

Algorithmique
Module INFO 0401
2^{ème} année informatique
Reims – Sciences Exactes

Cours n°1

A.HEBBACHE

Plan

1 Initiation à l'algorithme et à la complexité

2 Types de données abstraits

3 Types de données séquentielles

4 Pile , File, Liste linéaire chaînés

6 Types de données arborescentes

Plan

7

Les arbres binaires

8

Les AVL

9

Les arbres bicolores

10

Les algorithmes de tri

10

Les graphes

Sommaire

1 1. Introduction

2 2. Généralités sur l'algorithme

2.1 Quelques définitions du mot algorithme

2.2 But d'un algorithme

2.3 Intérêt

2.4 Propriétés d'un algorithme

3 3. Résolution de problèmes

4 4. Programme

4.1 Définition

4.2 Mise en œuvre d'un programme

5 5. Définir les objets élémentaires manipulés par un algorithme

Sommaire

5.1 Variable et constante

5.2 Les types

6 6. Les opérations

6.1 L'affectation

6.2 Lecture

6.3 Ecriture

7 7. La structure d'un algorithme

8 8. Les tests

9 9. Les boucles

10 10. Procédures et fonctions

11 11. Les différents types d'algorithmes

1.Introduction

➤ Al Khawarizmi



- Célèbre mathématicien à Bagdad, vers 780-850.
- « Kitâb **al-jabr** wa al-muqâbala ». Livre sur la science de la transposition et de la réduction : résolution systématique de l'équation du second degré.
- Traduit en latin au 12^e siècle par Gherardo di Cremona sous le titre « ***Dixit Algorismi*** ».
- Aussi : « Kitâb al Jami wa al Tafriq bi Hisab al Hind ». Livre de l'addition et de la soustraction d'après le calcul des indiens.
- [http ://trucsmaths.free.fr/alkhwarizmi.htm](http://trucsmaths.free.fr/alkhwarizmi.htm)
- <http://publimath.irem.univ-mrs.fr/glossaire/AL016.htm>

2. Généralités sur l'algorithme

2.1. Quelques définitions du mot algorithme

- Le mot « **algorithme** » vient du nom de l'auteur persan **Al-Khuwarizmi** (né vers 780 - mort vers 850) qui a écrit en langue arabe le plus ancien traité d'algèbre « abrégé de calcul par la complétion et la simplification » dans lequel il décrivait des procédés de calcul à suivre étape par étape pour résoudre des problèmes ramenés à des équations.
- un **algorithme** est une suite finie, séquentielle de règles que l'on applique à un nombre fini de données, permettant de résoudre des classes de problèmes semblables.

2. Généralités sur l'algorithme

2.1 Quelques définitions du mot algorithme (suite)

- Algorithme = description des étapes de la méthode utilisée
- Résultat d'une démarche logique de résolution d'un problème.
- C' est le résultat de l'analyse.
- Procédé de calcul, qui permet à partir de données numériques et en suivant un plan de calcul très précis, d'obtenir le résultat d'un calcul.

2. Généralités sur l'algorithme

2.2 But d'un algorithme

Résolution d'un problème algébrique, numérique ou décisionnel.

2.3 Intérêt

Explicite clairement les idées de solutions d'un problème indépendamment d'un langage de programmation.

2. Généralités sur l'algorithme

2.3 Propriétés d'un algorithme

En général un algorithme doit toujours être conçu de manière à envisager toutes les éventualités d'un traitement.

➤ **Finitude**

Un algorithme doit s'arrêter au bout d'un temps fini.

➤ **Dé finitude**

Toutes les opérations d'un algorithmes doivent être définies sans ambiguïtés

➤ **Répétitivité**

Généralement, un algorithme contient plusieurs itérations, c'est à dire des actions qui se répètent plusieurs fois.

➤ **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.

