Algorithmes récursifs

Licence 1 MASS - Introduction programmation JAVA

Sébastien Verel
verel@i3s.unice.fr
www.i3s.unice.fr/∼verel

Équipe ScoBi - Université de Nice Sophia-Antipolis

2 avril 2012

Objectifs de la séance 10

- Ecrire un algorithme récursif avec un seul test
- Etablir le lien entre définition par récurrence et algorithme récursif
- Recherche dichotomique d'un élément dans un tableau

Question principale du jour :

Comment écrire ce que l'on ne connait pas encore ?

Plan

Exemple du calcul du pgcd

Algorithme PGCD(a, b : entier) : entier
début
  si $b = 0$ **alors**
    retourner a
  sinon
    $c \leftarrow a$ modulo b
    retourner PGCD(b, c)
  fin si
fin

Exécution de l'algorithme

Pour $a = 70$ et $b = 462$

1. PGCD(70, 462)
2. $b \neq 0$
5. $c = 70$
6. PGCD(462, 70)
2. $b \neq 0$
5. $c = 42$
6. PGCD(70, 42)
2. $b \neq 0$
5. $c = 28$
6. PGCD(42, 28)
2. $b \neq 0$
5. $c = 14$

Exécution de l'algorithme

Pour $a = 462$ et $b = 70$

6. PGCD(28, 14)
2. $b \neq 0$
5. $c = 0$
6. PGCD(14, 0)
2. $b = 0$
3. PGCD $= 14$

Elements de l'algorithme

- Base : initialisation de la récurrence

  si $b = 0$ **alors**
  retourner a
  sinon
  ...
  fin si

- Hérédité : calcul à partir de paramètres plus "petits"

  si $b = 0$ **alors**
  ...
  sinon
  ...
  retourner PGCD(b, c)
  fin si

fin

Définition (informelle)

Algorithmes récursifs

Un algorithme récursif est un algorithme qui fait appel à lui-même dans le corps de sa propre définition.

Il existe deux types d'algorithmes récursifs :

- les algorithmes récursifs qui se terminent :
  au bout d'un nombre fini d'opérations, l'algorithme s'arrête.
- les algorithmes récursifs qui ne se terminent pas :
  on peut imaginer que l'algorithme continue "éternellement" de calculer.

Exemples

Algorithme suiteU(n : entier) : réel
début
  si $n = 0$ **alors**
    retourner 2
  sinon
    retourner $\frac{1}{2}$ suiteV$(n-1) + 2$
  fin si
fin

suiteU n'est pas un algorithme résursif

Exemples

Algorithme suiteV(n : entier) : réel
début
  si $n = 0$ **alors**
    retourner 2
  sinon
    retourner $\frac{1}{2}$ suiteV(n-1) +2
  fin si
fin

suiteV est un algorithme résursif qui se termine

Exemples

Algorithme suiteW(n : entier) : réel
début
  si $n = 0$ **alors**
    retourner 2
  sinon
    retourner $\frac{1}{2}$ suiteW(n-3)+2
  fin si
fin

suiteW(n) est un algorithme résursif :

- si n est un multiple de 3, suiteW se termine
- sinon suiteW ne se termine pas

Exemple d'exécution
Calcul de la puissance

Définition mathématique :

$$a^n = \begin{cases} 1, & \text{si } n = 0 \\ a^{n-1}.a, & \text{sinon} \end{cases}$$

Algorithme puissance(a : réel, n : entier) : réel
début
  si $n = 0$ **alors**
    retourner 1
  sinon
    retourner puissance(a, n-1) * a
  fin si
fin

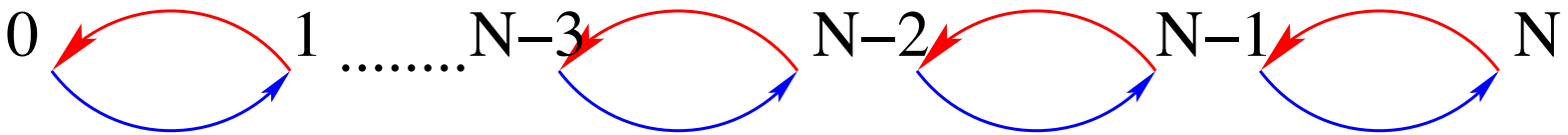
Exemple d'exécution

Calcul de la puissance

Calcul de $(2.5)^3$: $a = 2.5$ et $n = 3$.

