Corrigé du Contrôle d'Informatique INF 311

Sujet proposé par P. Chassignet, D. Monniaux, F. Nielsen, O. Serre

7 juillet 2008

Seuls les documents fournis dans le cadre du cours et les notes personnelles sont autorisés. Durée 2 heures.

Les 4 exercices qui suivent sont indépendants et peuvent être traités dans n'importe quel ordre. Nous donnons le nombre de lignes attendues pour la réponse à chaque question ; si la réponse que vous envisagez est beaucoup plus longue, il est probable qu'il s'agisse d'une méthode trop compliquée. Les questions les plus difficiles sont marquées par **.

On attachera une grande importance à la clarté, à la précision et à la concision de la rédaction.

Exercice 1. Fonction récursive mystérieuse

On considère le programme suivant qui se compile sans erreur :

```
public class MysteriousProgram {
  public static void display(int[] tab) {
    for (int i = 0; i < tab.length; i++)
      System.out.print(tab[i] + "␣");
    System.out.println();
  }
  public static void swap2(int a, int b) {
    int tmp = a;
    a = b;
    b = tmp;
  }
  public static void swap3(int[] tab, int i, int j) {
    int tmp = tab[i];
    tab[i] = tab[j];
    tab[j] = tmp;
  }

  public static void mysterious(int[] tab, int k) {
    for (int j = k; j < tab.length; j++) {
      swap3(tab, k, j);
      display(tab);
      swap3(tab, k, j);
    }
  }
```

```
  public static void mysteriousRecursive(int[] tab, int k) {
    if (k == tab.length - 1)
      display(tab);
    for (int j = k; j < tab.length; j++) {
      swap3(tab, k, j);
      mysteriousRecursive(tab, k + 1);
      swap3(tab, k, j);
    }
  }

  public static void init(int[] tab) {
    for (int i = 0; i < tab.length; i++)
      tab[i] = i + 1;
  }

  public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);
    int[] t = new int[n];
    init(t);
    swap2(t[0], t[n - 1]);
    mysterious(t, 0);
    //  mysteriousRecursive(t, 0);
  }

}
```

(1a) Après avoir compilé, quel est le résultat affiché dans la console par `java MysteriousProgram 4`? (réponse attendue : quelques lignes)
Décrivez de manière relativement détaillée le déroulement du programme et son effet, dans le cas général (pour `n > 0` quelconque). (réponse attendue : 10 à 20 lignes)

Solution: Pour `n = 4`, le programme affiche :

```
1 2 3 4
2 1 3 4
3 2 1 4
4 2 3 1
```

La fonction `init` remplit le tableau `t` qui est passé par référence avec les valeurs de `1` à `n`. La fonction `swap2` échange les copies des valeurs de deux cases du tableau et elle ne modifie donc pas le contenu de ce tableau qui reste dans l'ordre de `1` à `n`.

Dans la fonction `mysterious`, la valeur de `k` ne change pas et elle désigne une case du tableau. On utilise ici la fonction `swap3` qui permet d'échanger le contenu de deux cases du tableau. Au premier tour de la boucle, on échange une case avec elle-même. Ensuite, on échange successivement le contenu de cette case avec le contenu des cases suivantes, on affiche et on remet en place.

Ici `k = 0` et toutes les valeurs se retrouvent successivement en première position.
Remarque : Au retour de `mysterious`, le tableau est remis dans l'ordre initial.

(1b) ** Dans la fonction `main`, on remplace maintenant l'appel `mysterious(t, 0);` par `mysteriousRecursive(t, 0);`.
Après avoir recompilé, quel est le résultat affiché par `java MysteriousProgram 3` ? (réponse attendue : quelques lignes)
Décrivez de manière relativement détaillée le déroulement du programme et son effet, dans le cas général (pour `n > 0` quelconque). (réponse attendue : 10 à 20 lignes)

Solution: Pour `n = 3`, l'appel `mysteriousRecursive(t, 0);` affiche :

```
1 2 3
1 3 2
2 1 3
2 3 1
3 2 1
3 1 2
```

Le programme affiche toutes les permutations du tableau.
Remarque : La récursion termine car le corps de la boucle `for` n'est pas exécuté pour `k > tab.length-1`.
Hypothèse de récurrence : Pour tout `k`, l'appel `mysteriousRecursive(tab, k);` affiche le contenu du tableau pour toutes les permutations de `tab[k] ...tab[n-1]` et laisse finalement le tableau dans l'ordre qu'il avait à l'entrée dans la fonction.
C'est vrai pour `k=n-1`.
Si c'est vrai pour `k+1`, ..., `n-1`, alors c'est vrai pour `k` comme vu en (a).