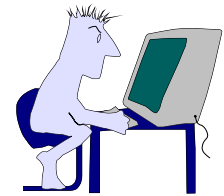
3. Résolution de problèmes

- **Lire** l'énoncé du problème, être certain de bien le comprendre
 - ressortir les informations pertinentes
 - données ? résultats ?
- **Réfléchir** à la résolution du problème en ignorant l'existence de l'ordinateur
 - déterminer les points principaux à traiter
 - exploiter l'ensemble de vos connaissances
 - adapter ou réutiliser des recettes existantes
 - encore difficile ?

Décomposer le problème!! *Analyse descendante* !!

3. Résolution de problèmes

- **Écrire** formellement la solution (algorithme) sur papier
 - utiliser un pseudo langage
- **Vérifier** votre solution sur un exemple
 - preuve formelle de l'algorithme : encore mieux !!
- **Traduire** dans un langage de programmation
- **Tester** le programme sur différents jeux de tests
 - le jeux de tests doit être « suffisant »
 - ne pas oublier les cas particuliers, ...



3. Résolution de problèmes

Exemple : Conversion de nombre

- Énoncé

Écrire un programme qui simule la conversion d'un nombre décimale en tant que chaîne de caractères en sa valeur réelle.

Exemples de nombres à lire : 122,51 12 (notation décimale)

- Analyse

- Les nombres ne sont pas des valeurs isolées.
- Font partie d'un texte.

Exemples :

- « la valeur de x est 1526.32 »
- « 12 * 5.01 + x = y »

 Le programme doit :

- lire tous les caractères jusqu'au premier caractère chiffre ;
- lire la partie entière du nombre,
- et s'il trouve le point, la partie décimale

3. Résolution de problèmes

- La construction de l'algorithme de conversion de nombre

- 1. Rechercher un chiffre

- répéter*** lire un caractère ***jusqu'à*** trouver un caractère chiffre.

- 2. Lire la partie entière du nombre

- répéter***

- 2.1. convertir le caractère chiffre en sa valeur entière

- 2.2. rajouter cette valeur à (nombre déjà calculé)*10

- 2.3. lire le caractère suivant

- jusqu'à*** ce que le caractère lu ne soit plus un caractère chiffre

3. Résolution de problèmes

- La construction de l'algorithme de conversion de nombre (suite)

3. Lire la partie décimale

si le caractère lu précédemment est un point

alors

3.1. lire le caractère suivant

tant que le caractère lu est un caractère chiffre faire

3.1.1. convertir le caractère chiffre en sa valeur entière

3.1.2. rajouter cette valeur à (nombre déjà calculé)*10

3.1.3. lire le caractère suivant

fin tant que

3.2. diviser le nombre obtenu par 10 puissance (le nombre de chiffres lus après la virgule)

