INFORMATIQUE
Sciences
Université Paris Cité

Langage: Java

IP1 Java -Cours TD 10

Lélia Blin

lelia.blin@irif.fr

2024

Objectifs

- Appréhender la notion de récursivité

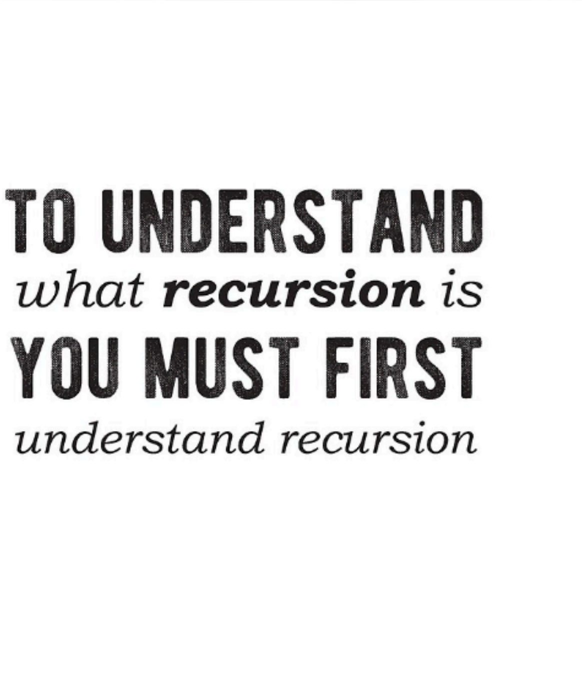
La récursivité

La **récursivité** est une **technique de programmation** où une **fonction s'appelle elle-même**, directement ou indirectement, pour résoudre un problème. Elle repose sur deux éléments essentiels :

1. **Cas de base** : une condition où la fonction arrête de s'appeler elle-même. Cela empêche une boucle infinie.
2. **Appel récursif** : une situation où la fonction s'appelle avec des données modifiées pour progresser vers le cas de base.

Remarque

- Elle caractérise des objets finis
- La **récursion se termine**!



Non !

Il faut penser à une représentation finie !

