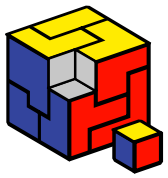
Prologin 2012



Concours National d'Informatique

Questionnaire de sélection

Correction des questions d'algorithmique

Jill-Jênn Vie Antoine Pietri *

4 janvier 2012

Table des matières

1 Éligibilité

1.1 Énoncé

Cette année, pour participer à Prologin, il faudra être né le 1er mai 1991 ou après cette date[1]. Écrivez une fonction prenant en argument la date de naissance d'une personne et retournant 1 si elle peut participer à Prologin cette année, 0 sinon.

1.2 Solution

La plupart des candidats ont opté pour une simple structure conditionnelle utilisant des opérateurs booléens. La première chose à voir est que le jour n'a aucune influence sur le résultat : « le 1er mai 1991 ou après » inclut en effet tout le mois de mai. La solution évidente est donc de vérifier si la date de naissance est strictement plus grande que 1991 ou qu'elle est égale à 1991 et que le mois est plus grand ou égal à 5, soit :

$$Y > 1991 \textbf{ OU } (Y = 1991 \textbf{ ET } M \geqslant 5).$$

*Pour l'équipe des correcteurs 2012 : Rémi Audebert, Guillaume Davy, Julien Freche, Paul Hervot, Jérémie Marguerie, Franck Michea, Antoine Pietri, Jill-Jênn Vie, Benoît Zanotti.

1. Certes, depuis le 11/11/11, il suffit d'être né en 1991 pour pouvoir participer.

Listing 1 – Une solution de l'exercice 1 en C

```c
#include <stdio.h>

int main(void) {
  int j, m, a;
  scanf("%d %d %d", &j, &m, &a);
  printf("%d\n", (a > 1991 || (a == 1991 && m >= 5)) ? 1 : 0);
  return 0;
}
```

1.3 Solution d'un candidat

Un des candidats a proposé une solution astucieuse basée sur un ordre lexicographique [2] ; la limite supérieure des années (2042) est en effet inférieure à 2×1991, ce qui permet d'établir la relation suivante :

$$10Y + M \geqslant 19915 \Leftrightarrow (Y, M) \geqslant_{lex} (1991, 5)$$

Listing 2 – Solution de l'exercice 1 en OCaml par thizanne

```ocaml
let estEligible' d m y =
  (10 * y + m) / 19915
```

2 Équilibre

2.1 Énoncé

Avant l'épreuve écrite, par souci d'équité, nous veillons à ce que chaque candidat ingurgite la même quantité de nourriture. Par conséquent, tous nos pains au chocolat font exactement le même poids. Cependant, cette fois-ci, l'un des N pains au chocolat dont nous disposons est plus lourd que les autres. À l'aide d'une balance à deux plateaux, vous devez le dépister, en manipulant le moins de pains au chocolat possible à chaque pesée.

Lors d'une pesée, vous manipulez trois tas de pains au chocolat : un tas sur le plateau gauche de la balance, un tas sur le plateau droit, ainsi que le tas restant, à l'extérieur de la balance. On vous demande de décrire le pire cas : celui où le pain au chocolat le plus lourd se trouve toujours dans un des plus gros tas, en indiquant à chaque pesée le nombre d'éléments qu'il vous reste à analyser.

2.2 Solution

On peut songer à une dichotomie [3] pour diviser par deux à chaque pesée le nombre d'éléments restant à analyser, mais comme suggéré dans l'exemple du serveur d'entraînement, diviser par trois réduit le nombre de pesées.

En effet, une pesée partitionne les $N \geqslant 2$ pains au chocolat en trois tas :

- le tas de gauche, qui contiendra p pains ;
- le tas de droite, qui contiendra également p pains (sinon la pesée n'a pas d'intérêt) ;
- le tas restant (en cas d'équilibre entre les deux tas ci-haut).

À l'étape d'après, puisque l'on considère le pire cas, on se retrouvera avec le tas comportant le plus de pains au chocolat. Il faut donc minimiser ce maximum, ce qui permet d'extraire les cas suivants.

