Université Félix Houphouët-Boigny
(UFHB)

Cours d'algorithme avancée
L3-Info

Enseignant :

Dr GBAME Gbede Sylvain

Assistant, enseignant chercheur à l'UFHB

gbamegbedesylvain@gmail.com

MODULE : Algorithme avancée

PLAN

Chapitre 6

Les sous-algorithme-Partie 4- Notions de récursivité

6.1 Notion de récursivités

1. Exemple introductif

Calculer la factorielle d'un entier naturel.
En mathématiques, il existe deux méthodes pour calculer la factorielle d'un entier naturel n.

❖ **Première méthode** : selon la formule classique (cf. définition de la factorielle)

Si n = 0 ou n = 1, alors n! = 1
Sinon n! = n × (n -1) × (n -2) × … × 2 × 1 = 1 × 2 × … × (n - 2) × (n - 1) × n.

Ainsi : **5!** = 5 × 4 × 3 × 2 × 1 = 1 × 2 × 3 × 4 × 5 = 120.

Les algorithmes mettant en œuvre cette méthode utilisent des boucles. Ce sont des ***algorithmes itératifs***.

6.1 Notion de récursivités

❖ **Deuxième méthode** : selon une formule de récurrence

On démontre à partir de la formule précédente que :

Si n = 0 alors n! = 1 Sinon **n! = n × (n - 1)!**

Or : (n-1)! = (n-1) × (n-2)! avec (n-2)! = (n-2) × (n-3)! et ainsi de suite jusqu'à 0!

Donc :

Calculer la factorielle d'un entier naturel revient à multiplier cet entier naturel par la factorielle de son précédent : on utilise une factorielle pour calculer une factorielle.

6.1 Notion de récursivités

Ainsi : **5!** = 5 × **4!**

= 5 × (4 × **3!**)

= 5 × [4 × (3 × **2!**)]

= 5 × [4 × [3 × (2 × **1!**)]]

= 5 × [4 × [3 × [2 × (1 × **0!**)]]]

= 5 × [4 × [3 × [2 × (1 × 1)]]]

= 5 × 4 × 3 × 2 × 1

5! = 120

Chaque étape demande le calcul d'une autre factorielle. On doit alors écrire un algorithme de calcul de factorielle *qui va s'appeler lui-même* : c'est un algorithme ***récursif***.

6.1 Notion de récursivités

2. Définitions

❖ La **récursivité** est la propriété pour un algorithme de s'appeler lui-même un nombre fini de fois. Un algorithme ayant une telle propriété est un ***algorithme récursif.***

❖ Un ***algorithme récursif*** est un algorithme ***A*** qui s'appelle lui-même ou qui appelle un autre algorithme *A'* contenant un appel de ***A***.

3. L'intérêt de la récursivité

La récursivité est un puissant outil algorithmique (et mathématique où elle est désignée par le terme *récurrence*) qui permet de décomposer un problème en plusieurs problèmes de même nature mais sur des données plus petites.

Elle est très adaptée à la résolution de certains problèmes n'appartenant pas forcément au domaine des mathématiques (par exemples : la recherche d'un élément dans un tableau trié, le jeu des Tours de Hanoï).

6.1 Notion de récursivités

4. Propriétés des algorithmes récursifs

Un algorithme récursif a deux propriétés fondamentales :

- il possède des conditions d'arrêt des appels ;
- chaque appel direct ou indirect le rapproche de ses conditions d'arrêt.

5. Les principaux types de récursivité

Il existe plusieurs types de récursivité dont les plus courants sont :

5.1. la récursivité directe (ou récursivité simple)

Elle a lieu quand un algorithme s'appelle lui-même.

Exemple : calcul de la factorielle d'un nombre.

6.1 Notion de récursivités

5.2. la récursivité multiple

Elle a lieu quand un algorithme s'appelle lui-même au moins deux fois simultanément.

Exemple : calcul d'un terme de la suite de Fibonacci.

La suite de Fibonacci **(Fn) n ∈ N** est définie par : $F_0 = 0$, $F_1 = 1$, et la relation de récurrence

$$F_n = F_{n-1} + F_{n-2} \quad \text{pour} \quad n \geq 2$$

Les cinq premiers termes de la suite de Fibonacci sont :

1. $F_0 = 0$
2. $F_1 = 1$
3. $F_2 = 1$
4. $F_3 = 2$
5. $F_4 = 3$

6.1 Notion de récursivités

5.3. la récursivité imbriquée

Elle a lieu quand un algorithme appelé dans un autre algorithme s'appelle lui-même (donc elle consiste à effectuer un appel récursif à l'intérieur d'un autre appel récursif).