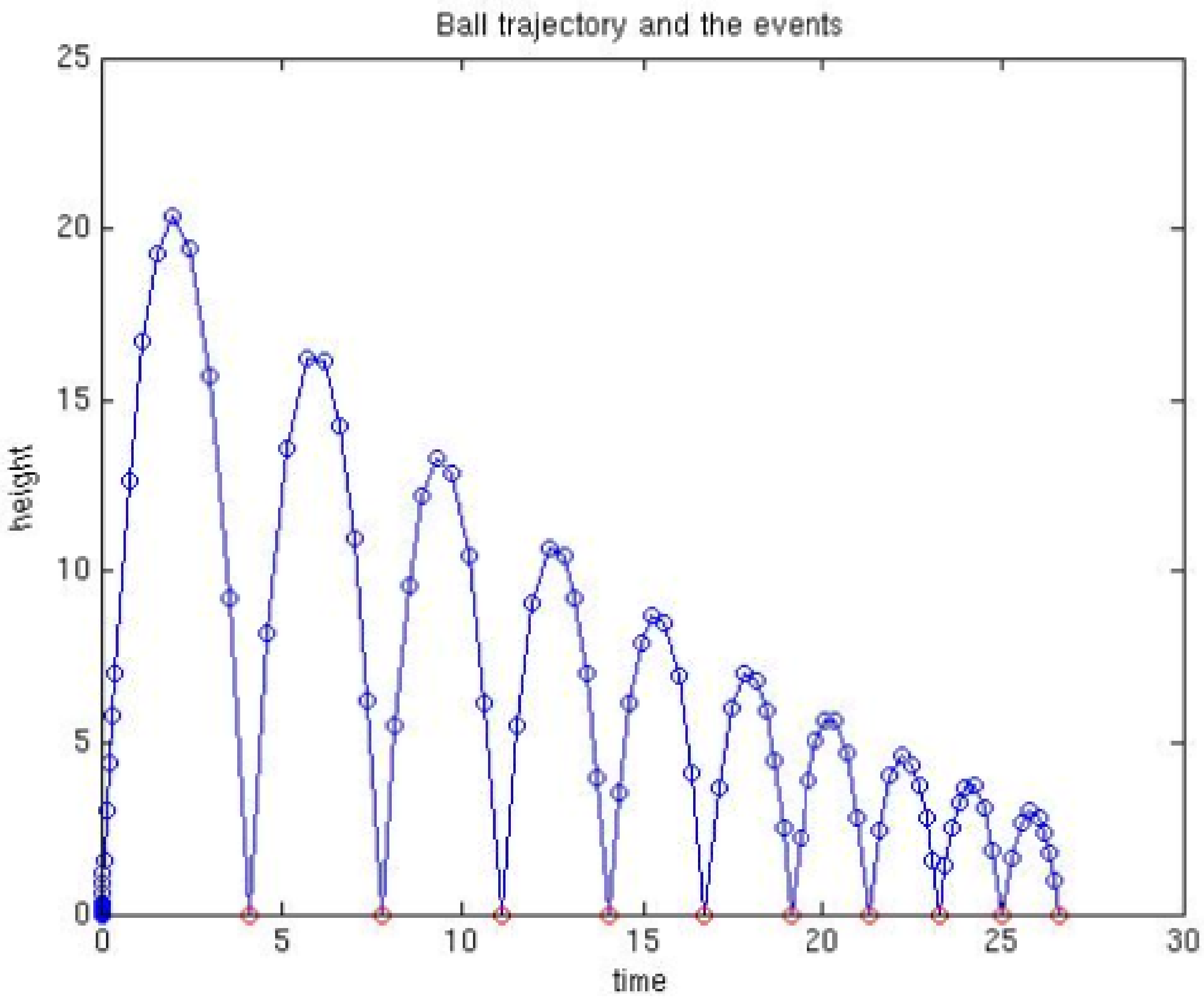
Raisonner en anticipant la fin

La dernière poupée existe

Il faut penser à une représentation finie !

Raisonner en anticipant la fin

La balle arrête de rebondir



Exemples mathématique

Factorielle

La **factorielle** d'un entier naturel n, notée $n!$, est définie par :

- $n! = 1$ si $n = 0$
- $n \times (n-1)!$ si $n > 0$

Rm: pour calculer $n!$ il faut avoir calculer $(n-1)!$, mais pour calculer $(n-1)!$ il faut avoir calculer $(n-2)! \ldots$ ceci jusqu'à $n = 0$.

Itératif vs Récursif

Approche itérative

Utilisation d'une boucle **for**, approche du bas vers le haut, on construit la solution depuis le cas de base:

```java
public class FactorielleIterative {
    public static int factorielle(int n) {
        int resultat = 1;
        for (int i = 1; i <= n; i++) {
            resultat *= i;
        }
        return resultat;
    }

    public static void main(String[] args) {
        int nombre = 5;
        System.out.println("La factorielle de " + nombre + " est " + factorielle(nombre)); // Affiche 120
    }
}
```

Exemple

- Factoriel 5:

i	resultat
1	$1 \times 1 = 1$
2	$1 \times 2 = 2$
3	$2 \times 3 = 6$
4	$6 \times 4 = 24$
5	$24 \times 5 = 120$

Approche récursive

```java
public class FactorielleRecursive {
    public static long fact(int n) {
        if (n == 0) { // Cas de base : 0! = 1
            return 1;
        }
        return n * fact(n - 1); // Appel récursif
    }

    public static void main(String[] args) {
        int nombre = 5; // Exemple : calculer la factorielle de 5
        long resultat = fact(nombre);

        System.out.println("La factorielle de " + nombre + " est " + resultat);
    }
}
```

Exemple

Appel:

$$5 \times fact(4) \rightarrow 4 \times fact(3) \rightarrow 3 \times fact(2) \rightarrow 2 \times fact(1) \rightarrow 1 \times 1$$

Construction

$$((((1 \times 1) \times 2) \times 3) \times 4) \times 5) = 120$$

Autre fonction mathématique

La somme des n premiers entiers positifs, notée $S(n)$, est définie comme :

- $S_n = 0$ si $n = 0$
- $S_n = n + S(n-1)$ si $n > 0$

Solution Itérative ou récursive??

