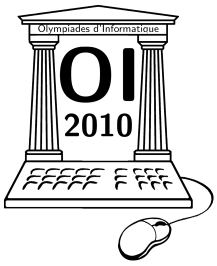


Olympiades d'Informatique

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

Introduction à l'algorithmique

Ce document propose une introduction à l'algorithmique et la résolution de problème. La première partie décrit pas-à-pas les concepts de base de la programmation, tout en introduisant un pseudo-code, une notation pour les différentes structures de programmation étudiées. Tout ceci mènera à la seconde partie où il sera question d'algorithmique et résolution de problèmes. Le document est rédigé dans un style pédagogique, les concepts étant introduit graduellement et à chaque fois sur base d'exemples concrets. Il s'agit d'un document préliminaire, toute remarque est bienvenue et peut être adressée à Sébastien Combéfis à l'adresse suivante : sebastien.combefis@uclouvain.acm-sc.be.

1 Introduction

Commençons avec un problème très simple dont voici l'énoncé :

> Je désire carreler une pièce rectangulaire qui mesure 2,5 m de large sur 4,9 m de long. Quelle est la surface, en m^2, que je vais devoir carreler ?

La première chose à faire est bien entendu de lire attentivement l'énoncé du problème à résoudre. Pendant cette lecture, il faut tout d'abord identifier deux choses : les *entrées*, c'est-à-dire les données qui sont fournies et les *sorties*, c'est-à-dire les résultats qu'il faut calculer. Pour le problème en cours de résolution, on peut identifier deux données : une largeur de 2,5 m et une longueur de 4,9 m. Le résultat qu'il faut calculer est la surface de la pièce de largeur et longueur fournis en données.

Une fois les entrées et sorties identifiées, il faut décrire comment le résultat désiré est calculé à partir de ceux-ci. Il s'agit de proposer une *méthode de résolution systématique* du problème. Dans notre cas, on doit calculer une surface d'un rectangle connaissant sa largeur et longueur, il suffit dès lors de multiplier ces deux valeurs pour obtenir le résultat voulu : $2{,}5\,\text{m} \times 4{,}9\,\text{m} = 12{,}25\,\text{m}^2$.

Le problème qu'on vient d'analyser est très *spécifique*, c'est-à-dire qu'il correspond exactement à une seule situation précise qui est de carreler une pièce rectangulaire mesurant 2,5 m sur 4,9 m. On ne s'intéresse que très rarement à ce genre de problème, mais plutôt à des problèmes plus *génériques*, c'est-à-dire plus généraux et qui peuvent correspondre à plusieurs situations concrètes.

Voyons une version plus générique du problème de carrelage :

> Je désire carreler une pièce rectangulaire d'une certaine largeur de l et d'une certaine longueur de L, toutes deux données en mètres. Quelle est la surface, en m^2, que je vais devoir carreler ?

On voit que cette fois-ci, il n'y a plus des valeurs précises pour les données du programme. Il s'agit en fait de *variables* pouvant correspondre à n'importe quelle valeur. Il y a deux variables l et L. Le résultat du problème se calcule simplement en multipliant ces deux valeurs : $l \times L$. On va pouvoir écrire la solution à ce problème un peu plus systématiquement grâce à un *algorithme*.

Algorithme 1 : Calcul de la surface de carrelage d'une pièce rectangulaire.

Input : l et L, largeur et longueur de la pièce en m
Output : Surface de la pièce rectangulaire

```
1 return l × L
```

On peut voir l'utilisation de trois mots en gras dans l'algorithme. Chacun de ces mots permet de définir différents éléments de l'algorithme. Le tableau ci-dessous reprend ces différents mots avec l'élément qu'ils définissent.

Mot	définit
Input	les entrées
Output	les sorties
return	le résultat

Ce problème générique peut être utilisé pour résoudre le premier problème qu'on a vu. Il suffit pour cela de donner la valeur 2,5 à l et la valeur 4,9 à L. En faisant ça, on a *instancié* le problème générique pour en obtenir un spécifique en donnant des valeurs concrètes aux variables d'entrée. Le résultat calculé par l'algorithme sera dans ce cas également une valeur concrète, résultat du problème spécifique concret.

1.1 Exercices

1. Partez du problème spécifique suivant et donnez le problème générique lui correspondant. Enfin, écrivez l'algorithme complet qui permet de résoudre le problème.