Exercice 2. Modélisation de molécules

Dans cet exercice, on va modéliser une molécule comme un tableau d'atomes où chaque atome est défini comme une sphère avec un centre et un rayon (le rayon de Van der Waals). On cherchera ensuite à détecter des "collisions" entre atomes et molécules.

(2a) Écrivez une classe `Point3D` dont chaque objet définit un point de l'espace par ses coordonnées x, y et z de type `double`. Munissez cette classe d'un constructeur `Point3D(double x0, double y0, double z0)` qui permet d'initialiser un objet de ce type. (réponse attendue : 5 à 10 lignes)

Solution: voir ci-dessous

(2b) Écrivez une fonction statique `double distance(Point3D p, Point3D q)` qui prend comme arguments deux points p et q et retourne la distance euclidienne $\|q - p\|$ entre ces deux points. Cette fonction sera placée dans la classe `Point3D`. Pour calculer un carré, on utilisera la fonction `static double sqr(double x) { return x*x; }` qui

sera placée aussi dans `Point3D`.

Pour calculer la racine carrée, on utilisera la fonction `static double sqrt(double x)` de la classe `Math`. (réponse attendue : quelques lignes)

Solution: voir ci-dessous

(2c) Écrivez une fonction statique `Point3D add(Point3D p, Point3D q)` qui prend comme arguments deux points p et q et qui retourne un **nouveau** point égal à $p + q$. Cette fonction sera placée dans la classe `Point3D`. (réponse attendue : quelques lignes)

Solution: voir ci-dessous

(2d) Écrivez une fonction statique `void scale(Point3D p, double k)` qui multiplie les coordonnées du point p par le nombre k, autrement dit, p devient $k.p$. Cette fonction sera placée dans la classe `Point3D`. (réponse attendue : quelques lignes)

Solution:

```
public class Point3D {
  double x, y, z;

  public Point3D(double x0, double y0, double z0) {
    this.x = x0;
    this.y = y0;
    this.z = z0;
  }

  static double sqr(double x) { return x*x; }

  public static double distance(Point3D p, Point3D q) {
    return Math.sqrt(sqr(q.x - p.x) + sqr(q.y - p.y)
                     + sqr(q.z - p.z));
  }

  public static Point3D add(Point3D p, Point3D q) {
    return new Point3D(p.x + q.x, p.y + q.y, p.z + q.z);
  }

  public static void scale(Point3D p, double k) {
    p.x *= k;
    p.y *= k;
    p.z *= k;
  }
}
```

(2e) Écrivez une classe `Atom` qui permet de définir des atomes avec **deux** champs d'objet :

• `center` de type `Point3D` qui représente la position (x, y, z) du centre de cet atome, et
• `radius` de type `double` qui est le rayon de cet atome.
Munissez cette classe d'un constructeur
`Atom(double x, double y, double z, double rad)` qui initialise un objet de ce type.
Ajoutez dans cette classe la définition de deux constantes `H_RADIUS = 1.2` et `O_RADIUS = 1.5` qui seront les rayons (en ångström) respectivement pour des atomes d'hydrogène et d'oxygène. (réponse attendue : une dizaine de lignes)

Solution: voir ci-dessous

(2f) Écrivez une fonction statique `boolean bump(Atom a, Atom b)` qui prend deux atomes a et b et qui retourne `true` si et seulement si la distance entre leurs deux centres est strictement inférieure à la somme de leurs rayons (c'est ce qu'on appellera une "collision" entre les deux atomes). Cette fonction sera dans la classe `Atom`. (réponse attendue : quelques lignes)

