Matthieu GRALL

Cours d’algorithmique

Partie 1

Attention : ce document est un complément aux séances de cours et de travaux pratiques.

Table des matières

1 Bases algorithmiques

Pourquoi l'algorithmique ?

- Apprendre à communiquer avec une machine
- Lui transmettre des connaissances
- Exploiter les capacités spécifiques des machines → efficacité
- Maîtriser les défauts incontournables des machines → optimiser la performance

Des mots, une grammaire, des données (stockées, représentées) et des moyens…

Conseils : adopter une écriture soignée et une indentation claire (tabulations…), commenter, choisir l'efficacité quand plusieurs solutions sont possibles, décomposer les programmes en sous-programmes

1.1 Types de données

1.1.1 Le statut des données

- Constantes explicites : 'A' 3 14.5 "bonjour"
- Constantes nommées : UN=1 PI=3.14
- Variables : X : entier TMPS : réel

1.1.2 Types standards

- Caractères : 'A'
- Entiers : 3
- Réels : 3.14
- Booléens : FAUX

1.1.3 Tableaux

- Déclaration
  <NomTableau> : tableau [1 : <entier constant maximum>] de <type>
- Repérage
  <NomTableau> [<position>]
- Saisie
  <NomTableau> [<position>] ← <valeur>
- Affichage
  Afficher (<NomTableau> [<position>])

1.2 Fonctions de base

- Saisir une valeur : Saisir (prénom)
- Afficher texte et valeurs de variables : Afficher ("bonjour", prénom)
- Affecter une valeur à une variable : ←
- Opérateurs arithmétiques : +, -, *, /, div, mod…
- Fonctions arithmétiques : rac(x), sin(x), log(x), …
- Commentaires : {…}

1.3 Opérateurs logiques

Les propositions logiques donnent des valeurs de vérité : VRAI (0) ou FAUX(1)

- Négation : non

P	Non P
Vrai	Faux
Faux	Vrai

- Conjonction : et

P	Q	P et Q
Vrai	Vrai	Vrai
Vrai	Faux	Faux
Faux	Vrai	Faux
Faux	Faux	Faux

- Disjonction : ou

P	Q	P ou Q
Vrai	Vrai	Vrai
Vrai	Faux	Vrai
Faux	Vrai	Vrai
Faux	Faux	Faux

- Attention à l'ordre d'évaluation : (A et non B ou C) s'évalue comme ((A et (non B)) ou C)
- propriétés :
  - commutativité
    p et q ⇔ q et p
    p ou q ⇔ q ou p
  - associativité
    p et (q et r) ⇔ (p et q) et r
    p et (q ou r) ⇔ (p ou q) ou r
  - distributivité
    p et (q ou r) ⇔ (p et q) ou (p et r)
    p ou (q et r) ⇔ (p ou q) et (p ou r)
  - Loi de deMorgan
    non (p et q) ⇔ non p ou non q
    non (p ou q) ⇔ non p et non q

1.4 Instructions conditionnelles

Si <expression logique>
    alors <instructions>
    sinon <instructions>   {optionnel}
Fin de Si

Selon <variable>   {équivaut à Si…Sinon Si….Sinon Si…Sinon… … , en plus propre}
    <valeur1> : <instructions1>
    <valeur2> : <instructions2>
    …
    autres : <instructions>   {optionnel}
Fin Selon

1.5 Boucles avec arrêt sur événement et instructions itératives

Tant Que <expression logique> faire
    <instructions>
Fin Tant Que

Exemple :

Fonction Puissance (x, puiss) retourne (entier)
{retourne le résultat de x^{puiss}}
Paramètres :
    (D) x : entier
    (D) puiss : entier
Variables :
    Résultat : entier
Début
    Résultat ← x
    Tant Que (puiss >1)
        puiss ← puiss – 1
        *Résultat ← Résultat * x*
    Fin Tant Que
    Retourne (Résultat)
Fin

