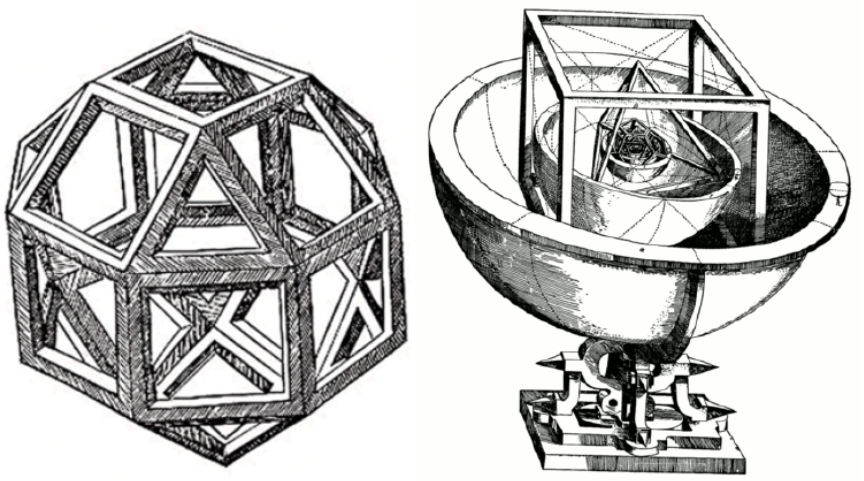

Complexité algorithmique

Julien Noyer

2018-09-20

Compléxité algorithmique *Définition et calcul*

L'analyse de la complexité d'un algorithme consiste en l'étude formelle de la quantité de ressources nécessaire à l'exécution de cet algorithme, celle-ci ne doit pas être mal comprise : lorsque l'on par de complexité c'est celle que la machine qui exécutera l'algorithme devra passer et pas la personne qui rédige l'algorithme.

Quand les scientifiques ont voulu énoncer formellement et rigoureusement ce qu'est l'efficacité d'un algorithme ou au contraire sa complexité, ils se sont rendu compte que la comparaison des algorithmes entre eux était nécessaire et que les outils pour le faire à l'époque1 étaient primitifs. Dans la préhistoire de l'informatique (les années 1950), la mesure publiée quand elle existait, était souvent dépendante de facteurs non-relatif à l'algorithme à traiter mais à la machine qui l'exécutera.

L'approche la plus classique utiliser pour calculer la complexité d'un algorithme est de calculer le temps de calcul dans le pire des cas nécessaire pour l'exécution de l'algorithme.

Calculer la complexité d'un algorithme

Il est possible d'écrire différents algorithmes pour atteindre un même résultat, par exemple calculer le fait qu'un utilisateur soit majeur ou pas, mais comment savoir lequel sera le plus efficace cotés

machine et qui demandera le moins de ressources matérielles ? Pour le savoir nous allons analyser un algorithme simple dont voici la définition :

```
FUNCTION check_user_age( a: INTEGER, b: INTEGER )
  START
    IF age > 18 THE
      PRINT("User is major")

    ELSE
      PRINT("User is not major")

    END IF
  END
/FUNCTION
```

L'analyse de cette algorithme nous permet de faire quelques constats :

- Le code est lu de haut en bas
- Pour répondre à la condition IF *age* doit être supérieur à 18
- La condition ELSE est déclencher uniquement si la condition IF n'est pas remplie

Nous pouvons donc considérer que pour que l'algorithme s'exécute le plus rapidement il faut que le paramètre *age* soit supérieur à 18 ce que nous nommerons lexécution **idéale** de l'agorithme. Pour en calculer sa complexité nous allons donc le pire des scénarios : que ce passe-t-il si *age* est égal à 10 ?

Pour calculer la complexité d'un algorithme il faut prendre en compte le pire des scénarios et associer un point pour chaque opération élémentaire :

- affectation
- calcul
- comparaison

Passons à présent au calcul de la complexité de l'algorithme suivant :

```
FUNCTION check_number()
  // Declare
  a: INTEGER,
  b: INTEGER

  START
    a <- 14
    PRINT("Number a is ", a)
```

```
    b <- a - 5
    PRINT("Number b is ", b)

  END
/FUNCTION
```

Calcule de la complexité d'un algorithme simple

Nous commençons par calculer chaque opération élémentaires

- a <- 14 = 1
- b <- a - 5 = 1
- a - 5 = 1
- **Total** = 3

La complexité de l'algorithme est donc de **3** sur une échelle de temps, c'est à dire que qu'à chaque fois qu'il est répéter il faudra multiplier par 3 ne temps d'exécution. La syntaxe pour d'écrire la complexité de cette algorithme est :

```
T(n) = 3
```

Calcule de la complexité d'un algorithme compliqué

Passons à présent au calcule de la complexité d'un algorithme plus compliqué :

```
FUNCTION total(n: NUMBER): NUMBER
  // Declare
  1: INTEGER,
  response: INTEGER

  START
    response <- 0

    FOR i FROM 0 TO n [ i <- i + 1 ]
      response <- response + i

    END FOR

    RETURN response
```

```
  END
/FUNCTION
```

Commençons par calculer les opérations élémentaires :

- response <- 0 = 1
- **Total** = 1

Nous retenons le résultat et passons au calcule de la boucle

- i FROM 0 = 1
- 0 TO n = 1
- i + 1 = 1
- response <- response + i = 1
- response + i = 1
- **Total** = 5 x n

La complexité de l'algorithme est donc de **5n + 1** sur une échelle de temps dont la syntaxe pour la d'écrire est :

```
T(n) = 5n + 1
```