- Si $N = 3k$, chacun des trois tas comportera k pains.
- Si $N = 3k + 1$, les tas sur la balance comporteront k pains et le tas restant $k + 1$ pains.
- Si $N = 3k + 2$, les tas sur la balance comporteront $k + 1$ pains et le tas restant k pains.

2. Et de la bidouille.
3. Division par deux. http://fr.wikipedia.org/wiki/Dichotomie

Il s'agit donc de rappeler la fonction[4] sur $N/3$ dans le cas où 3 divise N, et sur $\lfloor N/3 \rfloor + 1$ sinon. On s'arrête lorsque $N = 1$.

Listing 3 – Une solution de l'exercice 2 en C

```c
void peser(int n) {
  if(n == 1)
    return;
  printf("%d\n", n);
  if(n % 3 == 0)
    peser(n / 3);
  else
    peser((n / 3) + 1);
}
```

2.3 Solution d'un candidat

Il est possible de se passer d'une condition grâce à l'astuce $\lfloor (N+2)/3 \rfloor$, qui donnera bien $N/3$ si N est un multiple de 3 et $\lfloor N/3 \rfloor + 1$ dans les autres cas.

Listing 4 – Solution de l'exercice 2 en OCaml par thizanne

```ocaml
let rec nombreDePesees n =
  if n = 1 then ""
  else string_of_int n ^ "\n" ^ nombreDePesees ((n+2) / 3)
```

3 Hydratation

3.1 Énoncé

En salle machine, il fait souvent très chaud, c'est pourquoi nous abreuvons nos candidats. Par souci de précision, nous leur permettons même de demander le nombre de centilitres de jus d'orange qu'ils souhaitent. Hélas, bien que nous ayons une quantité infinie de jus d'orange, nous ne disposons que de deux gobelets, respectivement de a et b centilitres ($a < b$).

Vous pouvez :

- remplir un gobelet à ras bord de jus d'orange ;
- vider un gobelet intégralement ;
- verser un gobelet dans un autre jusqu'à ce que l'un soit vide ou que l'autre soit plein.

Combien de commandes différentes comprises entre 1 et b centilitres pouvons-nous satisfaire ?

3.2 Solution

Ce problème, d'apparence difficile, admet en réalité une solution simple.

En essayant avec quelques valeurs, les volumes atteignables $\mathcal{V}$ semblent être exactement les multiples du PGCD de a et b noté p :

$$\mathcal{V} = \left\{ p, 2p, 3p, \ldots, \left(\frac{b}{p}\right) \cdot p \right\}$$

Soit $\dfrac{b}{p} = \dfrac{b}{\mathrm{PGCD}(a,b)}$ volumes en tout.

Listing 5 – Une solution de l'exercice 3 en C

```c
int pgcd(int a, int b) {
  if(b == 0)
    return a;
```

4. http://fr.wikipedia.org/wiki/Récursivité

```
4   return pgcd(b, a % b);
5 }
6
7 int nombreDeCommandes(int a, int b) {
8   return b / pgcd(a, b);
9 }
```

3.3 Preuve

On part des gobelets A (de a cl) et B (de b cl) vides, les seuls volumes atteignables sont 0, a et b. Soit $p = \text{PGCD}(a, b)$.

Les opérations possibles faisant découvrir de nouveaux volumes sont les suivantes.

- $\mathbf{V_1}$: verser A plein dans B contenant déjà v cl. Si après cela, B n'est pas plein, il contiendra $v+a$ cl ; s'il est plein, il restera $v + a - b$ cl dans A. On passe donc exactement de v à $(v + a)$ mod b et on note $v \xrightarrow{\mathbf{V_1}} (v + a) \bmod b$.
- $\mathbf{V_2}$: verser B plein dans A contenant déjà v cl. A est alors plein (car $a < b$) et B contient $v + b - a$ cl ($v \xrightarrow{\mathbf{V_2}} v + b - a$).
- $\mathbf{E}$: écoper B avec A (s'il y a plus de a cl dans B). $v \xrightarrow{\mathbf{E}} v - a$.

