les configurations suivantes : 9056, 8156, 8956, 8046, 8055, 8057.

Il est alors possible d'y arriver en tournant la première roue 8 fois dans l'autre sens (celui des aiguilles d'une montre) : 8056 → 7056 → 6056 → 5056 → 4056 → 3056 → 2056 → 1056 → 0056.

Mais en fait, il est possible d'y arriver en 4 étapes seulement car « les roues tournent » toutes en vérité ; il s'agit juste d'éviter les configurations impossibles. Donc, par exemple : 8056 → 8066 → 9066 → 0066 → 0056.

Etant données les configurations de départ, d'arrivée et les configurations par lesquelles il est impossible de passer, il vous est demandé de trouver le chemin à suivre.

Remarquez qu'il peut être impossible d'y arriver. Par exemple si la configuration de départ est 0000 et que les configurations 0001, 0009, 0010, 0090, 0100, 0900, 1000, 9000 sont toutes impossibles, aucune configuration autre que celle de départ n'est atteignable.

Remarquez enfin que dans certaines situations, au contraire, il peut être possible d'y arriver par plusieurs chemins différents de même longueur. Ils seront alors également valables.

Fonctionnement de votre programme

En pratique, votre programme doit lire sur l'entrée standard une suite de nombres, un nombre par ligne. Le premier nombre représentera la configuration de départ (sous la forme d'un nombre entier positif ou nul de 4 chiffres), le deuxième, la configuration à atteindre. Le nombre suivant, *nb*, indiquera combien de nombres suivront encore. Ces derniers *nb* nombres représenteront les configurations impossibles. Par exemple :

```
8056
0056
6
9056
8156
8956
8046
8055
8057
```

Remarquez que la configuration à atteindre est fournie avec le préfixe 00. En tant que nombre, ces deux 0 sont non-significatifs mais ils sont fournis car ils font bien partie de la configuration et montrent ainsi les positions des 4 roues.

Votre programme devra simplement afficher la suite de configurations à suivre pour résoudre le problème, toutes sur une seule ligne, séparées par un espace. Donc, pour l'exemple ci-dessus :

```
8056 8066 9066 0066 0056
```

S'il est impossible d'atteindre la configuration finale, votre programme n'affichera **rien**.

Consignes

Il est impératif que vous respectiez le format exact des Entrées/Sorties spécifié ci-dessus. Votre affichage doit donc respecter scrupuleusement les spécifications données, à savoir, afficher **uniquement** la suite de nombres, séparés par un espace. En particulier, il ne doit pas afficher de message pour la saisie des données (du genre « `entrez des nombres` » …) ni afficher un message du style « `pas de solution` ».

Cette consigne a pour objectif de permettre la mise en œuvre de tests automatiques de votre programme. Veillez à la respecter scrupuleusement. Votre programme pourrait être correct et néanmoins ne pas être validé pour cette raison.

Vous ne devez pas vérifier la saisie de l'utilisateur. Vous pouvez donc supposer que le format spécifié ci-dessus sera toujours respecté, l'utilisateur n'entrant jamais que des nombres entiers positifs ou nuls.

Afin de garantir que vous respectiez ce format d'E/S, un programme simple effectuant les mêmes entrées et sorties que celui demandé vous est fourni pour différents langages (voir annexes). **Veuillez les utiliser comme modèle**.

Par exemple, si votre programme se nomme `roues`, son exécution sur la ligne de commande pourrait donner lieu à l'interaction ci-dessous :

```
C:\>roues↵
8056↵
0056↵
6↵
9056↵
8156↵
8956↵
8046↵
8055↵
8057↵
8056 8066 9066 0066 0056
C:\>
```

Remarques

Dans cet exemple, la saisie de l'utilisateur est soulignée. Le reste est affiché par l'ordinateur. Remarquez bien : aucune autre entrée que celles spécifiées plus haut. Bien entendu, en *Java*, la commande serait `java Roues↵`.

L'énoncé mentionne partout des nombres (entiers positifs ou nuls) pour représenter la configuration des 4 roues. Il est clair que le nombre entier 56 doit être interprété comme la configuration des 4 roues 0056 et peut aussi être vu comme le string `"0056"` ou un tableau de 4 chiffres. Pour votre affichage, vous êtes libre d'utiliser le format abrégé (56) ou complet (0056). Votre affichage pourrait donc produire

```
8056 8066 9066 66 56
```

Redirection des Entrées/Sorties

Comme vous l'avez étudié par ailleurs, les systèmes d'exploitation permettent une redirection des E/S afin de soulager l'utilisateur de devoir entrer tous les nombres. Votre programme sera exécuté en fournissant des fichiers de données préparés. Donc, par exemple :

```
C:\>roues < Exemple.in > Exemple.out↵
```

Si le fichier `Exemple.in` contient le texte donné dans le premier cadre de la page 4, le fichier `Exemple.out` contiendrait au final le texte du deuxième cadre.

