



Olympiades d'Informatique

<http://uclouvain.acm-sc.be/olympiades>

Exemples de questions pour les secondaires



Ce document propose des exemples de questions pour le concours destiné aux élèves du secondaire. La première section donne des exemples de questions de logique tandis que la seconde propose des exemples de questions algorithmiques. Nous vous conseillons de lire le document « *Introduction à l'algorithmique* » pour comprendre les exemples de solution donnés.

1 Logique



1.1 Les sabliers

Au moyen de deux sabliers (un de 7 minutes et un de 4 minutes), vous devez mesurer 9 minutes. Vous pouvez d'abord manipuler les sabliers avant de commencer à compter les 9 minutes. On vous demande de trouver la solution qui prenne le moins de temps en tout.

On dispose de deux sabliers, permettant de mesurer 4 et 7 minutes. On va appeler le premier A et le second B . On doit pouvoir mesurer 9 minutes en tout, mais on ne doit pas commencer directement à compter, on peut d'abord effectuer un certain nombre d'opérations avec les sabliers. Essentiellement, une seule opération est possible : *retourner*. Par contre, on peut n'en retourner qu'un seul ou les deux à la fois. La question demande de trouver la solution qui prenne le moins de temps en tout. Voici une solution possible, chaque sablier est accompagné de deux chiffres x/y . Le chiffre x représente le temps qu'il reste à couler et y celui déjà écoulé.

Opération	Observation	 A	 B	
Retourner A et B		4/0	7/0	
	A est vide	0/4	3/4	(4 minutes écoulées)
Retourner A		4/0	3/4	début du chrono
	B est vide	1/3	0/7	(3 minutes écoulées)
Retourner A et B		3/1	7/0	
	A est vide	0/4	4/3	(3 minutes écoulées)
Retourner A et B		4/0	3/4	
	B est vide	1/3	0/7	(3 minutes écoulées)
				fin du chrono

Cette solution prend donc 13 minutes en tout, ce n'est pas la meilleure. Une autre solution est possible et ne prend qu'exactly 9 minutes, c'est la meilleure. Elle est reprise ci-dessous :

Opération	Observation	 A	 B	
Retourner A et B		4/0	7/0	
	A est vide	0/4	3/4	(4 minutes écoulées)
Retourner A		4/0	3/4	
	B est vide	1/3	0/7	(3 minutes écoulées)
Retourner B		1/3	7/0	
	A est vide	0/4	6/1	(1 minute écoulée)
Retourner B		0/4	1/6	
	B est vide	0/4	0/7	(1 minute écoulée)



2 Algorithmique

2.1 Le nombre de diviseurs

Étant donné un nombre entier $n > 0$, écrire un algorithme qui calcule le nombre de diviseurs entiers positifs que possède n .

On reçoit comme donnée un nombre entier strictement positif n . Il faut calculer le nombre de diviseurs entiers positifs que possède n . Il faut tout d'abord se rappeler de la définition de *diviseur*. Le nombre entier positif d est diviseur de n si d divise parfaitement n , c'est-à-dire qu'il existe un entier $k > 0$ tel que $n = k \times d$. Les diviseurs de n sont donc obligatoirement compris entre 1 et n . À partir de ce constat, on peut écrire un algorithme qui teste, pour chaque entier d compris entre 1 et n , si celui-ci divise n . Une solution en français peut donc s'écrire comme :

```
Initialiser un compteur cnt à zéro.  
Pour chaque entier  $d = 1, 2, \dots, n$  :  
    Si  $d$  divise  $n$ , alors :  
        Ajouter 1 à cnt.  
→ Le nombre  $n$  possède cnt diviseurs entiers positifs.
```

On peut également écrire une solution plus rigoureuse en utilisant du pseudo-code comme montré ci-dessous. L'opération $n \bmod d$ permet de connaître le reste de la division entière de n par d . Si ce dernier est égal à zéro, c'est que la division est exacte et que d est un diviseur de n .

Algorithme 1 : Le nombre de diviseurs.

Input : n , un nombre entier positif

Output : Le nombre de diviseurs entiers positifs de n

```
1   $cnt \leftarrow 0$   
2  for  $d \leftarrow 1$  to  $n$  do  
3      if  $n \bmod d = 0$  then  
4           $cnt \leftarrow cnt + 1$   
5  return  $cnt$ 
```

2.2 Le tapis de cartes

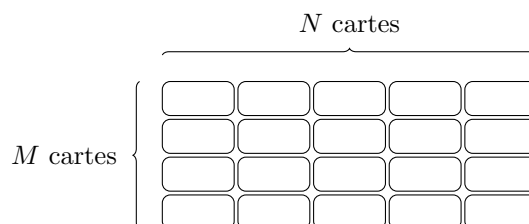
Alice et Bob disposent d'un très grand nombre (on le suppose illimité) de cartes à jouer rectangulaires, toutes de dimensions identiques. Ils désirent les disposer côte-à-côte, dans le même sens et sans les découper, afin de former un carré.

