

## Algorithmique et structures de données

### CM 4 – Piles et files

Jean-Marie Le Bars  
jean-marie.lebars@unicaen.fr

## Plan du CM 4

Présentation des piles et des files

Fonctions sur les piles et des files

Implémentation d'une pile

Implémentation d'une file

Exemples d'utilisation d'une pile

# Plan du CM 4

Présentation des piles et des files

Fonctions sur les piles et des files

Implémentation d'une pile

Implémentation d'une file

Exemples d'utilisation d'une pile

## Utilisation des piles et des files

Les piles et les files sont des **objets temporaires** utilisés dans de nombreux programmes.

### Mémorisation et restitution

- les données sont mémorisées
- elles sont restituées dans un certain ordre

### Ordre

- les piles utilisent l'ordre **LIFO** (Last in first out, Dernier entré premier sorti)
- les files utilisent l'ordre **FIFO** (First in first out, Premier entré premier sorti)

### Objet $\neq$ structure de données

Nous n'avons pas besoin de connaître la structure de données choisie pour utiliser une pile ou une file.

## Représentation d'une pile

Les piles utilisent l'ordre **LIFO** (dernier entré, premier sorti)

### Rôle de la représentation

- la représentation permet de simuler le fonctionnement de la pile
- elle ne dépend pas de la structure de données choisie

### Exemple d'application – Pile de dossiers

- le dernier dossier est posé en haut de la pile
- le dossier en bas de la pile est le premier dossier qui a été posé

### Exemple de pile

10
7
3
20
6

P

### Accessibilité

- 10 est la **première** valeur que l'on peut récupérer
- 7 est la **seconde** valeur que l'on peut récupérer
- 6 est la **dernière** valeur que l'on peut récupérer

## Représentation d'une file

Les files ou files d'attente utilisent l'ordre **FIFO** (premier entré, premier sorti)

### Rôle de la représentation

- la représentation permet de simuler le fonctionnement de la file
- elle ne dépend pas de la structure de données choisie

### Exemple d'application – Attente à un guichet

- à gauche se trouve la personne arrivée en premier
- à droite se trouve la personne arrivée en dernier

### Exemple de file

6	20	3	7	10
---	----	---	---	----

### Accessibilité

- 10 est la dernière valeur rentrée et la dernière que l'on peut récupérer
- 6 est la première valeur rentrée et la première que l'on peut récupérer

# Plan du CM 4

Présentation des piles et des files

**Fonctions sur les piles et des files**

Implémentation d'une pile

Implémentation d'une file

Exemples d'utilisation d'une pile

# Fonctions sur les piles et des files

## Fonctions essentielles

- on se limite aux fonctions strictement nécessaires
- ces fonctions sont utilisées pour tous les programmes utilisant des piles ou des files

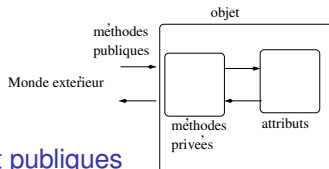
## Fonctions d'interface

- les fonctions (au sens **mathématique ou spécification**) ne dépendent pas de la structure de données choisie.
- l'**implémentation** des fonctions n'a pas à être connue de l'utilisateur



## Analogie avec la programmation orientée objet

### Interface entre l'objet et le monde extérieur



### Méthodes privées et publiques

- les méthodes privées ne sont accessibles que par l'objet
- les méthodes publiques sont accessibles dans le monde extérieur.
- les méthodes publiques permettent l'échange entre l'objet et le monde extérieur

### Analogie

- les piles et les files sont les objets
- les structures de données et les procédures liées à ces structures de données remplacent les attributs et les méthodes privées
- les fonctions des piles et des files forment l'interface entre l'objet et le monde extérieur

## Fonctions pour la pile – Interface

Comme notre langage algorithmique est typé, on supposera que toutes les données sont d'un seul type, ici des entiers.

