Université catholique de Louvain

Ecole Polytechnique de Louvain

Projet du cadran

Auteur:	*Matricule:*
Marie Visschers	0880-09-00
Audrey Wenders	3530-10-00
Helena Russello	5569-14-00
Wojciech Grynczel	8803-14-00

May 10, 2015

1 Théorie du problème

Tout d'abord, commençons par définir ce que l'on entend par cadran dans notre projet : un cadran est un cercle gradué par des nombres entiers. Le cadran se lit en commençant par son sommet et en continuant dans le sens horloger.

Définissons également la notion de rotation : une rotation dans un cadran est un déplacement d'une ou plusieurs unités de tous les éléments de ce cadran, l'ordre de ceux-ci restant inchangé.

Soit l le nombre d'éléments dans le cadran et k égal à $l-1$ (k correspond donc à l'indice du dernier élément du cadran (cfr cadran ci-dessous)).

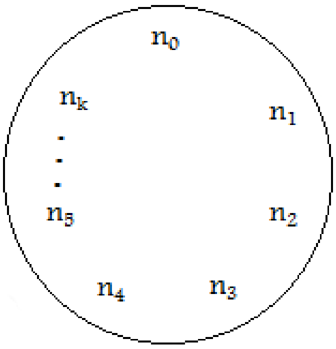

Figure 1: Le cadran de départ (avant rotation).

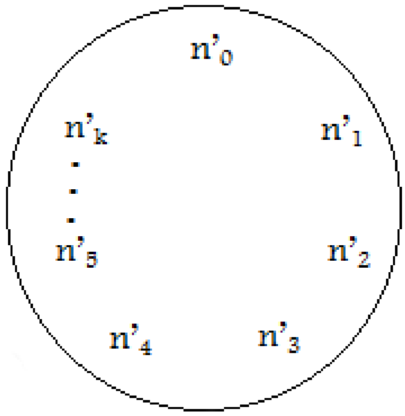

Figure 2: Le cadran après une rotation de longueur r.

Après une rotation de longueur r dans un cadran contenant au moins 2 éléments, tous les éléments de celui-ci sont déplacés de r positions vers la droite.

Nous pouvons donc constater que $\forall l > 1, \forall i \in [0, k] : si \quad i - r \geq 0 \; alors \; n'_i = n_{i-r} \; sinon \; n'_i = n_{l+(i-r)}$

Exemples:

- Si $i - r \geq 0$
  Dans cet exemple, $i = 4$ (c'est n_4). Et la rotation r est de 3. Donc pour trouver l'élément n'_4 on doit prendre la valeur n_{i-r} c'est-à-dire la valeur n_1.

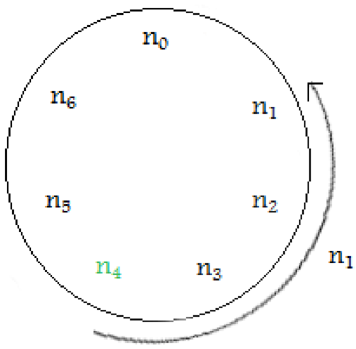

Figure 3

- Si $i - r < 0$
  Dans cet exemple, $i = 1$ (c'est n_1). Et la rotation r est de 3. Donc pour trouver l'élément n'_1, nous remplaçons l'élément n_1 par la valeur se trouvant 3 éléments vers la gauche. $n_1 - 3$ nous donne n_{-2} ce qui correspond à la valeur n_5 dans le cadran (car $n_5 = n_{l+(i-r)} = n_{7+(-2)}$). n'_1 sera donc remplacé par l'élément n_5.

En outre, nous remarquons qu'un cadran vide ou ne contenant qu'un seul élément ne peut pas être soumis à une rotation puisqu'une rotation demande de déplacer plusieurs nombres, or un tableau vide n'en contient pas et un tableau d'un élément n'en contient qu'un.