Aidez-les à trouver quelles sont les dimensions minimales de ce carré. Si L est la largeur d'une carte et H sa hauteur (toutes deux des valeurs entières strictement positives), on vous demande d'écrire un algorithme pour calculer X , le nombre de cartes nécessaires pour former le carré.

La première chose à faire est de bien comprendre le problème. On va donc disposer des cartes rectangulaires côte à côte, dans le même sens, pour former un tapis de cartes. Connaissant la



largeur et hauteur d'une carte, on souhaite trouver le nombre de cartes qu'on va devoir utiliser pour former un carré de dimension minimale. Voici ce que ça donne en dessin si on place les cartes horizontalement :



La hauteur du tapis vaut donc $M \times L$ et sa largeur $N \times H$. On souhaite que le tapis soit carré et donc que $M \times L = N \times H$ et ce qu'on cherche, c'est le nombre minimal de cartes nécessaires, c'est-à-dire $M \times N$. Pour réussir à résoudre ce problème, il faut se rendre compte que pour trouver la solution, il suffit de calculer le plus petit commun multiple (PPCM) entre L et H . On trouve ensuite :

$$M = \frac{P}{H} \quad \text{et} \quad N = \frac{P}{L} \quad \text{avec } P = \text{PPCM}(L, H)$$

Le problème revient donc à trouver le PPCM entre deux nombres entiers strictement positifs L et H . Une façon de faire consiste à parcourir tous les multiples de L et tester s'ils sont également multiple de H . On garde bien entendu le plus petit. On peut écrire l'algorithme en français :

Pour chaque entier $m = 1, 2, \dots, H$:
Si $L \times m$ est un multiple de H , alors :
→ $L \times m$ est le PPCM de L et H .

$L \times m$ est multiple de H revient à dire que H divise $L \times m$. On peut dès écrire un algorithme en pseudo-code en utilisant l'opérateur mod qu'on a vu à la question précédente. On ne teste que les entiers de 1 à H , en effet, on sait bien qu'au pire le PPCM entre L et H vaut exactement $L \times H$ (ceci se produisant lorsque L et H n'ont aucun diviseurs communs).

Algorithme 2 : Le tapis de cartes.

Input : L et H , nombres entiers strictement positifs, largeur et hauteur d'une carte

Output : Nombre de cartes nécessaires pour former un tapis carré de côté de longueur minimale en plaçant des cartes côte à côte dans le même sens

```
1 // Calcul du PPCM entre L et H
2 ppcm ← 1
3 found ← false
4 while not found and ppcm ≤ H do
5     if L × ppcm mod H = 0 then
6         found ← true
7         ppcm ← ppcm + 1
8 // Calcul du nombre de cartes
9 M ← ppcm/H
10 N ← ppcm/L
11 return M × N
```



2.3 La gestion d'un hangar de stockage

Vous êtes en charge de la gestion du hangar de stockage d'une grande surface pouvant stocker X produits. Tous les jours, un camion vient, apporte Y nouveaux produits et enlève les Y produits les plus anciens du stock. Les produits livrés par le camion sont toujours considérés comme neufs. Les produits du stock et ceux du camion sont représentés par des listes $stock = \langle s_1, s_2, \dots, s_X \rangle$ et $camion = \langle c_1, c_2, \dots, c_Y \rangle$.

- (a) Écrivez un algorithme qui prend en entrée l'état du stock et le contenu du camion et renvoie en sortie l'état du stock en fin de journée.
- (b) Étendez l'algorithme précédent afin de gérer le cas où le camion apporte des produits d'âges différents. Vous pouvez connaître l'âge d'un produit a en utilisant la fonction $age(a)$.
- (c) Chaque soir, un ouvrier vérifie que les produits ne sont pas trop vieux pour être vendus. Un produit âgé de plus de 126 jours (au moment de la vérification) est considéré comme trop vieux et doit être détruit. Écrivez une fonction qui prend en entrée l'état du stock et renvoie l'état du stock après vérification et éventuelle destruction de certains produits.