fin si

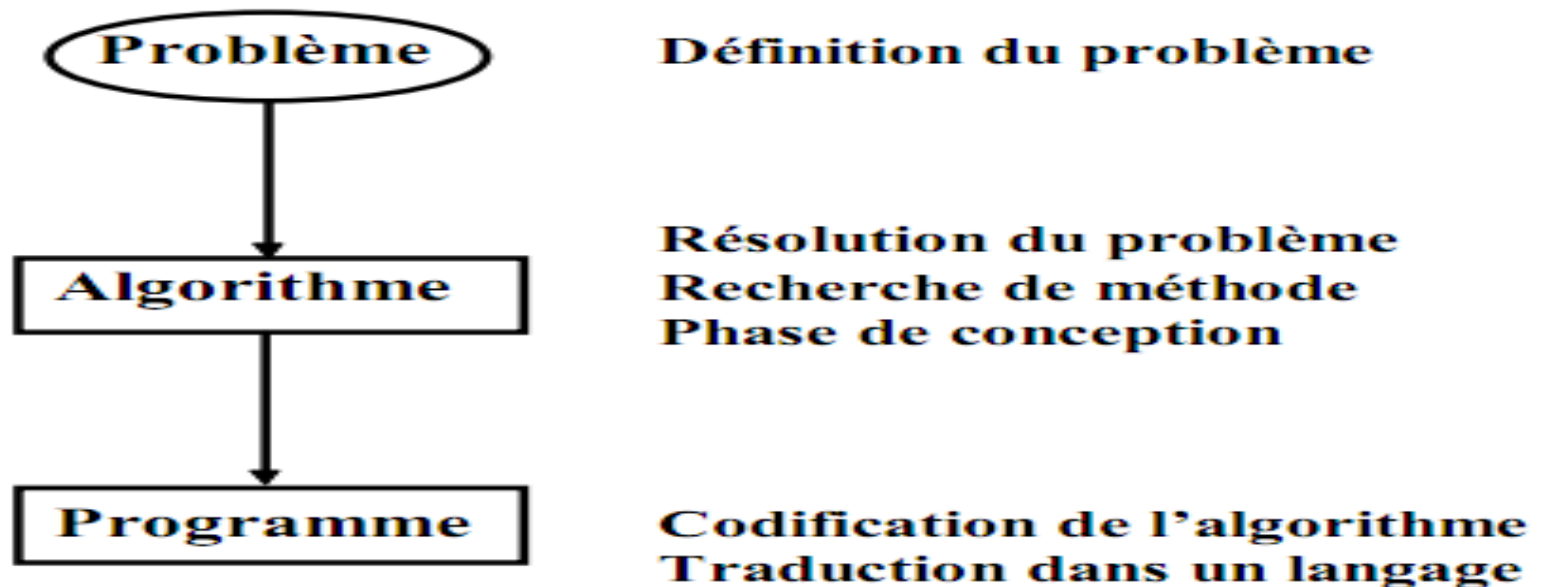
3. Résolution de problèmes

- L'algorithme final de conversion de nombre
- Algorithme
 1. rechercher un chiffre
 2. lire la partie entière du nombre
 3. **si** il y a un point décimal
 - alors**
 - 3.1. lire les chiffres suivants ce point
 - 3.2. remettre à l'échelle ce nombre
 - fin si**
 4. écrire le résultat

4. Programme

4.1 définition

Programmer consiste à rechercher et à exprimer, pour un problème donné, un algorithme compréhensible (exécutable) par l'ordinateur



4. Programme

Exemple 1

- *Un algorithme de résolution de l'équation $ax+b = 0$*

Données : *a et b entiers*

Algorithme :

Écrire(' résolution de l 'équation : $ax+b=0$ ')

lire(a), lire(b)

Si a est non nul,

alors on obtient la solution : $x = -b/a$

sinon si b est non nul,

alors l'équation est insoluble

sinon tout réel est solution

Résultat :

La solution de l'équation $ax+b=0$; si elle existe

4. Programme

Remarque

L'algorithme de calcul de l'exemple 1 ne peut être exécuté par une machine! (qui ne comprend que des 0 et des 1)

↳ Besoin d'un langage de programmation

Un programme est la traduction d'un algorithme dans un langage informatique particulier et parfaitement précis.

4. Programme

4.2 Mise en œuvre d'un programme

- Etape 1 : Définition du problème
- Etape 2 : Analyse du problème
- Etape 3 : Ecriture d'un algorithme avec un langage de description algorithmique.
- Etape 4 : Traduction de l'algorithme dans un langage de programmation.
- Etape 5 : Mise au point du programme

4. Programme

4.2 Mise en œuvre d'un programme

✓ Remarques

- La machine corrige l'orthographe, c'est ce qu'on appelle **syntaxe** dans le jargon de la programmation.
- La machine traduit le sens exprimé par le programme c'est ce qu'on appelle la **sémantique**.
- Si nous n'obtenons pas de résultats, on dira qu'il y a existence des **erreurs de logique**.