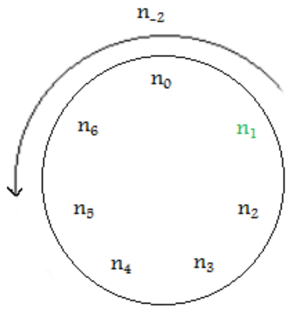

Figure 4

Nous remarquons également que toute rotation dans un cadran de n éléments dont la longueur r est un multiple de n ne modifiera pas le cadran (cela correspondra à des rotations complètes du cadran). Par exemple, si le cadran contient les nombres 1, 2, 3 et 4 et qu'on effectue une rotation de longueur 4, le cardan finale sera identique au cadran initial. Même chose si on effectue une rotation de 8, 12 ou 28 unités.

$\forall r, l \in N : si\ l\ mod\ r = 0\ alors\ \forall i \in [0, k] : n'_i = n_i$

De plus, nous pouvons admettre qu'une rotation de longueur r revient à faire une rotation de r modulo n. En effet, si r est supérieur ou égal à n, les rotations complètes ne sont pas nécessaires (en effet, ci-dessus nous avons constaté que cela ne modifiait pas le cadran), nous pourrons donc nous contenter de faire la rotation r modulo n. Et dans le cas où r est inférieur à n, nous aurons bien une rotation de r (puisque $\forall r < n : r\ mod\ n = r$).

En nous basant sur les propriétés ci-dessus nous constatons que pour arriver au cadran final, nous effectuons des rotations successives par saut de r. Chaque élément est remplacé par l'élément du cadran initial se trouvant r position(s) plus à gauche. Il est donc possible d'arriver au premier élément modifié sans que tous les éléments du cadran aient été déplacés. Typiquement cela arrive lorsque le *pgcd* de l et de r est inférieur à r.

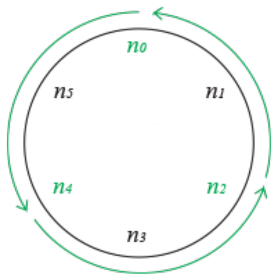

Figure 5

Pour parvenir à un cadran qui a effectué une rotation de longueur r il faut s'assurer que tous les éléments du cadran soient déplacés une seule fois ($O(n)$). De ce fait, lorsque l'on arrive au premier élément placé et que l'on n'a pas déplacé tous les éléments du tableau, on reprend la rotation à partir de l'élément à droite de l'élément courant. Lorsqu'on retournera à ce nouvel élément, si tous les éléments ont été déplacés, le cadran sera bien celui que nous devions obtenir après la rotation, sinon il faudra à nouveau reprendre la rotation à partir de l'élément à droite de l'élément courant et ainsi de suite jusqu'à ce que tous les éléments soient déplacés.

2 Conventions de représentation (fournies dans le projet)

Notre cadran sera représenté par un tableau de nombres entiers Java. Un entier est de type "int" en Java et appartient à [-(2^{31}) .. 2^{31} - 1[. Pour construire le tableau à partir du cadran, la convention est de dire que le premier élément du tableau sera l'élément au sommet du cadran. Les éléments suivants sont alors les éléments du cadran dans le sens des aiguilles d'une montre.

3 Problème principal

Pour ce problème, nous n'avons pas eu besoin de définir des sous-problèmes.

3.1 Spécifications

En-tête : public static void rotate(int[] c, int r)

Pré : c est un tableau d'entiers représentant le cadran à manipuler
&& $c \neq null$
& r est une entier représentant la rotation réelle à effectuer sur le cadran
&& $r \geq 0$

Post : le tableau c a été modifié afin de représenter le cadran après rotation de r.

Résultat : /

Invariant : $c \neq null$
&& $r \geq 0$
&& $0 \leq j \leq c.length$
&& $0 \leq i \leq nbElemVisite \leq c.length$
&& $-c.length \leq position \leq c.length$

3.2 Construction de l'algorithme

```
INIT
/* INV */
while(!H)
ITER
CLOT
```

Variables locales : int r, i, j, position, tmp, nbElemVisite;

INIT :
```
if c.length = 0
    r := 0
    tmp := 0
else
    r := r mod c.length;
    tmp := c[0];
i := 0;
j := i;
nbElemVisite := 0;
position := j - r;
```

H : $r = 0 \vee$ nbElemVisite $\geq$ c.length

ITER :
```
if position < 0
    position := position + c.length;
if i = position
    c[j] := tmp;
    i++;
    tmp:= c[i];
    j := i;
else
    c[j] := c[position];
    j := position;
nbElemVisite = nbElemVisite + 1;
position := j - r;;
```

CLOT : /

3.3 Exécutions symboliques

3.3.1 {PRE} INIT {INV}

Notons c_0 et r_0 la valeur des paramètres avant l'exécution de $INIT$.