### Signature des fonctions pour une pile d'entiers

- **initPile() : pile**  
→ la fonction retourne une pile vide.
- **pileVide(p : pile) : booléen**  
→ la fonction teste si la pile p est vide ou non.
- **sommet(p : pile) : entier**  
→ la fonction retourne l'élément placé au sommet de la pile p (elle retourne un message d'erreur si la pile est vide).
- **empiler(p : pile, x : entier) : pile**  
→ la fonction retourne la pile p avec l'élément x ajouté au sommet.
- **dépiler(p : pile) : pile**  
→ la fonction retourne la pile p (supposée non vide au départ) après avoir supprimé l'élément placé au sommet.

Parfois les fonctions *sommet* et *dépiler* sont réalisées simultanément

## Fonctions pour la file – Interface

Comme notre langage algorithmique est typé, on supposera que toutes les données sont d'un seul type, ici des entiers.

### Signature des fonctions pour une file d'entiers

- **initFile() : file**  
→ la fonction retourne une file vide.
- **fileVide(f : file) : booléen**  
→ la fonction teste si la file  $f$  est vide ou non.
- **tête(f : file) : entier**  
→ la fonction retourne l'élément placé en tête de la file  $f$  (elle retourne un message d'erreur si la file est vide).
- **enfiler(f : file, x : entier) : file**  
→ la fonction retourne la file  $f$  avec l'élément  $x$  ajouté en fin de file.
- **défiler(f : file) : file**  
→ la fonction retourne la file  $f$  (supposée non vide au départ) après avoir supprimé l'élément placé en tête.

Parfois les fonctions *tête* et *défiler* sont réalisées simultanément

## Plan du CM 4

Présentation des piles et des files

Fonctions sur les piles et des files

**Implémentation d'une pile**

Implémentation d'une file

Exemples d'utilisation d'une pile

## Implémentation d'une pile avec un tableau

### Choix effectués

- La pile  $P$  a une capacité maximale `maxPile` qui est une constante
- $P[0]$  contient  $n$  qui est le nombre d'éléments de la pile
- à cause de l'emplacement pour stocker  $n$ , la pile contient au plus `maxPile-1` éléments.
- les éléments sont placés des positions 1 à  $n$

### initPile

```
initPile() : tableau d'entiers
  P[maxPile] : tableau d'entiers
  P[0] = 0
  retourner P
```

### pileVide

```
pileVide(P : tableau d'entiers) : booléen
  retourner P[0] = 0
```

## Implémentation d'une pile avec un tableau

### sommet

```
sommet(P : tableau d'entiers) : entier  
    retourner P[P[0]]
```

- ici si la pile ne contient pas d'élément, on retourne la valeur 0, ce qui n'est pas correct
- pour gérer l'erreur, il faut d'abord utiliser la fonction `pileVide` avant d'appeler cette fonction.

## Implémentation d'une pile avec un tableau

### empiler

```
empiler(P : tableau d'entiers, x : entier) : tableau d'entiers
  si P[0] < maxPile - 1
    P[0] = P[0] + 1
    P[P[0]] = x
  retourner P
```

- si la pile est pleine, on ne fait rien
- en programmation, on peut aussi transmettre une **exception**

### dépiler

```
dépiler(P : tableau d'entiers) : tableau d'entiers
  P[0] = P[0] - 1
```

- on ne dépile que si la pile n'est pas vide
- il faut d'abord utiliser la fonction `pileVide` pour s'en assurer

# Implémentation d'une pile avec un tableau

## Complexité des fonctions

- toutes les procédures s'effectuent en temps constant
- en effet, le nombre d'instructions ne dépend pas du nombre d'éléments dans le tableau

## Bilan

- c'est un bon choix de structure de données
- `maxPile` doit être bien choisi



# Implémentation d'une pile avec une liste chaînée

## Choix effectués

- le sommet de la pile est la valeur du **premier nœud**
- pour empiler, il faut donc ajouter un nœud au **début de la liste chaînée**
- pour dépiler, il faut **supprimer le premier nœud**
- pas de problème de **pile pleine** avec une liste chaînée