Solution:

```
public class Atom {
  Point3D center;
  double radius;

  public static final double H_RADIUS = 1.2;
  public static final double O_RADIUS = 1.5;

  public Atom(double x, double y, double z, double rad) {
    this.center = new Point3D(x, y, z);
    this.radius = rad;
  }

  public static boolean bump(Atom a, Atom b) {
    return Point3D.distance(a.center, b.center)
         < a.radius + b.radius;
  }

} // fin provisoire de cette classe
```

(2g) Écrivez une classe `Molecule` qui permet de définir une molécule comme un objet ayant un tableau d'atomes et un constructeur qui prend en argument la référence sur ce tableau. (réponse attendue : quelques lignes)

Solution:

```
public class Molecule {
```

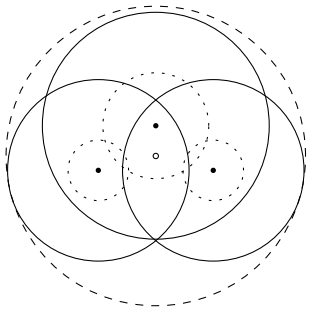
```
  Atom[] atoms;
  public Molecule(Atom[] t) {
    this.atoms = t;
  }
}
```

(2h) Écrivez dans une classe `Test` un programme qui construit une molécule d'eau H2O avec son atome d'oxygène en $(0, 0.4, 0)$ et ses atomes d'hydrogène en $(0.76, -0.19, 0)$ et $(-0.76, -0.19, 0)$, l'unité est toujours en ångström. (réponse attendue : 5 à 10 lignes)

Solution:

```
public class Test {
  public static void main(String[] args) {
    Atom o = new Atom(0, 0.4, 0, Atom.O_RADIUS);
    Atom h1 = new Atom(0.76, -0.19, 0, Atom.H_RADIUS);
    Atom h2 = new Atom(-0.76, -0.19, 0, Atom.H_RADIUS);
    Atom[] H2O = { o, h1, h2 };
    Molecule mol = new Molecule(H2O);

  }
}
```

Pour chaque molécule, on va maintenant construire une sphère englobante qui permettra d'accélérer les tests de collision. Pour simplifier, le centre de cette sphère sera l'isobarycentre (on ne tient pas compte des masses respectives) des centres de tous les atomes de la molécule, voir la figure ci-après. On utilisera un objet `Atom` pour représenter la sphère englobante.



Les • sont les centres des atomes. Le ∘ indique leur isobarycentre.
Les cercles en trait plein indiquent les sphères (de Van der Walls) de chaque atome.

La sphère englobante d'une molécule d'eau.

(2i) Écrivez une fonction statique `middle` qui prend en argument un tableau d'atomes (on le supposera non vide) et qui retourne l'isobarycentre de ces atomes. Cette fonction sera dans la classe `Atom` et elle utilisera les fonctions `add` et `scale` de la classe `Point3D`.

Cette fonction ne doit pas modifier les coordonnées des atomes du tableau. (réponse attendue : 5 à 10 lignes)

Solution:

```
public static Point3D middle(Atom[] t) {
  Point3D middle = t[0].center;
  for (int i = 1; i < t.length; ++i)
    middle = Point3D.add(middle, t[i].center);
  Point3D.scale(middle, 1.0/t.length);
  return middle;
}
```

(2j) Écrivez une fonction statique `double maxDistance(Point3D p, Atom a)` qui retourne la distance maximale entre le point p et tout point sur la sphère de l'atome a. Cette fonction sera dans la classe `Atom`. (réponse attendue : quelques lignes)

Solution:

```
public static double maxDistance(Point3D p, Atom a) {
  return Point3D.distance(p, a.center) + a.radius;
}
```

(2k) ** Écrivez maintenant les modifications à apporter à la classe `Molecule` pour construire la sphère englobante et l'utiliser dans les calculs de collision. Si on n'intersecte pas la sphère englobante d'une molécule, il est inutile de tester les collisions avec ses atomes. Écrivez une fonction statique `boolean bump(Atom a, Molecule b)` qui sera dans la classe `Molecule` et qui permet de tester si l'atome a entre en collision avec au moins un des atomes de la molécule b.
Écrivez aussi la fonction statique `boolean bump(Molecule a, Molecule b)` qui fait la même chose entre deux molécules. (réponse attendue : au total 20 à 30 lignes)