Répéter   {équivaut au Tant Que, mais les instructions sont exécutées au moins une fois}
    <instructions>
Jusqu'à <expression logique>

Pour <compteur> ← <valeur d'initialisation> à <valeur d'arrêt> par <pas> faire   {par défaut le pas est +1}
    <instructions>   {utilisé quand on connaît exactement le début et la fin du compteur}
Fin de Pour

1.6 Programmes et sous-programmes

1.6.1 Algorithme

Algorithme <NomAlgorithme>
Constantes :
    <const1> = <valeur1>
    <const2> = <valeur2>
    …
Variables :
    <var1> : <type1>
    <var2> : <type2>
    …
Début
    <instructions>
Fin

1.6.2 Procédures et fonctions

Afin de dégrossir un problème et de ne pas l'écrire séquentiellement, on peut créer différents modules :
les procédures et les fonctions.

Procédure <NomProcédure> (<paramètres>)
Paramètres :
    (<attribut1>) <param1> : <type1> {Attributs : D (donnée), R (résultat), ou D/R (donnée/résultat)}
    (<attribut2>) <param2> : <type2>
    …
Constantes :
    <const1> = <valeur1>
    <const2> = <valeur2>
    …
Variables :
    <var1> : <type1>
    <var2> : <type2>
    …
Début
    <instructions>
Fin

Fonction <NomFonction> (<paramètres>) retourne (<type>)
Paramètres :
    (<attribut1>) <param1> : <type1>
    (<attribut2>) <param2> : <type2>
    …
Constantes :
    <const1> = <valeur1>
    <const2> = <valeur2>
    …
Variables :
    <var1> : <type1>
    <var2> : <type2>
    …
Début
    <instructions>
    Retourne (<variable>)
Fin

1.6.3 Un exemple d'algorithme avec sous-programmes : le calcul de moyennes

```
Algorithme CalculNotes
{calcule la moyenne arrondie des notes saisies, arrêt sur note négative, on sait qu'il y a au moins une note}
Variables
        note, Snotes, MoyNotes : réel
        nbNotes : entier
Début
        Snotes ← 0              {initialisation de la somme}
        nbnotes← 0              {initialisation du nombre de notes}
        Afficher ("Donnez la première note : ")
        Saisir (note)
        Répéter
                COMPTER (note, Snotes, nbNotes)
                Afficher ("Donnez la note suivante : ")
                Saisir (note)
        Jusqu'à (note<0)
        MoyNotes ← MOYENNE (Snotes, nbNotes)
        Afficher (MoyNotes)
Fin
```

```
Procédure COMPTER (A, X, K)
{ajoute A à X et incrémente K de 1}
Paramètres
        (D)     A       :       réel
        (D/R)   X       :       réel
        (D/R)   K       :       entier
Début
        X←X+A
        K←K+1
Fin
```

```
Fonction MOYENNE (X, N) retourne (réel)
{calcule la moyenne W du total X par N}
Paramètres
        (D)     X       :       réel
        (D)     N       :       entier
Variable
        E : entier
        W : réel
Début
        W←X/N                           {division réelle}
        E←X/N                           {division entière}
        Si ((W-E)>0) alors              {soit on n'a pas besoin d'arrondir}
                Si ((W-E) < 0.5) alors  {soit on arrondit à 0.5 supérieur}
                        W← E+0.5
                Sinon                   {soir on arrondit à l'entier supérieur}
                        W←E+1
                Fin de Si
        Fin de Si
        Retourne (W)
Fin
```

1.7 Nouveaux types (agrégats) : quand les types standards ne suffisent plus

1.7.1 La définition de nouveaux types

- Redéfinition simple des tableaux
  Type <NomAgrégat> = tableau [1 : <max>] de <type>
- Nouvelles collections de types identiques ou hétérogènes
  Type <NomAgrégat> = agrégat
  <attribut1> : <type1>
  <attribut2> : <type2>
  …
  Fin