## Type pile

`type pile = pointeur sur noeud`

## initPile

```
initPile() : pile
    retourner None
```

## pileVide

```
pileVide(P : pile) : booléen
    retourner P = None
```

## Implémentation d'une pile avec une liste chaînée

### sommet

```
sommet(P : pile) : entier  
    retourner P->valeur
```

- ici si la pile ne contient pas d'élément, nous aurons une erreur
- pour gérer l'erreur, il faut d'abord utiliser la fonction `pileVide` pour tester si la pile est vide
- si la pile est vide alors cette procédure n'est pas appelée

# Implémentation d'une pile avec une liste chaînée

## empiler

```
empiler(P : pile, x : entier) : pile
    tmp : pointeur sur noeud
    tmp = Nouveau(noeud) ; tmp->valeur = x ; tmp->suivant = P
    retourner tmp
```

- il s'agit d'un ajout en début de liste chaînée

## dépiler

```
dépiler(P : pile) : pile
    tmp : pointeur sur noeud
    tmp = P->suivant
    désallouer(P)
    retourner tmp
```

On ne peut pas retourner directement P->suivant si l'on veut faire la désallocation

- on ne dépile que si la pile n'est pas vide
- il faut d'abord tester si la pile est vide
- si la pile est vide alors cette procédure n'est pas appelée

# Implémentation d'une pile avec une liste chaînée

## Complexité des fonctions

- toutes les procédures s'effectuent en temps constant
- en effet, le nombre d'instructions ne dépend pas du nombre d'éléments dans le tableau

## Bilan

- c'est un bon choix de structure de données
- pas besoin de définir une capacité maximale

# Plan du CM 4

Présentation des piles et des files

Fonctions sur les piles et des files

Implémentation d'une pile

**Implémentation d'une file**

Exemples d'utilisation d'une pile

## Implémentation d'une file avec un tableau

### Choix effectués

- La file  $F$  a une capacité maximale `maxFile` qui est une constante.
- $F[0]$  contient le nombre d'éléments de la file.
- $F[1]$  contient l'élément en tête de la file.
- nous pouvons mettre au plus `maxFile - 1` éléments.

### initFile

```
initFile() : tableau d'entiers
    F[maxFile] : tableau d'entiers
    F[0] = 0
    retourner F
```

### fileVide

```
FileVide(F : tableau d'entiers) : booléen
    retourner F[0] = 0
```

## Implémentation d'une file avec un tableau

### tête

```
tete(F : tableau d'entiers) : entier  
    retourner F[1]
```

- si la file ne contient pas d'élément, on retourne une valeur erronée
- pour gérer l'erreur, il faut d'abord utiliser la fonction `fileVide` pour tester que la file est vide
- si la file est vide alors cette procédure n'est pas appelée

# Implémentation d'une file avec un tableau

## enfiler

```
enfiler(F : tableau d'entiers, x : entier) : tableau d'entiers
  si F[0] < maxFile - 1
    F[0] = F[0] + 1
    F[F[0]] = x
  retourner F
```

- si la file est pleine, on ne fait rien
- en programmation, on peut aussi transmettre une **exception**

## défiler

```
défiler(F : tableau d'entiers) : tableau d'entiers
  pour i allant de 1 à F[0]-1 faire
    F[i] = F[i+1]
  F[0] = F[0] - 1
```

- on ne défile que si la file n'est pas vide
- la méthode consiste à supprimer le premier élément du tableau et à décaler tous les autres éléments d'un pas vers la gauche



# Implémentation d'une file avec un tableau

## Complexité des fonctions

- on choisit comme coût pour **défiler** le nombre de décalages
- le coût est égal au nombre d'éléments de la file.
- toutes les autres fonctions se font en temps constant

## Bilan

A cause du coût de la fonction `défiler`, la structure de tableau ne semble pas bien adaptée pour implémenter une file.