Solution:

```
public class Molecule {
  Atom[] atoms;
  Atom sphere;

  public Molecule(Atom[] t) {
    this.atoms = t;
    Point3D center = Atom.middle(atoms);
    double r = 0;
    for (int i = 0; i < atoms.length; ++i) {
      double ri = Atom.maxDistance(center, atoms[i]);
      if (r < ri) r = ri;
```

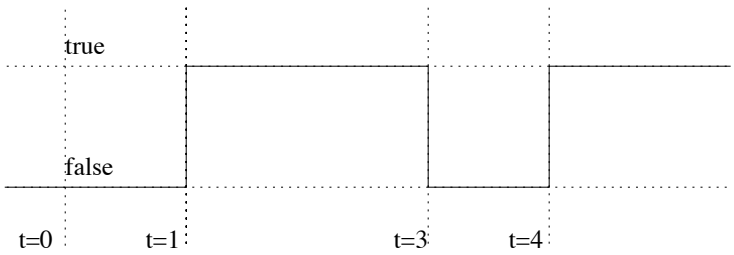
```
    }
    this.sphere = new Atom(center.x,center.y,center.z, r);
  }

  public static boolean bump(Atom a, Molecule b) {
    if (!Atom.bump(a, b.sphere))
      return false;
    for (int i = 0; i < b.atoms.length; ++i)
      if (Atom.bump(a, b.atoms[i]))
        return true;
    return false;
  }

  public static boolean bump(Molecule a, Molecule b) {
    if (!Atom.bump(a.sphere, b.sphere))
      return false;
    for (int i = 0; i < a.atoms.length; ++i)
      if (bump(a.atoms[i], b))
        return true;
    return false;
  }
}
```

Exercice 3. Codage de signaux électroniques

On veut représenter des signaux électroniques binaires, à valeurs dans $\{\mathsf{false}, \mathsf{true}\}$, qui ne changent de valeur qu'un nombre fini de fois au cours du temps. On considère que les changements de valeur se produisent à des instants qui correspondent à des "tops" d'une horloge, et on peut ainsi identifier un tel instant par un entier (positif) qui est le nombre de "tops" depuis l'origine des temps.

Un tel signal est défini par sa valeur initiale v_0, et une suite strictement croissante d'instants de transition $\tau_1, \ldots, \tau_n \in \mathbb{N}$. Dans $]-\infty, \tau_1[$ le signal prend la valeur v_0, puis dans $[\tau_1, \tau_2[$ il prend la valeur $v_1 = \neg v_0$ (où $\neg x$ est la négation logique de x, notée `!x` en Java), dans $[\tau_2, \tau_3[$ il prend la valeur $v_2 = \neg v_1$, etc., et dans $[\tau_n, +\infty[$ il prend la valeur $v_n = \neg v_{n-1}$. Un signal constant sera représenté par sa valeur v_0 et une suite vide ($n = 0$).

Par exemple, le signal illustré ci-après correspond à la valeur initiale faux et la suite d'instants de transitions $\tau_1 = 1, \tau_2 = 3, \tau_3 = 4$.



On utilisera la structure de données suivante :

```
class Transition {
  int time;
  Transition next;

  Transition(int time, Transition next) {
    this.time = time;
    this.suivante = next;
  }
}

class Signal {
  boolean initialValue;
  Transition transitions;

  Signal(boolean value, Transition transitions) {
    this.initialValue = value;
    this.transitions = transitions;
  }
}
```

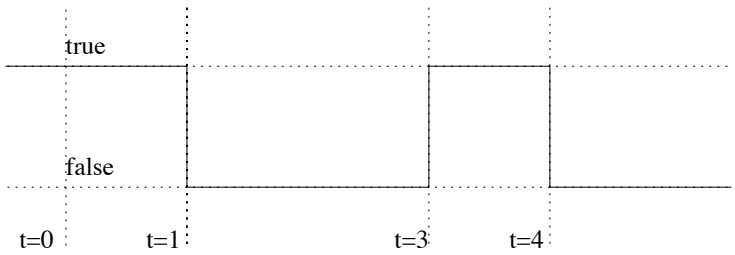
Le champ `initialValue` représente la valeur d'un objet de type `Signal` au voisinage de $-\infty$. Le champ `transitions` référence le début de la liste chaînée des instants des transitions pour ce signal. Si ce champ vaut `null`, alors le signal est constant. Sinon, les champs `time` successifs des éléments de la liste contiennent les valeurs $\tau_1, \ldots, \tau_n$ dans cet ordre (la tête de liste contient τ_1). On rappelle que la suite des τ_i est strictement croissante.