1.7.2 Exemples de définitions de nouveaux types

Type Mot = tableau [1:20] de caractères

Type chaîne40 = tableau [1:40] de caractères

Type Point = agrégat
Abscisse : réel
Ordonnée : réel
Fin
Type Adresse = agrégat
Numéro : entier
Voie : chaîne40
CP : chaîne5
Ville : chaîne40
Fin

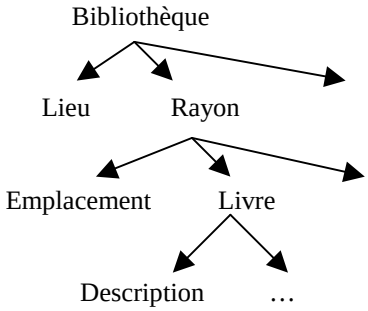
1.7.3 Utilisation des nouveaux types

- Déclaration
  <variable> : <NouveauType>
- Affectation globale
  <var1 de NouveauType> ← <var2 de NouveauType>
- Affectation d'un attribut
  <variable de NouveauType>.<attribut> ← <variable du type de l'attribut>
- Accès (saisie, affichage)
  <variable de NouveauType>.<attribut>

1.7.4 Structuration des types de données

Repérez les unités qui ont un sens afin de créer une structure modulaire.

Exemple : une bibliothèque organisée en rayons




```
Type Bibliothèque = agrégat
        Lieu : Adresse
        Salle : tableau [1 : 100] de Rayon
        Fin

Type Rayon = agrégat
        Emplacement = agrégat
                NumRangée, NumEtagère : entier
                Fin
        SesLivres : tableau [1:50] de Livre
        Fin

Type Livre = agrégat
        Description : bouquin
        PrêtLivre : Prêt
        Fin

Type Prêt = agrégat
        Nom : chaîne40
        DateDébut, DateFin : chaîne10
        Fin
…
```

1.8 Classes : quand les nouveaux types ne suffisent plus

1.8.1 Classes et instances de classes

Tout comme les nouveaux types, une classe permet de définir des agrégats de différents attributs de types différents, mais elle permet aussi de définir des outils pour travailler ses attributs : les méthodes.
Une variable d'une classe est appelée instance de la classe (objet).

1.8.2 Définition d'une classe

Classe <nom de la classe> {définition de l'objet}
Attributs : {partie descriptive de l'objet}
    <attributs>
Méthodes : {partie procédurale de l'objet}
    <Mprocédures, Mfonctions, et opérateurs redéfinis>

1.8.3 Utilisation des classes

- Déclaration
  <variable> : <classe>
- Affectation globale
  <var1 de classe> ← <var2 de la même classe>
- Utilisation des méthodes
  <variable de classe>.<méthode>
- Accès (saisie, affichage…)
  Les attributs d'une classe ne sont pas directement accessibles. L'accès se fait à l'aide de méthodes.

1.8.4 Un exemple de classe : le tableau de réels à deux dimensions

Classe T2D
Attributs :
MaxEnLigne : entier {utilisables}
MaxEnColonne : entier
NbLignes : entier {effectivement utilisées}
NbColonnes : entier
Table : [1 : MaxEnLigne ; 1 : MaxEnColonnes] de réels
Méthodes :
{méthodes de consultation des attributs de la classe}
MFonction RetourneMaxEnLigne retourne (entier)
MFonction RetourneMaxEnColonne retourne (entier)
MFonction RetourneNbLignes retourne (entier)
MFonction RetourneNbColonnes retourne (entier)
{méthodes de redéfinitions d'opérateurs et méthodes associées}
MProcédure Saisir
MProcédure Afficher
MProcédure SaisirLigne (NumLigne)
MProcédure AfficherLigne (NumLigne)
{méthodes diverses}
MProcédure InitTab (val)
MProcédure Copier (TabSource)
MProcédure Recherche (val, NumLigne, NumColonne)

MFonction RetourneMaxEnLigne retourne (entier)
{retourne la valeur de l'attribut MaxEnLigne}
Paramètres :
(D) Cible : T2D
Début
Retourne (Cible.MaxEnLigne)
Fin