# Implémentation d'une file avec une liste chaînée

## Choix effectués

- la tête de la file est la valeur du **premier nœud**
- pour enfiler, il faut ajouter un nœud **à la fin de la liste chaînée**
- pour défiler, il faut **supprimer le premier nœud**

## type file

```
type file = pointeur sur noeud
```

## initFile

```
initFile() : file
    retourner None
```

## fileVide

```
fileVide(F : file) : booléen
    retourner F = None
```

## Implémentation d'une file avec une liste chaînée

### tête

```
tete(F : file) : entier  
    retourner F->valeur
```

- si la file ne contient pas d'élément, nous aurons une erreur
- pour gérer l'erreur, il faut d'abord utiliser la fonction `fileVide` pour tester si la file est vide
- si la file est vide alors cette procédure n'est pas appelée

# Implémentation d'une file avec une liste chaînée

## enfiler

```

enfiler(F : File, x : entier) : file
  tmp1, tmp2 : pointeur sur noeud
  tmp1 = F
  tmp2 = Nouveau(noeud) ; tmp2->valeur = x ; tmp2->suivant = None
  si fileVide(F) alors retourner tmp2
  tant que tmp1->suivant <> None faire
    tmp1 = tmp1->suivant
  tmp1->suivant = tmp2
  retourner F

```

- la méthode utilisée est l'insertion en fin dans une liste chaînée

## Implémentation d'une file avec une liste chaînée

### défiler

```
défiler(F : file) : File
    tmp : pointeur sur noeud
    tmp = P->suivant
    désallouer(P)
    retourner tmp
```

- on ne défile que si la file n'est pas vide
- il faut d'abord tester si la file est vide
- si la file est vide alors cette procédure n'est pas appelée

# Implémentation d'une file avec une liste chaînée

## Complexité des fonctions

- on choisit comme coût pour **enfiler** le nombre de nœuds visités
- le coût est égal au nombre d'éléments de la file
- toutes les autres fonctions se font en temps constant

## Bilan

Comme pour le tableau, cette structure de données ne semble pas bien adaptée pour implémenter une file.

# Implémentation d'une file avec d'autres structures

## Autres structures de données

On montre que toutes les fonctions peuvent être implémentées en temps constant avec les deux structures de données suivantes.

### Liste avec deux pointeurs

On utilise une liste avec **un pointeur au début et un pointeur à la fin**.

### Liste circulaire

On utilise une liste **circulaire** avec un pointeur sur le dernier nœud

Voir CM 3 et TD 4

## Plan du CM 4

Présentation des piles et des files

Fonctions sur les piles et des files

Implémentation d'une pile

Implémentation d'une file

Exemples d'utilisation d'une pile



## Exemple 1 – bon parenthésage

### Algorithme

On parcourt l'expression caractère par caractère.

- si on lit le caractère (, on empile un ).
- si on lit le caractère ), on dépile un ).
- si on lit un autre caractère on ne fait rien.

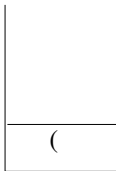
### Deux cas de mauvais parenthésage

- si on ne peut pas dépiler
  - ▶ plus de parenthèses fermantes que de parenthèses ouvrantes
- si à la fin la pile n'est pas vide
  - ▶ plus de parenthèses ouvrantes que de parenthèses fermantes

## Exemple 1 – bon parenthésage

### Exécution de l'algorithme sur un exemple

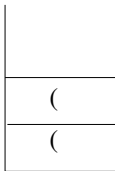
expression = ((2+5)\*(4-2)))



## Exemple 1 – bon parenthésage

### Exécution de l'algorithme sur un exemple

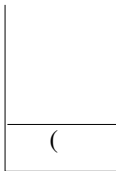
expression = ((2+5)\*(4-2)))



## Exemple 1 – bon parenthésage

### Exécution de l'algorithme sur un exemple

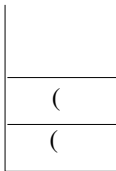
expression = ((2+5)\*(4-2)))



