X 2011 : Info pour non-informaticiens

```
> restart;
> m := 4; alloue :=proc(m) Array(1 ..m, 5) end;
```

$$m := 4$$

$$alloue := \mathbf{proc}(m)\ Array(1\ ..m, 5)\ \mathbf{end\ proc} \quad (1)$$

```
> P8 := alloue(8);
```

$$P8 := \left[\begin{array}{cccccccc} 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{array}\right] \quad (2)$$

```
> P8;
```

$$\left[\begin{array}{cccccccc} 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \end{array}\right] \quad (3)$$

```
> P15 := alloue(11);
```

$$P15 := \left[\begin{array}{c} 1\ ..\ 11\ Array \\ Data\ Type:\ anything \\ Storage:\ rectangular \\ Order:\ Fortran\_order \end{array}\right] \quad (4)$$

```
> taille := ArrayNumElems;
```

$$taille := ArrayNumElems \quad (5)$$

```
> taille(P8);
```

$$8 \quad (6)$$

```
> ttest1 := [4, 2, 1, 7, 3, 5, 9, 8, 6]; ttest2 := [1, 3, 2, 6, 5, 7, 4, 9, 10, 11, 8];
```

$$ttest1 := [4, 2, 1, 7, 3, 5, 9, 8, 6]$$

$$ttest2 := [1, 3, 2, 6, 5, 7, 4, 9, 10, 11, 8] \quad (7)$$

```
> ExempleT := [12, 5, 4, 8, 6, 13, 15, 2, 7, 14, 1, 10, 11, 3, 9]; ExempleS := [1, 2, 4, 5, 6, 7, 3, 8,
      10, 11, 12, 13, 9, 14, 15];
```

$$ExempleT := [12, 5, 4, 8, 6, 13, 15, 2, 7, 14, 1, 10, 11, 3, 9]$$

$$ExempleS := [1, 2, 4, 5, 6, 7, 3, 8, 10, 11, 12, 13, 9, 14, 15] \quad (8)$$

```
> ExTArray := convert(ExempleT, Array) : ExSArray := convert(ExempleS, Array) :
> AffGA :=proc(a) convert(a, list) end;
```

$$AffGA := \mathbf{proc}(a)\ convert(a, list)\ \mathbf{end\ proc} \quad (9)$$

Question 1

Idée : on crée un tableau "Vide", de même longueur que t et plein de "0"
on parcourt alors t et quand on lit t[k] on va mettre "1" dans Vide[t[k]]
si Vide[t[k]] ne contenait pas 0 ou si t[k] dépasse de Vide : c'est

perdu
Si on a pu effectuer cela jusqu'au bout de t : t était injectif, comme il avait même longueur que Vide c'est qu'il était aussi surjectif (aucun des t[k] n'a dépassé longueur(t) et les termes étaient supposés entiers)

Difficultés :
- au lieu de 0 et 1 j'avais voulu mettre true et false : Maple a refusé
- il faut éviter de parcourir tout t si on sait déja que c'est perdu
- comme toujours on se trompe entre Long et Long+1, entre < et <= ...

```
> estPermutation := proc(t) local Vide, Long, k;
                    Long := nops(t); Vide := Array(1..Long, 0); k := 1;
                    while(k ≤ Long) and (t[k] ≤ Long) and (Vide[t[k]] = 0)
                    do Vide[t[k]] := 1; k := k + 1 od;
                    evalb(k = Long + 1) end:
> estPermutation([4, 3, 1, 2]); estPermutation([4, 3, 1, 3]); estPermutation([7, 3, 1, 4]);
                                        true
                                        false
                                        false                                    (1.1)
```

