S 1.02
DEV

Complexité algorithmique

1 Vers une définition de complexité algorithmique

1.1 Objectif de la notion

Lors de la réalisation d'un logiciel, quand un développeur fait face à un problème algorithmique, il peut être amené à devoir choisir entre différentes structures de données (tableau, liste,...) et différents algorithmes permettant de résoudre le problème auquel il fait face.

Imaginons par exemple un développeur souhaitant réaliser un programme permettant à un utilisateur de gérer l'ensemble des fichiers vidéo présents sur son ordinateur et d'afficher ceux d'une catégorie qu'il aurait sélectionnée. Quelle structure de données utiliser pour stocker les informations sur les fichiers vidéo ? Quel algorithme choisir pour extraire les films de la catégorie désirée ? Et pourquoi faire ces choix-là et pas d'autres ?

Une façon assez naturelle de faire ces choix est de chercher les structures de données et les algorithmes les plus "efficaces" pour résoudre ce problème. La notion de complexité algorithmique va nous aider à évaluer "l'efficacité" d'un algorithme (et par extension d'une structure de données).

1.2 A partir d'exemples

Pour comprendre la notion de complexité algorithmique nous allons étudier deux exemples simples d'algorithmes :

Exercice 1 : Premier exemple
Entrée : Un tableau tab de 5 entiers
Algorithme :

```
for i :=1 to 5
    tab[i]  := tab[i] + 1 ;
```

1. Que fait cet algorithme.
2. Combien d'affectations et d'additions réalise-t-il ?
3. Et si votre tableau avait une taille de n ?

Exercice 2 : Deuxième exemple
Entrée : Un tableau tab de 5 entiers
Algorithme :

```
max  := tab[1]
for i :=1 to 5
    if(tab[i]>max) then max  := tab[i] ;
```

1. Que fait cet algorithme.
2. Combien d'affectations, de tests et d'additions réalise-t-il ?
3. Comment régler le problème rencontré dans la question précédente ?

1.3 Définitions

Attention, la notion de complexité algorithmique est une notion relativement complexe à formaliser correctement. Les définitions qui vont suivre sont donc juste une première approche "avec les mains" de ces notions qui nous sera suffisante pour le moment.

Définition : Complexité algorithmique

La complexité algorithmique est l'ordre de grandeur du nombre d'opérations élémentaires réalisées dans le pire des cas en fonction de la taille de l'entrée à traiter.

Cette définition repose sur deux autres notions, celle d'opération élémentaire et celle de taille de donnée.

Convention : Opération élémentaire

On considèrera comme opération élémentaire toute opération réalisable simplement par le processeur : affectation, test simple, addition, multiplication, division euclidienne...

Convention : Taille de donnée

On considèrera les tailles de données suivantes :

- *Pour un tableau unidimensionnel : son nombre d'éléments.*
- *Pour un tableau bidimensionnel : le maximum entre sa largeur et sa hauteur.*
- *Pour une liste : son nombre d'éléments.*
- *Pour un nombre entier : le nombre de bits utilisés pour le stocker.*

2 Exercices d'application

2.1 Algorithmes sur des tableaux

Exercice 3 : Parcours d'un tableau

On s'intéresse à un algorithme prenant en entrée un tableau d'entiers et ajoutant 1 à chacun des entiers de celui-ci.

1. Donnez le pseudo-code d'un tel algorithme.
2. Déterminez sa complexité.

Exercice 4 : Sommes des éléments d'un tableau

On s'intéresse à un algorithme prenant en entrée un tableau d'entiers et renvoyant la somme des éléments de ce tableau.

1. Donnez le pseudo-code d'un tel algorithme.
2. Déterminez sa complexité.

Exercice 5 : Accessibilité d'un élément dans un tableau

On s'intéresse à un algorithme prenant en entrée un tableau d'entiers et renvoyant le premier élément de ce tableau.

1. Donnez le pseudo-code d'un tel algorithme.
2. Déterminez sa complexité.
3. Même question si l'on souhaite renvoyer le dernier élément du tableau.

2.2 Algorithmes sur des listes

Exercice 6 : Accessibilité d'un élément dans une liste
On s'intéresse à un algorithme prenant en entrée une liste d'entiers et renvoyant le premier élément de cette liste.

1. Donnez le pseudo-code d'un tel algorithme.
2. Déterminez sa complexité.
3. Même question si l'on souhaite renvoyer le dernier élément de la liste.

Exercice 7 : Parcours d'une liste
On s'intéresse à un algorithme prenant en entrée une liste d'entiers et renvoyant le plus grand élément de cette liste.

1. Revoyez le principe d'un tel algorithme avec votre enseignant.
2. Déterminez sa complexité.

Exercice 8 : Ajout dans une liste
On s'intéresse à un algorithme prenant en entrée une liste d'entiers et un entier et ajoutant cet entier à la fin de la liste.

1. Revoyez le principe d'un tel algorithme avec votre enseignant.
2. Déterminez sa complexité.
3. Même question si l'on souhaite ajouter au début de la liste.
4. Même question si l'on manipule une pile.

2.3 Complexité et récursivité

Exercice 9 : Un premier exemple
On considère l'algorithme récursif suivant :
Entrée : un entier positif n
Algorithme :

```
function suite(n : integer);
    if(n=0) then suite :=1
    else suite := 2*suite(n-1)+1;
```

1. Que fait cet algorithme ?
2. Déterminez sa complexité.
3. Est-il possible de faire un algorithme plus efficace pour résoudre ce problème ?

Exercice 10 : Fibonacci
On considère l'algorithme récursif suivant calculant le n-ième terme de la célèbre suite de Fibonacci :
Entrée : un entier positif n
Algorithme :

```
function Fibonacci(n : integer);
    if(n=0) then Fibonacci :=0;
    else if(n=1) then Fibonacci :=1;
    else Fibonacci := Fibonacci(n-1) + Fibonacci(n-2);
```

1. Que fait cet algorithme ?
2. Déterminez sa complexité.
3. Est-il possible de faire un algorithme linéaire résolvant se problème ?

Exercice 11 : Les tours de Hanoï
On considère l'algorithme vu en TD de la ressource R1.01 sur le problème des tours de Hanoï

1. Rappelez rapidement le fonctionnement de l'algorithme.
2. Déterminez sa complexité.

2.4 Un peu de math :)

Exercice 12 : Une méthode naïve
On considère l'algorithme suivant :
Entrée : un entier positif a
Algorithme :

```
res := true;
for i :=2 to a-1
    if(a mod i=0) then res :=false;
```

1. Que fait cet algorithme ?
2. Déterminez sa complexité.

Exercice 13 : Indicatrice d'Euler
On admettra que l'algorithme d'Euclide est linéaire. On souhaite réaliser un algorithme calculant l'indicatrice d'Euler d'un nombre entier a. On rappelle qu'il s'agit du nombre d'entiers b inférieur à a tels que PGCD(a,b)=1.

1. Donnez le pseudo-code d'un tel algorithme.
2. Déterminez sa complexité.

Exercice 14 : Multiplication de matrices
On souhaite réaliser un algorithme calculant le résultat de la multiplication de deux matrices carrées de taille n.

1. Donnez le pseudo-code d'un tel algorithme.
2. Déterminez sa complexité.