Épreuve d'informatique de l'X - 2011 - MP/PC

Ordre d'une permutation

Question 1. Pour vérifier qu'une tableau représente bien une permutation, on utilise un tableau `test` qui permet de signaler les entiers déjà rencontrés. Dans la boucle des lignes 8 à 14, on vérifie que j le i^{eme} élément du tableau est compris entre 1 et n (la taille du tableau `t`) et que j n'était pas présent dans la partie du tableau étudié. Si l'une des conditions n'est pas vérifiée, on quitte le programme en retournant la valeur `false` (ligne 11) sinon on marque dans le tableau `test` la présence de j. Si on arrive à la ligne 15, c'est que les n éléments du tableau `t` sont distincts et compris entre 1 et n, t représente bien une permutation.

```
1  estPermutation := proc(t :: array)
2  local i, j, n, test;
3  n := taille(t);
4  test := allouer(n);
5  for i from 1 to n do
6       test[i]:=false;
7  od;
8  for i from 1 to n do
9    j := t[i];
10   if j<1 or j>n or test[j]
11      then return(false)
12      else test[j] := true
13   fi;
14 od;
15 return(true)
16 end;
```

Question 2. Pour tout entier i de $[\![1,n]\!]$, la valeur $(t \circ u)(i)$ est représenté par l'entier t[u[i]]. Dans la boucle des lignes 5 à 7, on remplit le tableau des résultats en conséquence.

```
1 composer := proc(t :: array, u :: array)
2 local i, n, res;
3 n:= taille(t);
4 res := allouer(n);
5 for i from 1 to n do
6   res[i] := t[u[i]]
7 end;
8 return(res)
9 end;
```

Question 3 : Si $t(i) = j$ alors $t^{-1}(j) = i$. On applique cette formule à la ligne 6 pour générer le tableau représentant t^{-1} à partir du tableau représentant t.

```
1 inverser := proc(t :: array)
2 local i, n, res;
3 n := taille(t);
4 res := allouer(n);
5 for i from 1 to n do
6     res[t[i]] := i
7 od;
8 return(res)
9 end;
```

Question 4 : La permutation $[1, 2, \ldots, n]$ est une permutation d'ordre 1 et la permutation $[2, 3, \ldots, n, 1]$ est une permutation d'ordre n.

Question 5 : On commence par écrire une fonction `EstIdentite` qui teste si une permutation est égale à l'identité. Elle retourne le booléen `true` si c'est le cas et le booléen `false` dans le cas contraire.
Pour calculer l'ordre d'une permutation, on calcule successivement les t^k jusqu'à ce que l'on obtienne l'identité. Le tableau `tk` représente la permutation t^k. On quitte la boucle conditionnelle des lignes 17 à 20 dès que $t^k = Id_{[\![1,n]\!]}$. La variable de comptage `compt` contient la valeur du k correspondant.

```
1  EstIdentite := proc(t :: array)
2  local i, n;
3  n := taille(t);
4  for i from 1 to n do
5    if t[i]<> i then return(false) fi;
6  od;
7  return(true);
8  end;
9
10 ordre := proc(t :: array)
11 local i, n, compt, tk;
12 compt := 1;
13 n := taille(t);
14 tk := allouer(n);
15
16 for i from 1 to n do tk[i] := t[i] od;
17 while not(EstIdentite(tk)) do
18   compt := compt+1;
19   tk := composer(t,tk);
20 od;
21 return(compt);
22 end;
```

II. Manipuler les permutations

Question 6 : À chaque test dans la boucle conditionnelle des lignes 5 à 8, la variable `j` contient $t^k(i)$. On quitte cette boucle pour le premier entier k tel que $t^k(i) = i$, c'est-à-dire quand k correspond à la période de i pour la permutation t.

```
1  periode := proc(t :: array,i :: posint)
2  local j, k;
3  k := 1;
4  j:=t[i];
5  while j <> i do
6    k := k+1;
7    j := t[j];
8  od;
9  return(k);
10 end;
```

Question 7 : Si p est la période de i alors l'orbite de i est $\{i, t(i), \dots, t^{p-1}(i)\}$. Pour savoir si l'entier j est dans l'orbite de t, on compare j avec les $t^\ell(i)$ ($0 \leqslant \ell < p$ où p est l'ordre de i). On quitte le programme si j coïncide avec l'un des points de l'orbite (ligne 5) et on retourne le booléen `true`. Si j n'est pas dans l'orbite de i, on exécute la boucle sans quitter le programme, celui-ci va donc retourner le booléen `false` (ligne 9).

```
1  estDansOrbite := proc(t :: array, i :: posint, j ::posint)
2  local k, l, res;
3  k:=i;
4  for l from 1 to periode(t,i) do
5     if k=j then return(true)
6            else k:=t[k]
7    fi;
8  od;
9  return(false);
10 end;
```