Question 2

Idée : on applique la formule ... si on compose des listes : rien à dire

Difficultés : Comp étant locale ne veut pas être renvoyée. On peut l'afficher avec eval mais on obtient une table, j'ai essayé la conversion en list et ça a marché
Avec la (stupide) gymnastique de composer2 on récupère un Array selon les ordres du concours (selon le concours que vous préparez, lisez ou non)

```
> composer :=proc(t, u) local Long, k, Comp;
          Long := nops(t);   k := 1;
          for k from 1 to Long do Comp[k] := t[u[k]] od ;
           convert(Comp, list) end:
> composer([4, 2, 1, 3], [4, 3, 2, 1]); composer([1, 3, 2, 4], composer([4, 2, 1, 3], [4, 3, 2,
      1]) );
                                      [3, 1, 2, 4]
                                      [2, 1, 3, 4]                               (2.1)
> composer2 :=proc(t, u) local Long, k, Comp;
          Long := nops(t);   k := 1;
          for k from 1 to Long do Comp[k] := t[u[k]] od ;
           convert(convert(Comp, list), Array) end:
> composer2([4, 2, 1, 3], [4, 3, 2, 1]); composer2([1, 3, 2, 4], composer2([4, 2, 1, 3], [4, 3,
      2, 1]) );
                                    [ 3   1   2   4 ]
```

$$\left[\begin{array}{cccc} 2 & 1 & 3 & 4 \end{array}\right] \quad \textbf{(2.2)}$$

```
> composeArray := proc(t, u)
                  local Long, k, Comp;
          Long := taille(t);   Comp := alloue(Long);
          for k from 1 to Long do Comp[k] := t[u[k]] od;
          Comp end:
> AffGA(composeArray(ExTArray, ExSArray));
```

$$[12, 5, 8, 6, 13, 15, 4, 2, 14, 1, 10, 11, 7, 3, 9] \quad \textbf{(2.3)}$$

Question 3

```
> inverser := proc(t) local Long, k, Inv;
          Long := nops(t);   k := 1;
          for k from 1 to Long do Inv[t[k]] := k od;
       convert(convert(Inv, list), Array)
    end:

> inverser([4, 2, 1, 7, 3, 5, 9, 8, 6]); AffGA(inverser(ExempleT));
       AffGA(inverser(ExempleS));
```

$$\left[\begin{array}{cccccccccc} 3 & 2 & 5 & 1 & 6 & 9 & 4 & 8 & 7 \end{array}\right]$$

$$[11, 8, 14, 3, 2, 5, 9, 4, 15, 12, 13, 1, 6, 10, 7]$$

$$[1, 2, 7, 3, 4, 5, 6, 8, 13, 9, 10, 11, 12, 14, 15] \quad \textbf{(3.1)}$$

Question 4

L'identité est d'ordre 1, la permutation circulaire 1->2->3 ... ->n ->1 est d'ordre n

Question 5

estId sert à dire si une permutation est l'Identité, ayant cela la fonction ordre va composer patiemment t avec t^k ... jusquà avoir Id

```
> estId := proc(t) local Long, k;
    Long := nops(t); k := 1;
              while(k ≤ Long) and (t[k] = k)
              do k := k + 1 od;
              evalb(k = Long + 1) end:
> estId([1, 2, 3, 4, 5, 6]); estId([7, 1, 2, 3, 4, 5, 6]);
```

true

false **(5.1)**

```
> boulot := proc(tk, t, k)
       if estId(tk) then k else boulot(composer(t, tk), t, k + 1) fi end;
boulot := proc(tk, t, k)                                                    (5.2)
    if estId(tk) then k else boulot(composer(t, tk), t, k + 1) end if
```

```
end proc
```

```
> ordre := proc(t) boulot(t, t, 1) end;
```

$$ordre := \mathbf{proc}(t)\ boulot(t, t, 1)\ \mathbf{end\ proc} \tag{5.3}$$

```
> ordre([3, 1, 2, 4, 6, 7, 8, 5]);
```

$$12 \tag{5.4}$$

Question 6

```
> periode := proc(t, i) local tki, boulot;
      boulot := proc(k) if tki = i then k else tki := t[tki]; boulot(k + 1) fi end;
      tki := t[i]; boulot(1)
   end:
> convert([3, 5, 2, 1, 4, 6, 8, 7], Array);
```