Solution itérative

```java
public class SommeIterative {
    public static int somme(int n) {
        int resultat = 0;
        for (int i = 1; i <= n; i++) {
            resultat += i;
        }
        return resultat;
    }

    public static void main(String[] args) {
        int nombre = 5;
        System.out.println("La somme des " + nombre + " premiers entiers est " + somme(nombre));
    }
}
```

Rm: la boucle for s'execute n fois il y a donc n additions

Solution récursive

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

    public static void main(String[] args) {
        int nombre = 5;
        System.out.println("La somme des " + nombre + " premiers entiers est " + somme(nombre));
    }
}
```

Rm: il y a n appels récursifs il y a donc n additions

Solution mathématique

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

Coût: 1 addition, 1 multiplication et 1 division $\rightarrow$ ne dépend pas de n.

Exercice

Donner une fonction récursive pour calculer calculer a^b:

$$\text{puissance}(a,b) = \begin{cases} 1 & \text{si } b = 0, \\ a \times \text{puissance}(a, b-1) & \text{si } b > 0. \end{cases}$$

```java
public class Puissance {
    public static int puissance(int a, int b) {
        if (b == 0) {
            return 1; // Cas de base : a^0 = 1
        }
        return a * puissance(a, b - 1); // Cas récursif : a^b = a * a^(b-1)
    }

    public static void main(String[] args) {
        int a = 2; // Base
        int b = 3; // Exposant