## Exemple 1 – bon parenthésage

### Exécution de l'algorithme sur un exemple

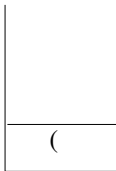
expression = ((2+5)\*(4-2)))



## Exemple 1 – bon parenthésage

### Exécution de l'algorithme sur un exemple

expression = ((2+5)\*(4-2)))



## Exemple 1 – bon parenthésage

### Exécution de l'algorithme sur un exemple

expression = ((2+5)\*(4-2)))



## Exemple 1 – bon parenthésage

### Exécution de l'algorithme sur un exemple

expression = ((2+5)\*(4-2)))

erreur la pile est vide



L'algorithme se finit sur une erreur



## Exemple 1 – bon parenthésage

### Procédure bienParenthese

```

bienParenthese(C : chaîne de caractères) : booléen
  P : pile ; P = initPile()
  L : entier ; L = longueur(C)
  i = 0
  tant que i < L faire
    si C[i] = '(' alors
      P = empiler(P, '(')
    si C[i] = ')' alors
      si pileVide(P) alors retourner Faux
      sinon P = dépiler(P)
    i = i + 1
  retourner pileVide(P)

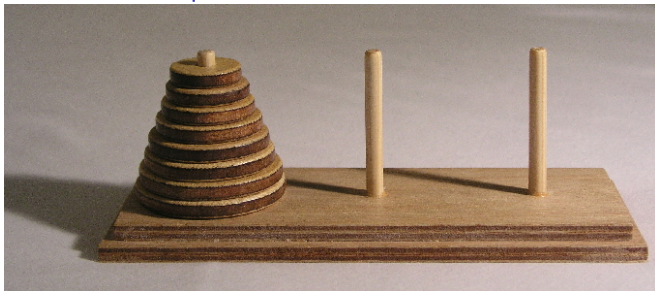
```

### Remarque

Nous pouvons nous passer d'une pile avec un seul type de parenthésage.  
En effet, nous pouvons remplacer la pile par un compteur.

