Induction et Récursivité

© Herve Flores

Exemple 1 : palindrome

Mot qui se lit de la même façon de gauche à droite ou de droite à gauche

Exemple :

ressasser, rever, Anna, oho, e sont des palindromes

papa n’est pas un palindrome

Connus

palindrome(ever)
FAUX

palindrome(v)
VRAI

palindrome(ver)
FAUX

palindrome(eve)
VRAI

palindrome(er)
FAUX

Inconnu

palindrome(rever)

palindrome(rever)

le premier et le dernier caractère sont égaux donc
rever est un palindrome si eve est un palindrome

palindrome(eve)
VRAI

donc rever est un palindrome

Cas général

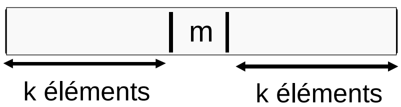
$\boxed{s_0 \mid s_1\ s_2\ s_3\ \ldots \mid s_{n-1}}$ est un palindrome ssi

- s0 = sn-1
- et $\boxed{s_1\ s_2\ s_3\ \ldots\ s_{n-2}}$ est un palindrome

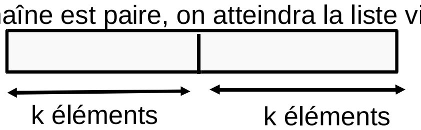
Comment s'arrêter ?

A chaque étape, on diminue la taille de la chaîne considérée de 2.

- si la taille de la chaîne est impaire, on atteindra une liste ayant un seul élément



- si la taille de la chaîne est paire, on atteindra la liste vide



Formulation récursive

si (taille(s) = 0 ou taille(s) =1) palindrome(s)=VRAI
sinon
    si (caractère(s,0) = caractère(s,taille(s)-1)
        palindrome(s) = palindrome(sousChaine(s,1, taille(s)-1)
    sinon palindrome(s) = FAUX

En Java

```
 public boolean palindrome(String c) {
   int taille = c.length();
   if ((taille == 0) || (taille == 1))
       return true;
   else
     if (c.charAt(0)==c.charAt(taille-1))
         return palindrome(c.substring(1,taille-1));
     else return false;
}
```

Exemple 2 : recherche dichotomique dans une liste ordonnée

Principe :

comparer x à l'élément m qui est au milieu de la partie du tableau considérée.

- Si x = m renvoyer l'indice de m
- Si x < m chercher x dans la partie du tableau à gauche de m
- Si x > m chercher x dans la partie du tableau à droite de m
- Si la partie considérée est vide, renvoyer –1

< m	m	> m

La méthode fait appel à une méthode privée.

Le méthode privée a pour paramètres les indices gauche et droite qui délimitent la partie du tableau à traiter.

```
public static int rechercheViteRecursif(int[] t,int x) {
    return rechercheViteRecursif(t,x,0,t.length-1);
}
```

A VOUS

```
/** pour rechercher l'indice d'un élément
* antécédent : t est un tableau d'entiers ordonné par ordre croissant,
*              x est un entier,
*              -1 <= g <= t.length et -1 <= d <= t.length
* conséquent : renvoie i si t[i] == x,
*              renvoie -1 si pour tout i, g<=i<=d, t[i]!=x  */

private int rechercheViteRecursif(int[] t,int x,int g,
       int d){
  if (........)return ........ ;
  int m = (g+d)/2;
  if (t[m]==x) return ........ ;
  if (t[m]<x) return ....................;
  return ................................;
}
```


Exemple 3 : l’incontournable factorielle

0! = 1

n! = n * (n-1) * (n-2) * … * 3 * 2 * 1

ou encore : 0! = 1

n! = n * (n-1)!

En java :

```
/** renvoie n!
*/
public int factorielle(int n) {
   if (n==0) return 1;
   else return n * factorielle(n-1);
}
```

Comment ça marche ?

- A chaque appel de procédure, les paramètres sont empilés dans la pile d'exécution
- Au moment du return, le calcul est effectué en fonction des résultats précedents (stockés dans un registre spécialisé *return*)
- à la sortie de la procédure les paramètres sont dépilés.