        // Calcul de a^b
        System.out.println(a + " élevé à la puissance " + b + " est : " + puissance(a, b));
    }
}
```

Un peu plus complexe: Fibonacci

$$F(n) = \begin{cases} 0 & \text{si } n = 0, \\ 1 & \text{si } n = 1, \\ F(n-1) + F(n-2) & \text{si } n > 1. \end{cases}$$

Explications :

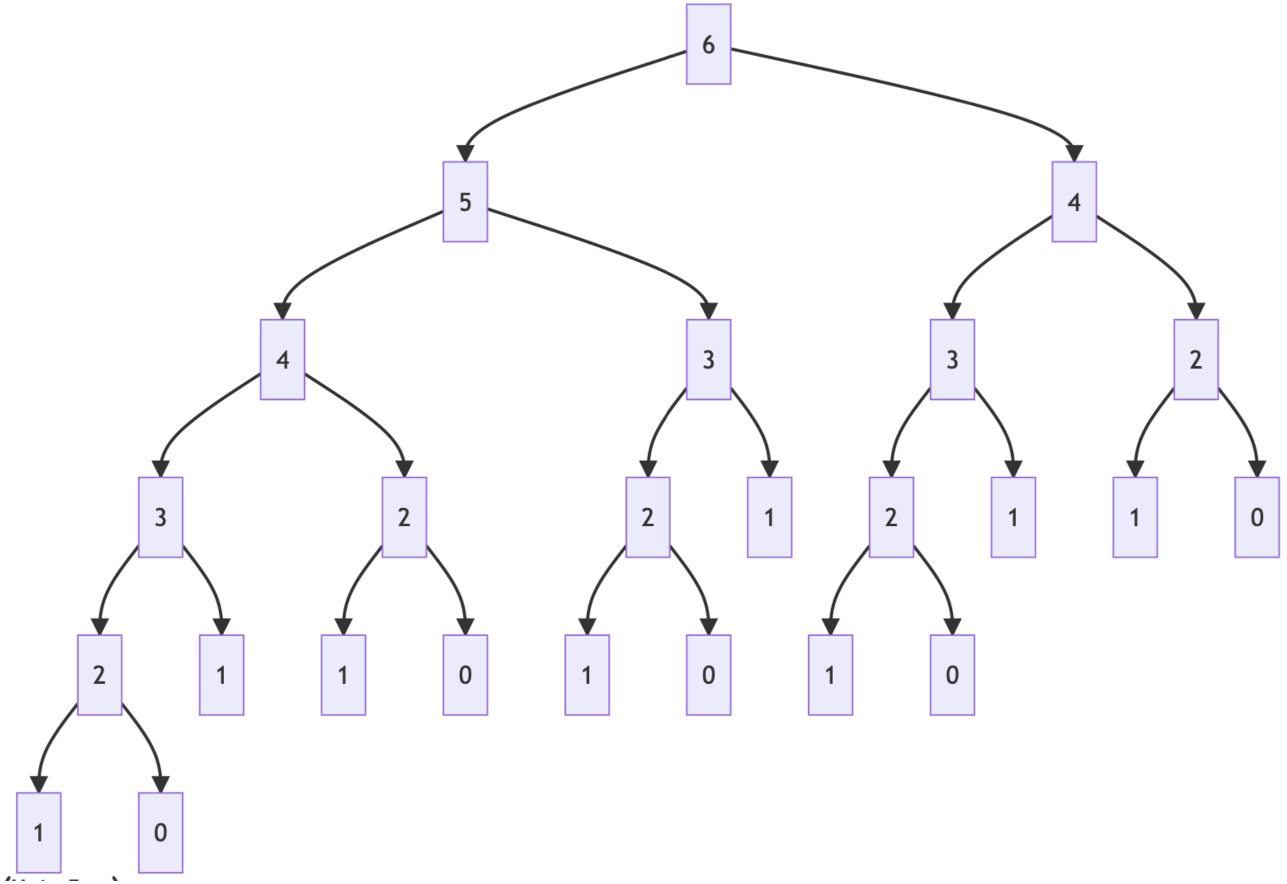
1. **Cas de base :**
   - $F(0) = 0$ et $F(1) = 1$.
2. **Relation de récurrence :**