   > On m'a appris un nouveau jeu de cartes dans lequel chaque joueur reçoit initialement une main de quatre cartes. Pour ce jeu, on utilise un jeu classique de 52 cartes. Combien de mains différentes pourrais-je recevoir initialement ?

2. L'algorithme suivant permet de calculer le volume d'un cylindre. Complétez-le.

Algorithme 2 : Calcul du volume d'un cylindre.

Input : R et H, rayon et hauteur du cylindre en m
Output : Volume du cylindre

```
1 // À compléter ...
```

2 Variable

Maintenant qu'on a vu les concepts de base de la résolution d'un problème grâce à un algorithme, on va pouvoir découvrir petit à petit différentes constructions qui vont nous permettre de résoudre des problèmes plus intéressants. Lorsqu'on fait des calculs, on a parfois besoin d'effectuer des calculs intermédiaires. Par exemple, si on doit calculer les racines d'une équation du second degré $P \equiv ax^2 + bx + c$, en supposant qu'on soit dans le cas où l'équation admet deux racines réelles distinctes, on fait le calcul en deux étapes. On commence par calculer le déterminant

$$\rho = b^2 - 4ac$$

et on calcule ensuite les deux racines

$$x_1 = \frac{-b + \sqrt{\rho}}{2a} \qquad \text{et} \qquad x_2 = \frac{-b - \sqrt{\rho}}{2a}.$$

Voici l'algorithme qui permet de résoudre ce problème. Remarquez que l'algorithme produit deux résultats, ce qu'on a écrit en écrivant les deux valeurs séparées par une virgule et écrites entre des crochets. On va revenir sur ce concept rapidement.

Algorithme 3 : Calcul des racines d'un polynôme du second degré.

Input : a, b et c, paramètres du polynôme $P \equiv ax^2 + bx + c$
Output : Les deux racines réelles distinctes du polynôme

1 $\rho \leftarrow b^2 - 4ac$
2 $x_1 \leftarrow \dfrac{-b + \sqrt{\rho}}{2a}$
3 $x_2 \leftarrow \dfrac{-b - \sqrt{\rho}}{2a}$
4 **return** $[x_1, x_2]$

L'algorithme nécessite trois entrées qui sont les variables a, b et c. Comme vous avez pu le lire dans l'algorithme, on les appelle *paramètres*, pour les distinguer des autres variables qu'on va pouvoir utiliser dans l'algorithme. On en a utilisées trois : ρ, x_1 et x_2. Les *variables* possèdent un *nom* et une *valeur* et on les utilise notamment pour effectuer des calculs intermédiaires. On peut les utiliser dans un calcul et on peut modifier leur valeur. On note $x \leftarrow v$ pour changer la valeur de la variable x en v. C'est donc bien ce qu'on fait dans l'algorithme ci-dessus. On calcule le déterminant $(b^2 - 4ac)$ et on stocke ce résultat dans la variable ρ. Ensuite, on va pouvoir utiliser ce résultat intermédiaire pour calculer les deux racines et les stocker dans les variables x_1 et x_2. Enfin, on termine en renvoyant le résultat qui est composé des valeurs de ces variables ($[x_1, x_2]$).

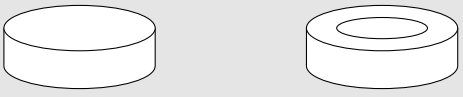
2.1 Expression

Lorsqu'on écrit un calcul, on écrit en fait une *expression*. Il s'agit donc d'un calcul qui utilise des nombres, des variables et des opérateurs et qui possède une certaine valeur, le résultat du calcul. On a déjà vu un type d'expression, à savoir les *expressions arithmétiques* dont la valeur est toujours un nombre. On a par exemple : $l \times L$, b^2, $\sqrt{x+1}$, ... On va donc combiner des nombres, des variables et des opérateurs pour construire ces expressions. On peut par exemple écrire : $x + y$, $x - y$, $x \times y$, x/y, $\sqrt{x}$, x^n, ...

Il y a également un second type d'expression, à savoir les *expressions booléennes*. Ces dernières peuvent prendre deux valeur différentes : elles peuvent être vraies (noté **true**) ou fausse (noté **false**). On obtient de telles valeurs en utilisant les opérateurs $=$, $\neq$, $<$, $>$, $\leq$ et $\geq$ qui vont permettre de comparer deux valeurs. On peut par exemple écrire : $x \neq 0$, $b^2 - 4ac > 0$, $x \neq y$, ...