Appels

appel de factorielle(4)

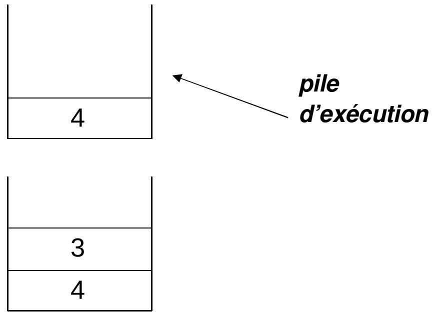
4 est empilé

appel de factorielle(3)

appel de factorielle(3)

3 est empilé

appel de factorielle(2)



appel de factorielle(2)

2 est empilé

appel de factorielle(1)

appel de factorielle(1)

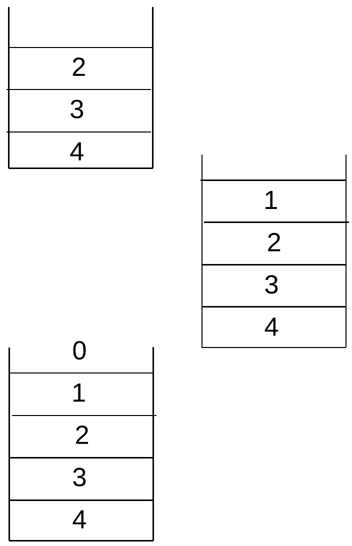
1 est empilé

appel de factorielle(0)

appel de factorielle(0)

0 est empilé

fin des appels récursifs



Retours

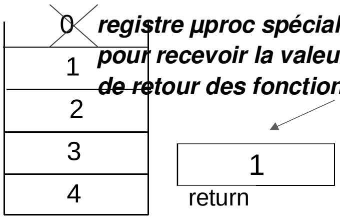
```
if (n==0) return 1;
```

retour de factorielle(0)

return = 1

0 est dépilé

revient à factorielle(1)



registre µproc spécialisé pour recevoir la valeur de retour des fonctions

```
else return n * factorielle(n-1);
```

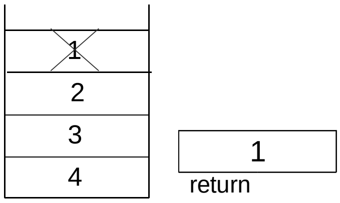
retour de factorielle(1)

return = sommetDePile * return

= 1 * 1

1 est dépilé

revient à factorielle(2)



retour de factorielle(2)

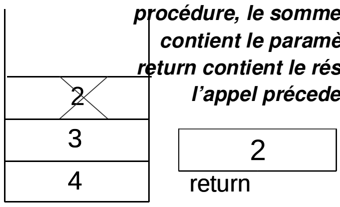
return = sommetDePile * return

= 2 * 1

2 est dépilé

revient à factorielle(3)

A chaque retour de procédure, le sommet de pile contient le paramètre et return contient le résultat de l'appel précedent



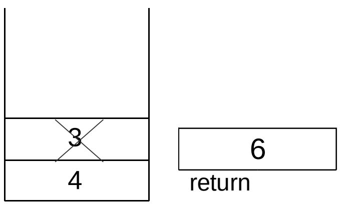
retour de factorielle(3)

return = sommetDePile * return

= 3 * 2

3 est dépilé

revient à factorielle(4)

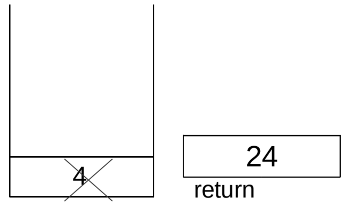


retour de factorielle(4)

return = sommetDePile * return

= 6 * 4

4 est dépilé



fin du retour des appels récursifs

le résultat de factorielle(4) est contenu dans return

Récursivité : Pourquoi ?

- Résoudre des problèmes complexes qui nécessitent un retour arrière
  - Faire un emploi du temps
  - Colorier une carte de façon à ce que deux pays voisins n’aient pas la même couleur
  - Le sudoku
- Parcourir des structures arborescentes
  - Vérifier qu’un fichier HTML est bien formé <tag> </tag>
  - Traiter un fichier XML en explorant sa structure DOM

Schémas d'induction

- Définir un ensemble de données de façon inductive
  - les éléments de base
  - les règles de construction de nouveaux éléments

- Prouver des propriétés sur les ensembles définis de façon inductive
  - montrer que P est vraie pour les éléments de la base
  - montrer que

    P vraie pour un élément e

    ⇒ P vraie pour tous les éléments construits à partir de e en appliquant une règle de construction

- Définir des fonctions inductives
  - définir f pour les éléments de la base
  - définir f sur les éléments construits en appliquant des règles à un élément e, en fonction de la valeur de f pour e.