1. puissance(2.5, 3)
2. ——> puissance(2.5, 2)
3. ————-> puissance(2.5, 1)
4. ——————-> puissance(2.5, 0) = 1
5. ————-> puissance(2.5, 1) = 1 * 2.5 = 2.5
6. ——> puissance(2.5, 2) = 2.5 * 2.5 = 6.25
7. puissance(2.5, 3) = 6.25 * 2.5 = 15.625

Principe d'exécution



- en rouge : parcours "aller"
- en bleu : parcours "retour"

En Java

```
/**************************************
* calcul de la puissance d'un nombre
*
* entrée :
*   - a : nombre
*   - n : puissance
*
* sortie :
*   - a^n
**************************************/
double puissance(double a, int n) {
  if (n == 0)
    return 1;
  else
    return puissance(a, n-1) * a;
}
```

PGCD en Java

```
/**************************************
* calcul du pgcd de a et b
*
* entrée :
*   - a, b : nombres entiers
*
* sortie :
*   - pgcd(a, b)
**************************************/
int pgcd(int a, int b) {
  if (b == 0)
    return a;
  else
    return pgcd(b, a % b);
}
```

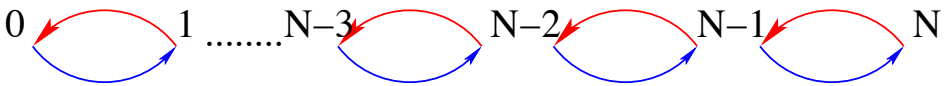
Intérêts

- bien adapté à la résolution de certains problèmes (et pas seulement mathématiques !)
- algorithmes souvent moins "laborieux" à écrire :
  moins de variables, beaucoups moins de boucles.
- une résolution par algorithme récursif nécessite souvent de prendre du recul pour résoudre le problème (avantage !)

A ne pas oublier !

- Hérédité : calcul à partir de paramètres plus "petits"

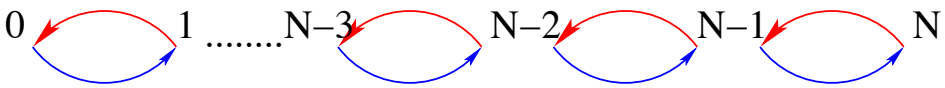
si $b = 0$ **alors**
  ...
sinon
  ...
  retourner PGCD(b, $b \mod a$)
fin si
fin



A ne pas oublier !

- Base : initialisation de la récurrence

si $b = 0$ **alors**
  retourner a
sinon
  ...
fin si



Parallèle entre principe de récurrence et algorithme récursif

définition mathématique par récurrence
très proche
définition d'un algorithme récursif (cf. puissance)

Modes de calcul proches :

$$a^3 = a.a^2 = a.a.a^1 = a.a.a$$

Souvent, définition mathématique valide lorsque algorithme récursif associé se termine.

A méditer
Calcul de la puissance

Algorithme puissanceTerminale(a : réel, n : entier, acc : réel) : réel
début
  si $n = 0$ **alors**
    retourner acc
  sinon
    retourner puissanceTerminale(a, $n - 1$, $acc * a$)
  fin si
fin

Comment s'exécute cet algorithme ? puissanceTerminale(2.5, 3, 1)

- récursivité terminale : équivalent à une itération

Calcul de la factorielle

Factorielle

$$n! = \begin{cases} 1, & \text{si } n = 0 \\ n.(n-1)!, & \text{sinon} \end{cases}$$

Algorithme factorielle(n : entier) : entier
début
  si $n = 0$ **alors**
    retourner 1
  sinon
    retourner n * factorielle(n-1)
  fin si
fin

Factorielle en Java

```
/**************************************
* calcul de la factorielle
*
* entrée :
*   - n : nombre
*
* sortie :
*   - n!
**************************************/
int factorielle(int n) {
  if (n == 0)
    return 1;
  else
    return n * factorielle(n - 1);
}
```

Exécution de l'algorithme

Calcul de 3!.

1. factorielle(3) = 3 * factorielle(2)
2. ——> factorielle(2) = 2 * factorielle(1)
3. ————-> factorielle(1) = 1 * factorielle(0)
4. ——————-> factorielle(0) = 1
5. ————-> factorielle(1) = 1 * 1 = 1
6. ——> factorielle(2) = 2 * 1 = 2
7. factorielle(3) = 3 * 2 = 6

Recherche dichotomique

Recherche d'une définition dans un dictionnaire

Algorithme rechercheDicho(*cible* : mot) : liste de mot
début
  premier ← premier mot du dictionnaire
  dernier ← dernier mot du dictionnaire
  Lire le mot médian entre premier et dernier
  tant que mot lu n'est pas le mot cible **faire**
    si mot cible est avant le mot lu **alors**
      dernier ← mot lu
    sinon
      premier ← mot lu
    fin si
    Lire le mot médian entre premier et dernier
  fin tant que
  retourner définition du mot lu
fin

