▶ **MPSI – Option Informatique**
Année 2001, Cinquième TP Caml
Vincent Simonet (`http://cristal.inria.fr/~simonet/`)

Permutations

Ce sujet décrit quelques problèmes classiques à propos des permutations d'ensemble fini. Une permutation σ de l'ensemble des entiers compris entre 0 et $n-1$, $[\![0, n-1]\!]$, est une bijection de cet ensemble dans lui-même.

1 Représentation des permutations

Nous représenterons une permutation σ de $[\![0, n-1]\!]$ par un tableau t de taille n tel que pour tout $i \in [\![0, n-1]\!]$, l'entier $t.(i)$ soit égal à $\sigma(i)$.

▶ **Question 1** *À quelle condition un tableau t de longueur n représente-t-il une permutation de $[\![0, n-1]\!]$? Écrivez une fonction* est_permutation *qui prend pour argument un tableau t et teste si ce tableau représente une permutation.*

Le support d'une permutation σ est l'ensemble des points non fixes de σ (i.e. l'ensemble des i tels que $\sigma(i) \neq i$).

▶ **Question 2** *Écrivez une fonction* support *qui prend pour argument un tableau représentant une permutation et calcule son support (sous forme d'une liste d'entiers).*

La composée de deux permutations de $[\![0, n-1]\!]$, σ et σ', est l'application $\sigma \circ \sigma'$, au sens habituel des fonctions. On vérifie facilement que la composée de deux permutations de $[\![0, n-1]\!]$ est une permutation de $[\![0, n-1]\!]$.

▶ **Question 3** *Programmez une fonction* compose *qui prend pour arguments deux tableaux représentant des permutations et calcule le tableau qui représente la permutation composée.*

2 Énumération des permutations

2.1 Ordre sur les permutations

On ordonne les permutations de $[\![0, n-1]\!]$ selon l'ordre lexicographique. Par exemple, si $n = 4$, on a $[|0;1;3;2|] < [|0;2;1;3|] < [|1;2;3;0|] < [|2;0;1;3|]$. Nous allons écrire une fonction qui étant donnée un tableau t représentant une permutation trouve son successeur immédiat. On peut pour cela appliquer l'algorithme suivant :

1. Déterminer le plus grand entier i tel que $t.(i) < t.(i+1)$.
2. Permuter $t.(i)$ avec le plus petit élément qui lui soit plus grand parmi $t.(i+1), \ldots, t.(n-1)$.
3. Trier la partie restante du tableau (i.e. $t.(i+1), \ldots, t.(n-1)$).

On commence par écrire quelques fonctions préliminaires.

▶ **Question 4** *Écrivez une fonction* max_croit *qui prend pour argument un tableau t et retourne le plus grand entier i tel que $t.(i) < t.(i+1)$.*

▶ **Question 5** *Écrivez une fonction* permute_max *qui prend pour argument un tableau t, un entier k et permute dans le tableau t le contenu de la case k avec le plus petit élément qui lui soit plus grand parmi $t.(k+1), \ldots, t.(n-1)$ (où n est la longueur de t).*

▶ **Question 6** *Écrivez une fonction* trie *qui prend pour argument un tableau t et un entier k. Cette fonction triera* en place *la partie du tableau située à droite de la case k.*

▶ **Question 7** *Déduisez-en une fonction* suivant *qui prend pour argument un tableau t et modifie t* sur place *de manière à obtenir la permutation suivante. Votre fonction sera de type int vect → unit.*

2.2 Affichage des permutations

On souhaite écrire une fonction qui, étant donné un entier n, affiche successivement toutes les permutations de $[\![0, n-1]\!]$.

▶ **Question 8** *Programmez une fonction* print_perm *qui prend pour argument un tableau représentant une permutation et l'affiche sur la* sortie standard *sous la forme de votre choix. Vous pourrez utiliser les fonctions prédéfinies* print_int, print_string *et* print_newline.

▶ **Question 9** *Déduisez-en une fonction* enumere *qui prend pour argument un entier n et énumère toutes les permutations de* $[\![0, n-1]\!]$. *Combien y en a-t-il ?*

3 Cycles

Un cycle est une permutation σ tel qu'il existe k éléments $x_0, \ldots, x_{k-1}$ de $[\![0, n-1]\!]$ tels que $\mathrm{dom}(\sigma) = \{x_0, \ldots, x_{k-1}\}$ et $\sigma(x_0) = x_1$, $\sigma(x_1) = x_0$, ..., $\sigma(x_{k-2}) = x_{k-1}$ et $\sigma(x_{k-1}) = x_0$. On peut représenter un tel cycle par la liste $[x_0; \ldots; x_{k-1}]$. On peut montrer que toute permutation peut se décomposer comme un (unique) produit de cycles de supports disjoints. On représentera un produit de cycles sous forme d'une liste de liste d'entiers.

▶ **Question 10** *Décomposez la permutation* $[|2; 3; 4; 1; 0; 5|]$ *en un produit de cycles de support disjoints.*

On s'intéresse tout d'abord à l'écriture d'une fonction qui permette de traduire un permutation représentée comme produit de cycles sous forme de tableau.

