Cours : Initiation à l'Algorithme

1 Introduction

Un algorithme est une suite d'instructions précises et ordonnées permettant de résoudre un problème ou d'accomplir une tâche. En informatique, les algorithmes sont la base de tout programme. Ils sont comparables à une recette de cuisine : une série d'étapes pour atteindre un objectif.

1.1 Objectifs du cours

— Comprendre ce qu'est un algorithme et son rôle en informatique.

— Apprendre à décomposer un problème en étapes simples.

— Découvrir les outils pour représenter un algorithme (pseudo-code, organigramme).

— Pratiquer la conception d'algorithmes simples.

2 Qu'est-ce qu'un algorithme ?

Un algorithme est une méthode systématique pour résoudre un problème. Il doit être :

— **Fini** : Il se termine après un nombre limité d'étapes.

— **Précis** : Chaque instruction est claire et non ambiguë.

— **Efficace** : Il utilise les ressources (temps, mémoire) de manière raisonnable.

2.1 Exemple quotidien

Pour faire un sandwich :

1. Prendre deux tranches de pain.
2. Étaler du beurre sur une tranche.
3. Ajouter une tranche de jambon.
4. Refermer avec l'autre tranche.
5. Servir.

C'est un algorithme simple, avec des étapes ordonnées.

2.2 Importance en informatique

Les algorithmes permettent de :

— Résoudre des problèmes complexes (tri, recherche, calculs).

— Optimiser les performances des programmes.

— Traduire des idées en instructions exécutables par un ordinateur.

3 Caractéristiques d'un bon algorithme

Un algorithme doit respecter ces critères :

1. **Entrées** : Les données nécessaires pour résoudre le problème.
2. **Sorties** : Le résultat attendu.
3. **Définition claire** : Chaque étape est compréhensible.
4. **Finitude** : L'algorithme s'arrête après un nombre fini d'étapes.
5. **Efficacité** : Il minimise le temps et les ressources.

4 Représentation d'un algorithme

Il existe plusieurs façons de décrire un algorithme :

— **Langage naturel** : Description en phrases simples.

— **Pseudo-code** : Langage intermédiaire, structuré.

— **Organigramme** : Représentation graphique avec des symboles.

4.1 Exemple en pseudo-code

Problème : Calculer la moyenne de trois nombres.

```
DEBUT
  LIRE nombre1, nombre2, nombre3
  somme <- nombre1 + nombre2 + nombre3
  moyenne <- somme / 3
  AFFICHER moyenne
FIN
```

5 Concepts de base

5.1 Variables

Une variable est une zone de mémoire qui stocke une valeur. Exemple : `age <- 25`.

5.2 Instructions

— **Affectation** : Donner une valeur à une variable (`x <- 10`).

— **Entrée/Sortie** : Lire des données (`LIRE x`) ou afficher un résultat (`AFFICHER x`).

— **Calculs** : Opérations mathématiques (`y <- x + 5`).

5.3 Structures de contrôle

1. **Séquencielle** : Les instructions s'exécutent dans l'ordre.
2. **Conditionnelle** : Exécuter une instruction selon une condition.

```
SI condition ALORS
  Faire ceci
SINON
  Faire cela
FIN SI
```

Exemple : Vérifier si un nombre est positif.

```
SI nombre > 0 ALORS
  AFFICHER "Nombre positif"
SINON
  AFFICHER "Nombre négatif ou zéro"
FIN SI
```

3. **Itérative (boucle)** : Répéter des instructions.
   — Boucle **POUR** :

```
POUR i DE 1 A 10 FAIRE
  AFFICHER i
FIN POUR
```

   — Boucle **TANT QUE** :

```
TANT QUE x < 10 FAIRE
  x <- x + 1
  AFFICHER x
FIN TANT QUE
```

6 Étapes pour concevoir un algorithme

1. Comprendre le problème : Identifier les entrées, sorties et contraintes.
2. Décomposer le problème : Diviser en sous-étapes simples.
3. Écrire l'algorithme : Utiliser pseudo-code ou organigramme.
4. Tester l'algorithme : Vérifier manuellement avec des exemples.
5. Optimiser : Simplifier ou améliorer l'efficacité si nécessaire.

6.1 Exemple : Trouver le maximum de trois nombres

Problème : Lire trois nombres et afficher le plus grand.
Pseudo-code :

```
DEBUT
  LIRE a, b, c
  max <- a
  SI b > max ALORS
    max <- b
  FIN SI
  SI c > max ALORS
```

```
    max <- c
  FIN SI
  AFFICHER "Le maximum est ", max
FIN
```

Test :

— Entrées : $a = 5$, $b = 10$, $c = 3$

— Étapes :

— `max` $\leftarrow 5$

— $10 > 5$? Oui, `max` $\leftarrow 10$

— $3 > 10$? Non, `max` reste 10

— Afficher : "Le maximum est 10"

7 Exercices pratiques

7.1 Exercice 1 : Calculer l'aire d'un rectangle

Écrire un algorithme qui lit la longueur et la largeur dun rectangle et affiche son aire.

Solution :

```
DEBUT
  LIRE longueur, largeur
  aire <- longueur * largeur
  AFFICHER "L'aire du rectangle est ", aire
FIN
```

7.2 Exercice 2 : Vérifier si un nombre est pair

Écrire un algorithme qui lit un nombre et affiche sil est pair ou impair.

Solution :

```
DEBUT
  LIRE nombre
  SI nombre MOD 2 = 0 ALORS
    AFFICHER "Le nombre est pair"
  SINON
    AFFICHER "Le nombre est impair"
  FIN SI
FIN
```

7.3 Exercice 3 : Compter jusqu'à 5

Écrire un algorithme qui affiche les nombres de 1 à 5.

Solution :

```
DEBUT
  POUR i DE 1 A 5 FAIRE
    AFFICHER i
  FIN POUR
FIN
```

8 Introduction à l'optimisation

Un algorithme peut être amélioré pour être plus rapide ou utiliser moins de mémoire. Exemple : pour trier une liste, un algorithme simple comme le **tri par large selection** peut être utilisé :

```
DEBUT
  LIRE liste
  POUR i DE 1 A longueur(liste) FAIRE
    min <- i
    POUR j DE i+1 A longueur(liste) FAIRE
      SI liste[j] < liste[min] ALORS
        min <- j
      FIN SI
    FIN POUR
    ÉCHANGER liste[i] ET liste[min]
  FIN POUR
  AFFICHER liste
FIN
```

Pour des listes très grandes, des algorithmes comme le **tri rapide** sont plus efficaces.

9 Conclusion

Les algorithmes sont au cur de linformatique. En maîtrisant leur conception, vous pourrez :

— Résoudre des problèmes de manière structurée.

— Préparer le terrain pour programmer dans des langages comme Python ou JavaScript.

— Développer une pensée logique et analytique.

9.1 Conseils pour progresser

— Pratiquer régulièrement avec des exercices simples.

— Tester vos algorithmes avec des cas limites (nombres négatifs, zéro).

— Passer progressivement à des problèmes plus complexes.

10 Ressources supplémentaires

— **Livres** : "Introduction to Algorithms" de Cormen (niveau avancé).

— **Sites** : Codecademy, LeetCode, Khan Academy (cours gratuits).

— **Outils** : Visual Studio Code pour tester vos algorithmes.