## Exemple 2 – tours d'Hanoï

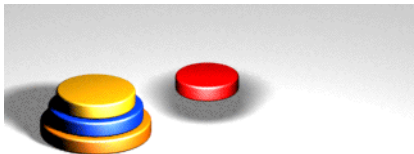
### Tour avec 8 disques



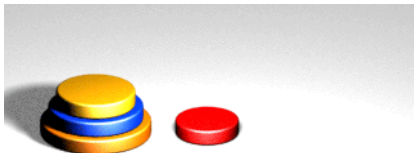
## Exemple 2 – tours d'Hanoï



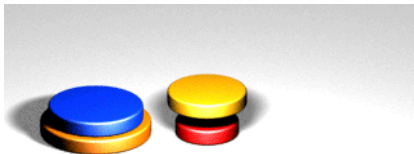
## Exemple 2 – tours d'Hanoï



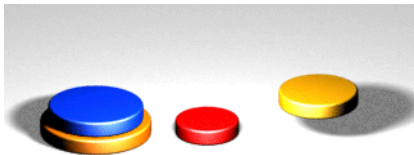
## Exemple 2 – tours d'Hanoï



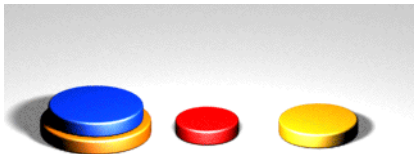
## Exemple 2 – tours d'Hanoï



## Exemple 2 – tours d'Hanoï

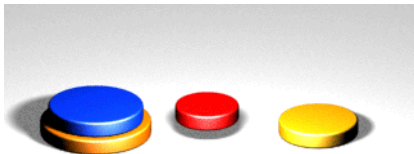


## Exemple 2 – tours d'Hanoï

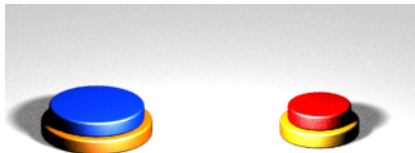




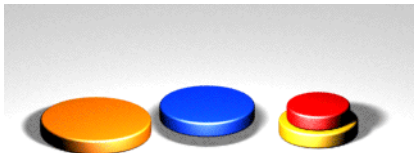
## Exemple 2 – tours d'Hanoï



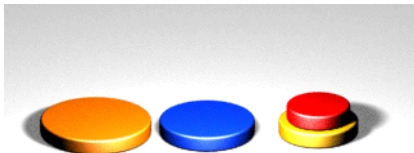
## Exemple 2 – tours d'Hanoï



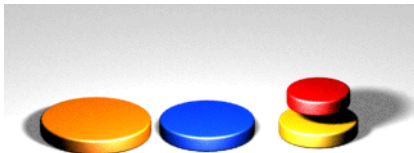
## Exemple 2 – tours d'Hanoï



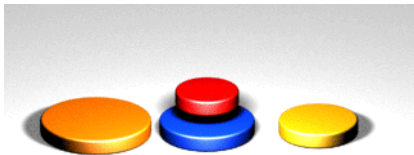
## Exemple 2 – tours d'Hanoï



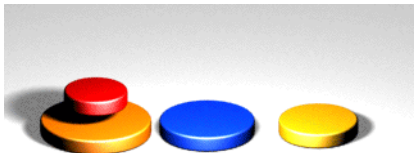
## Exemple 2 – tours d'Hanoï



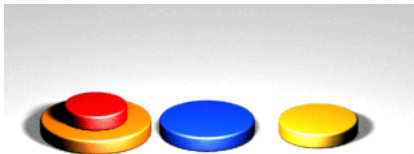
## Exemple 2 – tours d'Hanoï



## Exemple 2 – tours d'Hanoï

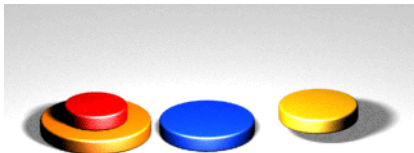


## Exemple 2 – tours d'Hanoï

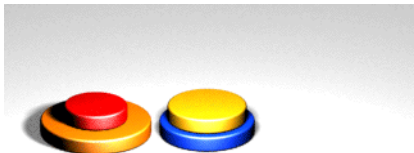




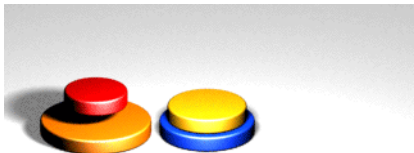
## Exemple 2 – tours d'Hanoï



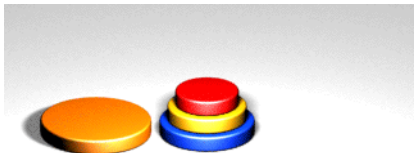
## Exemple 2 – tours d'Hanoï



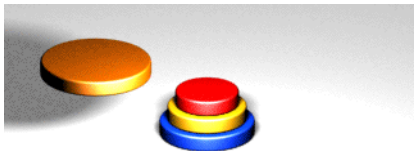
## Exemple 2 – tours d'Hanoï



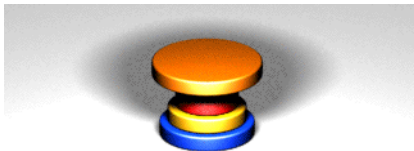
## Exemple 2 – tours d'Hanoï



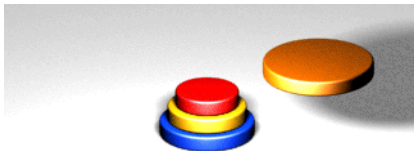
## Exemple 2 – tours d'Hanoï



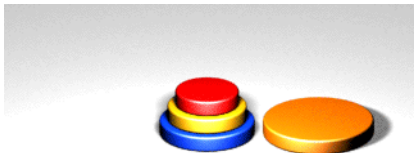
## Exemple 2 – tours d'Hanoï



## Exemple 2 – tours d'Hanoï

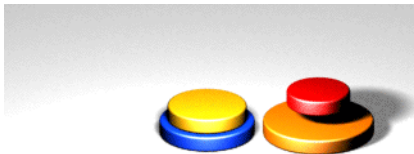


## Exemple 2 – tours d'Hanoï

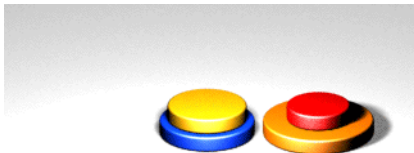