Par exemple, on peut créer le signal illustré ci-dessus par :

```
Signal signal1 = new Signal(false,
  new Transition(1, new Transition(3, new Transition(4, null))));
```

Pour les questions qui suivent, les fonctions demandées seront toutes placées dans une même classe, dont le nom importe peu.

(3a) Étant donné un signal s, on veut l'inverser, c'est-à-dire obtenir sa négation logique. Ainsi, l'inversion de `signal1` donne :



Écrivez pour cela une fonction `static Signal invert(Signal s)`.
Vous ne devez pas modifier le signal d'origine, autrement dit, il faut créer un **nouveau** `Signal`. Le temps de calcul de cette fonction ne devra pas dépendre de la longueur de la liste de transitions de `s`. (réponse attendue : quelques lignes)

Solution: voir ci-dessous

(3b) Étant donné un signal s et un instant t, on veut calculer $s(t)$, c'est-à-dire, la valeur (true ou false) prise par le signal s à l'instant t. Écrivez pour cela une fonction **itérative** `static boolean valueAt(Signal s, int time)`.
Vous ne devez pas modifier le signal `s` et le temps d'exécution sera au plus proportionnel au nombre de transitions dans `s`. (réponse attendue : une dizaine de lignes)

Solution: voir ci-dessous

(3c) Nous voulons une fonction qui affiche le signal sous la forme d'une suite d'intervalles avec ses valeurs successives. Par exemple :

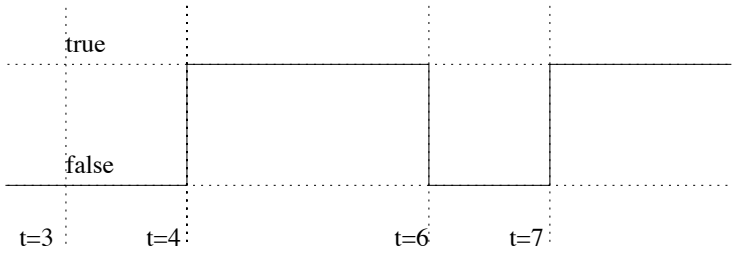
```
-inf -> +inf : false
```

veut dire que le signal est uniformément égal à false, tandis que pour l'exemple `signal1` on obtient :

```
-inf -> 1.0 : false
1.0 -> 3.0 : true
3.0 -> 4.0 : false
4.0 -> +inf : true
```

Écrivez la fonction **itérative** `static void print(Signal s)`. (réponse attendue : de 10 à 15 lignes)

(3d) Étant donné un signal s, on veut calculer un autre signal identique à s mais décalé dans le temps d'une durée $\delta > 0$. Autrement dit, on veut un signal de même valeur initiale mais dont la suite des instants (τ'_i) est définie par $\tau'_i = \tau_i + \delta$, $\forall i \in \{1, \ldots, n\}$.
Ainsi, le décalage de $\delta = 3$ de `signal1` donne :



Écrivez une fonction **récursive**
`static Transition shift(Transition t, int delta)`
qui retournera une **nouvelle** liste de transitions, identiques à `t` mais dont les valeurs sont décalées de `delta`. Attention, vous ne devez pas modifier la liste `t`.
Écrivez ensuite une fonction `static Signal shift(Signal s, int delta)`. (réponse attendue : au total une dizaine de lignes)

Solution: voir ci-dessous

(3e) Un `Signal` dont la suite des instants de transitions n'est pas strictement croissante est incorrect. Afin de pouvoir détecter des erreurs dans le programme, il est utile de disposer d'une fonction `static boolean isWellFormed(Signal s)` qui retourne `true` si et seulement si `s` est correct.
Écrivez une telle fonction de manière **récursive** à l'aide d'une fonction auxiliaire. Le temps d'exécution doit être proportionnel à la longueur de la liste de transitions associée à `s`. (réponse attendue : au total une petite dizaine de lignes)

Solution: voir ci-dessous

(3f) ** L'opération commutative "ou exclusif", dite XOR (eXclusive OR), entre deux valeurs binaires est définie par la table :

XOR	false	true
false	false	true
true	true	false

Autrement dit, b_1 XOR b_2 vaut true si et seulement si $b_1 = \neg b_2$. En Java, on peut l'écrire `b1 != b2`.

