IUT de Montpellier **TP : Jeux récursifs**
M3103 Algorithmique avancée **2016-2017**

Exercice 1. Les tours de Hanoï

Le but de ce TP est d'écrire le jeu des tours de Hanoï, célèbre chez les informaticiens [1]

Dans ce jeu on considère 3 poteaux (dénommés "1" (à gauche), "2" (au milieu), et "3" (à droite)), ainsi que N disques de diamètres deux à deux distincts. Les disques sont troués en leur centre, de telle sorte que l'on puisse les enfiler sur les poteaux.

Dans la situation initiale, les N disques sont sur le poteau gauche, et rangés "en pyramide" : c'est à dire de telle sorte que les plus petits disques sont au dessus (voir Figure 1).

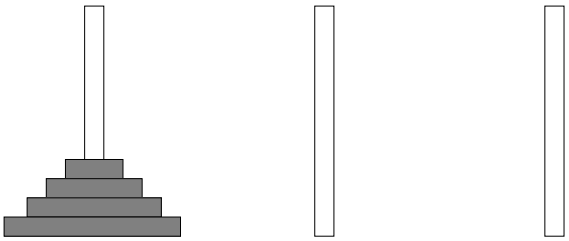


FIGURE 1 – Exemple avec $N = 4$.

Le but du jeu est de déplacer cette pyramide sur le poteau de droite, en sachant qu'un coup légal consiste à

- choisir un poteau de départ, et prendre le disque du dessus
- choisir un poteau d'arrivée, et déposer le disque au sommet
- s'assurer que sur chaque poteau les disques sont rangés par *ordre décroissant*, c'est à dire avec les plus petits disques au sommet (autrement dit, un disque ne peut être placé que sur un disque de plus grande taille).

Par exemple, la succession de coups "1 -> 2", "1 -> 3", "2 -> 3" est légale, alors que la succession de coups "1 -> 2", "1 -> 2" ne l'est pas.

On souhaite écrire une méthode `public void resoudre(int n)` qui pour tout $n \geq 1$, affiche une liste de coups qui en partant de l'état initial du jeu (les n disques sur le poteau 1) mène à l'état final (tous les disques sur le poteau 3).

Question 1.1.
Ecrire d'abord la méthode `public void resoudreAux(int n, int i, int j)` qui pour tout $n \geq 1$ affiche une liste de coups permettant, étant donné un jeu de Hanoï avec n disques initialement sur le poteau i, de déplacer ces n disques sur le poteau j (avec i et j dans $\{1, 2, 3\}, i \neq j$).
Par exemple, resoudreAux(2,2,1) affiche :
"2 -> 3"
"2 -> 1"
"3 -> 1"

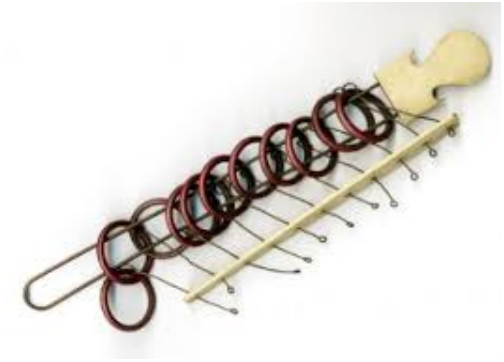
Question 1.2.
En déduire la méthode `public void resoudre(int n)`

Question 1.3.
Ecrire une méthode qui calcule u_n, le nombre de coups qu'utilise resoudre(n) pour résoudre Hanoï à n disques. Observez les valeurs .. finalement, que vaut u_n ?

1. En particulier pour la découverte de la récurrence !

Question 1.4.
(Bonus) Que faut-il changer si l'on souhaite maintenant écrire `public void resoudreAux(int n, int i, int j)` et `public void resoudre(int n)` pour tout $n \geq 0$? (et non plus $n \geq 1$)

Exercice 2. Le baguenaudier et la récursion croisée [2]**.**



Le jeu du Baguenaudier se modélise de la façon suivante. On considère un tablette à n cases. Chaque case peut être vide ou occupée. Les cases du Baguenaudier sont numérotées à partir de 0. Jouer sur la case i, $0 \leq i < n$ signifie ajouter un pion en case i (si la case était vide), ou enlever un pion (si la case était pleine). A chaque coup, les seules cases jouables sont la case 0 (tout à gauche), ou la case qui suit la première case occupée.

Nous considérons les deux problèmes suivant :

— remplir un Baguenaudier de n cases initialement vide

— vider un Baguenaudier de n cases initialement plein

Par exemple, voici une séquence permettant de remplir un Baguenaudier à 3 cases :

```
. . .

* . .

* * .

. * .

. * *

* * *
```

Question 2.1.
Ecrire les méthodes `public void remplir(int n)` qui pour $n \geq 1$ affiche une liste de coups qui en partant d'un baguenaudier (à n cases) vide permet d'arriver à un baguenaudier plein, et `public void vider(int n)` qui pour $n \geq 1$ affiche une liste de coups qui en partant d'un baguenaudier (à n cases) plein permet d'arriver à un baguenaudier vide. Indication : pensez à utiliser "vider" pour écrire "remplir", et à utiliser "remplir" pour écrire "vider" ! Cette technique s'appelle la récursion croisée. Autrement dit, posez vous la question suivante : comment remplir les cases d'indices $0..i$ en supposant que pour tout $j < i$, on sait vider ou remplir les cases d'indices $0..j$?

2. Cet exercice est directement inspiré du TP de Karine Zampieri, Stephane Riviere, Beatrice Amerein-Soltner, de Unisciel Algoprog.

Question 2.2.
On souhaite maintenant afficher non seulement les coups joués, mais aussi les états successifs du baguenaudier. Pour cela, écrire une classe Baguenaudier ayant 2 atributs : un entier nbCases ≥ 1 et un tableau de longueur nbCases (contenant des entiers, des caractères ou des booléens à votre choix), et contenant les méthodes, remplir, remplirAux, vider et viderAux dont les spècifications sont les suivantes.

```
public void remplir()
/** pré-requis : this est vide
* action : exécute une partie du jeu de baguenaudier sur this
permettant de le remplir, en affichant le déroulement du jeu
(c'est-à-dire en affichant l'état initial, puis pour chaque coup joué
le coup et le nouvel état de this)
* Par exemple, si nbCases = 1, la méthode affiche :
* le baguenaudier à une case vide,
* le coup "remplir la case 0"
* le baguenaudier à une case plein
*/ à la fin de l'exécution, this est plein

public void remplirAux(int n)
// pré-requis : 1 <= n <= this.nbCases, les n premières cases de
this, c'est-à-dire d'indices 0 à n-1, sont vides.
// action : exécute une partie du jeu de baguenaudier sur this
permettant d'en remplir les n premières cases sans changer l'état
des cases suivantes, en affichant le déroulement du jeu (sauf l'état
initial, c'est-à-dire en affichant pour chaque coup joué le coup et
le nouvel état de this)
```

Les spécifications de vider et viderAux sont similaires.
NB : Rappel : pour déterminer les cas de base, vous pouvez appliquer la méthode suivante :

1. déterminer les appels récursifs à effectuer
2. les cas de base sont les cas où les appels récursifs ne satifont pas les pré-requis