EPFC

Concours de programmation

10ème édition

Le 9 mai 2014

Editeur de Partition Facile et Convivial

EPFC

Concours de programmation

Règlement

Date et Lieu : La 10ème édition du concours de programmation de l'EPFC se déroulera le **vendredi 9 mai 2014** 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. 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++*, *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 l'exécutable et le code source de leur solution 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 répondant 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'efficacité de la solution
- L'analyse du problème
- L'algorithme choisi (et sa complexité)
- Le choix des structures de données
- L'élégance du code source

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

La proclamation des résultats aura lieu le vendredi 23 mai à 17h30.
Elle sera suivie d'un drink offert à tous les participants

Le problème: Vous contribuez à l'écriture du logiciel *Editeur de Partition Facile et Convivial* (**EPFC**) permettant de gérer les partitions d'un disque dur. Etant donnés la taille d'un disque dur et les positions des limites de partitions vous devez trouver la stratégie optimale de découpage, sachant que le temps pris pour découper un volume en deux ne dépend que de la taille du volume de départ.

Il est facile de se rendre compte que l'ordre dans lequel les partitions vont être sous-divisées est important. Par exemple si vous devez découper un volume de 10 Gb aux limites 2, 4 et 7 (pour former 4 partitions, respectivement de tailles 2, 2, 3 et 3, l'une commençant en position 0 et s'étendant jusqu'en position 2, une allant de 2 à 4, une de 4 à 7 et la dernière de 7 à 10), vous pourriez commencer par découper en position 2, ensuite en 4 et ensuite en 7. Le temps serait de 10 + 8 + 6 = 24 unités car le volume de départ est 10, le suivant à être subdivisé est de taille 8 et le dernier, de taille 6. Une autre possibilité est de commencer à découper en position 4, puis en 2 puis en 7. Dans ce cas, le temps sera de 10 + 4 + 6 = 20 unités.

Remarquez que l'idée d'un temps dépendant de la taille du volume de départ n'est pas réaliste. Cette hypothèse ne vaut que pour les besoins de l'énoncé.

En pratique, votre programme doit lire, sur l'entrée standard, un premier nombre positif T, représentant la taille du disque. Il doit lire ensuite le nombre N de limites de partitions ($N + 1$ étant le nombre final de partitions) et enfin, N nombres positifs p_i ($0 < p_i < T$) indiquant les positions des limites de partitions, données dans un ordre strictement croissant ($p_i < p_{i+1}$ pour tout i de 1 à $N - 1$).

Votre programme doit ensuite écrire sur la sortie standard le temps de la solution optimale. Par exemple, si votre programme se nomme `partition`, son exécution pourrait donner lieu à l'interaction ci-dessous :

```
C:\> partition↵
10↵
3↵
2↵
4↵
7↵
Solution optimale : 20
C:\>
```

Remarques

Dans cet exemple, la saisie de l'utilisateur est soulignée. Le reste est affiché par l'ordinateur. Bien entendu, en *Java*, la commande serait `java Partition`↵.

Les nombres sont fournis un par ligne.

Afin de faciliter la saisie, l'entrée pourra être obtenue au départ d'un fichier texte, par une re-direction de l'entrée standard.

Par exemple, le fichier texte *data1.txt*, correspondant à la partition proposée précédemment,

Data1.txt

```
10
3
2
4
7
```

pourrait donner lieu à l'interaction suivante :

```
C:\> partition < data1.txt↵
Solution optimale : 20
C:\>
```

D'autres exemples de fichiers d'inputs vous sont fournis avec la solution optimale attendue. Ils portent les noms `data1.txt`, `data2.txt` … pour les inputs et `resultat1.txt` … pour les résultats attendus.

En pratique, vous pouvez supposer $T < 1000$ et $N < 50$ (ce qui signifie un disque de taille inférieure à 1000 Gb et un maximum de 50 partitions).

Consignes

Vous ne devez pas vérifier la saisie de l'utilisateur. Vous pouvez donc supposer que le format spécifié ci-dessus sera toujours respecté. Quelques fichiers d'input vous sont fournis.

Par contre, votre affichage doit respecter scrupuleusement les spécifications données, à savoir, le texte « `Solution optimale :` » suivi d'une simple valeur numérique. Votre programme ne doit rien afficher d'autre. En particulier, il ne doit pas afficher de message pour la saisie des données (du genre « `entrez la taille` »…).

Annexes

Afin de vous illustrer l'usage des entrées/sorties dans différents langages (*Java, C++, C, C#, Scala* et *Haskell*), vous trouverez, ci-après, et/ou sur le serveur, au format électronique, les codes sources d'un programme simple qui utilise le même format d'Entrée/Sortie que celui de votre problème. Vous pouvez (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 dans les différents langages sont dans le dossier **`Demos`**, les quelques fichiers d'input, avec la solution attendue dans le dossier **`Datas`**.

Remarques

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

Bonne Chance

(et, surtout, bon amusement)

Question subsidiaire

Afin de départager d'éventuels concurrents ex aequo, une solution au problème suivant est demandée :

En plus de fournir la valeur de la solution optimale, votre programme fournira ensuite l'ordre du découpage optimal. Par exemple avec le fichier d'input proposé plus haut, votre programme afficherait

```
C:\> partition < data1.txt↵
Solution optimale : 20
Coupure en : 4
Coupure en : 2
Coupure en : 7
C:\>
```

Si plusieurs stratégies optimales existent, vous pouvez vous contenter de n'en afficher qu'une seule.

DemoIO.java

```
/*
Programme de démo des E/S pour le concours de programmation EPFC 2014.
Vous ne devez pas modifier le format de ces E/S
(si ce n'est enlever le message "DemoIO..."!)

Ce programme lit deux nombres length et nbCuts sur les 2 premières lignes de
l'entrée standard. Il lit ensuite nbCuts nombres (un par ligne) et écrit sur
la sortie standard le texte "Somme : " suivi de la somme de tous les nombres
lus et d'un saut de ligne.
*/

import java.util.*;

public class DemoIO {
  public static void main(String[] args) {
    System.out.println("DemoIO Java");
    Scanner s = new Scanner(System.in);
    int length = s.nextInt();
    int nbCuts = s.nextInt(), somme = 0;
    for(int k = 0; k < nbCuts; ++k) {
      int cutPlace = s.nextInt();
      somme += cutPlace;
    }
    System.out.println("Somme : " + (length + nbCuts + somme));
  }
}
```

DemoIO.cpp

```
/*
Programme de démo des E/S pour le concours de programmation EPFC 2014.
Vous ne devez pas modifier le format de ces E/S
(si ce n'est enlever le message "DemoIO..."!)

Ce programme lit deux nombres length et nbCuts sur les 2 premières lignes de
l'entrée standard. Il lit ensuite nbCuts nombres (un par ligne) et écrit sur
la sortie standard le texte "Somme : " suivi de la somme de tous les nombres
lus et d'un saut de ligne.
*/

#include <iostream>

using namespace std;

int main() {
  cout << "DemoIO C++\n"; // Enlevez ce message dans votre programme!
  unsigned length;
  cin >> length;
  unsigned nbCuts, somme = 0;
  cin >> nbCuts;
  for(unsigned k = 0; k < nbCuts; ++k) {
    unsigned cutPlace;
    cin >> cutPlace;
    somme += cutPlace;
  }
  cout << "Somme : " << length + nbCuts + somme << endl;;
}
```