Exemple : calcul d'une valeur de la fonction d'Ackermann

5.4. la récursivité croisée (ou récursivité indirecte ou récursivité mutuelle)

Elle a lieu quand un algorithme est appelé via une série d'appels d'algorithmes qu'il a lui-même initiés. (Cas simple : l'algorithme A_1 appelle l'algorithme A_2 , et A_2 appelle A_1 .)

Exemple : détermination de la parité d'un nombre.

Remarque

Le terme *algorithme récursif* fait référence aux *sous-algorithmes récursifs*, c'est-à-dire aux *procédures récursives* et *fonctions récursives*. Toutefois, dans la pratique, les *fonctions récursives* sont plus utilisées que les procédures récursives.

6.1 Notion de récursivités

6. Concevoir et écrire une fonction récursive

Écrire une fonction récursive est délicat. Il faut suivre rigoureusement certaines règles, notamment :

6.1. vérifier la décomposabilité du problème

Le problème doit être décomposable en problèmes de même nature sur des données plus petites.

6.2. déterminer la (les) condition(s) d'arrêt des appels

Il faut déterminer la (ou les) condition d'arrêt (ou cas de base) afin de définir le (les) cas pour lequel la fonction ne s'appelle pas elle-même. Si on ne définit pas la (les) condition(s) d'arrêt, la fonction ne terminera pas, c'est-à-dire que la fonction s'appellera indéfiniment.
Une condition d'arrêt provoque l'arrêt des appels récursifs.

6.1 Notion de récursivités

6.3. déterminer la (les) condition(s) de continuité des appels

Il faut déterminer la (les) condition(s) de continuité (ou cas inductif) afin de définir le (les) cas pour lequel la fonction s'appelle elle-même. Une condition de continuité provoque un appel récursif.

6.4. écrire la fonction récursive

Dans le corps de la fonction récursive, on écrit l'instruction (ou les instructions) contenant le (les) cas de base avant celle(s) contenant le (les) cas inductif(s).
Chaque appel récursif doit "*se rapprocher*" d'un cas de base de sorte à favoriser la terminaison de la fonction.
Techniquement, on utilise la fonction que l'on n'a pas encore écrite en supposant qu'elle donne déjà un résultat. Une fonction récursive se compose de deux parties :

- la (les) condition(s) d'arrêts des appels récursifs : dans cette partie les valeurs à déterminer sont directement connues ;
- un (des) appel(s) récursif(s).

6.1 Notion de récursivités

D'où le schéma général d'une fonction récursive :

```
Fonction frecursiv (liste des paramètres formels avec leurs types) : type de la valeur de retour
Début
    Si (condition d'arrêt)
    Alors retourner (r) // valeur de retour de la condition d'arrêt
    Sinon retourner (frecursiv (liste des nouveaux paramètres))
    FinSi
Fin
```

L'instruction contenant l'appel récursif peut se présenter sous deux formes telles que :

- ❖ la fonction récursive retourne, sans aucun autre calcul, la valeur obtenue par son appel récursif. On parle alors de récursivité terminale.
- ❖ la fonction récursive retourne, après un autre calcul, la valeur obtenue par son appel récursif. On parle alors de récursivité non terminale.

Exemple : l'appel récursif de la fonction de calcul de la factorielle d'un nombre.

6.1 Notion de récursivités

7. Applications

Exemple d'application 1 : Calcul récursif de la factorielle d'un nombre

Écrire une fonction qui calcule la factorielle d'un entier naturel n selon la formule de récurrence $n! = n \times (n-1)!$ sachant que $0! = 1$.

Analyse et méthode

On utilisera une fonction de type entier. De plus :

- puisque $n! = n \times (n-1)!$ et que $(n-1)$ décroît jusqu'à atteindre la valeur 0, alors le problème est décomposable en problèmes de même nature sur des données plus petites ;
- le résultat de $0!$ est directement fourni par la formule de récurrence, alors 0 est le cas de base ; en effet $0! = 1$;
- pour $n \geq 1$, le calcul de $n!$ nécessite une décomposition en un produit contenant une factorielle, alors tout entier naturel supérieur ou égal à 1 est un cas inductif.