Voyons un nouveau problème qui va nous permettre d'utiliser ces nouveaux concepts :

J'ai deux types de pièces cylindriques de même hauteur H à ma disposition. Les cylindres de premier type sont pleins tandis que ceux du deuxième type sont creux comme vous pouvez le voir sur le dessin ci-dessous. Un cylindre de type I de rayon R est-il plus volumineux qu'un cylindre de type II de rayon extérieur R_O et de rayon intérieur R_I ?



Voyons maintenant comment résoudre ce problème. Il y a quatre entrées : H la hauteur des cylindres, R le rayon du cylindre de type I et R_O et R_I les rayons extérieur et intérieur du cylindre de type II. Le résultat à produire est booléen, c'est oui (**true**) ou non (**false**). Vous avez dû retrouver la formule permettant d'obtenir le volume d'un cylindre à l'exercice 1.2. Il s'agit en effet de $V = \pi R^2 H$. L'algorithme va donc être assez simple, on calcule de volume des deux cylindres et on les compare. Pour le second cylindre, il faudra calculer le volume du cylindre extérieur et lui soustraire celui du cylindre intérieur. Voici un algorithme qui va résoudre ce problème :

Algorithme 4 : Comparaison du volume de cylindres de différents types.

Input : H, hauteur des cylindres, R rayon du cylindre de type I, R_O, R_I rayons extérieur et intérieur du cylindre de type II
Output : **true** si le cylindre de type I est plus volumineux que celui de type II et **false** sinon

```
1 // Volume du premier cylindre
2 V_1 ← πR²H
3 // Volume du second cylindre
4 V_O ← πR_O²H
5 V_I ← πR_I²H
6 V_2 ← V_O − V_I
7 return V_1 ≥ V_2
```

On a donc calculé les volumes des deux cylindres dans les variables V_1 et V_2. Pour le second cylindre, on a fait le calcul en plusieurs étapes en stockant des résultats intermédiaires dans les variables V_O et V_I. Enfin, le résultat est un booléen qui est $V_1 \geq V_2$ (le volume du premier cylindre est-il plus grand que celui du second).

2.2 Division entière et reste

Lorsqu'on travaille uniquement avec des nombres entiers, la division se comporte différemment. En effet, on parle de *division entière* : le quotient de a par b est un certain entier q, résultat de cette division entière. Par exemple, 12 divisé par 3 a pour quotient 4. On va noter la division entière avec : « div ». La division ne tombe pas toujours juste. En effet, si b n'est pas un diviseur de a (ou si a n'est pas divisible par b), il y a un reste r. On peut vouloir calculer ce reste et on va le noter avec « mod ». Par exemple, le reste de la division de 12 par 7 est 4. En effet, on peut réécrire 12 comme $1 \times 7 + 4$. De manière générale, on peut écrire :

$$a = q \times b + r$$

avec q le quotient de la division entière de a par b et r le reste de cette même division. Voyons tout de suite un exemple concret qui utilise cette notion. Écrivons un algorithme qui teste si un nombre entier est pair. L'algorithme doit calculer une valeur booléenne qui va être **true** si le nombre est pair et **false** sinon. Pour savoir cela, il suffit de calculer le reste de la division par 2. Ce dernier sera en effet nul si le nombre est pair, et non-nul dans le cas contraire.

Algorithme 5 : Test de la parité d'un nombre naturel.

Input : n, un nombre entier positif
Output : **true** si n est pair et **false** sinon

```
1 return n mod 2 = 0
```

2.3 Exercices

1. Écrivez un algorithme qui prend deux nombres naturels a et b en entrée et qui doit tester si a est un multiple de b. Le résultat de l'algorithme doit être **true** si a est un multiple de b et **false** sinon.

2. Écrivez un algorithme qui calcule le volume compris entre un cylindre de rayon R et de hauteur H et le parallélépipède rectangle minimal qui entoure ce cylindre, c'est-à-dire un parallélépipède de longueur et largeur $2R$ et de hauteur H.

3. J'ai une bouteille avec une capacité de 1,5 l et une tasse d'une capacité de 210 ml. Écrivez un algorithme qui calcule combien de tasses remplies complètement il est possible de verser dans la bouteille.

