

Introduction aux algorithmes I

Algorithmes et structures de données, 2025-2026

P. Albuquerque (B410) et O. Malaspinas (A401), ISC, HEPIA
2025-09-16

En partie inspiré des supports de cours de P. Albuquerque

Qu'est-ce qu'un algorithme ?

Définition informelle (recette)

- des entrées (les ingrédients, le matériel utilisé) ;
- des instructions élémentaires simples (faire, flamber, etc.), dont les exécutions dans un ordre précis amènent au résultat voulu ;
- un résultat : le plat préparé.

Qu'est-ce qu'un algorithme ?

Définition informelle (recette)

- des entrées (les ingrédients, le matériel utilisé) ;
- des instructions élémentaires simples (frire, flamber, etc.), dont les exécutions dans un ordre précis amènent au résultat voulu ;
- un résultat : le plat préparé.

Histoire et étymologie

- Existent depuis 4500 ans au moins (algorithme de division, crible d'Eratosthène).
- Le mot algorithme est dérivé du nom du mathématicien perse *Muḥammad ibn Musā al-Khwārizmī*, qui a été “latinisé” comme *Algoritmi*.

Qu'est-ce qu'un algorithme ?

Définition informelle (recette)

- des entrées (les ingrédients, le matériel utilisé) ;
- des instructions élémentaires simples (frire, flamber, etc.), dont les exécutions dans un ordre précis amènent au résultat voulu ;
- un résultat : le plat préparé.

Histoire et étymologie

- Existent depuis 4500 ans au moins (algorithme de division, crible d'Eratosthène).
- Le mot algorithme est dérivé du nom du mathématicien perse *Muḥammad ibn Musā al-Khwārizmī*, qui a été “latinisé” comme *Algoritmi*.

Définition formelle

En partant d'un état initial et d'entrées (peut-être vides), une séquence finie d'instruction bien définies (ordonnées) implémentables sur un ordinateur, afin de résoudre typiquement une classe de problèmes ou

Variable

Notions de base d'algorithmique

Variable

- Paire : identifiant - valeur (assignation) ;

Séquence d'instructions / expressions

Variable

- Paire : identifiant - valeur (assignation) ;

Séquence d'instructions / expressions

- Opérateurs (arithmétiques / booléens)
- Boucles ;
- Structures de contrôle ;
- Fonctions ;

Algorithme de vérification qu'un nombre est premier (1/3)

Nombre premier : nombre possédant deux diviseurs entiers distincts.

Algorithme de vérification qu'un nombre est premier (1/3)

Nombre premier : nombre possédant deux diviseurs entiers distincts.

Algorithme naïf (problème)

```
booléen est_premier(nombre)
  si
    pour tout i, t.q.  $1 < i < \text{nombre}$ 
      i ne divise pas nombre
  alors vrai
  sinon faux
```

Algorithme de vérification qu'un nombre est premier (1/3)

Nombre premier : nombre possédant deux diviseurs entiers distincts.

Algorithme naïf (problème)

```
booléen est_premier(nombre)
  si
    pour tout i, t.q.  $1 < i < \text{nombre}$ 
      i ne divise pas nombre
  alors vrai
  sinon faux
```

Pas vraiment un algorithme : pas une séquence ordonnée et bien définie

Algorithme de vérification qu'un nombre est premier (1/3)

Nombre premier : nombre possédant deux diviseurs entiers distincts.

Algorithme naïf (problème)

```
booléen est_premier(nombre)
  si
    pour tout i, t.q.  $1 < i < \text{nombre}$ 
      i ne divise pas nombre
  alors vrai
  sinon faux
```

Pas vraiment un algorithme : pas une séquence ordonnée et bien définie

Problème : Comment écrire ça sous une forme algorithmique ?

Algorithme de vérification qu'un nombre est premier (2/3)

Algorithme naïf (une solution)

```
booléen est_premier(nombre) // fonction
    soit i = 2           // variable, type, assignation
    tant que i < nombre // boucle
        si nombre modulo i == 0 // expression typée
            retourne faux // expression typée
        i = i + 1
    retourne vrai // expression typée
```

Algorithme de vérification qu'un nombre est premier (3/3)

Algorithme naïf (une solution en C)

```
bool est_premier(int nombre) {  
    int i; // i est un entier  
    i = 2; // assignation i à 2  
    while (i < nombre) { // boucle avec condition  
        if (0 == nombre % i) { // is i divise nombre  
            return false; // i n'est pas premier  
        }  
        i = i + 1; // sinon on incrémente i  
    }  
    return true;  
}
```

Algorithme de vérification qu'un nombre est premier (3/3)

Algorithme naïf (une solution en C)

```
bool est_premier(int nombre) {  
    int i; // i est un entier  
    i = 2; // assignation i à 2  
    while (i < nombre) { // boucle avec condition  
        if (0 == nombre % i) { // is i divise nombre  
            return false; // i n'est pas premier  
        }  
        i = i + 1; // sinon on incrémente i  
    }  
    return true;  
}
```

Exercice : Comment faire plus rapide ?

