Algorithmique

Le mot algorithme vient d'un mathématicien perse du IXème siècle, Al Khawarizmi. Mais, des algorithmes, il en existait bien avant. En effet Euclide a donné une série d'instructions permettant de trouver le PGCD de deux nombres. On retiendra le nom d'Ada Lovelace (première moitié du XIXème) auteur du premier algorithme destiné à être implémenter par une machine …qui ne fut jamais terminée !

Un algorithme est donc une série d'instructions finies visant à résoudre un problème. Le mot est aujourd'hui très souvent utilisé et en effet, la vie quotidienne peut être vu comme une suite d'algorithmes.

Ex : Si je ne suis pas malade ou en vacances, je me lève à 6h30. Sinon, je reste au lit.

Une recette de cuisine, une tactique sportive (tant que je gagne, je ne change pas de stratégie…) tout peut être vu sous forme d'algorithme.

Dans ce cours, nous allons présenter différents types d'algorithmes et nous intéresser à certaines notions relatives aux algorithmes : la complexité et la preuve des algorithmes.

1. Exemple introductif

On va s'intéresser à un algorithme qui calcule la somme des n premiers entiers, n étant défini par l'utilisateur.

En algorithmique, on doit spécifier les valeurs qui sont fournies pour l'exécution de l'algorithme et la sortie que l'on attend.

On va s'intéresser au coût temporel d'exécution d'un algorithme. Ce que l'on appelle **la complexité en temps**. En effet, les ressources d'un ordinateur ne sont pas illimitées et l'idée sera d'optimiser nos algorithmes afin que l'implémentation de ceux-ci dans la machine coûte le moins de ressources possibles. Attention, il existe aussi la complexité en mémoire, non au programme.

On va aussi s'assurer que l'algorithme se termine bien (on parle de **terminaison**) et qu'il fait bien ce que l'on attend de lui (**correction**). On parle de **preuve** d'un algorithme. On le fera dans un chapitre à part. Ici donc, on met en place notre algorithme en précisant les entrées, la ou les sorties et on étudie son coût en temps.

```
####Algorithme Somme des n premiers entiers ####


Algorithme Somme_des_n_premiers_entiers (n)

  Entrée : Un nombre entier n
  Sortie : La somme des n premiers entiers

  Somme <- 0
  i <-0
  Tant que i ≤ n faire
      Somme <- Somme +i
      i<- i+1

  retourner Somme
```

2. Notion de complexité

On va utiliser deux notations en rapport avec la complexité : O et Θ.

Dire qu'un algorithme à **une complexité en O(n)** par exemple, signifie que dans le pire des cas, le temps d'exécution sera de l'ordre de n. Dans notre exemple n est l'entier entré.

On rencontrera aussi **la complexité en Θ (n).** On est alors capable de dire que le temps d'exécution de l'algorithme est compris entre deux fonctions d'ordre n et non plu seulement majorée comme pour O(n).

On peut rencontrer différents types de complexité : Constante, linéaire…

complexité constante en $O(1)$;
complexité logarithmique en $O(\log_2 n)$;
complexité linéaire en $O(n)$;
complexité quasi-linéaire en $O(n \log_2 n)$;
complexité polynomiale en $O(n^k)$;
complexité exponentielle en $O(2^n)$;

Rappel $\log_2(x) = \frac{\ln(x)}{\ln 2}$ x >0.

1. Faites la trace de cet algorithme pour n= 5.
2. Etude du coût.

Alors comment calcule-t-on la complexité de notre algorithme ?

```
####Algorithme Somme des n premiers entiers ####


Algorithme Somme_des_n_premiers_entiers (n)

 1.  Entrée : Un nombre entier n
     Sortie : La somme des n premiers entiers

 1.  Somme <- 0
 1.  i <-0
 n.  Tant que i ≤ n faire
 1.      Somme <- Somme +i                  Ici, on va faire n fois 2 opérations
 1.      i<- i+1                            soit un coût de 2n .On est en O(n), la constante 2
                                            ne change pas l'ordre du coût (linéaire)
 1.  retourner Somme


Au total , on a un coût de 1+1+1+ 2n+1, soit un complexité en O(n)
```

3. A vous de jouer : Recherche d'un élément dans un tableau

1. Ecrire un algorithme ayant :

- En entrée, un tableau de données numériques et une valeur à chercher.
- En sortie, l'indication de la présence ou non dans le tableau de l'élément cherché.

2. Etudier la complexité de cet algorithme.