Dans le cas où l'algorithme ne donne pas de résultat :

- Revoir en priorité si l'algorithme a été bien traduit (**programmation**)
- Sinon revoir l'étape analyse (**méthode**).

4. Programme

4.2 Mise en œuvre d'un programme

❖ Exemple 1

Algorithme :

debut

Écrire(' Résolution de l'équation : $ax+b=0$ ')

lire(a), lire(b)

Si a est non nul,

alors on obtient la solution : $x = -b/a$

sinon si b est non nul,

alors l'équation est insoluble

sinon tout réel est solution

fin

Choisir



Programme

```
equation (int input, output);
```

```
{int a,b;
```

```
printf("Résolution de l'équation  
:  $ax+b=0$ ");
```

```
printf(' 'a ? ');scanf(a);
```

```
printf(' 'b? '); scanf (b);
```

```
if ( a !=0)
```

```
{ printf(" la solution est x =", -b/a)
```

```
else if (b!=0)
```

```
printf("insoluble");
```

```
else printf(" tout réel est  
solution");
```

```
}
```

4. Programme

4.2 Mise en œuvre d'un programme

❖ Exemple 2:

Calcul du PGCD de deux entiers positifs :

Données : a et b entiers positifs

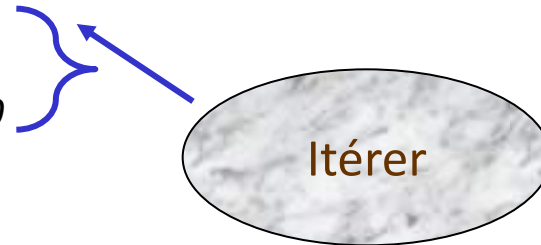
Algorithme :

Appliquer récursivement les deux règles suivantes :

$R1) - \text{PGCD}(a, 0) = a$

$R2) - \text{PGCD}(a, b) = \text{PGCD}(b, a \bmod b), b \neq 0$

Résultat : le pgcd entre a et b



Exécution : $\text{PGCD}(36, 24)?$

$$\text{PGCD}(36, 24) = {}^{R2} \text{PGCD}(24, 12) = {}^{R2} \text{PGCD}(12, 0) = {}^{R1} 12$$

4. Programme

4.2 Mise en œuvre d'un programme

program Calcul_pgcd (input, output)

Int main()

{

int a,b, reste;

 printf(a ? ');scanf(a);

 printf(b ? ');scanf(b);

 while(b!=0)

 {

reste=a mod b;

 a= b;

 b=reste;

 }

 printf(' pgcd = ',a);

}

5. Définir les objets élémentaires manipulés par un algorithme

5.1 Variable et constante

- Un algorithme opère sur des objets.
- A tout objet est associé un nom qui permet de l'identifier de façon unique. C'est une suite de caractères alphanumériques.
- Des objets qui peuvent varier durant le déroulement d'un algorithme : **Variables**.
- Des objets qui ne peuvent pas varier par le déroulement d'un algorithme : **Constantes**.

5. Définir les objets élémentaires manipulés par un algorithme

5.2 les types

On peut répartir l'ensemble des objets en sous ensembles appelés **classe** ou **type**.

Il existe 4 type standard :

- ENTIER : L'ensemble des entiers relatifs
- REEL : L'ensemble des réels
- BOOLEEN : Les valeurs Vrai et Faux
- CAR : L'ensemble des chaînes de caractères.

6. les opérations

6.1 Affectation

C'est l'opération de base. Elle permet d'affecter la valeur d'une expression calculable à une variable. On utilisera le symbole '<-'

Exemple : **A ← B / D**

6.2 Lecture

Elle permet d'introduire les données dans les variables. Nous utiliserons instruction

Exemple : **LIRE (X, Y, ...)**

Ce qui est équivalente à

- **X ← première donnée**
- **Y ← deuxième donnée**

6. les opérations

6.3 Ecriture

Elle permet de restituer les résultats. Nous utiliserons l'opération.