Définition. On dit que v est *atteignable* s'il peut être obtenu à partir de 0 cl en appliquant un certain nombre d'opérations.

Exemple. Dans *Une journée en enfer*, Bruce Willis doit réussir à obtenir 4 gallons avec $a = 3$, $b = 5$.

- Il verse B plein dans A vide. B contient $0 + 5 - 3 = 2$ gallons, qu'il transvase dans A une fois celui-ci vidé.
- Il verse B plein dans A contenant 2 gallons. B contient alors $2 + 5 - 3 = 4$ gallons.

Donc $0 \xrightarrow{\mathbf{V_2}} 2 \xrightarrow{\mathbf{V_2}} 4$, $v = 4$ est atteignable.

Propriété. Montrons que v est atteignable **si et seulement si** [5] v est un multiple du PGCD de a et b noté p, compris entre 0 et b.

Preuve. Montrons que si v est atteignable, alors v est un multiple de p. La propriété « Le volume que je manipule est un multiple de p » est un invariant des opérations : elle est évidemment vraie pour 0, et les opérations ne font qu'ajouter ou retirer a ou b, qui sont des multiples de p. Donc tous les volumes atteignables sont des multiples de p.

Montrons à présent que si $v = kp$ pour un certain k tel que $0 \leqslant v \leqslant b$, alors v est atteignable. D'après le théorème de Bézout [6], il existe $m, n > 0$ tels que [7] $ma - nb = p$.
Alors $kma - knb = kp$.
$kp \leqslant b$, donc $kma \bmod b = kp$.
Donc si l'on verse km fois A plein dans B, on obtiendra :

$$0 \xrightarrow{\mathbf{V_1}} a \bmod b \xrightarrow{\mathbf{V_1}} 2a \bmod b \xrightarrow{\mathbf{V_1}} \dots \xrightarrow{\mathbf{V_1}} kma \bmod b = kp = v$$

Donc v est atteignable.

5. Il ne s'agit pas seulement de prouver que les volumes atteignables sont multiples de p, mais aussi que **tous** les multiples sont effectivement atteints.

6. « L'arithmétique c'est comme l'amour, ça commence par un Bézout et ça finit par un Gauss. » http://fr.wikipedia.org/wiki/Théorème_de_Bachet-Bézout

7. En réalité le théorème donne $m, n \in \mathbb{Z}$ tels que $ma + nb = p$, mais ce corollaire est également vrai.

Extension. Que se passe-t-il dans le cas où a et b sont réels [8] ?
Félicitations à ordiclic pour sa réponse !

4 Relaxation

4.1 Énoncé

À l'issue de la finale, nous remplissons la cour de l'EPITA de mousse, de manière à relaxer nos candidats.

Pour cela, nous disposons de N bidons de volumes distincts triés dans l'ordre décroissant que nous pouvons remplir à loisir, et procédons de la manière suivante : nous remplissons la cour avec le premier bidon autant de fois qu'il est possible de le déverser intégralement sans dépasser le volume souhaité, auquel cas nous passons au deuxième et ainsi de suite. Comme le dernier bidon a toujours une capacité d'un litre, nous pouvons réaliser tous les volumes possibles.

Si vous parvenez à trouver un volume de mousse pour lequel vous serez à même de remplir la cour avec nos bidons en effectuant moins de versements que nous, vous gagnez. Sinon, vous perdez.

On vous donne les volumes des bidons, écrivez une fonction qui retourne 1 si vous pouvez gagner, 0 sinon.

4.2 Intérêt

Ce problème, qui est un rendu de monnaie déguisé, revient à déterminer pour un système de bidons donné si l'algorithme glouton [9] : « Tant qu'il reste un volume à verser, choisir le plus gros bidon ne dépassant pas le volume souhaité [10] » est optimal. Nous l'appellerons « algorithme de Prologin » dans toute la suite de la correction.