- Pour $n > 1$, $F(n) = F(n-1) + F(n-2)$.

Donner une fonction java pour résoudre Fibonacci

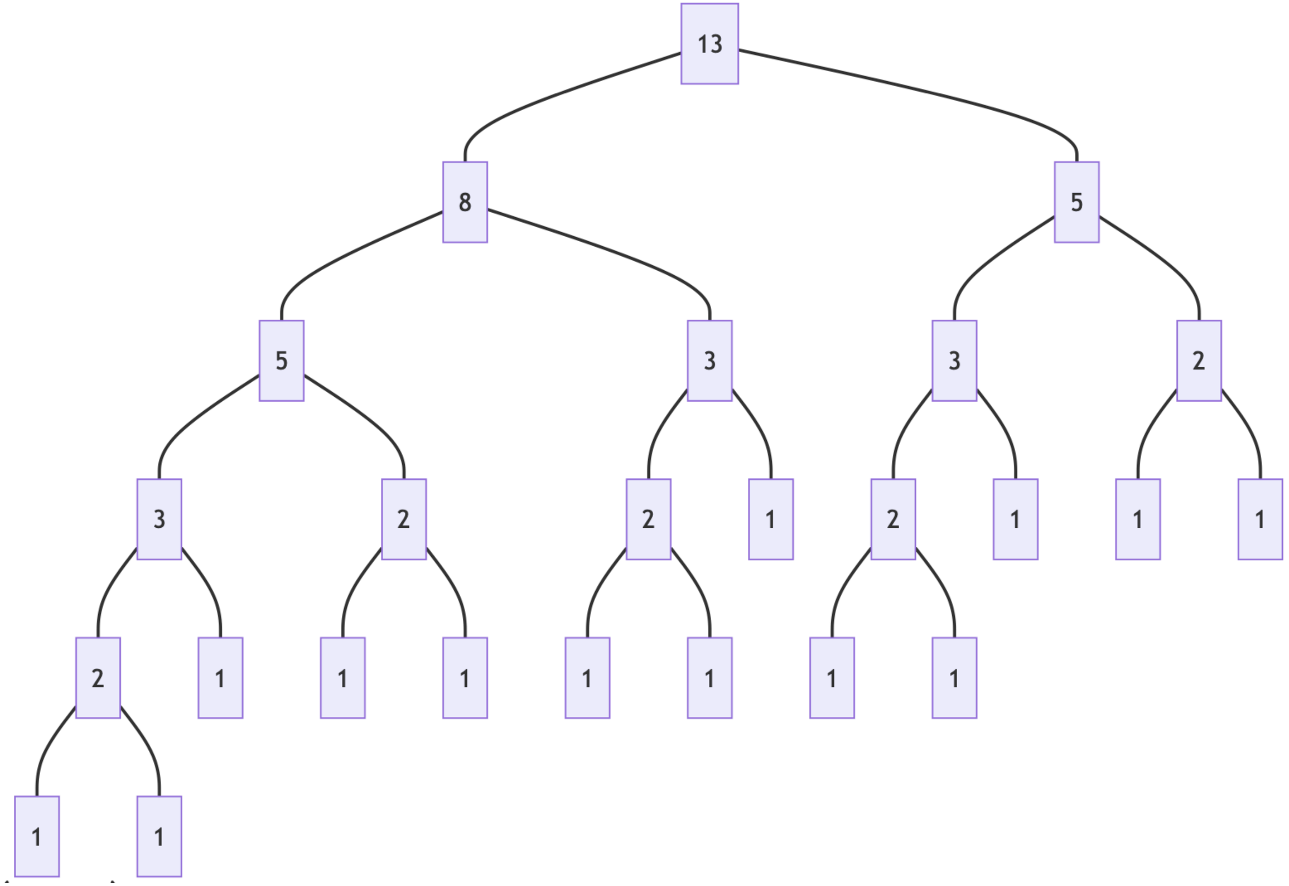
```java
public class FibonacciRecursive {
    public static int fibonacci(int n) {
        if (n == 0) {
            return 0; // Cas de base
        } else if (n == 1) {
            return 1; // Cas de base
        }
        return fibonacci(n - 1) + fibonacci(n - 2); // Appel récursif
    }