6.1 Notion de récursivités

Donc : la fonction peut être écrite sous forme récursive avec pour cas de base ***: n = 0***, et pour cas inductif : ***n ≥ 1***.

Puisque cette fonction s'appelle elle-même directement jusqu'à l'atteinte de sa condition d'arrêt alors c'est un cas de *récursivité directe* (ou *récursivité simple*).

```
Fonction factorielle (n : entier) : entier
Début
    Si (n = 0)
    Alors retourner (1)
    Sinon retourner (n*factorielle (n - 1)) // appel récursif de la fonction factorielle
    FinSi
Fin
```

6.1 Notion de récursivités

Peut-on écrire cette fonction *sous forme récursive* ?

❖ puisque la suite de Fibonacci est une suite dans laquelle chaque terme est la somme des deux termes précédents, alors le problème est décomposable en problèmes de même nature sur des données plus petites ;

❖ l'obtention des termes de rang 0 et de rang 1 (n = 0 et n = 1) de la suite ne nécessite pas de calcul. Alors 0 et 1 sont les cas de base (voir énoncé) ;

❖ tout entier naturel strictement supérieur à 1 est un cas inductif (voir énoncé).

Donc :

- la fonction peut être écrite sous forme récursive ;
- cas de base : $n = 0$ et $n = 1$;
- cas inductif : $n > 1$.

De plus, d'après l'énoncé, cette fonction s'appelle elle-même simultanément deux fois de suite jusqu'à l'atteinte de ses conditions d'arrêt : c'est un cas de récursivité double.

D'où la fonction récursive :

6.1 Notion de récursivités

La fonction récursive retourne la valeur obtenue par son appel récursif après un autre calcul ***(n*factorielle (n - 1))***. Cette fonction est alors une fonction récursive non terminale.

Exemple d'application 2 : Calcul d'un terme de la suite de Fibonacci

Écrire une fonction qui calcule le terme de la suite F_n de Fibonacci pour le nombre n sachant que, n étant un entier naturel :

- si n = 0 ou n = 1 alors F_n = n
- si n > 1 alors $F_n = F_n - 1 + F_n - 2$.

Analyse et méthode

Le terme de rang n de la suite F_n de Fibonacci est un entier naturel. On peut alors le calculer à l'aide d'une fonction de type entier.

Peut-on écrire cette fonction sous forme récursive ?

6.1 Notion de récursivités

```
Fonction fibonacci (n : entier) : entier
Début
    Si ((n = 0) OU (n = 1))
    Alors retourner (n)
    Sinon retourner (fibonacci (n – 1) + fibonacci (n – 2)) // double appel de la fonction
    FinSi
Fin
```

6.1 Notion de récursivités

Exemple d'application 3 : **Calcul d'une valeur de la fonction d'Ackermann**

Étant donné les entiers naturels m et n, on définit la fonction d'Ackermann $A(m, n)$ par :

- $A(m, n) = A(m - 1 ; A(m ; n - 1))\ pour\ m > 0\ et\ n > 0$
- $A(0, n) = n + 1\ pour\ n > 0$
- $A(m, 0) = A(m - 1 ; 1)\ pour\ m > 0$

On considère que cette fonction retourne la valeur $1\ pour\ m = 0\ et\ n = 0$.

Écrire une fonction qui calcule une valeur de la fonction d'Ackermann.

6.1 Notion de récursivités

Exercice d'application 4 : Tester la parité d'un nombre

On suppose que l'on ne connait que la parité du nombre 0 (0 est un nombre pair). Alors pour tester la parité de tout entier naturel n, on utilise les fonctions pair et impair telles que définies ci-dessous :

$$\text{pair}(n) = \begin{cases} \textit{vrai} & \text{si } n = 0; \\ \text{impair}(n-1) & \text{sinon;} \end{cases} \quad \text{et} \quad \text{impair}(n) = \begin{cases} \textit{faux} & \text{si } n = 0; \\ \text{pair}(n-1) & \text{sinon.} \end{cases}$$

Écrire un algorithme qui :

- d'abord reçoit un entier naturel n ;
- ensuite affiche l'affirmation "n est pair" ;
- et enfin affiche "VRAI" ou "FAUX" en réponse à l'affirmation précédente.

FIN DU COURS