$$\begin{bmatrix} 3 & 5 & 2 & 1 & 4 & 6 & 8 & 7 \end{bmatrix} \tag{6.1}$$

```
> ttest := convert([3, 5, 2, 1, 4, 6, 8, 7], Array);
```

$$ttest := \begin{bmatrix} 3 & 5 & 2 & 1 & 4 & 6 & 8 & 7 \end{bmatrix} \tag{6.2}$$

```
> map(k→periode(ttest, k), [$1..8]);;
```

$$[5, 5, 5, 5, 5, 1, 2, 2] \tag{6.3}$$

Question 7

```
> estDansOrbite := proc(t, i, j)
    local longO, k, val, trouve;
    longO := periode(t, i);
    k := 0; val := i; trouve := false;
    while ((k < longO) and not(trouve)) do
      if val = j then trouve := true else k := k + 1; val := t[val] fi od;
    trouve end:
> estDansOrbite(ttest, 8, 7);
```

$$true \tag{7.1}$$

Question 8

Idée : on applique la fonction periode de la question 6, on veut que "tous les résultats soient 1 sauf deux (qui seront donc 2)"

```
> estTransposition := proc(t)
      local bon, nombre2, k;
      nombre2 := 0; k := 1; bon := true;
      while (k ≤ taille(t)) and bon do
        if periode(t, k) ≥ 3
        then bon := false
        else if periode(t, k) = 2
               then nombre2 := nombre2 + 1; if nombre2 = 3 then bon := false else k := k + 1
      fi;
               else k := k + 1
```

```
                fi;
            fi; od;
        bon end:
```

```
> estTransposition(ttest);
```

false (8.1)

```
> estTransposition(convert([1, 2, 3, 4, 5, 6, 8, 7], Array));
```

true (8.2)

Question 9

perT : contient les périodes des éléments de t
setperT : c'est l'ensemble des périodes de t
N = card(setperT) = le nombre de périodes (distinctes!) de t
triperT : l'ensemble des périodes de t, trié en croissant

pour trier j'ai utilisé sort ... selon l'X il ne faut rien utiliser

Si N >=3 on a perdu
Si N=1 : il faut que notre unique élément soit 1 ou la longueur de t
Si N=2 : on a [1,k] et il reste à compter le nombre de termes qui sont de période k

```
> estCycle := proc(t)
    local perT, setperT, N, triperT, compte, k;
    perT := map(k→periode(t, k), [$1 ..taille(t)]);
    setperT := convert(convert(perT, set), list); N := nops(setperT);
#   print(perT, " ", setperT, " ", N);
    if N ≥ 3
    then false
    else if N = 1
        then (evalb(perT[1] = taille(t)) or evalb(perT[1] = 1))
        else triperT := sort(setperT);
            if triperT[1] ≠ 1
            then false
            else compte := 0;
                for k from 1 to taille(t)
                do if perT[k] ≠ 1 then compte := compte + 1 else fi od
            fi;
            evalb(compte = triperT[2])
        fi
    fi
end:
```

```
> estCycle(convert([1, 2, 3, 4, 5, 6, 8, 7], Array));
```

true (9.1)

```
> estCycle(ttest);
```

false (9.2)

```
> estCycle(convert([1, 2, 4, 3, 5, 6, 8, 7], Array))
```

false (9.3)

```
> estCycle(convert([1, 2, 3, 4, 5, 6, 7, 8], Array))
                                    true                                    (9.4)
```

Question 10

Pour réutiliser ce que l'on va faire dans les questions qui viennent, on ne va pas se contenter de calculer le tableau des périodes, on va aussi calculer les cycles

.... c'est peu malin, la suite du problème n'a pas fait intervenir les cycles de t, j'aurais dû lire mieux la fin du problème ...