4. Écrivez un algorithme qui effectue la conversion de degrés Fahrenheit en degrés Celsius. Pour rappel, les températures sont liées par $t_C = \frac{5}{9}(t_F - 32)$.

3 Condition

Dans un algorithme, on est parfois amené à faire des choix. On va voir une structure qui permet de faire de tels choix, ceux-ci étant fait sur base d'une *condition* qui est une expression booléenne. Pour comprendre cela, écrivons un algorithme qui permet de calculer la valeur absolue d'un nombre n :

Algorithme 6 : Calcul de la valeur absolue d'un nombre.

Input : n, un nombre
Output : La valeur absolue du nombre

```
1 if n < 0 then
2     n ← (−1) × n
3 return n
```

Pour calculer la valeur absolue d'un nombre n, si celui-ci est négatif, il faut le multiplier par -1 pour avoir sa valeur absolue. C'est précisément ce qu'on fait des lignes 1 à 2. par contre, dans l'autre cas (le nombre est positif), il n'y a rien à faire. Pour exprimer une condition, on utilise donc le mot **if** suivi d'un condition et du mot **then**. Vient ensuite tout ce qu'il faut faire si la condition est satisfaite.

Voyons maintenant la construction qui permet de faire une action dans le cas où la condition est satisfaite et une autre action si elle n'est pas satisfaite. On va cette fois-ci écrire un algorithme qui permet de comparer deux nombres et d'obtenir le plus grand des deux.

Algorithme 7 : Calcul du maximum de deux nombres.

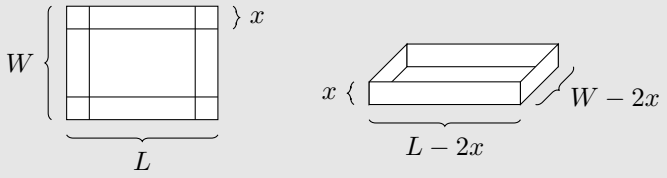
Input : a, b, deux nombres
Output : Le plus grand nombre entre a et b

```
1 if a > b then
2     max ← a
3 else
4     max ← b
5 return max
```

Le **if** teste donc si la valeur du nombre a est plus grande que celle de b. Si c'est bien le cas, on stocke la valeur de a dans max. Sinon, ce n'est pas le cas et on indique cela avec **else**. Dans ce cas, la valeur de b est plus grande ou égale à celle de a et c'est donc b qu'on stocke dans max. Ainsi, à la fin, max contient la plus grande valeur entre a et b, ce qui est bien ce qu'on devait calculer.

Voyons tout de suite un nouveau problème qui va nous permettre d'exploiter l'instruction **if-else** qu'on vient de voir :

J'ai une feuille de papier de longueur L et de largeur W. À partir de celle-ci, on peut construire un récipient d'une certaine hauteur x. Quelle est la hauteur x qui permet d'obtenir un récipient de volume maximal ?



Le volume du parallélépipède peut être calculé comme suit :

$$V = x(L-2x)(W-2x) = 4x^3 - 2(L+W)x^2 + LWx$$

Pour trouver le maximum de cette fonction, il faut calculer sa dérivée :

$$V' = 12x^2 - 4(L+W)x + LW$$

Et on cherche ensuite les points où la fonction s'annule, c'est-à-dire les racines de l'équation. On va donc procéder comme on a fait à la section 2, si ce n'est qu'on va devoir distinguer les différents cas pour la valeur du déterminant. En effet, soit le déterminant est nul et l'équation n'admet qu'une seule racine réelle ou alors le déterminant est strictement positif et l'équation admet deux racines réelles. Dans ce cas, il faudra prendre celle qui donne le plus grand volume. On a :

$$\rho = \big(-4(L+W)\big)^2 - 4 \cdot 12 \cdot LW = 16(L^2+W^2+2LW) - 48LW = 16(L^2+W^2-LW)$$

Les racines de l'équation $V' = 0$ sont donc données par (dans le cas où $\rho > 0$) :

$$x_1, x_2 = \frac{4(L+W) \pm 4\sqrt{L^2+W^2-LW}}{24} = \frac{(L+W) \pm \sqrt{L^2+W^2-LW}}{6}$$

Si $\rho = 0$, l'équation n'admet qu'une seule racine réelle qui est donnée par :

$$x = \frac{-\big(-4(L+W)\big)}{24} = \frac{L+W}{6}$$

Enfin, si $\rho < 0$, l'équation n'admet aucune racine réelle. Voyons maintenant un algorithme pour résoudre ce problème.

