Complexité d'un algorithme

Alexandre Benoit

BCPST

I Coût d'un algorithme

Un premier exemple

Pour traiter un même problème, il existe souvent plusieurs algorithmes. Pour décider quel algorithme choisir, l'un des critères est celui du temps d'exécution, qu'on appelle aussi coût de l'algorithme.

Exemple

```
def diviseurs(n):
    L = []
    for i in range(1, n+1):
        if n % i == 0:
            L.append(i)
    return L
```

- À quoi sert cet algorithme ?
- Combien de calculs de restes et de divisions euclidiennes, combien de comparaisons, combien d'affichages effectue cet algorithme ?

Une alternative à cet algorithme

Exemple

```
import math
def diviseurs2(n):
    racine = int(math.sqrt(n))+1
    L = []
    for i in range(1, racine):
        if n % i == 0:
            L.append(i)
            diviseur = n // i
            if diviseur != i :
                L.append(diviseur)
    return L
```

- À quoi sert cet algorithme ?
- Combien de calculs de restes et de divisions euclidiennes, combien de comparaisons, combien d'affichages effectue cet algorithme

Exercice

- Si le premier algorithme avec comme entrée n s'exécute en 10 secondes, en combien de secondes s'exécutera-t'il avec en entrée $100n$?
- Même question avec le second algorithme.

Déterminer le coût d'un algorithme

Pour déterminer le coût d'un algorithme, on se fonde en général sur le modèle de complexité suivant :

- Une affectation, une comparaison ou l'évaluation d'une opération arithmétique ayant en général un faible temps d'exécution, celui-ci sera considéré comme l'unité de mesure d'un algorithme.
- Le coût d'un test `if b: p else: q` est inférieur ou égal au maximum des coûts des instructions `p` et `q`, plus le temps d'évaluation de l'instruction `b`
- Le coût d'une boucle `for i in iterable : p` est égal au nombre d'éléments de l'itérable multiplié par le coût de l'instruction p si ce dernier ne dépend pas de la valeur de i.
  Quand le coût du corps de la boucle dépend de la valeur de i, le coût total de la boucle est la somme des coûts de corps de la boucle pour chaque valeur de i.
- Pour les boucles while, on doit majorer le nombre de répétions de la boucle de la même façon qu'on démontre sa terminaison.

II Applications : Complexités de certains algorithmes

Exemple 1 : Une premier test

Quel est le coût de l'algorithme suivant ?

```
def test_de_presence(L)
    if 2 in L:
        print ("Il y est")
    else:
        print ("Il n'y est pas")
```

Dans le pire cas, 2 n'est pas dans L. Dans ce cas, on doit effecteur $len(L)$ tests. La complexité est alors de $len(L) + 1$ opérations.

Exemple 2 : Un second test

Quel est le coût de l'algorithme suivant ?

```
def test_de_presence(a, L)
    if a == 0:
        print ("a doit etre non nul")
    else:
        L.remove(a)
```

La commande L.remove(a) retire la première occurrence de l'élément a dans L
Dans tous les cas, le coût du test est : 1 opération.
Si $a \neq 0$, et que $a \notin L$, on doit parcourir toute la liste pour chercher a ce qui fait $len(L)$ opérations.
Si $a \in L$, on doit décaler tous les éléments se situants après a, ce qui coûte $len(L)$ opérations.
On a donc $len(L) + 1$ opérations.

Exemple 3 : Une première boucle

Quel est le coût de l'algorithme suivant ?

```
    for i in range(11):
        print(i * n)
```

Cet algorithme effectue 11 multiplications. Il effectue donc 11 opérations.

Exemple 4 : Une seconde boucle

Quel est le coût de l'algorithme suivant ?

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

Cet algorithme effectue n multiplications. Il effectue donc n opérations.

Exemple 5 : Une troisième boucle

Quel est le coût de l'algorithme suivant ?

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

Cet algorithme construit une table de multiplications pour tous les entiers de 1 à n en donnant tous leurs multiples jusqu'au n-ième.
Il comprend deux boucles imbriquées, chacune effectuant n répétitions de son corps ; le corps de la boucle interne ne comporte qu'une multiplication.
La complexité est ici n^2 opérations.

Exemple 5 : Une dernière boucle

Quel est le coût de l'algorithme suivant ?

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

Cet algorithme construit une table de multiplications pour tous les entiers de 1 à n en donnant tous leurs multiples jusqu'au n-ième ; en prenant en compte la commutativé de la multiplication (On affichera 10×5 mais pas 5×10).
Il comprend deux boucles imbriquées, la première effectuant n répétitions de son corps. La boucle interne dépend du compteur de la boucle externe. Le corps de la boucle interne ne comporte qu'une multiplication. Le nombre d'opérations sera donc :

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2}.$$

III Complexité et équivalences entre les suites

Définition d'une classe de complexité

Dans la plupart des cas, on a pas besoin d'aller jusqu'au niveau de détails des exemples précédents.
On sait que

$$n(n+1) \underset{n\to\infty}{\sim} n^2,$$

donc les complexités des algorithmes 4 et 5 sont du même « ordre ».

Définition

On dit qu'un algorithme a une complexité $g(n)$ de l'ordre de $f(n)$ s'il existe des constantes strictement positives c et c' telles que

$$cf(n) \leqslant g(n) \leqslant c'f(n).$$

Ordre de grandeur

Ordres de grandeur des temps d'exécution d'un problème de taille 10^6 sur un ordinateur à un milliard d'opérations par seconde.

Ordre	Nom	Temps pour $n = 10^6$
1	Temps constant	$1ns$
$\log n$	logarithmique	$10ns$
n	linéaire	$1ms$
n^2	quadratique	$1/4h$
n^k	polynomiale	30 ans si $k = 3$
2^n	exponentielle	$> 10^{300000}$ milliards d'années

IV Différentes nuances de complexité

Les différents types de complexité

- Complexité au pire : Notre modèle de complexité
- Complexité dans le meilleurs cas : Modèle de complexité peu utile
- Complexité en moyenne : à partir d'une répartition probabiliste des tailles de données, tente d'évaluer le temps moyen que l'on peut attendre de l'évaluation d'un algorithme sur une donnée d'une certaine taille.
- Complexité en espace : On ne s'occupe plus du temps d'exécution mais de la mémoire utile pour exécuter un algorithme.