$\{c = c_0, r = r_0\}$ tq (c_0 est un tableau d'entiers représentant le cadran à manipuler
&& $c_0 \neq null$
& r_0 est une entier représentant la rotation réelle à effectuer sur le cadran
&& $r_0 \geq 0$)

if(c.lenght == 0)
↓ r := 0

$\{c = c_0, r = r_0\}$ tq (c_0 est un tableau d'entiers représentant le cadran à manipuler
&& $c_0 \neq null$
& r_0 est une entier représentant la rotation réelle à effectuer sur le cadran
&& $r_0 \geq 0$
&& $c_o.length = 0$)

↓ tmp := 0

$\{c = c_0, r = r_0, tmp = 0\}$ tq (c_0 est un tableau d'entiers représentant le cadran à manipuler
&& $c_0 \neq null$
& r_0 est une entier représentant la rotation réelle à effectuer sur le cadran
&& $r_0 \geq 0$
&& $c_o.length = 0$)

↓ i := 0

$\{c = c_0, r = r_0, tmp = 0, i = 0\}$ tq (c_0 est un tableau d'entiers représentant le cadran à manipuler
&& $c_0 \neq null$
& r_0 est une entier représentant la rotation réelle à effectuer sur le cadran
&& $r_0 \geq 0$
&& $c_o.length = 0$)

↓ j := i

$\{c = c_0, r = r_0, tmp = 0, i = 0, j = 0\}$ tq (c_0 est un tableau d'entiers représentant le cadran à manipuler
&& $c_0 \neq null$
& r_0 est une entier représentant la rotation réelle à effectuer sur le cadran
&& $r_0 \geq 0$
&& $c_o.length = 0$)

↓ nbElemVisite := 0

$\{c = c_0, r = r_0, tmp = 0, i = 0, j = 0, nbElemVisite = 0\}$ tq
(c_0 est un tableau d'entiers représentant le cadran à manipuler
&& $c_0 \neq null$
& r_0 est une entier représentant la rotation réelle à effectuer sur le cadran
&& $r_0 \geq 0$
&& $c_o.length = 0$)

↓ **position := j-r**

$\{c = c_0, r = r_0, tmp = 0, i = 0, j = 0, nbElemVisite = 0, position = 0\}$ tq
(c_0 est un tableau d'entiers représentant le cadran à manipuler
&& $c_0 \neq null$
& r_0 est une entier représentant la rotation réelle à effectuer sur le cadran
&& $r_0 \geq 0$
&& $c_o.length = 0$)

Prouver l'invariant :

- $c = c_0$ et $c_0 \neq null \rightarrow c \neq null$
- $r = 0 \rightarrow r \geq 0$
- $j = 0$ et $c_0.length = 0$ et $c = c_0 \rightarrow 0 \leq j \leq c.length$
- $i = 0$ et $nbElemVisite = 0$ et $c_0.length = 0$ et $c = c_0 \rightarrow 0 \leq i \leq nbElemVisite \leq c.length$
- $position = 0$ et $c_0.length = 0$ et $c = c_0$ et c_0 est un tableau ($c.length \geq 0$) $\rightarrow -c.length \leq position \leq c.length$

else // if(c.length != 0)
↓ **r := r mod c.length**

$\{c = c_0, r = r_0 \bmod c_0.length\}$ tq (c_0 est un tableau d'entiers représentant le cadran à manipuler
&& $c_0 \neq null$
& r_0 est une entier représentant la rotation réelle à effectuer sur le cadran
&& $r_0 \geq 0$
&& $c_o.length \neq 0$)