Algorithme 8 : Hauteur optimale pour une boite construite à partir d'une feuille de papier.

Input : W, L, deux nombres strictement positifs, largeur et longueur d'une feuille de papier
Output : x, la hauteur de la boite ouverte de telle sorte que son volume soit maximal

```
1  // Calcul du déterminant
2  ρ = 16(L² + W² − LW)
3  // On traite les différents cas pour ρ
4  // ρ < 0, pas de racines, l'algorithme renvoie 0
5  if ρ < 0 then
6      x ← 0
7  // ρ = 0, une seule solution donnée par −b/2a
8  else if ρ = 0 then
9      x ← (L + W)/6
10 // ρ > 0, deux solutions (−b ± √ρ)/2a
11 // Il faut choisir celle qui donne le plus grand volume, c'est-à-dire la plus grande des deux
12 else
13     rac ← √(L² + W² − LW)
14     x1 ← (L + W + rac)/6
15     x2 ← (L + W − rac)/6
16     // On prend le plus grand entre x1 et x2
17     if x1 > x2 then
18         x ← x1
19     else
20         x ← x2
21 return x
```

Il y a plusieurs remarques à faire à propos de cet algorithme. On remarque tout d'abord qu'il y a trois cas à distinguer, selon la valeur de ρ. Pour ce faire, on va pouvoir enchainer plusieurs **if-else** et on se retrouve avec la structure suivante : **if** (pour $\rho < 0$), **else if** (pour $\rho = 0$) et **else** (pour $\rho < 0$). On va pouvoir utiliser la même structure lorsqu'on a a plusieurs cas à distinguer. Lorsqu'il n'y a qu'un seul cas, on se contentera d'un simple **if**, et lorsqu'il n'y a que deux cas, un **if-else** suffira.

La deuxième remarque concerne le cas $\rho > 0$. On calcule les deux racines, et on doit ensuite trouver la plus grande des deux. Vous aurez surement remarqué qu'il s'agit exactement de ce qu'on a fait dans l'algorithme 7 de la page 5. On verra plus loin qu'on peut exploiter cela pour éviter de réécrire des choses déjà existantes.

Enfin, la dernière remarque concerne le calcul de x_1 et x_2. Pour rendre l'algorithme plus lisible, on a d'abord calculé la valeur de $\sqrt{L^2 + W^2 - LW}$ qu'on a stocké dans la variable *rac*. Ceci permet d'éviter de recopier deux fois la même chose et de rendre le code plus embrouillé. C'est une assez bonne pratique.

3.1 Algèbre de Boole

Les conditions qu'on a utilisées jusqu'à présent étaient toutes *simples*. Elles sont composées d'un opérateur de comparaison et de deux opérandes. Les opérateurs qu'on peut utiliser sont : $=$, $\neq$, $<$, $>$, $\leq$ et $\geq$. On va maintenant voir qu'il est possible de combiner plusieurs conditions simples avec des *opérateurs logiques*. Imaginons par exemple qu'on souhaite tester si la valeur de x est comprise entre 12 et 20. On va écrire : ($x \geq 12$ **and** $x \leq 20$). L'opérateur logique **and** permet donc

de combiner deux conditions. La valeur d'une condition A **and** B est **true** si A et B valent tous les deux **true**, elle vaudra **false** dans tous les autres cas. L'opérateur logique **and** permet donc de tester que deux conditions soient satisfaites.

Il y a également l'opérateur **or** qui permet de tester si une condition ou une autre est satisfaite. La valeur d'une condition A **or** B est **true** si A ou B ou les deux valent **true** et elle vaudra **false** lorsque A et B valent tous les deux **false**.

Enfin, il reste l'opérateur **not** qui s'applique à une condition. La valeur d'une condition **not** A est **true** si A vaut **false** et elle vaut **false** dans l'autre cas.

Le tableau ci-dessous reprend les trois opérateurs **and**, **or** et **not** avec leurs valeurs lorsqu'ils sont appliqués aux conditions A et B.