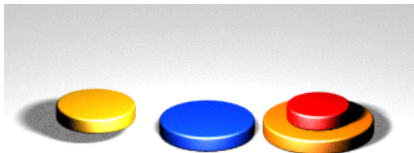
## Exemple 2 – tours d'Hanoï



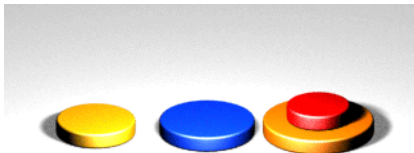
## Exemple 2 – tours d'Hanoï



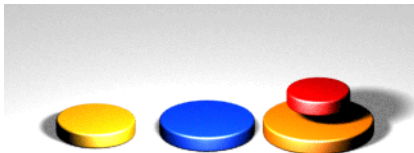
## Exemple 2 – tours d'Hanoï



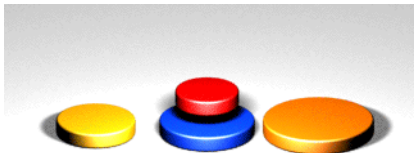
## Exemple 2 – tours d'Hanoï



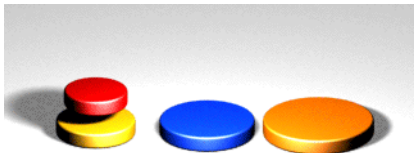
## Exemple 2 – tours d'Hanoï



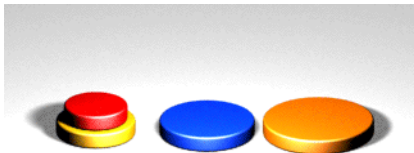
## Exemple 2 – tours d'Hanoï



## Exemple 2 – tours d'Hanoï

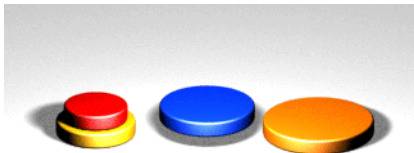


## Exemple 2 – tours d'Hanoï





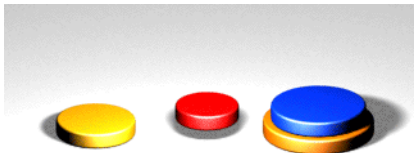
## Exemple 2 – tours d'Hanoï



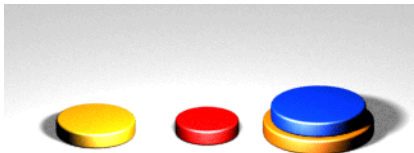
## Exemple 2 – tours d'Hanoï



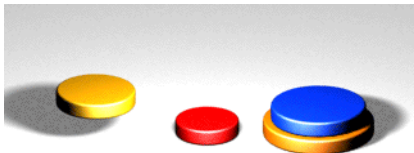
## Exemple 2 – tours d'Hanoï



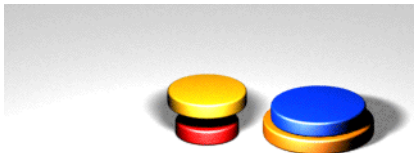
## Exemple 2 – tours d'Hanoï



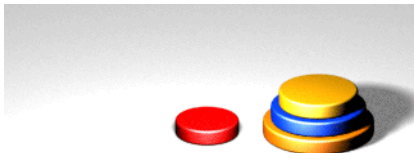
## Exemple 2 – tours d'Hanoï



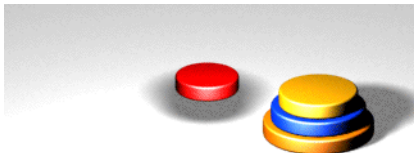
## Exemple 2 – tours d'Hanoï



## Exemple 2 – tours d'Hanoï



## Exemple 2 – tours d'Hanoï

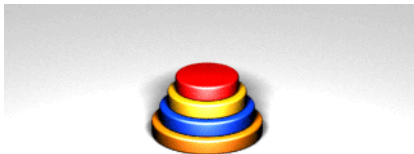




## Exemple 2 – tours d'Hanoï



## Exemple 2 – tours d'Hanoï



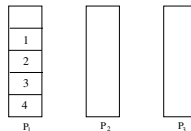
## Exemple 2 – tours d'Hanoï

### Formalisation du problème

- trois piles  $P_1$ ,  $P_2$  et  $P_3$
- $n$  disques de taille  $1, \dots, n$
- les trois piles sont empilées par ordre décroissant

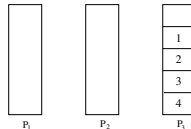
### Configuration du début

$P_1$  contient les  $n$  disques.



### Configuration de la fin

$P_3$  contient les  $n$  disques.



## Exemple 2 – tours d'Hanoï

### Déplacement autorisé

On fixe deux tours de Hanoï  $P_i$  et  $P_j$ ,  $i, j \in \{1, \dots, n\}$ ,  $i \neq j$ .