↓ **tmp := c[0]**

$\{c = c_0, r = r_0 \bmod c_0.length, tmp = c_0[0]\}$ tq (c_0 est un tableau d'entiers représentant le cadran à manipuler
&& $c_0 \neq null$
& r_0 est une entier représentant la rotation réelle à effectuer sur le cadran
&& $r_0 \geq 0$
&& $c_o.length \neq 0$)

↓ **i := 0**

$\{c = c_0, r = r_0 \bmod c_0.length, tmp = c_0[0], i = 0\}$ tq
(c_0 est un tableau d'entiers représentant le cadran à manipuler
&& $c_0 \neq null$
& r_0 est une entier représentant la rotation réelle à effectuer sur le cadran
&& $r_0 \geq 0$
&& $c_o.length \neq 0$)

↓ **j := i**

$\{c = c_0, r = r_0 \bmod c_0.length, tmp = c_0[0], i = 0, j = 0\}$ tq
(c_0 est un tableau d'entiers représentant le cadran à manipuler
&& $c_0 \neq null$
& r_0 est une entier représentant la rotation réelle à effectuer sur le cadran
&& $r_0 \geq 0$
&& $c_o.length \neq 0$)

↓ **nbElemVisite := 0**

$\{c = c_0, r = r_0 \mod c_0.length, tmp = c_0[0], i = 0, j = 0, nbElemVisite = 0\}$ tq

(c_0 est un tableau d'entiers représentant le cadran à manipuler
&& $c_0 \neq null$
& r_0 est une entier représentant la rotation réelle à effectuer sur le cadran
&& $r_0 \geq 0$
&& $c_o.length \neq 0$)

↓ **position := j-r**

$\{c = c_0, r = r_0 \mod c_0.length, tmp = c_0[0], i = 0, j = 0, nbElemVisite = 0, position = 0\}$ tq

(c_0 est un tableau d'entiers représentant le cadran à manipuler
&& $c_0 \neq null$
& r_0 est une entier représentant la rotation réelle à effectuer sur le cadran
&& $r_0 \geq 0$
&& $c_o.length \neq 0$)

Prouver l'invariant :

- $c = c_0$ et $c_0 \neq null \rightarrow c \neq null$
- $r = r_0 \mod c_0.length$ et $r_0 \geq 0$ et $c_0.length \neq 0$ et c_0 est un tableau d'entiers représentant le cadran à manipuler
  $\rightarrow$ La longeur d'un tableau (ici c_0) est forcément $\geq$ à 0. Comme $c_0.length \neq 0$, on peut dire que $c_0.length > 0$. Sachant que: $\forall x \geq 0, \forall y > 0 : x \mod y \geq 0$, sachant que $c_0.lenght > 0$ et $r_0 \geq 0$ nous pouvons dire que $r_0 \mod c_0.length \geq 0$ donc $r \geq 0$.
- $j = 0$ et $c_0.length \neq 0$ et $c = c_0$ et c_0 est un tableau d'entiers représentant le cadran à manipuler
  $\rightarrow 0 \leq j \leq c.length$ (car la longueur d'un tableau non vide est forcément supérieure à 0)
- $i = 0$ et $nbElemVisite = 0$ et $c_0.length \neq 0$ et $c = c_0$ et c_0 est un tableau d'entiers représentant le cadran à manipuler
  $\rightarrow 0 \leq i \leq nbElemVisite \leq c.length$ (car la longueur d'un tableau non vide est forcément supérieure à 0)
- $position = 0$ et $c_0.length = 0$ et $c = c_0$ et c_0 est un tableau ($c.length \geq 0$)
  $\rightarrow -c.length \leq position \leq c.length$

L'invariant est vérifié pour les 2 conditions. Notons désormais que $0 \leq r \leq c.length$. En effet,$\forall x \geq 0, \forall y > 0 : x \mod y \leq y$. Par conséquent,$r_0 \mod c_0.length \leq c_0.length = c.length$ à la fin de l'$INIT$. C'est aussi vrai dans le cas où $r_0 = 0$ puisque $c_0.length \geq 0$.