    public static void main(String[] args) {
        int terme = 6;
        System.out.println("Fibonacci(" + terme + ") = " + fibonacci(terme));
    }
}
```

Appels récursifs: pour $n = 6$



Lélia Blin

Calcul pour $n = 6$



Suite de Fibonacci recursive

- Approche récursive: **Simple**
- L'approche récursive est-elle efficace?
- Quel est le nombre d'appels récursifs en fonction de l'éntrée n?

Suite de Fibonacci recursive

- Approche récursive: **Simple**
- L'approche récursive est-elle efficace?
- Quel est le nombre d'appels récursifs en fonction de l'éntrée n?
- 2^n appels
- Méthode très redondante
- **Coûteux voire rédhibitoire**

Suite de Fibonacci itératif

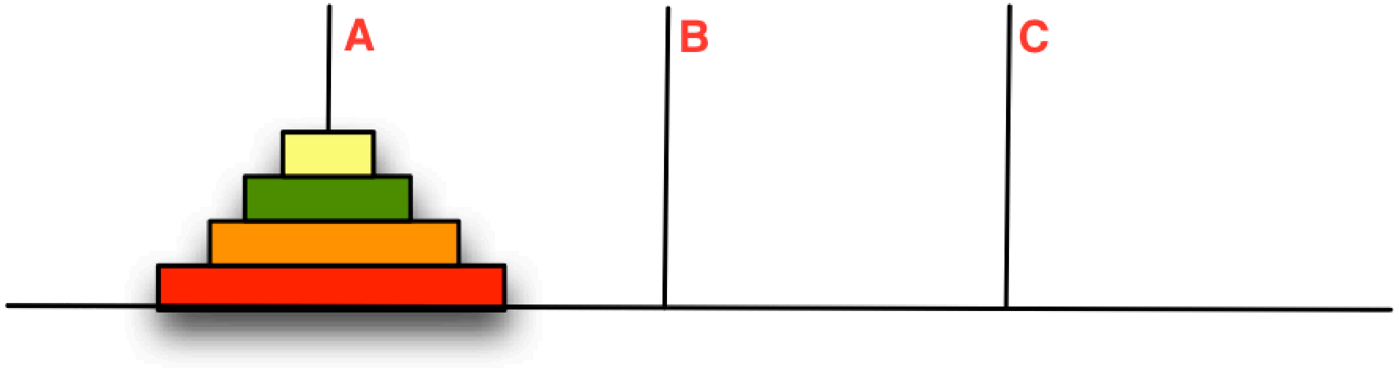
```java
public class FibonacciIterative {
    public static int fibonacci(int n) {
        if (n == 0) {
            return 0;
        } else if (n == 1) {
            return 1;
        }
        int a = 0, b = 1, resultat = 0;
        for (int i = 2; i <= n; i++) {
            resultat = a + b;
            a = b;
            b = resultat;
        }
        return resultat;
    }
    public static void main(String[] args) {
        int terme = 6;
        System.out.println("Fibonacci(" + terme + ") = " + fibonacci(terme));
    }
} //Resultat [1, 1, 2, 3, 5, 8, 13]
```

Les tours de Hanoï

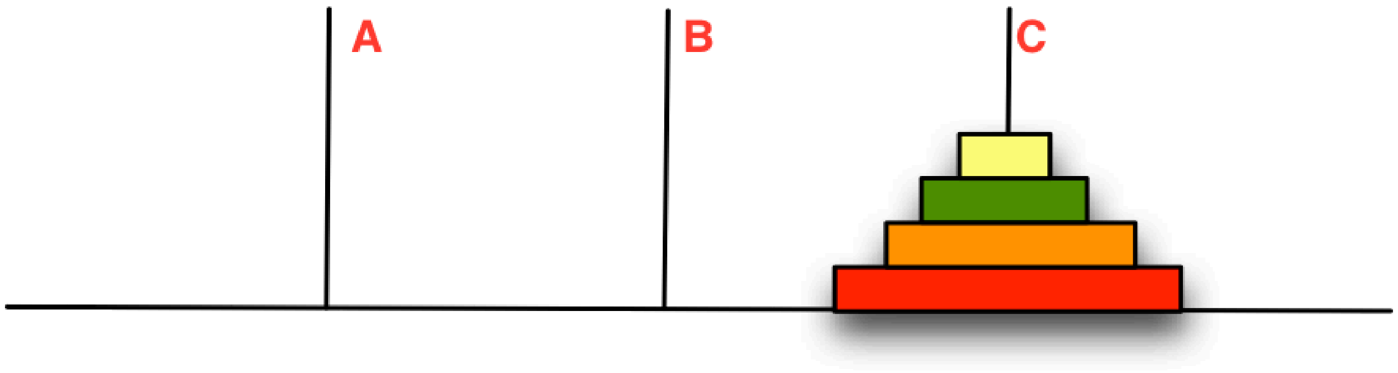
Légende:

- Dans un temple dieu a installé trois aiguilles de diamant.
- Dieu enfila au commencement des siècles, 64 disques d'or pur, sur une des aiguilles, du plus gros au plus petit.
- Nuit et jour, les prêtres se succèdent sur les marches de l'autel, occupés à transporter les disques de la première aiguille sur la troisième.
- Ils doivent respecter deux règles:
    - On ne peut déplacer qu'un disque à la fois.
    - Un disque ne peut pas être disposé sur une disque plus petit.
- Quand tout sera fini, la troisième tour tombera et ce sera la fin du monde!
- Quand est prévu la fin du monde?
  Jeu de réflexion imaginé par le mathématicien français Édouard Lucas

Tour intiale



Tour finale



Quelle stratégie de jeu adopter?

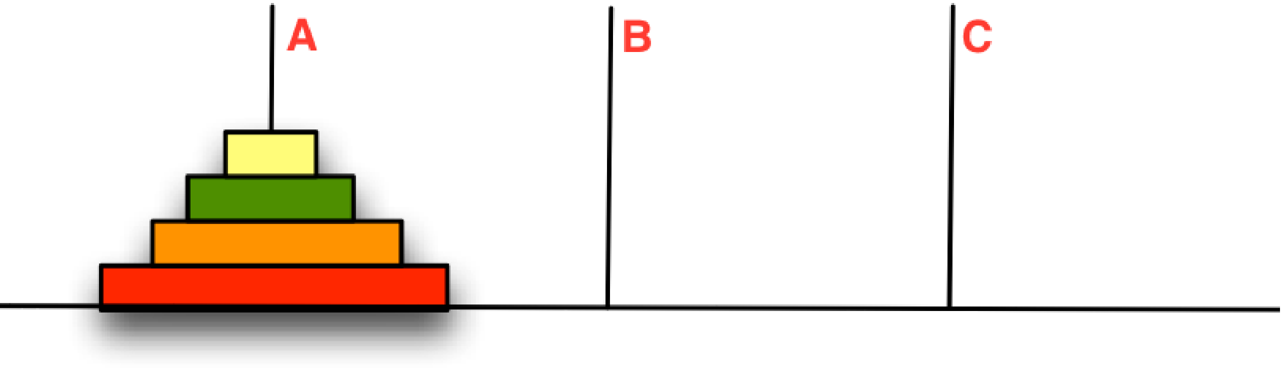
Stratégie

- déplacer la tour des $n-1$ premiers disques de A vers B.
- déplacer le plus grand disque de A vers C.
- déplacer la tour des $n-1$ premiers disques de B vers C.

Jouons un peu

[https://codepen.io/finnhvman/pen/gzmMaa?editors=1111]