A	B	**not** A	A **and** B	A **or** B
false	**false**	**true**	**false**	**false**
false	**true**	**true**	**false**	**true**
true	**false**	**false**	**false**	**true**
true	**true**	**false**	**true**	**true**

3.2 Exercices

1. Écrivez un algorithme qui calcule la valeur de la fonction $f(x) = |x^2 - 2x - 10|$ pour un x donné.

2. Écrivez un algorithme qui transforme une note en points sur 20 en une lettre selon la table de correspondance suivante : strictement en-dessous de 8 : F, entre 8 et 10 : D, entre 10 et 13 : C, entre 14 et 16 : B et entre 17 et 20 : A.
   (Pour afficher un texte, vous pouvez écrire : **print** "texte à afficher")

4 Boucle

On vient de voir qu'on pouvait effectuer des tests et n'exécuter une partie d'un algorithme que si une certaine condition était vérifiée. On va maintenant voir une autre structure très importante, à savoir les *boucles*. Cette dernière permet de répéter plusieurs fois une partie d'algorithme. Partons d'un exemple pour comprendre le principe, écrivons un algorithme pour calculer la somme $1 + 2 + \cdots + n$, sans utiliser la formule $n(n+1)/2$.

Algorithme 9 : Calcul de la somme des n premiers entiers positifs.

Input : n, un nombre naturel
Output : La somme $1 + 2 + \cdots + n$

```
1 i ← 1
2 sum ← 0
3 while i ≤ n do
4 |   sum ← sum + i
5 |   i ← i + 1
6 return sum
```

Voyons maintenant comment cet algorithme fonctionne. On commence par définir deux variables i et sum. La variable sum va nous permettre de calculer la somme et la variable i va stocker successivement les termes qu'il faut additionner, à savoir $1, 2, \cdots, n$.

On écrit ensuite **while** suivi d'une condition suivie de **do**. Tout ce qui vient ensuite va être répété successivement tant que la condition du **while** est satisfaite, c'est-à-dire que sa valeur est **true**. Et donc, tant que i est plus petit ou égal à n, on ajoute la valeur de i à la variable sum et on augmente la valeur de i de 1. Prenons un exemple pour comprendre le principe de la boucle. Soit n qui vaut 3. La variable i a pour valeur initiale 1 et la boucle va se répéter tant que $i \leq 3$. Sachant que la valeur de i est augmentée de 1 à chaque boucle, cette dernière va se répéter 3 fois : pour i valant 1, 2 et 3. Si on écrit tout ce qui va se passer, sans utiliser la boucle, ça donnerait :

$i \leftarrow 1$
$sum \leftarrow 0$
// Maintenant, i vaut 1 et donc, $i \leq 3$.
$sum \leftarrow sum + i$
$i \leftarrow i + 1$
// Maintenant, i vaut 2 et donc, $i \leq 3$.
$sum \leftarrow sum + i$
$i \leftarrow i + 1$
// Maintenant, i vaut 3 et donc, $i \leq 3$.
$sum \leftarrow sum + i$
$i \leftarrow i + 1$
// Maintenant, i vaut 4 et donc, la boucle s'arrête.
// La variable sum contient $1 + 2 + 3$, c'est-à-dire 6.

Voyons maintenant un exemple de problème qu'on va pouvoir résoudre à l'aide d'une boucle.

> Un nombre entier positif n est premier s'il admet exactement deux diviseurs (qui sont 1 et n). Écrivez un algorithme qui teste si un nombre donnée n est premier ou non.

Un algorithme simple pour résoudre le problème consiste à tester tous les entiers compris entre 1 et n. On compte ensuite le nombre de ceux-ci qui sont des diviseurs de n (avec le reste de la division). Le nombre n est premier s'il admet exactement deux diviseurs.

Algorithme 10 : Teste si un nombre est premier.

Input : $n > 0$, un nombre naturel
Output : **true** si n est premier et **false** sinon

```
1  i ← 1
2  cnt ← 0
3  // On teste tous les i compris entre 0 et n
4  while i ≤ n do
5      // Si i est un diviseur de n, on augmente la valeur de cnt de 1
6      if n mod i = 0 then
7          cnt ← cnt + 1
8      // On passe au i suivant
9      i ← i + 1
10 // n est premier s'il admet exactement 2 diviseurs
11 return cnt = 2
```