MFonction RetourneMaxEnColonne retourne (entier)
{retourne la valeur de l'attribut MaxEnColonne}
Paramètres :
(D) Cible : T2D
Début
Retourne (Cible.MaxEnColonne)
Fin

MFonction RetourneNbLignes retourne (entier)
{retourne la valeur de l'attribut NbLignes}
Paramètres :
(D) Cible : T2D
Début
Retourne (Cible.NbLignes)
Fin

MFonction RetourneNbColonnes retourne (entier)
{retourne la valeur de l'attribut NbColonnes}
Paramètres :
(D) Cible : T2D
Début
Retourne (Cible.MaxEnLigne)
Fin

MProcédure Saisir
{redéfinition de l'opérateur de saisie : saisie de tous les éléments du tableau}
Paramètres :
(R) Cible : T2D {la Cible est l'objet qui déclenche cette méthode et sur lequel la méthode va travailler}
Variables :
NumLigne : entier
Début
Pour NumLigne ← 1 à Cible.NbLignes faire
SaisirLigne (NumLigne) {saisie des éléments de la ligne NumLigne de Table}
Fin de Pour
Fin

MProcédure SaisirLigne (NumLigne)
Paramètres :
(R) Cible : T2D
(D) NumLigne : entier
Variables :
Ind : entier
Début
Pour Ind ← 1 à Cible.NbColonnes faire
Saisir (Cible.Table[NumLigne, Ind]) {saisie d'un réel et affectation à une case de la table}
Fin de Pour
Fin

MProcédure Afficher
{redéfinition de l'opérateur d'affichage : affichage des valeurs de tous les attributs}
Paramètres :
(D) Cible : T2D
Variables :
NumLigne : entier
Début
Afficher (Cible.RetourneMaxEnLigne)
Afficher (Cible.RetourneMaxEnColonne)
Afficher (Cible.RetourneNbLignes)
Afficher (Cible.RetourneNbColonnes)
Pour NumLigne ← 1 à Cible.NbLignes faire
AfficherLigne (NumLigne) {affichage des éléments de la ligne NumLigne de Table}
Fin de Pour
Fin

MProcédure AfficherLigne (NumLigne)
{affiche les NbColonnes éléments de la ligne NumLigne}
Paramètres :
(D) Cible : T2D
(D) NumLigne : entier
Variables :
Ind : entier
Début
Pour Ind ← 1 à Cible.NbColonnes faire
Afficher (Cible.Table[NumLigne, Ind]) {affichage d'une réel}
Fin de Pour
Fin

MProcédure InitTab (Val)
{donne à tous les éléments de Table la valeur Val}
Paramètres :
(R) Cible : T2D
(D) Val : réel
Variables :
NumLigne : entier
NumColonne : entier
Début
Pour NumLigne ← 1 à Cible.MaxEnLigne faire
Pour NumColonne ← 1 à Cible.MaxEnColonne faire
Cible.Table[NumLigne,NumColonne] ← Val
Fin de Pour
Fin de Pour
Fin

MProcédure Copier (TabSource)
{copie tous les attributs de l'instance TabSource dans l'instance Cible}
Paramètres :
(R) Cible : T2D
(D) TabSource : T2D
Variables :
NumLigne : entier
NumColonne : entier
Début
{Cible est l'objet sur lequel on travaille, on peut donc consulter directement ses attributs
mais avec TabSource, on doit utiliser les méthodes de consultation de la classe}
Cible.MaxEnLigne ← TabSource.RetourneMaxEnLigne
Cible.MaxEnColonne ← TabSource.RetourneMaxEnColonne
Cible.NbLignes ← TabSource.RetourneNbLignes
Cible.NbColonnes ← TabSource.RetourneNbColonnes
Pour NumLigne ← 1 à Cible.NbLignes faire
Pour NumColonne ← 1 à Cible.NbColonnes faire
Cible.Table[NumLigne, NumColonne] ← TabSource.Table[NumLigne, NumColonne]
Fin de Pour
Fin de Pour
Fin