On veut maintenant calculer le signal qui résulte de la composition par XOR de deux signaux. On peut remarquer que, à un instant donné, si un seul signal change de valeur, alors le résultat change et que si les deux signaux changent en même temps, alors le résultat ne change pas. Écrivez une fonction
`static Transition xorTransitions(Transition t1, Transition t2)`
qui retournera une **nouvelle** liste de transitions qui correspond au résultat attendu. Attention, vous ne devez pas modifier les listes `t1` et `t2` mais, dans certains cas, il est possible de partager une sous-liste. Le temps d'exécution doit être au plus proportionnel à la somme des longueurs des deux listes.

Écrivez ensuite une fonction `static Signal xorSignals(Signal s1, Signal s2)`. (réponse attendue : au total une vingtaine de lignes)

Solution:

```
public class WorkingSignals {

  public static Signal invert(Signal s) {
    return new Signal(!s.initialValue, s.transitions);
```

```
  }

  public static boolean valueAt(Signal s, int time) {
    boolean v = s.initialValue;
    Transition t = s.transitions;
    while (t != null) {
      if (time < t.time)
        return v;
      v = !v;
      t = t.next;
    }
    return v;
  }

  public static void print(Signal s) {
    boolean v = s.initialValue;
    Transition t = s.transitions;
    System.out.print("-inf␣->␣");
    while (t != null) {
      System.out.println(t.time + "␣:␣" + v);
      System.out.print(t.time + "␣->␣");
      v = !v;
      t = t.next;
    }
    System.out.println("+inf␣:␣" + v);
  }

  public static Transition shift(Transition t,
      int delta) {
    if (t == null)
      return null;
    else
      return new Transition(t.time + delta,
          shift(t.next, delta));
  }

  public static Signal shift(Signal s, int delta) {
    return new Signal(s.initialValue,
        shift(s.transitions, delta));
  }

  public static boolean isWellFormed(Transition t) {
    if (t == null || t.next == null)
      return true;
```

```
    else
      return (t.time < t.next.time)
        && isWellFormed(t.next);
  }

  static boolean isWellFormed(Signal s) {
    return isWellFormed(s.transitions);
  }

  public static Transition xorTransitions(Transition t1,
                                          Transition t2) {
    if (t1 == null)
      return t2;
    else if (t2 == null)
      return t1;
    else {
      int tt1 = t1.time;
      int tt2 = t2.time;
      if (tt1 < tt2)
        return new Transition(tt1,
            xorTransitions(t1.next, t2));
      else if (tt2 < tt1)
        return new Transition(tt2,
            xorTransitions(t1, t2.next));
      else
        return xorTransitions(t1.next, t2.next);
    }
  }

  public static Signal xorSignals(Signal s1, Signal s2) {
    return new Signal(
        (s1.initialValue) != (s2.initialValue),
        xorTransitions(s1.transitions, s2.transitions));
  }

}
```

Exercice 4. Le jeu de Nim

Le jeu de Nim se joue à deux joueurs. On considère m vases (indicés de 0 à $m-1$), le vase d'indice i contenant x_i fruits. Une configuration du jeu sera donc donnée par un tableau d'entiers.

Les joueurs jouent à tour de rôle. Un coup consiste à retirer autant de fruits que l'on désire (mais au moins un) dans un **même** vase. Le joueur qui gagne est celui qui enlève le dernier

fruit du jeu (tous les vases sont alors vides).
Ce jeu est en fait totalement déterminé : l'un des deux joueurs (qui dépend de la configuration initiale des vases et du joueur qui commence) possède une stratégie pour gagner quoi que fasse son adversaire. C'est cette stratégie que l'on va calculer ici.