3.3.2 {INV && !H} ITER {INV} :

$c \neq null$
&& $r \geq 0$
&& $0 \leq j \leq c.length$
&& $0 \leq i \leq nbElemVisite \leq c.length$
&& $-c.length \leq position \leq c.length$
&& $(r \neq 0$ && $nbElemVisite < c.length)//!H$

Nous commençons par prouver que l'évaluation de la condition d'arrêt ne conduit pas à une erreur si l'invariant est vrai :

- L'invariant impose une condition sur r. Cela veut implicitement dire que r est initialisée. Il n'y a donc pas d'erreur lors de l'évaluation de la première partie de la condition d'arrêt.
- Supposons que $r = 0$, la condition d'arrêt est vraie et la deuxième partie de la condition d'arrêt n'est pas évaluée. L'invariant nous dit que $r \geq 0$, il n'y a donc pas d'erreur d'exécution.
- Supposons que $r \neq 0$, alors la condition $r \geq 0$ de l'invariant implique que $r > 0$. On évalue alors la deuxième partie de la condition d'arrêt qui vérifie si $nbElemVisite \geq c.length$. Si cette condition est vérifiée, en la combinant avec l'invariant nous obtenons $nbElemVisite = c.length$, ce qui implique qu'il n'y aura pas d'erreur d'exécution.

Passons maintenant à l'exécution symbolique. Notons i_1 la valeur de i avant l'exécution de $ITER$ (idem pour les autres variables). Nous simplifions un peu $\{INV$ && $!H\}$, et ajoutons la condition trouvée après l'exécution d'$INIT$ ($r_1 \leq c_1.length$ nous pouvons convenir que cela restera vrai si nous prouvons à la fin de l'exécution symbolique que $r = r_1$). Nous obtenons alors les conditions vraies suivantes :

$c_1 \neq null$
&& $0 < r_1 \leq c_1.length$ (1)
&& $0 \leq j_1 \leq c_1.length$
&& $0 \leq i_1 \leq nbElemVisite1 < c_1.length$ (2)
&& $-c.length \leq position_1 \leq c_1.length$

Notons que (1) et (2) résultent non seulement de l'invariant mais aussi du fait que la condition d'arrêt est fausse.

Cas 1 : $position < 0$

$\{position = position_1\}$

$\downarrow position := position + c.length$

$\{position = position_1 + c_1.length\}$

- $-c.length \leq position \leq c.length$
  - cette condition est vérifiée puisque nous sommes dans le cas où $position_1 < 0$, nous savons que $c_1.length \geq 0$ (en effet, la longueur d'un tableau ne peut pas être inférieure à 0) et nous savons que $-c_1.length \leq position_1 \leq c_1.length.$
    Donc $-c.length \leq position = position_1 + c_1.length \leq c1.length = c.length$ Comme $-c_1.length \leq position_1 \leq c_1.length$ et $position = position_1 + c_1.length$ on sait pour la suite que $position \geq 0.$

Pour la suite nous noterons $position_2$ la position après l'execution du cas 1 c'est à dire $position = position_1 + c_1.length.$ Pour rappel $0 \leq position_2 \leq c_1.length.$

Cas 2a : $i = position$

$\{i = i_1, j = j_1, r = r_1, tmp = tmp_1, c[j_1] = c_1[j], nbElemVisite = nbElemVisite_1, position = position_2\}$

↓ c[j] := tmp

$\{i = i_1, j = j_1, r = r_1, tmp = tmp_1, c[j_1] = tmp_1, nbElemVisite = nbElemVisite_1, position = position_2\}$

↓ i := i+1

$\{i = i_1{+}1, j = j_1, r = r_1, tmp = tmp_1, c[j_1] = tmp_1, nbElemVisite = nbElemVisite_1, position = position_2\}$

↓ tmp := c[i]

$\{i = i_1 + 1, j = j_1, r = r_1, tmp = c[i_1 + 1], c[j_1] = tmp_1, nbElemVisite = nbElemVisite_1, position = position_2\}$

↓ nbElemVisite := nbElemVisite +1

$\{i = i_1 + 1, j = j_1, r = r_1, tmp = c[i_1 + 1], c[j_1] = tmp_1, nbElemVisite = nbElemVisite + 1, position = position_2\}$

↓ position := j - r

$\{i = i_1 + 1, j = j_1, r = r_1, tmp = c[i_1 + 1], c[j_1] = tmp_1, nbElemVisite = nbElemVisite + 1, position = i_1 + 1 - r_1\}$