Le déplacement d'un disque de  $P_i$  vers  $P_j$  est autorisé lorsque les deux conditions suivantes sont vérifiées :

1.  $P_i$  est non vide
2.  $P_j$  est vide ou  $\text{sommet}(P_i) < \text{sommet}(P_j)$

### Déplacement effectué

On déplace  $\text{sommet}(P_i)$  vers  $P_j$ .

## Exemple 2 – tours d'Hanoï

### Codage d'un déplacement

On suppose ici que les tests pour savoir si le déplacement est possible est effectué avant l'appel de la procédure.

```
deplacementTours(P : tableau de piles, i,j : entier):tableau de piles
    d : entier
    d = sommet(P[i])
    P[i] = depiler(P[i])
    P[j] = empiler(P[j],d)
    retourner P
```

## Exemple 2 – tours d'Hanoï

Meilleure solution pour  $n = 4$ ,  $15 = 2^4 - 1$  déplacements.

étape 1	<code>P = deplacementTours (P, 1, 2)</code>
étape 2	<code>P = deplacementTours (P, 1, 3)</code>
étape 3	<code>P = deplacementTours (P, 2, 3)</code>
étape 4	<code>P = deplacementTours (P, 1, 2)</code>
étape 5	<code>P = deplacementTours (P, 3, 1)</code>
étape 6	<code>P = deplacementTours (P, 3, 2)</code>
étape 7	<code>P = deplacementTours (P, 1, 2)</code>
étape 8	<code>P = deplacementTours (P, 1, 3)</code>
étape 9	<code>P = deplacementTours (P, 2, 3)</code>
étape 10	<code>P = deplacementTours (P, 2, 1)</code>
étape 11	<code>P = deplacementTours (P, 3, 1)</code>
étape 12	<code>P = deplacementTours (P, 2, 3)</code>
étape 13	<code>P = deplacementTours (P, 1, 2)</code>
étape 14	<code>P = deplacementTours (P, 1, 3)</code>
étape 15	<code>P = deplacementTours (P, 2, 3)</code>

Pour une tour de Hanoï contenant  $n$  disques

Le nombre de déplacements pour la meilleure solution vaut  $2^n - 1$  déplacements.

Méthodes pour trouver la solution

Elles peuvent être récursives ou itératives.