Les entiers positifs

- Construction
  - base : 0 est un entier
  - règle : si n-1 est un entier positif, n est un entier positif

- Récurrence :

  P(n) est vrai ssi
  - P(0) est vrai
  - $P(i-1) \Rightarrow P(i)$ est vrai $\forall$ $i>0$

Exemple : somme des n premiers entiers

- Propriété : $\sum_{i=0}^{n} i = n\,(n+1) / 2$
  - vrai pour 0 : si n = 0, $\sum_{i=0}^{0} i = 0$ et $0\,(0 + 1) / 2 = 0$
  - vrai pour n-1 $\Rightarrow$ vrai pour n:

$$\sum_{i=0}^{n} i = (n) + \sum_{i=0}^{n-1} i$$

$$=(n) + (n-1)\,n/2$$ *Par hypothèse*

$$= n\,(n+1)/2$$

- Définition inductive de la fonction somme :

  somme(0) = 0
  somme(n) = n+ somme(n-1)

- En java

```
public int somme(int n) {
  if (n==0) return 0;
       else return n + somme(n-1);
}
```


A VOUS : un autre schéma pour les entiers positifs

- BASE : ....
- REGLE : si n-2 est un entier positif, alors n est un entier positif
- Fonction somme en suivant ce schéma :

```
public int somme(int n) {
.....................................................
.....................................................
.....................................................
.....................................................
}
```

Les listes

- Construction
  - base : vide est une liste
  - règle : si x est un élément
    si L est une liste
    alors la liste notée <x, L> dont x est le premier élément et L la suite de liste, est une liste

- Schéma de preuve par induction :
  P(L) est vrai ssi
  - P(vide) est vrai
  - P(LL) $\Rightarrow$ P(<x,LL>) est vrai $\forall$ x, $\forall$ LL

- Exemple : définition inductive de la longueur

  longueur(vide) = 0
  longueur(<x,L>) = 1 + longueur(L)

- En java :
  Utilisation de deux classes pour la tête de liste et le chaînage (chapitre IV)

Les arbres binaires

- Construction
  - base : vide est un arbre binaire
  - règle : si x est un élément
    si g est un arbre binaire
    si d est un arbre binaire
    <x,g,d> est un arbre binaire

Schéma de preuve par induction :

P(l) est vrai ssi

- P(vide) est vrai
- P(g) et P(d) $\Rightarrow$ P(<x,g,d>) est vrai $\forall$ x, $\forall$ g, $\forall$ d

Récursivité

<u>Forme générale d'une fonction récursive</u>

public *<type>* fonctionRecursive(*<parametres>*) {
    if (*<parametres> correspond à la base*)
        return *valeur pour la base*
        else return
            valeur calculée à partir de
        fonctionRecursive (*fonction(<parametres>)*)
}

où *fonction(<parametres>)* permet d'accéder aux éléments à partir desquels ont été construits *<parametres>*

Terminaison

S'assurer que la fonction qui définit les paramètres de l'appel récursif en fonction des paramètres d'entrée est une fonction qui *converge* vers une des valeurs de la base.

```
public int jeBoucle(int i) {
  if (i==0) return 3;
  else return jeBoucle(i-2);
}
```

Si i n'est pas un nombre pair, on n'atteint jamais la valeur 0.

Autre cas d'erreur ?

Correction

- s'assurer que la valeur renvoyée pour la base est correcte
- en supposant que la valeur retournée par l'appel récursif est correcte, s'assurer que la valeur retournée pour le cas général est correcte

Remarque :

La récursivité peut ne pas suivre les schémas d'induction structurelle usuels; dans ce cas, il faut s'assurer que le schéma d'induction utilisé est correct pour les données considérées.

- Schéma usuel

```
public int factorielle(int n) {
   if (n==0) return 1;
   return n * factorielle(n-1);
}
```

- Schéma non usuel

```
public int factorielle2(int n) {
   if (n==0) return 1;
   if (....) return ...; // incorrect sans ce test d'arrêt
   return n * (n-1) * factorielle2(n-2);
}
```


<u>Calcul de la complexité</u>

résolution d'une *relation de récurrence* donnant la complexité de la méthode en fonction de la complexité des appels récursifs.

$$\boxed{\begin{array}{l} C(0) = c_0 \\ C(n) = a * C(f(n)) + c_n \end{array}}$$

n : taille des données
c_0: complexité pour une donnée de taille nulle
a : nombre d'appels récursifs
f(n) : nouvelle valeur de n pour l'appel récursif
c_n: complexité du calcul pour la donnée de taille n

```
public int factorielle(int n) {
   if (n==0) return 1;
   return n * factorielle(n-1);
}
```

$$C(0) = \Theta\ (1)$$
$$C(n) = C(n\text{-}1) + \Theta\ (1)$$

```
private int rechercheViteRecursif(int[] t, int x,
       int g,int d){
 if (g > d) return - 1;
 else {
   int milieu = (g + d) / 2 ;
   if (x==t[milieu]) return milieu;
     else if (x<t[milieu])
       return rechercheViteRecursif(t,x,g,milieu-1);
       else return rechercheViteRecursif(t,x,milieu+1,d);
   }
}
```