▶ **Question 11** *Écrivez une fonction qui prend pour argument un cycle c (sous forme d'une liste d'entier) et un tableau t* égal à l'identité sur le support du cycle. *Cette fonction modifiera* en place *le tableau t de telle sorte qu'il représente la composée des deux permutations passées en arguments.*

▶ **Question 12** *Déduisez-en une fonction* produit *qui prend pour argument un produit de cycles de supports disjoints ainsi que l'entier n et retourne la permutation de* $[\![0, n-1]\!]$ *représentée par le produit de cycles.*

On s'attache maintenant à la question inverse : décomposer une permutation en un produit de cycles.

▶ **Question 13** *Déduisez-en une fonction* cycles *qui prend pour argument un tableau représentant une permutation et décompose cette permutation en un produit de cycles de supports disjoints.*

▶ **MPSI – Option Informatique**
Année 2001, Cinquième TP Caml
Vincent Simonet (`http://cristal.inria.fr/~simonet/`)

Permutations

Un corrigé

▶ **Question 1** Une condition nécessaire et suffisante pour qu'un tableau t de longueur n représente une permutation de $[\![0, n-1]\!]$ est que tous les éléments de t soient différents et compris entre 0 et $n-1$. On en déduit la fonction *Caml* suivante :

```
let est_permutation t =
  let n = vect_length t in
  let resultat = ref true in
  for i = 0 to n - 1 do
    if t.(i) >= n or t.(i) < 0 then
      resultat := false;
    for j = i + 1 to n - 1 do
      if t.(i) = t.(j) then
        resultat := false;
    done
  done;
  ! resultat
;;
```

▶ **Question 2**

```
let support t =
  let n = vect_length t in
  let liste = ref [] in
  for i = 0 to n - 1 do
    if t.(i) <> i then
      liste := i :: !liste
  done;
  !liste
;;
```

▶ **Question 3**

```
let compose t t' =
  let n = vect_length t in
  let s = make_vect n 0 in
  for i = 0 to n - 1 do
    s.(i) <- t.(t'.(i))
  done;
  s
;;
```

▶ **Question 4** On définit une exception qui sera levée lorsqu'on aura atteint la dernière permutation.

```
exception Fini;;
```

```
let max_croit t =
  let rec aux = function
      0 -> raise Fini
    | i ->
      if t.(i - 1) < t.(i) then i - 1
      else aux (i - 1)
  in
  aux (vect_length t - 1)
;;
```

▶ **Question 5**

```
let swap t a b =
  let tampon = t.(a) in
  t.(a) <- t.(b);
  t.(b) <- tampon
;;

let permute_max t k =
  let j = ref k in
  for i = k + 1 to vect_length t - 1 do
    if t.(i) > t.(k) && (t.(i) < t.(!j) || !j = k)
    then j := i
  done;
  swap t !j k
;;
```

▶ **Question 6**

```
let trie t k =
  let n = vect_length t in
  let rec select_min i =
    if i = n - 1 then n - 1 else
    let i' = select_min (i + 1) in
    if t.(i') < t.(i) then i' else i
  in

  for i = k + 1 to n - 1 do
    let i' = select_min i in
    swap t i i'
  done
;;
```

▶ **Question 7**

```
let suivant t =
  let i = max_croit t in
  permute_max t i;
  trie t i
;;
```

▶ Question 8

```
let print_perm t =
  let n = vect_length t in
  for i = 0 to n - 1 do
    print_int i;
    print_string "->";
    print_int t.(i);
    print_string " "
  done;
  print_newline ()
;;
```

▶ Question 9

```
let enumere n =
  let t = make_vect n 0 in
  for i = 1 to n - 1 do
    t.(i) <- i
  done;

  try
    while true do
      print_perm t;
      suivant t
    done;
  with
    Fini -> ()
 ;;
```

▶ Question 11

```
let compose_cycle t c =
  match c with
    [] -> failwith "cycle vide"
  | x :: _ ->
    let rec aux = function
        [] -> ()
      | y :: [] -> t.(y) <- x
      | y :: z :: q ->
        t.(y) <- z;
        aux (z :: q)
    in
    aux  c
;;
```

▶ Question 12

```
let produit cycles n =
  let t = make_vect n 0 in
  for i = 0 to n - 1 do
    t.(i) <- i
  done;
  let rec itere = function
      [] -> ()
    | c :: q ->
      itere q;
      compose_cycle t c
  in
  itere cycles;
  t
;;
```

▶ Question 13

```
let cycles t =
  let n = vect_length t in
  let vu = make_vect n false in
  let liste = ref [] in
  let rec aux i =
    if vu.(i) then [] else
    begin
      vu.(i) <- true;
      i :: (aux t.(i))
    end
  in
  for i = 0 to n - 1 do
    let c = aux i in
    if c <> [] then liste := c :: !liste
  done;
  !liste
;;
```