Recherche dichotomique : récursive

Recherche de l'existence d'un mot dans un dictionnaire

Algorithme recherche(m : mot, l : liste) : booléen
début
variable lu : mot
  lu ← median(l)
  si lu = m **alors**
    retourner Vrai
  sinon
    si lu < m **alors**
      retourner recherche(m, liste à droite de lu)
    sinon
      retourner recherche(m, liste à gauche de lu)
    fin si
  fin si
fin

⟶ Incomplet (jamais Faux) ! !

Recherche dichotomique : récursive

Recherche de l'existence d'un mot dans un dictionnaire

```
Algorithme  recherche(m : mot, l : liste) : booleen
début
variable  lu : mot
   si l est vide alors
      retourner Faux
   sinon
      lu ← median(l)
      si lu = m alors
         retourner Vrai
      sinon
         si lu < m alors
            retourner recherche(m, liste à droite de lu)
         sinon
            retourner recherche(m, liste à gauche de lu)
         fin si
      fin si
   fin si
fin
```

Recherche dichotomique

Recherche dans un tableau de nombres entiers ordonnés

Algorithme recherche(x : entier, t : tableau d'entiers, a : entier, b : entier) : booléen
début
variable c : entier
  si $a > b$ **alors**
    retourner Faux
  sinon
    $c \leftarrow (a + b)/2$
    si $t[c] = x$ **alors**
      retourner Vrai
    sinon
      si $t[c] < x$ **alors**
        retourner recherche(x, t, c+1, b)
      sinon
        retourner recherche(x, t, a, c-1)
      fin si
    fin si
  fin si
fin

Exécution de l'algorithme

t :	2	5	6	8	11	15	20

1. recherche(6, t, 0, 6)
1. $c = 3$
1. $t[3] > 6$
2. ——> recherche(6, t, 0, 2)
2. $c = 1$
2. $t[1] < 6$
3. ————-> recherche(6, t, 2, 2)
3. $c = 2$
3. $t[2] = 6$
3. ————-> recherche(6, t, 2, 2) = Vrai
4. ——> recherche(6, t, 0, 2) = Vrai
5. recherche(6, t, 0, 6) = Vrai

En Java

```
/**************************************
* recherche du nombre x dans le tableau entre les bornes a et
*
* entrée :
*   - x   : nombre à rechercher
*   - tab : tableau de recherche
*   - a, b :borne de la recherche
*
* sortie :
*   - vrai ssi le nombre est dans le tableau
**************************************/
boolean recherche(int x, int [] t, int a, int b) {
  if (a > b)
    return false;
  else {
    int c = (a + b) / 2;
    if (t[c] == x)
      return true;
    else
```

En Java

```
boolean recherche(int x, int [] t, int a, int b) {
  if (a > b)
    return false;
  else {
    int c = (a + b) / 2;
    if (t[c] == x)
      return true;
    else
      if (t[c] < x)
        return recherche(x, t, c + 1, b);
      else
        return recherche(x, t, a, c - 1);
  }
}
```

Quand utiliser un algorithme récursif ?

- Est-ce que le problème dépend d'un (ou plusieurs) paramètre(s) ?
- Est-il possible de résoudre le problème lorsque la (les) valeur(s) du paramètre est "petite(s)" ?
- Est-il possible de résoudre le problème à l'aide de la résolution du problème portant sur une (des) "plus petite(s)" valeur(s) du paramètre ?

Si oui, oui, oui
alors la résolution par un algorithme récursif est à envisager.

Tours de Hanoï (Édouard Lucas 1842 - 1891)

Le problème des tours de Hanoï consiste à déplacer N disques de diamètres différents d'une tour de départ à une tour d'arrivée en passant par une tour intermédiaire et ceci en un minimum de coups, tout en respectant les règles suivantes :

- on ne peut déplacer plus d'un disque à la fois,
- on ne peut placer un disque que sur un autre disque plus grand que lui ou sur un emplacement vide.

Comment résoudre ce problème ?

- Est-ce que le problème dépend d'un (ou plusieurs) paramètre(s) ?
  $\longrightarrow$ Oui le nombre de disques.
- Est-il possible de résoudre le problème lorsque la (les) valeur(s) du paramètre est "petite(s)" ?
  $\longrightarrow$ Oui lorsque le nombre de disque est 1.
- Est-il possible de résoudre le problème à l'aide de la résolution du problème portant sur une (des) "plus petite(s)" valeur(s) du paramètre ?
  $\longrightarrow$ Oui...