Exemple:

ECRIRE(Expression1, Expression2, ...)

7. Structure d'un algorithme

➤ **DECLARATION**

- des variables

➤ **ENTREE**

- avec éventuellement initialisation des variables

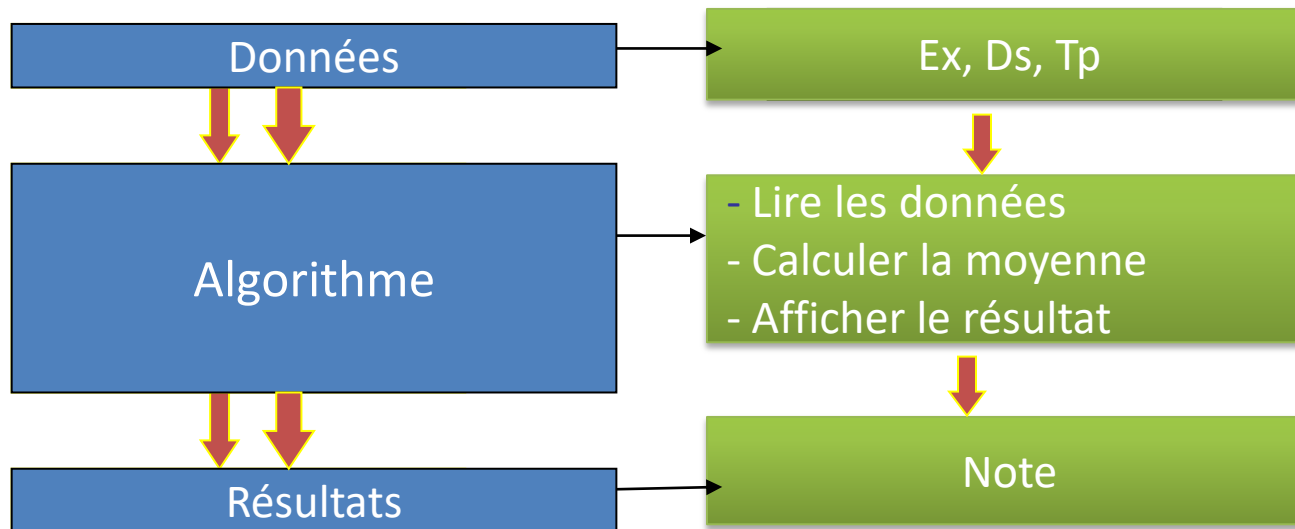
➤ **TRAITEMENT**

- dont on décrit la procédure, c'est le corps de l'algorithme

➤ **SORTIE**

- avec un affichage de ce que l'on souhaite.

7. Structure d'un algorithme



7. Structure d'un algorithme


Exemple

- **Déclaration :**

const NOM_CONST = valeur;

type nomVar ;

type de la variable
(int, float, char)

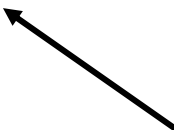


Exemples :

const PI = 3.14;

Int rayon, surface;

nom de la variable
(identificateur)



8 .Les instructions conditionnelles

En pseudo :

```
si condition
    alors instruction_1
    sinon instruction_2;
fin si
```

En C :

```
if condition
    instruction_1 ;
    instruction_2;
```

condition est une expression booléenne

Exemple :

```
if (nb mod 2 == 1)
{
    printf (" le nombre est impair");
}
else
{
    printf("Le nombre' ", nb ,' est pair");
}
```


9. Boucles

En **pseudo** : tantque (condition) faire instructions; fin tanque

En **C** : while (condition) instructions;

➤ *condition* est une expression booléenne

Exemple :

Main()

{

int a, b, reste;

a=36;

b=24;

while (b!=0)

 {

reste=a mod b;

a= b;

b=reste;

 }

printf("Le pgcd('a', 'b')=", a);

}

10. Procédures et fonctions

- Si vous avez un gros programme à mettre au point, et que certaines parties sont semblables, ou d'autres très complexes , alors il faut le structurer et chercher à utiliser au maximum "l'existant".
 - Utiliser une procédure (sous-programme) si un même traitement est effectué à plusieurs reprises dans le programme.
- ↳ permet d'alléger le programme, de faciliter sa maintenance, de le structurer et d'améliorer sa lisibilité.