```java
public class Hanoi {

    public static void hanoi(int n, String a, String b, String c) {
        if (n > 0) {
            hanoi(n - 1, a, c, b);
            System.out.println("De " + a + " vers " + c);
            hanoi(n - 1, b, a, c);
        }
    }

    public static void main(String[] args) {
        System.out.println("Déplace un anneau");
        hanoi(3, "A", "B", "C");
    }
}
```




```
Déplace un anneau
de A vers C
de A vers B
de C vers B
de A vers C
de B vers A
de B vers C
de A vers C
```

Les tours de Hanoï : approche récursive

Avantages: Simplicité

Nombres d'appels récursifs : $2^n - 1$

Alors la fin du monde

- Pour jeu à 64 disques requiert un minimum de 2^{64}-1 déplacements
- Soit 18 446 744 073 709 551 615 déplacements.
- En admettant qu'il faille une seconde pour déplacer un disque
- Soit 86 400 déplacements par jour.
- La fin du jeu aurait lieu au bout d'environ 213 000 milliards de jours
- Ce qui équivaut à peu près à 584,5 milliards d'années.
- Soit 43 fois l'âge estimé de l'univers (13,7 milliards d'années selon certaines sources)

Exercice : Inverser une chaîne de caractères

Écrire un programme récursif pour inverser une chaîne de caractères. Par exemple, la chaîne "hello" deviendrait "olleh".

Solution