Génération d'un exécutable

- Pour pouvoir être exécuté un code C doit être d'abord compilé (avec gcc ou clang).
- Pour un code prog.c la compilation "minimale" est

```
$ gcc prog.c  
$ ./a.out # exécutable par défaut
```

- Il existe une multitude d'options de compilation :

```
$ gcc -O1 -std=c11 -Wall -Wextra -g prog.c -o prog  
-fsanitize=address
```

1. -std=c11 utilisation de C11.
2. -Wall et -Wextra activation des warnings.
3. -fsanitize=... contrôles d'erreurs à l'exécution (coût en performance).
4. -g symboles de débogages sont gardés.
5. -o définit le fichier exécutable à produire en sortie.
6. -O1, -O2, -O3 : activation de divers degrés d'optimisation

La simplicité de C ?

32 mots-clé et c'est tout

<code>auto</code>	<code>double</code>	<code>int</code>	<code>struct</code>
<code>break</code>	<code>else</code>	<code>long</code>	<code>switch</code>
<code>case</code>	<code>enum</code>	<code>register</code>	<code>typedef</code>
<code>char</code>	<code>extern</code>	<code>return</code>	<code>union</code>
<code>const</code>	<code>float</code>	<code>short</code>	<code>unsigned</code>
<code>continue</code>	<code>for</code>	<code>signed</code>	<code>void</code>
<code>default</code>	<code>goto</code>	<code>sizeof</code>	<code>volatile</code>
<code>do</code>	<code>if</code>	<code>static</code>	<code>while</code>

Déclaration et typage

En C lorsqu'on veut utiliser une variable (ou une constante), on doit déclarer son type

```
const double two = 2.0; // déclaration et init.  
int x; // déclaration (instruction)  
char c; // déclaration (instruction)  
x = 1; // affectation (expression)  
c = 'a'; // affectation (expression)  
int y = x; // déclaration et initialisation en même temps  
int a, b, c; // déclarations multiples  
a = b = c = 1; // init. multiples
```

Les variables (1/2)

Variables et portée

- Une variable est un identifiant, qui peut être liée à une valeur (un expression).
- Une variable a une **portée** qui définit où elle est *visible* (où elle peut être accédée).
- La portée est **globale** ou **locale**.
- Une variable est **globale** est accessible à tout endroit d'un programme et doit être déclarée en dehors de toute fonction.
- Une variable est **locale** lorsqu'elle est déclarée dans un **bloc**, `{...}`.
- Une variable est dans la portée **après** avoir été déclarée.

Les variables (2/2)

Exemple

```
float max; // variable globale accessible partout
int foo() {
    // max est visible ici
    float a = max; // valide
    // par contre les variables du main() ne sont pas visibles
}

int main() {
    // max est visible ici
    int x = 1; // x est locale à main
    {
        // x est visible ici, y pas encore
        // on peut par exemple pas faire x = y;
        int y = 2;
    } // y est détruite à la sortie du bloc
} // x est à la sortie de main
```

Quiz : compile ou compile pas ?

Quiz : compile ou compile pas

Types de base (1/4)

Numériques

Type	Signification (gcc pour x86-64)
<code>char, unsigned char</code>	Entier signé/non-signé 8-bit
<code>short, unsigned short</code>	Entier signé/non-signé 16-bit
<code>int, unsigned int</code>	Entier signé/non-signé 32-bit
<code>long, unsigned long</code>	Entier signé/non-signé 64-bit
<code>float</code>	Nombre à virgule flottante, simple précision
<code>double</code>	Nombre à virgule flottante, double précision

La signification de `short`, `int`, ... dépend du compilateur et de l'architecture.

Types de base (2/4)

Voir `<stdint.h>` pour des représentations **portables**

Type	Signification
<code>int8_t</code> , <code>uint8_t</code>	Entier signé/non-signé 8-bit
<code>int16_t</code> , <code>uint16_t</code>	Entier signé/non-signé 16-bit
<code>int32_t</code> , <code>uint32_t</code>	Entier signé/non-signé 32-bit
<code>int64_t</code> , <code>uint64_t</code>	Entier signé/non-signé 64-bit

Types de base (2/4)

Voir `<stdint.h>` pour des représentations **portables**

Type	Signification
<code>int8_t</code> , <code>uint8_t</code>	Entier signé/non-signé 8-bit
<code>int16_t</code> , <code>uint16_t</code>	Entier signé/non-signé 16-bit
<code>int32_t</code> , <code>uint32_t</code>	Entier signé/non-signé 32-bit
<code>int64_t</code> , <code>uint64_t</code>	Entier signé/non-signé 64-bit

Prenez l'habitude d'utiliser ces types-là !

Booléens

- Le ANSI C n'offre pas de booléens.
- L'entier `0` signifie *faux*, tout le reste *vrai*.
- Depuis C99, la librairie `stdbool` met à disposition un type `bool`.
- En réalité c'est un entier :
 - `1` \Rightarrow `true`
 - `0` \Rightarrow `false`
- On peut les manipuler comme des entier (les sommer, les multiplier, ...).

Quiz : booléens

Quiz : booléens

Conversions

- Les conversions se font de manière :
 - Explicite :

```
int a = (int)2.8;  
double b = (double)a;  
int c = (int)(2.8+0.5);
```

- Implicite :

```
int a = 2.8; // warning, si activés, avec clang  
double b = a + 0.5;  
char c = b; // pas de warning...  
int d = 'c';
```

Quiz : conversions

Quiz : conversions