10. Procédures et fonctions

- **Quand réaliser une procédure ?**

On doit réaliser une procédure à chaque fois que l'analyse d'un problème conduit à identifier une tâche.

- **Quand réaliser une fonction?**

En général, le rôle d'une fonction est le même que celui d'une procédure.

Il y a cependant quelques différences :

- renvoi d'une valeur unique
- la fonction est typée
 - Type de la fonction : scalaire, réel, intervalle, string, structure

10. Procédures et fonctions

Exemple 1

```
int puissance(int a,b) {  
    int a,b;           // données  
    // résultat: a puissance b  
    // spécifications : élever a à la puissance b  
    int p, i;  
    p=1;  
    for( i=1; i<=b; i++) {  
        p=p*a;  
    }  
    puissance = p;  
}
```

10. Procédures et fonctions

Exemple 2 : Afficher récursivement les éléments d'une suite

afficherSuite(int n)

début

 si $n=0$ alors ne rien faire ;

 Sinon afficher n ;

 afficherSuite($n-1$) ;

finSi

finProcédure

11. Les différents types d'algorithmes

On distingue deux types d'algorithmes :

Algorithme séquentiel: Les opérations élémentaires sont exécutées de manière séquentielle c.-à-d. l'une après l'autre, une seule fois.

Algorithme parallèle: Des opérations élémentaires peuvent être exécutées en même temps.

- **Remarque :**

L'étude de l'efficacité d'un algorithme porte sur deux principaux facteurs : le temps d'exécution, et l'espace mémoire nécessaire pour résoudre un problème.

Ces deux facteurs seront mesurés en fonction de la taille du problème en entrée.

11. Les différents types d'algorithmes

Programme – Itératif vs récursif

Les langages de programmation offrent tous des structures de programmes itératives (boucles).

Un programme qui en emploie est dit **itératif**.

Un sous-programme (procédure ou fonction) est dit **récursif** s'il est susceptible de s'appeler lui-même directement (récursivité directe) ou indirectement via l'appel d'autres sous programmes (récursivité indirecte).

11. Les différents types d'algorithmes

Programme – Itératif vs récursif

Action récursive

« Une action A est exprimée de façon récursive si la décomposition de A fait appel à A. »

Exemple

➤ Action calculer $S(n)$:

Appliquer les deux règles suivantes:

$$1) S(0) = 0$$

$$2) S(n) = S(n-1) + n, n > 0$$

➤ $S(3)$?

$$= S(2) + 3 = (S(1) + 2) + 3$$

$$= ((S(0) + 1) + 2) + 3 = ((0 + 1) + 2) + 3 = 6$$

11. Les différents types d'algorithmes

Programme – Itératif vs récursif

- Objet récursif

« La définition d'un objet est récursive lorsqu'elle se formule en utilisant l'objet même qu'elle entend définir. »

- Exemple (déf. récursive d'une chaîne)

Une chaîne de caractère est :

- soit la chaîne vide
- soit un caractère suivi d'une chaîne de caractère

11. Les différents types d'algorithmes

Programme – Itératif vs récursif

La récursivité est un mécanisme puissant de définition d'objets !