On va commencer par écrire deux fonctions de conversion : la première convertira un nombre écrit en base 2 vers une représentation en base 10, la seconde effectuera la conversion réciproque.

(4a) La représentation d'un entier n en base 2 sur k bits sera donnée par un tableau d'entiers `binaryRepresentation` de longueur k. On supposera que toutes les valeurs contenues dans ce tableau sont égales à 0 ou à 1 et `binaryRepresentation[i]` est le `i`-ème bit de la décomposition de n en base 2 :

$$\texttt{n} = \sum_{\texttt{i=0}}^{\texttt{i<k}} \texttt{binaryRepresentation[i]} * 2^{\texttt{i}}$$

Écrivez une fonction `static int binaryToDecimal(int[] binaryRepresentation)` qui retourne l'entier dont `binaryRepresentation` est la représentation binaire. (réponse attendue : une petite dizaine de lignes)

Solution: voir ci-dessous

(4b) On supposera ici que `k` est assez grand et on se donne la fonction :

```
public static int[] decimalToBinary(int n, int k) {
  int[] binaryRepresentation = new int[k];
  decimalToBinaryAux(n, 0, binaryRepresentation);
  return binaryRepresentation;
}
```

Écrivez de manière **récursive** la fonction `decimalToBinaryAux` qui est utilisée ici. (réponse attendue : quelques lignes)

Solution: voir ci-dessous

Dans la suite, étant donnés un nombre y, et un entier k, suffisamment grand pour que l'on puisse écrire y en base 2 sur k bits, on notera $y[i]$ le i-ème bit de la décomposition en base 2 de y, c'est-à-dire que $y = \sum_{i=0}^{k} y[i] * 2^i$.

On considère la fonction suivante, appelée fonction de Grundy, qui prend en argument les nombres de fruits dans chaque vase et qui retourne un entier $Grundy(x_0, x_2, \ldots, x_{m-1}) = a$ dont la représentation en base 2 est définie en posant $a[i] = \left(\sum_{l=0}^{m-1} x_l[i] \right) \mod 2$.

Par exemple, si l'on souhaite calculer $Grundy(6, 9, 1, 2)$, on écrit tout d'abord 6, 9, 1 et 2 en base 2 (sur le même nombre de bit k choisi suffisamment grand) et on fait la somme par colonne modulo 2, ce qui donne la représentation en base 2 de a. On n'a alors plus qu'a convertir a en base 10 :

6	=	0	1	1	0
9	=	1	0	0	1
1	=	0	0	0	1
2	=	0	0	1	0
a	=	1	1	0	0

Ainsi, on a $Grundy(6, 9, 1, 2) = a = 12$.

(4c) ** Écrivez la fonction `static int Grundy(int[] decimalTab)` qui retourne la valeur de la fonction de Grundy pour la configuration du jeu décrite par le tableau `decimalTab`.
Il est conseillé d'écrire un certain nombre de fonctions auxiliaires pour :
- calculer le nombre de bits k qui est nécessaire,
- construire un tableau `int[][]` qui contient les décompositions en binaire des valeurs de `decimalTab`,
- faire le calcul de Grundy en binaire.

Ces fonctions pourront être réutilisées par la suite. (réponse attendue : quelques dizaines de lignes)

Solution: voir ci-dessous

On admet le résultat suivant :

> Soit la configuration de jeu où les vases contiennent respectivement $x_0, x_1, \ldots, x_{m-1}$ fruits. Alors le joueur dont c'est le tour a une stratégie gagnante si et seulement si $Grundy(x_0, \ldots, x_{m-1}) \neq 0$.

Dans le cas où $Grundy(x_0, \ldots, x_{m-1}) \neq 0$, le coup à jouer pour gagner est le suivant :
- On considère $Grundy(x_0, \ldots, x_{m-1})$ en base 2 et on détermine l'indice maximal j d'un bit valant 1 dans cette décomposition (un tel indice existe car $Grundy(x_1, \ldots, x_{m-1}) \neq 0$).
- On trouve un indice i correspondant à un vase contenant x_i fruits où le bit d'indice j de la décomposition en base 2 de x_i vaut 1 (un tel i existe par définition de la fonction de Grundy). C'est du vase i que l'on va retirer des fruits.
- On retire du vase i autant de fruits qu'il faut pour qu'il contienne x'_i fruits où x'_i est défini à l'aide de sa décomposition en base 2 comme suit, pour tout rang h :

$$x'_i[h] = \begin{cases} 1 - x_i[h] & \text{si } grundy[h] = 1 \\ x_i[h] & \text{sinon.} \end{cases}$$

Dans l'exemple de la question précédente, l'indice j vaut 3 et le vase à modifier est le vase d'indice 1 (contenant 9 fruits). On retire 4 fruits, de sorte à ce qu'il ne reste plus que 5 fruits.