Par exemple : pour t=[1,3,2,6,5,7,4,9,10,11,8] on obtiendra

[[1] [2,3] [4,6,7] [5] [8,9,10,11]]

Le tableau des périodes sera alors obtenu en mappant nops sur cette liste de cycles

On "raye au fur et à mesure" les termes que l'on lit dans le tableau : pour cela le tableau a été copié et dès qu'un terme intervient on met son indice à 0

La fonction ChercheSuivant prend un indice k est un tableau et elle cherche le premier indice supérieur à k de valeur non nulle. S'il n'y en a pas on aura long(t)+1 pour dire "c'est fini"

J'ai utilisé la fonction Reverse de ListTools pour retourner les listes. Selon l'esprit de l'X je n'aurais pas du, il aurait fallu recopier ... ce que l'on a fait dans les T.D. du début d'année

```
> with(ListTools):
> ChercheSuivant := proc(k, tab)
                    if tab[k] = 0
                    then if k = nops(tab)
                         then (k + 1)
                         else ChercheSuivant(k + 1, tab)
                         fi;
                    else k fi end:
> ChercheSuivant(10, [1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0]);
                                    11                                      (10.1)
> DecomposeEnCycles := proc(t) local TousCycles, copieT, long, ChercheCycle, k;
                    copieT := t; long := nops(copieT); k := 1;
                    ChercheCycle := proc(premier, p, debut)
                         if t[p] ≠ premier
                         then copieT[t[p]] := 0; ChercheCycle(premier, t[p], [t[p], op(debut)])
                         else Reverse(debut)
                         fi end;
                    copieT[1] := 0; TousCycles := ChercheCycle(1, 1, [1]);
                    k := ChercheSuivant(1, copieT); #print(k); print(1 + long);
                    while (k ≠ (1 + long))
                    do copieT[k] := 0;
```

```
        TousCycles := ChercheCycle(k, k, [k]), TousCycles; #print(TousCycles);
        k := ChercheSuivant(k, copieT)
    od;
    Reverse([TousCycles])
end:
> testC := [1, 3, 2, 6, 5, 7, 4, 9, 10, 11, 8] : DecomposeEnCycles(testC);
                    [[1], [2, 3], [4, 6, 7], [5], [8, 9, 10, 11]]                    (10.2)
```

Il reste à transformer la liste de liste en liste de ses longueurs :
la premiere fonction passe d'une Liste à une Sequence de Longueurs
la seconde d'une ListedeListes à sa ListedeLongueurs

```
> deLaSL :=proc(l) op(map(k→nops(l), [$1 ..nops(l)])) end;
        deLaSL := proc(l) op(map(k→nops(l), [`$`(1 ..nops(l))])) end proc            (10.3)
> deLaSL([4, 84, 2]);
                                3, 3, 3                                              (10.4)
> LLaLL :=proc(l) map(deLaSL, l) end;
                    LLaLL := proc(l) map(deLaSL, l) end proc                         (10.5)
> LLaLL([[1], [2, 3], [4, 6, 7], [5], [8, 9, 10, 11]]);
                    [1, 2, 2, 3, 3, 3, 1, 4, 4, 4, 4]                                (10.6)
> periodes :=proc(t) LLaLL(DecomposeEnCycles(t)) end;
                periodes := proc(t) LLaLL(DecomposeEnCycles(t)) end proc             (10.7)
> periodes(testC);
                    [1, 2, 2, 3, 3, 3, 1, 4, 4, 4, 4]                                (10.8)
```