- $c \neq null$
  - $c_1 = c$ et $c_1 \neq null$ donc $c \neq null$
- $0 < r \leq c.length$ nous n'utilisons pas $r(= r_1)$ donc $\{INV$ && $!H\}$ est vérifié
- $0 \leq j \leq c.length$
  - $\{INV$ && $!H\}$ nous dit que $0 \leq i_1 \leq nbElemVisite_1 < c_1.length$ donc $0 \leq i_1 < c_1.length$. Comme $j = i_1 + 1$ et $c_0 = c$, nous avons $0 \leq j \leq c_1.length = c.length$.
- $nbElemVisite_1 < c_1.length$ et $nbElemVisite = nbElemVisite_1$ et $c_1 = c$ donc {INV} est vérifié.
- $0 \leq i \leq c.length$
  - $\{INV$ && $!H\}$ nous dit que $0 \leq i_1 \leq nbElemVisite_1 < c_1.length$. Donc $0 \leq i_1 < c_1.length$. Comme $i = i_1 + 1$ et $c_0 = c$, nous pouvons en conclure que $0 \leq i \leq c.length$ est bien respecté.
- $-c.length \leq position \leq c.length$
  - $position = i_1 + 1 - r_1 = i - r_1$ et comme $0 < r_1 \leq c.length$ et $0 \leq i \leq c.length$ (prouvé ci-dessus) alors $-c.length \leq position \leq c.length$ est bien respecté.

<u>Cas 2b</u> : $i < position$

$\{c[j_1] = c_1[j], r = r_1, position = position_2, j = j_1, i = i_1, nbElemVisite = nbElemVisite_1\}$

↓ **c[j] := c[position]**

$\{c[j_1] = c[position_2], r = r_1, position = position_2, j = j_1, i = i_1, nbElemVisite = nbElemVisite_1\}$

↓ **j := position**

$\{c[j_1] = c[position_2], r = r_1, position = position_2, j = position_2, i = i_1, nbElemVisite = nbElemVisite_1\}$

↓ **nbElemVisite = nbElemVisite + 1**

$\{c[j_1] = c[position_2], r = r_1, position = position_2, j = position_2, i = i_1, nbElemVisite = nbElemVisite_1 + 1\}$

↓ **position = j - r**

$\{c[j_1] = c[position_2], r = r_1, position = position_2 - r_1, j = position_2, i = i_1, nbElemVisite = nbElemVisite_1 + 1\}$

- $c \neq null$ idem que cas 2a
- $0 < r \leq c.length$ idem que cas 2a
- $0 \leq j \leq c.length$
  - $j = position_2$ et $0 \leq position_2 \leq c_1.length$ donc $0 \leq j \leq c_1.length = c.length$
- $0 \leq i \leq c.length$, nous n'utilisons pas $i(= i_1)$ donc $\{INV$ && $!H\}$ est vérifié
- $-c.length \leq position \leq c.length$
  - $position = position_2 - r1$. Comme $0 \leq position_2 \leq c_1.length$ et $0 < r_1 \leq c_1.length$, nous pouvons en déduire que $-c_1.length \leq position < c_1.length$ donc nous avons bien prouver que $-c.length \leq position \leq c.length$.

Cas 2c : $i > position$

$$\{c[j_1] = c[position_2], r = r_1, position = position_2 - r_1, j = position_2, i = i_1, nbElemVisite = nbElemVisite_1 + 1\}$$

- $c \neq null$ idem que cas 2b
- $0 < r \leq c.length$ idem que cas 2b
- $0 \leq j \leq c.length$
  - $j = position_2$ et $0 \leq position_2 \leq c_1.length$ donc $0 \leq j \leq c_1.length = c.length$
- $0 \leq i \leq c.length$ idem que 2b
- $-c.length \leq position \leq c.length$ idem que 2b

Nous voyons bien que dans les différents cas $r = r_1$ nous pouvons donc en conclure que la condition $r \leq c.length$ ajoutée dans l'$INIT$ est bien vérifiée.