Annexes

Afin de vous illustrer l'usage des entrées/sorties dans différents langages (*Java, C++, C#, Python* et *Haskell*), vous trouverez, ci-après, et/ou sur le serveur, au format électronique, les codes sources d'un programme simple qui, comme indiqué plus haut, utilise le même format d'Entrée/Sortie que celui de votre problème. Vous devriez utiliser le format de ces E/S sans modification pour votre programme !

Tous les fichiers utiles à votre travail sont disponibles sur le serveur au point **`PROFS-PUBLIC\CONCOURS`**. Les programmes d'exemples d'entrées/sorties dans les différents langages sont dans le dossier **`DemosIO`**. Quelques fichiers d'exemples avec des solutions possibles sont dans le dossier **`Exemples`**.

Réponses

Vous copierez vos solutions (sources et exécutables) dans un répertoire portant votre nom sur le serveur au point **`PROFS-PUBLIC\CONCOURS\REPONSES`**.

Bon Amusement

DemoIO.java

```
/*
Programme de démo des E/S pour le concours de programmation EPFC 2017.
Vous ne devez pas modifier le format de ces E/S

Ce programme lit sur l'entrée standard une suite d'au minimum 3 nombres entiers,
positifs ou nuls, les deux premiers de 4 chiffres, un nombre par ligne.
Les deux premiers nombres représentent les configurations des roues
du problème (voir énoncé).
Le troisième nombre indique si, et combien de nombres suivent.
Le programme réaffiche toutes les configurations qui contiennent
au moins un zéro, même non significatif, séparées par un espace.
*/

import java.util.*;

public class DemoIO {
    private static final int NB_WHEELS = 4;
    private static Scanner scan = new Scanner(System.in);

    private static int nbZero(int n) {
        int result = 0;
        for(int i = 0; i < NB_WHEELS; ++i) {
            if(n % 10 == 0)
                ++result;
            n /= 10;
        }
        return result;
    }

    public static void main(String[] args) {
        List<Integer> hasZeros = new ArrayList<>();
        int config;
        for(int i = 0; i < 2; ++i) {
            config = scan.nextInt();
            if(nbZero(config) > 0) hasZeros.add(config);
        }

        int nbImpossibles = scan.nextInt();
        for(int i = 0; i < nbImpossibles; ++i) {
            config = scan.nextInt();
            if(nbZero(config) > 0) hasZeros.add(config);
        }

        for(int x : hasZeros)
            System.out.print("" + x + ' ');
    }
}
```

DemoIO.cpp

```
/*
Programme de démo des E/S pour le concours de programmation EPFC 2017.
Vous ne devez pas modifier le format de ces E/S

Ce programme lit sur l'entrée standard une suite d'au minimum 3 nombres entiers,
positifs ou nuls, les deux premiers de 4 chiffres, un nombre par ligne.
Les deux premiers nombres représentent les configurations des roues
du problème (voir énoncé).
Le troisième nombre indique si, et combien de nombres suivent.
Le programme réaffiche toutes les configurations qui contiennent
au moins un zéro, même non significatif, séparées par un espace.
*/

#include <iostream>
#include <vector>

using namespace std;

const unsigned NB_WHEELS = 4;

int nbZero(unsigned n) {
    int result = 0;
    for(unsigned i = 0; i < NB_WHEELS; ++i) {
        if(n % 10 == 0)
            ++result;
        n /= 10;
    }
    return result;
}

int main() {
    vector<unsigned> hasZeros;
    unsigned config;
    for(unsigned i = 0; i < 2; ++i) {
        cin >> config;
        if(nbZero(config) > 0) hasZeros.push_back(config);
    }

    unsigned nbImpossibles;
    cin >> nbImpossibles;
    for(unsigned i = 0; i < nbImpossibles; ++i) {
        cin >> config;
        if(nbZero(config) > 0) hasZeros.push_back(config);
    }

    for(unsigned x : hasZeros)
        cout << x << ' ';
}
```