Itération

```
procédure Itération( )  
  Tantque (condition) faire  
    <Instructions>  
fin tantque
```

```
fonction S(n) : entier  
  S ← 0  
  Tant que (n>0) faire  
    S ← S+n  
    n ← n-1  
  fin tant que  
  retourner S
```

Récursivité

```
procédure Itération( )  
  Si (condition) alors  
    <Instructions>  
    Itération()  
  fin si
```

```
fonction S(n) : entier  
  Si (n=0)  
    alors S ← 0  
    sinon S ← S(n-1)+n  
  fin si
```

11. Les différents types d'algorithmes

Programme – Itératif vs récursif

- Exemple 1 : Calcul de la factorielle

Équations de récurrences :

- $0! = 1$ (base)
- $n! = n(n-1)!$ (récurrence)

fonction fact(n:entier): entier

début

si $n = 0$

alors fact $\leftarrow 1$

*sinon fact $\leftarrow n * \text{fact}(n-1)$*

fin si

fin

✱ Que se passe-t-il si $n < 0$? \Rightarrow *récursivité infinie*

11. Les différents types d'algorithmes

Programme – Itératif vs récursif

Exemple 2 : Suite de fibonacci

Équations de récurrences :

- $u(0) = 0, u(1) = 1$ (Base)
- $u(n) = u(n-1) + u(n-2), n > 1$ (récurrence)

fonction fib(n:entier) : entier

début

si (n=0) ou (n=1)

alors fib \leftarrow n

sinon fib \leftarrow (n-1) + fib(n-2)

fin si

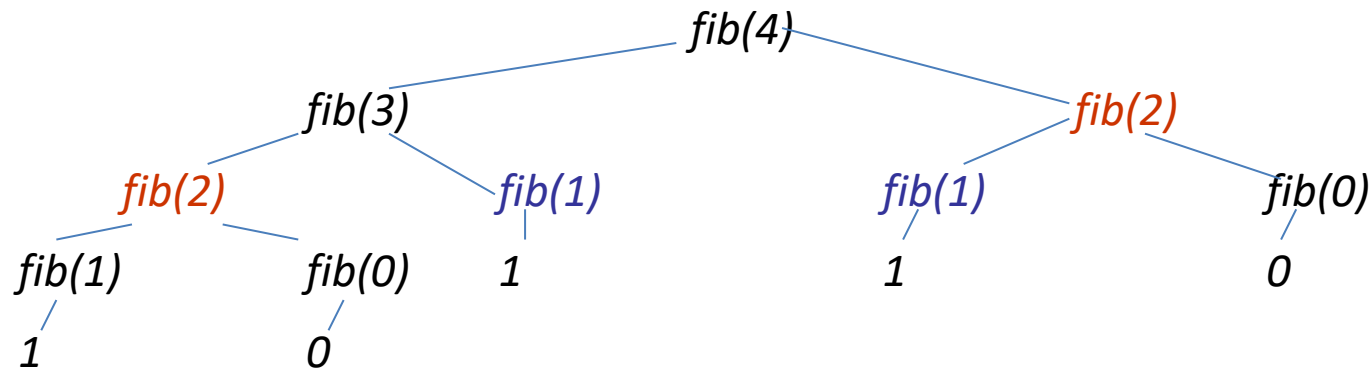
fin

11. Les différents types d'algorithmes

Programme – Itératif vs récursif

- Exemple 2 (suite)

Voyons l'exécution : $\text{fib}(4)$?



** Une solution récursive n'est pas toujours viable*

11. Les différents types d'algorithmes

Programme – Itératif vs récursif

Exemple 2 (suite) : solution itérative

fonction fib(n:entier) : entier

début

var x, y, z, i : entier

si (n=0) ou (n=1) alors retourner (n)

sinon x ← 0 {fib(0)}

y ← 1 {fib(1)}

pour i allant de 2 à n faire

z ← x+y {fib(n) = fib(n-1)+fib(n-2)}

x ← y

y ← z

fin pour

retourner (z)

fin si

fin

Conclusion

La récursivité est un principe puissant nous permettant de :

- **définir des structures de données complexes**
- **de simplifier les algorithmes opérant sur ces structures de données**