L'intérêt principal de ce problème réside dans le fait que le problème de rendu de monnaie est $\mathcal{NP}$-complet [11] ; en d'autres termes, il est difficile de dire si l'algorithme de Prologin est optimal pour un volume donné (« Est-ce que je peux gagner pour 27 L ? »). Mais s'il s'agit de savoir s'il existe **un** volume où l'algorithme de Prologin n'est pas optimal (« Est-ce qu'un volume me fait gagner ? »), il existe un algorithme plus simple. La raison intrinsèque est que s'il existe un volume pour lequel Prologin a perdu, alors le plus petit volume faisant perdre Prologin a une forme particulière simple à rechercher. Si cette forme est introuvable, c'est qu'un tel volume n'existe pas.

4.3 Solution

En 1994, David Pearson a prouvé [12] le théorème suivant [13] :

Théorème. Notons $c_1 > \ldots > c_n$ les volumes des bidons. S'il existe un volume qui fait perdre Prologin, alors si l'on considère le plus petit volume v_{min} faisant perdre Prologin, ainsi que la plus petite liste L (dans l'ordre lexicographique) des nombres de bidons de chaque volume utilisés pour faire perdre Prologin avec le volume v_{min} (L_i et L_j étant respectivement les première et dernière composantes non nulles) ; si $(0, \ldots, 0, m_i, m_{i+1}, \ldots, m_j, \ldots)$ représente la liste des nombres de bidons de chaque volume choisis par l'algorithme de Prologin pour faire $c_{i-1} - 1$ litres de mousse, alors :

$$L = (0, \ldots, 0, m_i, m_{i+1}, \ldots, m_j + 1, 0, \ldots, 0).$$

Or, l'algorithme de Prologin s'exécute en temps linéaire : pour chaque bidon, il suffit de calculer combien de fois on peut le déverser intégralement, puis passer au suivant et ainsi de suite. Nous avons donc accès au nombre minimal de versements (la somme des éléments de L) en temps linéaire.

8. Nous avons trouvé une merveilleuse réponse à cette question. Mais ce pied de page est trop étroit pour la contenir.
9. http://fr.wikipedia.org/wiki/Algorithme_glouton
10. Proverbe chinois.
11. http://fr.wikipedia.org/wiki/Problème_NP-complet
12. http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.57.3243
13. Enfin, presque.

Seulement, nous ne connaissons pas i et j, c'est pourquoi il faut tester toutes les paires $(i, j) \in \{1, \ldots, n\}$ telles que $i \leqslant j$:

Pour chaque i de 1 à $n-1$
    Pour chaque j de i à $n-1$
        Si L compte moins de versements que l'algorithme de PROLOGIN pour le même volume,
            On a trouvé un volume qui fait perdre Prologin.
Sinon, il n'existe pas de volume faisant perdre Prologin (sinon, on serait déjà tombé sur v_{min}).

Listing 6 – Une solution de l'exercice 4 en C

```c
int nbBidons;
int volumes[100000];
int versements[100000];

int prologin(int n, int j) {
  int i, s, nbVersements;
  s = n;
  nbVersements = 0;
  for(i = 0 ; i <= j ; i++) {
    versements[i] = s / volumes[i];
    nbVersements += versements[i];
    s %= volumes[i];
  }
  return nbVersements;
}

int volume(int n, int j) {
  int i, s, volume;
  s = n;
  volume = 0;
  for(i = 0 ; i <= j ; i++) {
    versements[i] = s / volumes[i];
    volume += versements[i] * volumes[i];
    s %= volumes[i];
  }
  return volume;
}

int puisJeGagner() {
  int i, j;
  for(i = 1 ; i < nbBidons ; i++)
    for(j = i ; j < nbBidons ; j++)
      if(prologin(volumes[i - 1] - 1, j) + 1
        < prologin(volume(volumes[i - 1] - 1, j) + volumes[j], nbBidons - 1))
        return 1;
  return 0;
}
```

Merci à tous d'avoir participé !

Si vous avez des questions, n'hésitez pas à nous contacter à l'adresse info@prologin.org.