(4d) ** Écrivez une fonction `pick` qui prend en argument un tableau d'entiers représentant une configuration du jeu, et retourne un tableau d'entiers de taille deux : le premier entier donnera l'indice du vase à modifier, et le second donnera le nombre de fruits à retirer. Dans le cas où la fonction de Grundy est nulle (on sait que l'on va perdre) on retirera un fruit du premier vase non vide. (réponse attendue : quelques dizaines de lignes)

Solution:

```
public class Nim {

  public static int binaryToDecimal(
                           int[] binaryRepresentation) {
    int n = 0;
    int p2 = 1;
    for (int i=0; i < binaryRepresentation.length; i++) {
      n = n + binaryRepresentation[i] * p2;
      p2 = p2 * 2;
    }
    return n;
  }

  public static void decimalToBinaryAux(int n, int i,
      int[] binaryRepresentation) {
    if (n > 0) {
      binaryRepresentation[i] = n % 2;
      decimalToBinaryAux(n/2, i+1, binaryRepresentation);
```

```
  }
}

public static int[] decimalToBinary(int n, int k) {
  int[] binaryRepresentation = new int[k];
  decimalToBinaryAux(n, 0, binaryRepresentation);
  return binaryRepresentation;
}

public static int getBinaryLength(int[] decimalTab) {
  // search the largest value
  int max = decimalTab[0];
  for (int i = 1; i < decimalTab.length; i++) {
    if (decimalTab[i] > max) {
      max = decimalTab[i];
    }
  }
  int k = 0;
  int p = 1;
  while (p <= max) {
    ++k;
    p *= 2;
  }
  return k;
}

public static int[][] decomposition(int[] decimalTab) {
  int k = getBinaryLength(decimalTab);
  int[][] binaryTab = new int[decimalTab.length][];
  for (int i = 0; i < decimalTab.length; i++) {
    binaryTab[i] = decimalToBinary(decimalTab[i], k);
  }
  return binaryTab;
}

public static int[] binaryGrundy(int[][] binaryTab) {
  int k = binaryTab[0].length;
  int[] grundyBinaire = new int[k];
  for (int j = 0; j < k; j++) {
    for (int i = 0; i < binaryTab.length; i++) {
      grundyBinaire[j] =
        (grundyBinaire[j] + binaryTab[i][j]) % 2;
    }
  }
```

```
    return grundyBinaire;
  }

  public static int Grundy(int[] decimalTab) {
    int[][] binaryTab = decomposition(decimalTab);
    return binaryToDecimal(binaryGrundy(binaryTab));
  }

  public static int[] loserPick(int[] decimalTab) {
    int i = 0;
    while (decimalTab[i] == 0) {
      i++;
    }
    int[] play = { i, 1 };
    return play;
  }

  public static int[] pick(int[] decimalTab) {
    int[][] binaryTab = decomposition(decimalTab);
    int k = binaryTab[0].length;
    int[] grundyBin = binaryGrundy(binaryTab);

    if (binaryToDecimal(grundyBin) == 0) {
      return loserPick(decimalTab);
    }
    int j = k - 1;
    while (grundyBin[j] == 0) {
      j--;
    }
    int i = 0;
    while (binaryTab[i][j] == 0) {
      i++;
    }
    for (int h = 0; h < grundyBin.length; h++) {
      if (grundyBin[h] == 1) {
        binaryTab[i][h] = 1 - binaryTab[i][h];
      }
    }
    int[] play = new int[2];
    play[0] = i;
    play[1] = decimalTab[i] -
         binaryToDecimal(binaryTab[i]);
    return play;
  }
```

```
}
```