Complexité

Cours de CPGE PSI, 19 septembre 2018

Adeline Pierrot

Un premier algorithme

```
Fonction1(n) :
        Pour tout i de 1 à n :
                Si i divise n :
                        Afficher i
```

Que fait cet algorithme ?

Il affiche la liste des diviseurs d'un nombre entier n.

Un second algorithme

```
Fonction2(n) :
    Pour tout i de 1 à √n :
        Si i divise n :
            Afficher i
            Si n/i différent de i :
                Afficher n/i
```

Que fait cet algorithme ?

Il affiche la liste des diviseurs d'un nombre entier n.

Plusieurs algorithmes pour un même problème

```
Fonction1(n) :
    Pour tout i de 1 à n :
        Si i divise n :
            Afficher i
```

```
Fonction2(n) :
    Pour tout i de 1 à √n :
        Si i divise n :
            Afficher i
            Si n/i différent de i :
                Afficher n/i
```

Comparaison d'algorithmes pour un même problème

- Le premier est plus simple à comprendre
- Quel est l'avantage du second ?

Le premier algorithme

```
Fonction1(n) :
    Pour tout i de 1 à n :
        Si i divise n :
            Afficher i
```

Cet algorithme effectue n fois :
un test et peut-être un affichage

$\hookrightarrow$ entre n et 2n opérations au total, soit $\mathcal{O}(n)$.

Le second algorithme

Fonction2(n) :
    Pour tout i de 1 à $\sqrt{n}$:
        Si i divise n :
            Afficher i
            Si n/i différent de i :
                Afficher n/i

Cet algorithme effectue $\sqrt{n}$ fois :
    un test, peut-être un affichage, peut-être un autre test, peut-être un autre affichage

$\hookrightarrow$ entre $\sqrt{n}$ et $4\sqrt{n}$ opérations au total, soit $\mathcal{O}(\sqrt{n})$.

$\hookrightarrow$ temps d'exécution plus court.

Coût d'un algorithme (complexité)

Pour déterminer le coût d'un algorithme, l'un des modèles utilisés est de compter son nombre d'opérations parmi les opérations suivantes :

- opérations arithmétiques (+, -, * ...)
- comparaisons (==, < ...)
- affectations de variables
- affichages

Le nombre d'opérations dépend souvent d'un entier n à définir proprement.
On dit que la complexité est $\mathcal{O}(f(n))$ si $\exists c_1, c_2, n_0$ tels que $\forall n > n_0,\ c_1 f(n) \leq$ nombre d'opérations pour n $\leq c_2 f(n)$.

Propriétés du calcul du coût

- Si l'algorithme est composé de plusieurs blocs d'instructions les uns à la suite des autres, le coût de l'algorithme est la somme du coût de chacun des blocs.

- Le coût d'un test *if b: bloc1 else: bloc2* est inférieur ou égal au maximum des coûts des blocs 1 et 2, plus le temps d'évaluation de l'expression b.

- Le coût d'une boucle est égal à la somme des coûts de chacun des passages dans la boucle (en comptant les mises à jour et tests).

Exemples

```
def table1(n):
    for i in range(10):
        print(i * n)
```

complexité $\mathcal{O}(1)$: constant

```
def table2(n):
    for i in range(n):
        print(i * i)
```

complexité $\mathcal{O}(n)$: linéaire

```
def table3(n):
    for i in range(n):
        for j in range(n):
            print(i * j)
```

complexité $\mathcal{O}(n^2)$: quadratique

Différentes nuances de complexité

- Lorsque l'entrée de l'algorithme n'est pas un entier, il faut définir une notion de taille sur l'entrée pour pouvoir définir la complexité de l'algorithme. Par exemple lorsque l'entrée est une liste (ou une chaîne de caractères), on calcule usuellement la complexité en fonction du nombre d'éléments de la liste (ou du nombre de caractères de la chaîne de caractères).
- Pour différentes entrées de même taille n, la complexité de l'algorithme peut être différente en fonction de l'entrée (par exemple $\mathcal{O}(1)$ ou $\mathcal{O}(n)$). On considère usuellement la complexité la pire sur l'ensemble des entrées possibles (complexité pire cas).
- La complexité calculée à partir du nombre d'opérations est appelée complexité en temps (elle est liée au temps d'exécution de l'algorithme). On peut aussi s'intéresser à la complexité en espace, qui est liée à la place mémoire occupée lors de l'exécution de l'algorithme.