Question 8 : Pour tester si une permutation est une transposition, on compte le nombre d'entiers qui ne sont pas invariants par la permutation `t`. Si ce nombre est égal à 2, c'est que `t` est une transposition et sinon ce n'est pas le cas.

```
estTransposition := proc(t :: array)
local i, n, compt;
n := taille(t);
compt := 0;
for i from 1 to n do
  if t[i] <> i then compt := compt+1 fi;
od;
return(evalb(compt=2));
end;
```

Question 9 : Plus généralement, pour tester si une permutation t est un cycle, il suffit de vérifier que le nombre d'éléments qui ne sont pas invariants par t est égal à l'ordre de la permutation t. C'est le principe du programme `estCycle` :

```
estCycle := proc(t :: array)
local i, n, compt;
n := taille(t);
compt := 0;
for i from 1 to n do
  if t[i] <> i then compt := compt+1 fi;
od;
return(evalb(compt = ordre(t)));
end;
```

Question 10 : On commence par créer un tableau `p` des périodes que l'on initialise avec des 0. Puis, pour chaque entier `i`, si la période de i n'a pas été encore attribuée (donc `p[i]=0`) alors, on calcule la période p_i de l'élément i. Les éléments de l'orbite de i ayant la même période que i, on met la valeur p_i dans les cases du tableau `p` correspondant aux éléments de l'orbite de i. Le tableau des périodes est parcouru deux fois ainsi que celui représentant la permutation t. La complexité est bien linéaire.

```
periodes := proc(t :: array)
local i, j, k, n, p, p_i;
n := taille(t);
p := allouer(n);
for i from 1 to n do p[i]:=0 od;
for i from 1 to n do
  if p[i] = 0 then
    p_i := periode(t,i);
    k := i;
    for j from 1 to p_i do
      p[k] := p_i;
      k := t[k];
    od;
  fi;
od;
return(p)
end;
```

Question 11 : Si p est la période de l'élément i et k' le reste de la division euclidienne de k par p, on a $t^k(i) = t^{k'}(i)$. Ce résultat est exploité dans le programme `itererEfficace` ci-dessous. On commence par calculer le tableau des périodes (ligne 5), puis pour chaque élément i, on calcule $t^{k'}(i)$ grâce à la boucle des lignes 8 à 9. Le résultat est stocké dans le tableau `res` (ligne 11).

```
1  itererEfficace := proc(t :: array, k :: posint)
2  local i, j, n, l, res, tab_periodes;
3  n := taille(t);
4  res := allouer(n);
5  tab_periodes := periodes(t);
6  for i from 1 to n do
7     l := i;
8     for j from 1 to reste(k,tab_periodes[i]) do
9        l := t[l];
10    od;
11    res[i] := l;
12 od;
13 return(res);
14 end;
```

Question 12 : La permutation $[5, 4, 2, 3, 1]$ est d'ordre 6 et de taille 5.

Question 13 : On écrit une fonction `pgcd` qui repose sur l'algorithme d'Euclide.

```
pgcd := proc(a :: nonnegint, b :: nonnegint)
local u, v, tmp;
u := a;
v := b;
while v <> 0 do
    tmp := v;
    v := reste(u,v);
    u := tmp;
od;
return(u);
end;
```

Question 14 : La fonction `ppcm` repose sur l'identité $\text{ppcm}(a,b) = \dfrac{a.b}{\text{pgcd}(a,b)}$.

```
ppcm := proc(a :: nonnegint, b :: nonnegint)
  return(a*b/pgcd(a,b));
end;
```

Question 15 : On calcule le plus petit commun multiple des périodes des éléments. Pour limiter le nombre d'appels à la fonction `ppcm`, on utilise un tableau `marquage` qui permet de mémoriser les k qui ont déjà été pris en compte dans le calcul du ppcm.

```
ordreEfficace := proc(t :: array)
local i, k, n, periodes_t, res, marquage;
n := taille(t);
periodes_t := periodes(t);
marquage := allouer(n);
res := 1;
for i from 1 to n do
    marquage[i] := true
od;
for i from 1 to n do
    k := periodes_t[i];
    if marquage[k] then marquage[k] := false; res := ppcm(res,k) fi;
od;
return(res);
end;
```