Question 11

La fonction itere est "bête" : on utilise n fois t en partant de i. C'est elle qui est utilisée par itereEfficace ... mais un nombre de fois inférieur

```
> itere :=proc(t, i, n) local k, val; val := i;
        if n = 0 then val else for k from 1 to n do val := t[val] od fi end:
> ttest1; ttest2;
                    [4, 2, 1, 7, 3, 5, 9, 8, 6]
                    [1, 3, 2, 6, 5, 7, 4, 9, 10, 11, 8]                              (11.1)
> itere(ttest1, 1, 4); itere(ttest2, 1, 4);
                                6
                                1                                                    (11.2)
> itereEfficace :=proc(t, k) local pt; pt := periodes(t); map(p→itere(t, p, irem(k, pt[p])), [
      $1 ..nops(t)]) end;
itereEfficace := proc(t, k)                                                          (11.3)
    local pt;
    pt := periodes(t); map(p→itere(t, p, irem(k, pt[p])), [`$`(1 ..nops(t))])
```

```
end proc
```

```
> DecomposeEnCycles(ttest1);
```

$$[[1, 4, 7, 9, 6, 5, 3], [2], [8]] \quad (11.4)$$

```
> itereEfficace(ttest1, 3);
```

$$[9, 2, 7, 6, 4, 1, 5, 8, 9] \quad (11.5)$$

```
> periodes(ttest1);
```

$$[7, 7, 7, 7, 7, 7, 7, 1, 1] \quad (11.6)$$

```
> itereEfficace(ttest1, 10); itereEfficace(ttest1, 3);
```

$$[9, 2, 7, 6, 4, 1, 5, 8, 9]$$

$$[9, 2, 7, 6, 4, 1, 5, 8, 9] \quad (11.7)$$

Question 12

[1 2] [3 4 5] [6 7 8 9 10]
Comme on a un 2-cycle : pour revenir au départ 1 et 2 devront s'être échangés 2k fois
Comme on a un 3-cycle : pour revenir au départ 3,4,5 devront s'être échangés 3k fois
Comme on a un 5-cycle : pour revenir au départ 6...10 devront s'être échangés 5k fois

L'ordre est donc 30 > 10

Question 13

```
> pgcd := proc(a, b) if b = 0 then a else pgcd(b, irem(a, b)) fi end;
```

$$pgcd := \mathbf{proc}(a, b)\ \mathbf{if}\ b = 0\ \mathbf{then}\ a\ \mathbf{else}\ pgcd(b, irem(a, b))\ \mathbf{end\ if}\ \mathbf{end\ proc} \quad (13.1)$$

```
> pgcd(10, 500); pgcd(105, 500); pgcd(30, 100); pgcd(10000, 5000);
> trace(pgcd);
```

$$pgcd \quad (13.2)$$

```
> pgcd(456789, 123456); pgcd(456789, 1234561)
{--> enter pgcd, args = 456789, 123456
{--> enter pgcd, args = 123456, 86421
{--> enter pgcd, args = 86421, 37035
{--> enter pgcd, args = 37035, 12351
{--> enter pgcd, args = 12351, 12333
{--> enter pgcd, args = 12333, 18
{--> enter pgcd, args = 18, 3
{--> enter pgcd, args = 3, 0
                                    3
<-- exit pgcd (now in pgcd) = 3}
                                    3
<-- exit pgcd (now in pgcd) = 3}
                                    3
```

```
<-- exit pgcd (now in pgcd) = 3}
                                        3

<-- exit pgcd (now in pgcd) = 3}
                                        3

<-- exit pgcd (now in pgcd) = 3}
                                        3

<-- exit pgcd (now in pgcd) = 3}
                                        3

<-- exit pgcd (now in pgcd) = 3}
                                        3

<-- exit pgcd (now at top level) = 3}
                                        3

Warning, inserted missing semicolon at end of statement
{--> enter pgcd, args = 456789, 1234561
{--> enter pgcd, args = 1234561, 456789
{--> enter pgcd, args = 456789, 320983
{--> enter pgcd, args = 320983, 135806
{--> enter pgcd, args = 135806, 49371
{--> enter pgcd, args = 49371, 37064
{--> enter pgcd, args = 37064, 12307
{--> enter pgcd, args = 12307, 143
{--> enter pgcd, args = 143, 9
{--> enter pgcd, args = 9, 8
{--> enter pgcd, args = 8, 1
{--> enter pgcd, args = 1, 0
                                        1

<-- exit pgcd (now in pgcd) = 1}
                                        1

<-- exit pgcd (now in pgcd) = 1}
                                        1

<-- exit pgcd (now in pgcd) = 1}
                                        1

<-- exit pgcd (now in pgcd) = 1}
                                        1

<-- exit pgcd (now in pgcd) = 1}
                                        1

<-- exit pgcd (now in pgcd) = 1}
                                        1

<-- exit pgcd (now in pgcd) = 1}
                                        1

<-- exit pgcd (now in pgcd) = 1}
                                        1

<-- exit pgcd (now in pgcd) = 1}
                                        1

<-- exit pgcd (now in pgcd) = 1}
                                        1

<-- exit pgcd (now in pgcd) = 1}
                                        1

<-- exit pgcd (now in pgcd) = 1}
                                        1

<-- exit pgcd (now at top level) = 1}
```