3.3.3 {INV && H} CLOT {POST} :

Les variables $r_2, tmp_2, j_2, i_2, nbElemVisite_2$ et $position_2$ sont les valeurs avant et après $CLOT$ puisque nous sommes dans le cas d'une méthode sans valeur de retour. Il ne se passe rien lors de la clôture.

$$\begin{aligned}
&\{r = r_2, tmp = tmp_2, j = j_2, i = i_2, nbElemVisite = nbElemVisite_2, position = position_2\} \text{ tq(}\\
&\qquad c \neq null\\
&\qquad \&\&\ r \geq 0\\
&\qquad \&\&\ 0 \leq j \leq c.length\\
&\qquad \&\&\ 0 \leq i \leq nbElemVisite \leq c.length\\
&\qquad \&\&\ -c.length \leq position \leq c.length\\
&\qquad \&\&\ (r = 0 \,||\, nbElemVisite \geq c.length))
\end{aligned}$$

- Dans le cas où $r = 0$, on sait qu'il n'y a pas de rotation à effectuer (on ne rentre pas dans $ITER$) donc le tableau renvoyé n'a pas été modifié et correspond bien à la clôture: "le tableau c a été modifié afin de représenter le cadran après rotation de r".
- Dans le cas où $nbElemVisite = c.length$ (c'est-à-dire $nbElemVisite \geq c.length$ && $0 \leq i \leq nbElemVisite \leq c.length$), cela voudra dire que tous les éléments du tableau auront effectué une rotation de r. En effet, le nombre d'éléments distincts déplacés ($nbElemVisite$) est égal au nombre d'éléments du tableau ($c.length$). Donc, le tableau renvoyé correspondra également à la clôture.

3.3.4 VARIANT

Prouver le variant : (E représente l'environnement)

- $INV \implies 0 \leq f(E)$
  Si l'invariant est vrai, alors $0 \leq nbElemVisite \leq c.length$ donc $0 \leq c.length - nbElemVisite$
- $f(E') < f(E)$

$$\begin{aligned}
E = &\{r = r_1, tmp = tmp_1, j = j_1, i = i_1, nbElemVisite = nbElemVisite_1, position = position_1\} \text{ tq(}\\
&\qquad c \neq null\\
&\qquad \&\&\ r_1 \geq 0\\
&\qquad \&\&\ 0 \leq j_1 \leq c_1.length\\
&\qquad \&\&\ 0 \leq i_1 \leq nbElemVisite_1 < c_1.length\\
&\qquad \&\&\ -c_1.length \leq position_1 \leq c_1.length)
\end{aligned}$$

3.3.5 ITER

$E' = \{..., nbElemVisite = nbElemVisite1 + 1, ...\}$ tq(

$c \neq null$

$\&\& \ r_1 \geq 0$

$\&\& \ 0 \leq j_1 \leq c_1.length$

$\&\& \ 0 \leq i_1 \leq nbElemVisite_1 < c_1.length$

$\&\& \ position_1 \leq c_1.length)$

($c.length$ n'est pas modifié durant $ITER$) $Var(E') = c_1.length - (nbElemVisite_1 + 1) \leq c_1.length - nbElemVisite_1 = Var(E)$ C'est vrai puisque $c_1 \neq null$(donc $c_1.length \geq 0$) et que $0 \leq nbElemVisite_1 < c_1.length$.

L'variant est donc prouvé puisque son résultat est toujours supérieur à 0 et qu'il décroît à chaque itération.

4 Code Java

```
public static void rotate(int[] c, int r) {
        // INIT
        int i, j, nbElemVisite, position, tmp;
        if (c.length == 0) {
                r = 0;
                tmp = 0; // valeur sentinelle
        } else {
                r = r % c.length;
                tmp = c[0];
        }
        i = 0; j = i; nbElemVisite = 0; position = j - r;
        while (r != 0 && nbElemVisite < c.length) { // !H
                // ITER
                if (position < 0) {
                        position = position + c.length;
                }

                if (i == position) {
                        c[j] = tmp;

                        i++;
                        tmp = c[i];
                        j = i;
                } else {
                        c[j] = c[position];
                        j = position;
                }
                nbElemVisite++;
                position = j - r;
        }
}
```