Énoncé

On définit une suite $(u_n)_{n\in\mathbb{N}}$ par les relations suivantes

$$u_0 = 0, \qquad \forall n \in \mathbb{N} : \begin{cases} u_{2n} = u_n \\ u_{2n+1} = 1 - u_n \end{cases}$$

On utilisera dans cet exercice `iquo(u,2)` et `u mod(2)` qui renvoient respectivement le quotient et le reste de la division euclidienne d'un entier `u` par 2.

1. a. Quel est l'ensemble des valeurs de la suite $(u_n)_{n\in\mathbb{N}}$?
   b. Former un algorithme (diagramme ou langage usuel) permettant de remplir un tableau indexé de 0 à un entier n avec $u_0, u_1, \cdots, u_n$.
   c. Définir en langage Maple une procédure `suite` prenant comme paramètre une variable entière `n` et renvoyant un tableau indexé de 0 à `n` et contenant les valeurs $u_0, \cdots u_n$.
2. a. Former un algorithme (diagramme ou langage usuel) permettant de calculer le nombre de 1 dans le développement binaire d'un entier x. Ce nombre est noté $s(x)$.
   b. Définir en langage Maple une procédure `nbd1` prenant comme paramètre une variable entière `x` et renvoyant le nombre de 1 dans le développement binaire de x.
3. Comment peut-on vérifier que $s(n) \equiv u_n \mod 2$ (une syntaxe précise est demandée). Démontrer que ce résultat est valable pour tous les entiers.

Corrigé

1. a. La suite ne prend que les valeurs 0 et 1.
   b. Algorithme de calcul des valeurs de la suite en langage usuel.
   La variable `n` désigne un entier naturel donné. La variable `i` désigne un entier naturel
   - Assigner à une variable `U` un tableau vide indexé de 0 à `n`.
   - Assigner à `A[0]` la valeur 0.
   - Pour `i` de 1 à `n`
     - Si `i` est pair : `A[i]<-- A[i/2]`
     - Si `i` est impair : `A[i]<-- 1-A[(i-1)/2]`

   c. Implémentation Maple de l'algorithme présenté au dessus dans une procédure

```
suite := proc(n)
  local i,U;
  U:= array(0..n);
  U[0] := 0;
  for i from 1 to n do
    if i mod 2 = 0 then U[i] := U[i/2] fi;
    if i mod 2 = 1 then U[i] := 1-U[(i-1)/2] fi;
  od;
  return(eval (U));
end proc:
```

2. a. On modifie l'algorithme de cours pour développer un entier dans une base donnée. Les restes sont additionnés dans une variable au lieu d'être stockés dans une liste ou un tableau.
   `x` est une variable désignant l'entier donné.
   `s` est une variable entière qui sert à compter les 1 .
   `r` est une variable entière.
   - `s <- 0`
   - tant que `x > 0` faire :
     - `s <- s +` reste de la division de `x` par 2
     - `x <-` quotient de la division de `x` par 2

   b. Implémentation Maple de la procédure :

```
nbd1 := proc(x)
  local xl,s;
  xl :=x;
  s:= 0;
```

```
    while xl >0 do
      s:= s + (xl \mod 2);
      xl := iquo(xl,2);
    od;
    return s;
  end proc:
```

3. On peut vérifier que u_n est congru à $s(x)$ en appelant les procédures pour un certain nombre de x. Remarquer le crochet après la parenthèse pour la procédure qui renvoie un tableau.

```
for n from 1 to 50 do
  (nbd1(n) \mod 2) - suite(n)[n];
od;
```

L'exécution de ce code ne renvoie que des 0.
Pour démontrer que la congruence est valable pour tous les n, considérons la suite $(e_n)_{n\in\mathbb{N}}$ formée par les restes modulo 2 des $s(n)$. On va montrer que $e_n = u_n$ en vérifiant que $(e_n)_{n\in\mathbb{N}}$ satisfait aux *mêmes* relations de récurrence que $(u_n)_{n\in\mathbb{N}}$.
En effet, $e_0 = 0$ car le développement de 0 ne contient que 0.
Considérons un entier n et son développement binaire $\overline{b_k b_{k-1} \cdots b_1 b_0}$. On peut écrire les développements suivants

$$\text{pour } 2n : \overline{b_k b_{k-1} \cdots b_1 b_0 0}$$
$$\text{pour } 2n+1 : \overline{b_k b_{k-1} \cdots b_1 b_0 1}$$

Dans le cas de $2n$, le nombre de 1 ne change pas et $e_{2n} = e_n$. Dans le cas de $2n+1$, le nombre de 1 est incrémenté de 1 donc change de parité et $e_{2n+1} = 1 - e_n$. Les suites sont donc égales.