1 **(13.3)**

Question 14

```
> ppcm := proc(a, b) a·b / pgcd(a, b) end;
```

$$ppcm := \mathbf{proc}(a, b)\ a * b / pgcd(a, b)\ \mathbf{end\ proc}$$ **(14.1)**

```
> ppcm(23, 230); ppcm(123, 456); pgcd(123, 456);
{--> enter pgcd, args = 23, 230
{--> enter pgcd, args = 230, 23
{--> enter pgcd, args = 23, 0
                                    23
<-- exit pgcd (now in pgcd) = 23}
                                    23
<-- exit pgcd (now in pgcd) = 23}
                                    23
<-- exit pgcd (now in ppcm) = 23}
                                   230
{--> enter pgcd, args = 123, 456
{--> enter pgcd, args = 456, 123
{--> enter pgcd, args = 123, 87
{--> enter pgcd, args = 87, 36
{--> enter pgcd, args = 36, 15
{--> enter pgcd, args = 15, 6
{--> enter pgcd, args = 6, 3
{--> enter pgcd, args = 3, 0
                                     3
<-- exit pgcd (now in pgcd) = 3}
                                     3
<-- exit pgcd (now in pgcd) = 3}
                                     3
<-- exit pgcd (now in pgcd) = 3}
                                     3
<-- exit pgcd (now in pgcd) = 3}
                                     3
<-- exit pgcd (now in pgcd) = 3}
                                     3
<-- exit pgcd (now in pgcd) = 3}
                                     3
<-- exit pgcd (now in pgcd) = 3}
                                     3
<-- exit pgcd (now in ppcm) = 3}
                                 18696
{--> enter pgcd, args = 123, 456
{--> enter pgcd, args = 456, 123
{--> enter pgcd, args = 123, 87
{--> enter pgcd, args = 87, 36
{--> enter pgcd, args = 36, 15
{--> enter pgcd, args = 15, 6
```

```
{--> enter pgcd, args = 6, 3
{--> enter pgcd, args = 3, 0
                                        3

<-- exit pgcd (now in pgcd) = 3}
                                        3

<-- exit pgcd (now in pgcd) = 3}
                                        3

<-- exit pgcd (now in pgcd) = 3}
                                        3

<-- exit pgcd (now in pgcd) = 3}
                                        3

<-- exit pgcd (now in pgcd) = 3}
                                        3

<-- exit pgcd (now in pgcd) = 3}
                                        3

<-- exit pgcd (now in pgcd) = 3}
                                        3

<-- exit pgcd (now at top level) = 3}
                                        3
```

(14.2)

Question 15

Il s'agit donc de calculer le ppcm des valeurs fournies par la fonction periodes.

Ci-dessous : ept : Ensemble des Periodes de T

```
> ordreEfficace := proc(t) local ept, long, val, k;
                   ept := convert(convert(periodes(t), set), list); long := nops(ept);
                   val := 1; for k from 1 to long do val := ppcm(val, ept[k]) od;
    end:
> ordreEfficace([1, 3, 2, 6, 5, 7, 4, 9, 10, 11, 8]);
```