MProcédure Recherche (val, NumLigne, NumColonne)
{retourne la position de Val dans Table si Val appartient à Table, NumLigne=-1 sinon}
Paramètres :
(D) Cible : T2D
(D) val : réel
(R) NumLigne : entier
(R) NumColonne : entier
Début
NumLigne ← 0
Tant Que ((NumLigne ≤ Cible.NbLignes) ET (Cible.Table[NumLigne,NumColonne] ≠ val))
NumColonne ← 0
NumLigne ← NumLigne + 1
Tant Que ((NumColonne ≤ Cible.NbColonnes) ET (Cible.Table[NumLigne,NumColonne] ≠ val))
NumColonne ← NumColonne + 1
Fin Tant Que
Fin Tant Que
Si (Cible.Table[NumLigne,NumColonne] ≠ val) alors
NumLigne ← -1
Fin de Si
Fin

2 Les Tris

2.1 Les tris, généralités

- Entrée : une séquence de n nombres
- Sortie : un arrangement de la séquence d'entrée afin que les valeurs soient ordonnées
- Structure : clés d'enregistrements contenant aussi des données satellites

On s'intéressera aux différents principes des mécanismes de tris à l'aide d'exemples et d'algorithmes simplifiés. Les algorithmes prévus pour résoudre un même problème différent souvent énormément par leur efficacité, il faudra donc choisir l'algorithme le plus efficace selon ce qu'on souhaite trier.

2.2 Tris par comparaison

2.2.1 Tri par insertion

- Exemple : tri d'une poignée de cartes dans la main
- Exécution en $\Theta(n^2)$ dans le pire des cas
- Peu d'entrées mais opère sur place

```
Algorithme TriInsertion(A)          {A est le tableau à trier}
Pour j←2 à longueur[A] faire
    | clé←A[j]
    | {insertion de A[j] dans la séquence triée A[1..j-1]}
    | i←j-1
    | tant que i>0 et A[i]>clé faire
    |     | A[i+1]←A[i]
    |     | i←i-1
    | A[i+1]←clé
```

A :	5	2	4	6	1	3	{déplacement du 2}
	2	5	4	6	1	3	{déplacement du 4}
	2	4	5	6	1	3	{pas de déplacement du 6}
	2	4	5	6	1	3	{déplacement du 1}
	1	2	4	5	6	3	{déplacement du 3}
	1	2	3	4	5	6	

2.2.2 Tri par sélection de place

- Exécution en $\Theta(n^2)$ dans le pire des cas

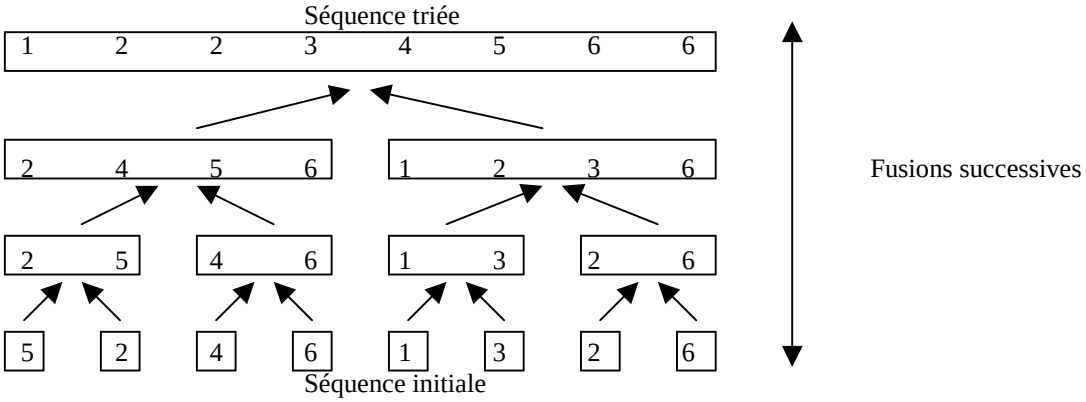
```
Algorithme TriSélection(A)          {A est le tableau à trier}
Pour i←2 à longueur[A] faire
    | Ind←Sélection(A, i)
    | Echanger(i,Ind)

Fonction Sélection(A, i) retourne(entier)
    | Indice←A[i]
    | Pour j←(indice+1) à longueur[A] faire
    |     | Si A[indice]>A[j] alors
    |     |     Indice←j
    | Retourne (indice)
```

