Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider - Biskra
Faculté des Sciences Exactes et des Sciences de la Nature et de la Vie
Département d'Informatique

2ème année LMD

Cours d'Algorithmique et Structures de Données 3

Pr. Abdelhamid DJEFFAL
www.abdelhamid-djeffal.net

Année Universitaire 2021/2022

Plan du cours

Références

[1] D. Beauquier, J. Berstel, and P. Chrétienne. *Éléments d'algorithmique*. Masson, Version 6, 2005.

[2] T.H. Cormen, C.E. Leiserson, R.L. Rivest, T.H. Cormen, and T.H. Cormen. *Introduction à l'algorithmique*. Dunod, 1996.

[3] J. Courtin and I. Kowarski. *Initiation à l'algorithmique et aux structures de données*. Dunod, 1990.

[4] M.C. Gaudel, M. Soria, and C. Froidevaux. Types de données et algorithmes 2 : Recherche, tri, algorithmes sur les graphes. 1987.

[5] Guillaume Poupard. *Algorithmique*. Ecole Nationale Supérieure des Techniques Avancées, 2000.

Chapitre 1

Introduction

L'utilisation d'un ordinateur pour la résolution d'un problème nécessite tout un travail de préparation. N'ayant aucune capacité d'invention, l'ordinateur ne peut en effet qu'exécuter les ordres qui lui sont fournis par l'intermédiaire d'un programme. Ce dernier doit donc être établi de manière à envisager toutes les éventualités d'un traitement.
Exemple : le problème Div(a,b), n'oubliez pas le cas b=0 !

1.1 Résolution d'un problème en informatique

Plusieurs étapes sont nécessaires pour résoudre un problème en informatique :

- Etape 1 : Définition du problème

  Il s'agit de déterminer toutes les informations disponibles et la forme des résultats désirés.

- Etape 2 : Analyse du problème

  Elle consiste à trouver le moyen de passer des données aux résultats. Dans certains cas on peut être amené à faire une étude théorique. Le résultat de l'étape d'analyse est un algorithme. Une première définition d'un algorithme peut être la suivante :
  "On appelle algorithme une suite finie d'instructions indiquant de façon unique l'ordre dans lequel doit être effectué un ensemble d'opérations pour résoudre tous les problèmes d'un type donné."

  Sachez aussi qu'il existe des problèmes pour lesquels on ne peut trouver une solution et par conséquent il est impossible de donner l'algorithme correspondant.

- Etape 3 : Ecriture d'un algorithme avec un langage de description algorithmique

  Une fois qu'on trouve le moyen de passer des données aux résultats, il faut être capable de rédiger une solution claire et non ambiguë. Comme il est impossible de le

faire en langage naturel, l'existence d'un langage algorithmique s'impose.

– Etape 4 : Traduction de l'algorithme dans un langage de programmation

  Les étapes 1, 2 et 3 se font sans le recours à la machine. Si on veut rendre l'algorithme concret ou pratique, il faudrait le traduire dans un langage de programmation. Nous dirons alors qu'un programme est un algorithme exprimé dans un langage de programmation.

– Etape 5 : Mise au point du programme

  Quand on soumet le programme à la machine, cette dernière le traite en deux étapes :

  1. La machine corrige l'orthographe, c'est ce qu'on appelle syntaxe dans le jargon de la programmation.
  2. La machine traduit le sens exprimé par le programme.

  Si les résultats obtenus sont ceux attendus, la mise au point du programme se termine. Si nous n'obtenons pas de résultats, on dira qu'il y a existence des erreurs de logique. Le programme soit ne donne aucun résultat, soit des résultats inattendus soit des résultats partiels. Dans ce cas, il faut revoir en priorité si l'algorithme a été bien traduit, ou encore est-ce qu'il y a eu une bonne analyse.

1.2 Notion d'algorithme

1.2.1 Définition

On peut définir un algorithme comme suit :

Résultat d'une démarche logique de résolution d'un problème. C'est le résultat de l'analyse.

Ou encore :

Une séquence de pas de calcul qui prend un ensemble de valeurs comme entrée (input) et produit un ensemble de valeurs comme sortie (output).

1.2.2 Propriétés

On peut énoncer les cinq propriétés suivantes que doit satisfaire un algorithme :

1. Généralité : un algorithme doit toujours être conçu de manière à envisager toutes les éventualités d'un traitement.
2. Finitude : Un algorithme doit s'arrêter au bout d'un temps fini.
3. Définitude : toutes les opérations d'un algorithme doivent être définies sans ambiguïté

4. Répétitivité : généralement, un algorithme contient plusieurs itérations, c'est à dire des actions qui se répètent plusieurs fois.
5. Efficacité : Idéalement, un algorithme doit être conçu de telle sorte qu'il se déroule en un temps minimal et qu'il consomme un minimum de ressources.

1.2.3 Exemples

- PGCD (Plus Grand Commun Diviseur) de deux nombres u et v.
  - Algorithme naïf : on teste successivement si chaque nombre entier est diviseur commun.
  - Décomposition en nombres premiers.
- Algorithmes de tri
- Algorithmes de recherche
  - Recherche d'une chaîne de caractère dans un texte (Logiciels de traitement de texte).
  - Recherche dans un dictionnaire.
  - ... etc.

1.2.4 Remarque

Attention, certains problèmes n'admettent pas de solution algorithmique exacte et utilisable. On utilise dans ce cas des algorithmes heuristiques qui fournissent des solutions approchées.

1.3 Langage algorithmique utilisé

Durant ce cours, on va utiliser un langage algorithmique pour la description des différentes solutions apportées aux problèmes abordés. L'algorithme suivant résume la forme générale d'un algorithme et la plupart des déclarations et instructions utilisées.

```
Algorithme PremierExemple;
Type  TTab = tableau[1..10] de reel ;
Const Pi = 3.14 ;
Procédure Double( x : reel);
Début
    | x ← x * 2 ;
Fin;
Fonction Inverse( x : reel) : reel;
Début
    | Inverse ← 1/x ;
Fin;

Var  i, j, k : entier ;
     T : TTab ;
     S : chaine ;
     R : reel ;
Début
 | Ecrire (' Bonjour, donner un nombre entier ≤ 10 :') ;
 | Lire (i) ;
 | Si (i>10) Alors
 |     | Ecrire ('Erreur : i doit être ≤ 10')
 | Sinon
 |     | Pour j de 1 à i faire
 |     |     | Lire(R) ;
 |     |     | Double(R) ;
 |     |     | T [j] ← R ;
 |     | Fin Pour;
 |     | k ← 1 ;
 |     | Tant que (k ≤ i) faire
 |     |     | Ecrire (T [k] * Inverse(Pi)) ;
 |     |     | k ← k + 1 ;
 |     | Fin TQ;
 |     | S ← 'Programme terminé' ;
 |     | Ecrire(S) ;
 | Fin Si;
Fin.
```

Algorithme 1: Algorithme type