Algorithme récursif

Algorithme hanoi(n : entier, A : caractère, B : caractère, C : caractère) : rien
début
  si $n = 1$ **alors**
    écrire("déplacer ", A, " vers ", C)
  sinon
    hanoi(n-1, A, C, B) ;
    écrire("déplacer ", A, " vers ", C)
    hanoi(n-1, B, A, C) ;
  fin si
fin

En Java

```
/**************************************
* résolution des tours de Hanoi
*
* entrée :
*   - n : nombre de disques du problème
*   - A : pic initial
*   - B : pic intermédiaire
*   - C : pic final
*
* sortie :
*   - aucune
**************************************/
void hanoi(int n, char A, char B, char C) {
  if (n == 1)
    println("déplacer " + A + " vers " + C);
  else {
    hanoi(n-1, A, C, B);
    println("déplacer " + A + " vers " + C);
    hanoi(n-1, B, A, C);
```

Exécution de l'algorithme

- Hanoi(2, 'a', 'b', 'c')
- Hanoi(3, 'a', 'b', 'c')
- Hanoi(4, 'a', 'b', 'c')

Quel est le nombre de déplacements en fonction de n ?

Pour tout entier $n \geq 1$, $C_n = 2^n - 1$. A démontrer par récurrence...

Pour $n = 64$, les moines d'Hanoi y sont encore...

Temps de diffusion

Calculer $T_a(n) = a + 2a + 3a + \ldots + n.a$

- Est-ce que le problème dépend d'un (ou plusieurs) paramètre(s) ?
  $\longrightarrow$ Oui n.
- Est-il possible de résoudre le problème lorsque la (les) valeur(s) du paramètre est "petite(s)" ?
  $\longrightarrow$ Oui pour $n = 0$ ou $n = 1$
- Est-il possible de résoudre le problème à l'aide de la résolution du problème portant sur une (des) "plus petite(s)" valeur(s) du paramètre ?
  $\longrightarrow$ Oui, $T_a(n) = T_a(n-1) + n.a$

Temps de diffusion
Algorithme

Algorithme diffusion(a : réel, n : entier) : réel
début
  si $n = 0$ **alors**
    retourner 0
  sinon
    retourner diffusion(a, n-1) $+n.a$
  fin si
fin

En Java

```
/**************************************
* calcul du temps de diffusion
*
* entrée :
*   - n : nombre de particule à diffuser
*   - a : temps pour diffusion 1 particule sur le bord
*
* sortie :
*   - temps total
**************************************/
float diffusion(float a, int n) {
  if (n == 0)
    return 0;
  else
    return diffusion(a, n-1) + a * n;
}
```

Exécution de l'algorithme

Calcul de $T_4(3)$

1. diffusion(4, 3)
2. ——> diffusion(4, 2)
3. ————-> diffusion(4, 1)
4. ——————-> diffusion(4, 0) = 0
5. ————-> diffusion(4, 1) = 0 + 4 = 4
6. ——> diffusion(4, 2) = 4 + 2.4 = 12
7. diffusion(4, 3) = 12 + 3.4 = 24

Régionnement du plan

Etant donné un nombre n de droites, calculer le nombre R_n maximum de régions du plan obtenus

- Est-ce que le problème dépend d'un (ou plusieurs) paramètre(s) ?
  $\longrightarrow$ Oui le nombre n de droites.
- Est-il possible de résoudre le problème lorsque la (les) valeur(s) du paramètre est "petite(s)" ?
  $\longrightarrow$ Oui pour $n = 0$ ou $n = 1$
- Est-il possible de résoudre le problème à l'aide de la résolution du problème portant sur une (des) "plus petite(s)" valeur(s) du paramètre ?
  $\longrightarrow$ Oui, en comptant le nombre régions ajoutées lorsqu'on ajoute une droite à $n - 1$ droites : une région supplémentaire par droite coupée, plus une dernière région.

Régionnement du plan
Algorithme

Algorithme region(n : entier) : entier
début
  si $n = 0$ **alors**
    retourner 1
  sinon
    retourner region(n-1) $+n$
  fin si
fin

Objectifs de la séance 10

- Ecrire un algorithme récursif avec un seul test
- Etablir le lien entre preuve par récurrence et algorithme récursif
- Recherche dichotomique d'un élément dans un tableau

Question principale du jour :

Comment écrire ce que l'on ne connait pas encore ?