2.2.3 Tri par fusion

- Récursif
- Exécution en Θ(n lg n)
- N'opère pas sur place

Algorithme TriFusion(A, p, r) {A est le tableau à trier, p l'indice du premier élément, r celui du dernier}
Si p<r alors
    q←((p+r)/2)
    TriFusion(A,p,q)
    TriFusion(A,q+1,r)
    Fusionner(A,p,q,r) {utiliser un tri linéaire}



2.2.4 Tri rapide

- Récursif
- Exécution en Θ(n²) mais en O(n lg n) en moyenne
- Grands tableaux d'entrées
- Tri sur place

Algorithme TriRapide(A,p,r) {A est le tableau à trier, p l'indice du premier élément, r celui du dernier}
Si p<r alors
    q←Partitionner(A,p,r)
    TriRapide(A,p,q)
    TriRapide(A,q+1,r)

Fonction Partitionner(A,p,r) retourne (entier)
x←A[p]
i←p-1
j←r+1
Tant que i≥j faire
    Répéter
        j←j-1
    Jusqu'à (A[j]≤x)
    Répéter
        i←i+1
    jusqu'à (A[i]≥x)
    Si i<j alors
        Echanger (A[i],A[j])
    Sinon
        Retourner (j)

5 3 2 6 4 1 3 7 **5 3 2 6 4 1 3** 7 3 **3 2 6 4 1** 5 7 3 3 2 **6 4 1** 5 7 3 3 2 1 4 6 5 7
i j i j i j i j j i

2.3 Tris linéaires

- Hypothèse sur la nature de l'entrée
- Exécution stable en O(n)

2.3.1 Tri par dénombrement

- Entrée : entiers dans l'intervalle 1 à k

Algorithme TriDénombrement(A,B,k) {A est le tableau à trier, B est le tableau trié, k est le maximum de l'intervalle}

```
{boucle 1 : initialisation}
Pour i←1 à k faire
        C[i]←0
{boucle 2}
Pour j←1 à longueur[A] faire
        C[A[j]]←C[A[j]]+1
        {C[i] contient à présent le nombre d'éléments égaux à i}
{boucle 3}
Pour i←2 à k faire
        C[i]←C[i]+C[i-1]
        {C[i] contient à présent le nombre d'éléments inférieurs ou égaux à i}
{boucle 4}
Pour j←longueur[A] à 1 faire
        B[C[A[j]]]←A[j]
        C[A[j]]←C[A[j]]-1
```

	A :	3	6	4	1	3	4	1	4
{boucle 2}	C :	2	0	2	3	0	1		
{boucle 3}	C :	2	2	4	7	7	8		
{boucle 4}	B :	/	/	/	/	/	/	4	/
	C :	2	2	4	**6**	7	8		
	…								
	B :	1	1	3	3	4	4	4	6

2.3.2 Tri par base

- Exemple : Tri des machines à trier les cartes perforées : 80 colonnes, 1 trou à une position parmi 12, position croissante de chiffres significatifs

Algorithme TriBase(A,d) {A est le tableau à trier, d est le nombre de chiffres}

```
Pour i←1 à d faire
    |   Utiliser un tri stable pour trier A selon i
```

329	720	720	329
457	355	329	355
657	436	436	436
839 →	457 →	839 →	457
436	657	355	657
720	329	457	720
355	839	657	839