```
public class InverserChaine {

    public static String inverser(String s) {
        if (s.isEmpty()) {
            return s; // Cas de base : une chaîne vide est son propre inverse
        }
        return inverser(s.substring(1)) + s.charAt(0); // Appel récursif
    }

    public static void main(String[] args) {
        String s = "hello"; // Exemple
        System.out.println("Chaîne inversée : " + inverser(s));
    }
}
```

Exercice:

Ecrire une version récursive de la recherche dichotomique d'un entier dans un tableau trié.

La recherche dichotomique récursive fonctionne de la même manière que la version itérative, sauf que la recherche est effectuée à travers des appels récursifs au lieu d'une boucle. À chaque appel, le tableau est divisé en deux parties, et on continue la recherche dans la moitié appropriée

- **Méthode récursive** :
    - La fonction `dichoto` prend en paramètre le tableau, l'élément recherché, ainsi que les indices `gauche` et `droite` qui délimitent la partie du tableau à explorer.

- **Base de la récursion** :
  - Si l'indice `gauche` dépasse l'indice `droite`, cela signifie que l'élément n'a pas été trouvé, et la fonction retourne `-1`.

- **Diviser le tableau** :
  - Le milieu du tableau est calculé par $milieu = \frac{gauche+droite}{2}$.
  - Si l'élément à l'indice `milieu` correspond à l'élément recherché, l'indice est retourné.
  - Si l'élément à `milieu` est plus petit que l'élément recherché, la recherche continue dans la moitié droite du tableau (en mettant à jour `gauche = milieu + 1`).
  - Si l'élément à `milieu` est plus grand, la recherche continue dans la moitié gauche du tableau (en mettant à jour `droite = milieu - 1`).

- **Appels récursifs** :
  - À chaque appel récursif, l'espace de recherche est réduit de moitié, ce qui permet de trouver l'élément rapidement.

```java
public class RechercheDichotomiqueRecursive {
    public static int dichoto(int[] tableau, int x, int gauche, int droite) {
        if (gauche > droite) {
            return -1; // L'élément n'a pas été trouvé
        }

        int milieu = (gauche + droite) / 2;

        if (tableau[milieu] == x) {
            return milieu; // L'élément a été trouvé
        }

        if (tableau[milieu] < x) {
            // Si l'élément recherché est plus grand, on recherche dans la moitié droite
            return dichoto(tableau, x, milieu + 1, droite);
        } else {
            // Si l'élément recherché est plus petit, on recherche dans la moitié gauche
            return dichoto(tableau, x, gauche, milieu - 1);
        }
    }
```

```java
    public static void main(String[] args) {
        int[] tableau = {1, 3, 5, 7, 9, 11, 13, 15}; // Tableau trié
        int x = 7; // Élément à rechercher
        int resultat = dichoto(tableau, x, 0, tableau.length - 1);

        if (resultat != -1) {
            System.out.println("L'élément " + x + " a été trouvé à l'indice : " + resultat);
        } else {
            System.out.println("L'élément " + x + " n'est pas dans le tableau.");
        }
    }
}
```

Exercice: le triangle ce Pascal

```
      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
 1 5 10 10 5 1
1 6 15 20 15 6 1
```

Ecrire une fonction récursive pour afficher le triangle de Pascal

Le triangle de Pascal est une représentation visuelle des coefficients binomiaux, et chaque élément du triangle peut être calculé récursivement à l'aide de la relation
$C(n, k) = C(n-1, k-1) + C(n-1, k)$
avec $C(n, 0) = 1$ et $C(n, n) = 1$

```java
public class TrianglePascal {
    public static int coefficientBinomial(int n, int k) {
        if (k == 0 || k == n) {
            return 1; // Cas de base : C(n, 0) = 1 et C(n, n) = 1
        }
        // Cas récursif : C(n, k) = C(n-1, k-1) + C(n-1, k)
        return coefficientBinomial(n - 1, k - 1) + coefficientBinomial(n - 1, k);
    }
    public static void afficherTrianglePascal(int n) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j <= i; j++) {
                System.out.print(coefficientBinomial(i, j) + " ");
            }
            System.out.println();
        }
    }
    public static void main(String[] args) {
        int n = 6; // Nombre de lignes à afficher
        afficherTrianglePascal(n);
    }
}
```