$C(0) = \Theta\ (1)$

$C(n) = C(n/2) + \Theta\ (1)$

taille des données : n = nombre d'éléments entre g et d
complexité pour une donnée de taille nulle : 1
nombre d'appels récursifs : un seul
nouvelle valeur de n : n/2 car on travaille sur la moitié du tableau
complexité du calcul : 1 car constant (test et affectation)

A VOUS : relation de récurrence de la méthode somme

```
public int somme(int n) {
if (n==0) return 0;
    else return n + somme(n-1);
}
```

C(0) = ...

C(n) = ... C(...) + ...

Relations de récurrence

Arbre de récursivité

$C(0) = \theta\ (1)$
$C(n) = C(n\text{-}1) + \theta\ (1)$

1 + C(n-1)

1 + 1 + C(n-2)

1 + 1 + 1 + C(n-3)

1 + 1 + 1 + 1 ...... C(0)

hauteur n+1

$\boxed{C(n) = \theta\ (n)}$

- $C(0) = \Theta(1)$
  $C(n) = C(n/2) + \Theta(1)$

1 + C(n/2)

1 + 1 + C(n/4)

1 + 1 + 1 + C(n/8)

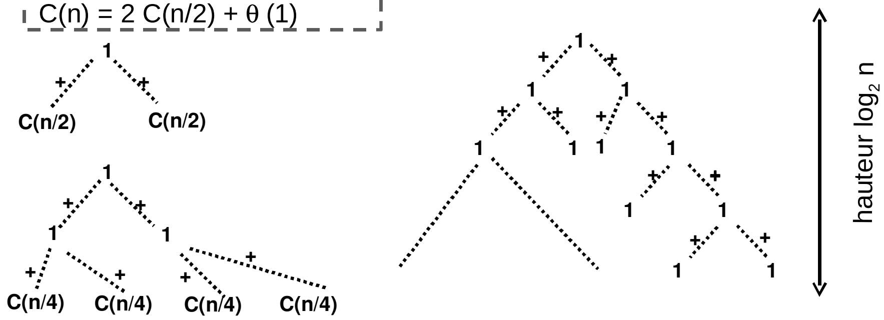
1 + 1 + 1 + 1 + ... C(0)

hauteur $\log_2 n$

$\boxed{C(n) = \Theta(\log_2 n)}$

$C(0) = \theta(1)$

$C(n) = 2\,C(n/2) + \theta(1)$



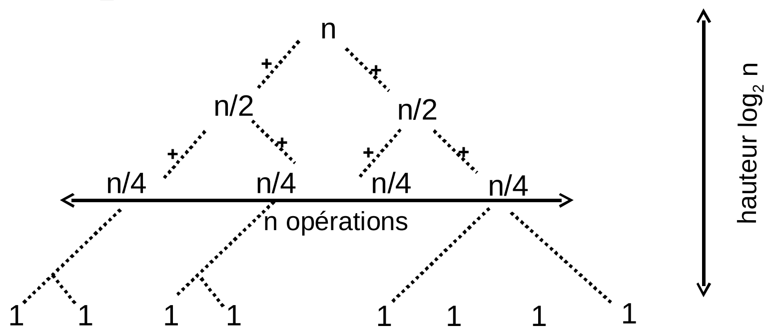
$\boxed{C(n) = \theta(n)}$

arbre binaire complet de hauteur $\log_2 n$

donc n nœuds, une opération par nœud

- $C(0) = 1$

  $C(n) = 2\,C(n/2) + \Theta(n)$



$\boxed{C(n) = \Theta(n \log_2 n)}$

n opérations à chaque niveau

$\log_2 n$ niveaux

A VOUS : complexités

- `Méthode somme : ......`
- `Méthode factorielle : ......`
- `Méthode rechercheViteRecursif : ......`

- Cas général

$C(0) = \theta((1)$
$C(n) = a\,C(n/b) + n^k$

- si $k < \log_b a$: $\boxed{C(n) = \theta(n^{\log_b a})}$
- si $k = \log_b a$: $\boxed{C(n) = \theta(n^k \log_2 n)}$
- si $k > \log_b a$: $\boxed{C(n) = \theta(n^k)}$