2.3.3 Tri par paquets

- Entrée : réels dans [0 ;1[

Algorithme TriPaquets(A) {A est le tableau à trier}
n←longueur[A]
Pour i←1 à n faire
    Insérer A[i] dans la liste B[nA[i]]
Pour i← 0 à n-1 faire
    Trier la liste B[i] à l'aide du tri par insertion
Concaténer les listes B[0], B[1], …B[n-1] dans l'ordre

	A	B	
1	.78	0	
2	.17	1	→.12
3	.39	2	→.21→.23→.26
4	.26	3	→.39
5	.72	4	
6	.94	5	
7	.21	6	→.68
8	.12	7	→.72→.78
9	.23	8	
10	.68	9	→.94

3 La récursivité

3.1 Le principe : diviser pour régner

- Diviser le problème en sous-problèmes
- Régner sur les sous-problèmes en les résolvant récursivement ou directement
- Combiner les solutions aux sous-problèmes en une solution complète

Le raisonnement itératif :
Expliciter dans le schéma de l'algorithme que l'on résoud un problème en N étapes, au cours desquelles le même travail est réalisé sur des variables différentes.

Le raisonnement récursif :
Expliciter dans le schéma de l'algorithme la spécificité de la $N^{ième}$ étape, exprimée en faisant appel globalement au travail réalisé par ce même algorithme au cours des N-1 autres étapes.

3.2 Exemples de traitements récursifs

3.2.1 Calcul du Plus Grand Commun Diviseur (PGCD) de deux nombres

- Définition mathématique

Soient deux nombres a et b.
Si a est positif,
le PGCD de a et de 0 est a ;
Si b est strictement positif et a est positif,
le PGCD de a et b est égal au PGCDD de b et du reste de la division entière de a par b.

- Exemple

A=30, b=18
PGCD (30,18) = PGCD(18,12) = PGCD(12,6) = PGCD(6,0)
Le PGCD de 30 et de 18 est donc égal à 6

- Algorithme

Fonction PGCD(A,B) retourne (entier)
{retourne le PGCD de A et B}
Paramètres :
(D) A, B : entier
Variables :
Résultat : entier
Début
Si (B = 0) alors {condition d'arrêt}
Résultat ← A
Sinon
Résultat ← PGCD (B, A mod B)
Fin de Si
Retourne (Résultat)
Fin

3.2.2 Puissance d'un nombre

Fonction PUISSANCE (x, puiss) retourne (entier)
{retourne x^{puiss}}
Paramètres :
(D) x, puiss : entier
Variables :
Résultat : entier
Début
Si (puiss = 1) alors {condition d'arrêt}
Résultat ← x
Sinon
*Résultat ← x * PUISSANCE (x, (puiss – 1))*
Fin de Si
Retourne (Résultat)
Fin

3.2.3 Le jeu des tours de Hanoï

- Principe

On dispose d'un plateau sur lequel sont fixés trois axes. Sur l'un des axes sont enfilés, par taille décroissante, NbDisques disques de tailles différentes et toutes distinctes. L'objectif est d'empiler les disques de cette même façon sur l'un des deux autres axes en déplaçant les disques un par un, sans jamais passer par une situation où les disques enfilés ne respectent pas un empilement pyramidal.

- Algorithme

Procédure HANOÏ (A, B, C, NbDisques)
{déplace NbDisques disques de A vers B en utilisant C comme axe auxiliaire}
Paramètres :
(D) NbDisques : entier
(D) A, B, C : caractère
Début
Si (NbDisques = 1) alors {condition d'arrêt}
Afficher ("Déplacez un disque de ", A, " vers ", B)
Sinon
{déplacer tous les disques sauf le (NbDisques)ième de l'axe de départ vers l'axe auxiliaire}
HANOÏ (A, C, B, NbDisques-1)

{déplacer le (NbDisques)ième disque vers l'axe destinataire}
Afficher ("Déplacez un disque de ", A, " vers ", B)

{déplacer tous les disques de l'axe auxiliaire vers l'axe destinataire}
HANOÏ (C, B, A, NbDisques-1)
Fin de Si
Fin