Recherche de la permutation suivante, explication

Michel Billaud (michel.billaud@laposte.net)

3 novembre 2021

Table des matières

Dernières corrections : 6 novembre.

1 Le problème

Le problème est une question classique d'algorithmique combinatoire :

> Donnez un algorithme efficace qui, à partir d'une permutation, détermine la permutation suivante dans l'ordre lexicographique.

Par exemple, l'algorithme trouvera que la permutation $p = (1, 4, 5, 3, 2)$ est suivie par $next(p) = (1, 5, 2, 3, 4)$.

Notation. La permutation $(1, 4, 5, 3, 2)$ est la bijection p sur $1, 2 \ldots$ telle que $p(1) = 1, p(2) = 4, p(3) = 5, p(4) = 3$ et $p(5) = 2$.

On trouve facilement cet algorithme sur internet, je tente ici de donner quelques éclaircissements.

Cet algorithme permet de construire un programme de génération de permutations dans l'ordre lexicographique, en remarquant que pour tout N

- la première permutation est la suite croissante $(1, 2, \ldots, N)$, facile à construire ;
- la dernière est la suite décroissante $(N, …, 2,1), facile à détecter.

2 Construction de *next*

La construction de *next* repose sur quelques observations simples.

2.1 Comment compléter une permutation

Quand on connait le début d'une permutation, il est facile de déterminer l'ensemble des permutations qui la complètent.

Par exemple, pour compléter $(4, 6, 3, ?, ?, ?)$, permutation de longueur 6

- les éléments manquants sont 1, 2, 5 ;
- il suffit de combler les "trous" par les permutations de $\{1, 2, 5\}$.

Nous sommes intéressés par la **séquence** de permutations dans l'ordre lexicographique, obtenue en complétant le "préfixe" $(4, 6, 3)$ successivement par les permutations du "reste" $\{1, 2, 5\}$ dans l'ordre.

Cette séquence commence par $(4, 6, 3, 1, 2, 5)$ et finit par $(4, 6, 3, 5, 2, 1)$, qui ont le "reste" en ordre croissant et décroissant respectivement.

2.2 Décomposition d'une permutation

De façon évidente, chaque permutation se décompose *de façon unique* en concaténation d'un préfixe et d'un suffixe qui soit la plus longue suite décroissante possible.

Exemple, pour $p = (1, 4, 5, 3, 2)$

- le suffixe est $(5, 3, 2)$,
- le préfixe est $(1, 4)$;

On remarque que

- le suffixe est non vide (le dernier élément est une suite décroissante à lui tout seul) ;
- le préfixe est vide si et seulement si la permutation est la dernière,
- par définition, si le préfixe n'est pas vide, son dernier élément est plus grand que le premier du suffixe (sinon cet élément ferait partie du fixe).

2.3 Passer d'une permutation à la suivante

Une propriété intéressante de cette décomposition est que toute permutation est la plus grande, dans l'ordre lexicographique, de celles qui ont le même préfixe.

La permutation suivante, si elle existe (ce qui se voit au préfixe non vide), commence donc par la séquence qui s'obtient en augmentant le dernier élément du préfixe, et en complétant. Exemple,

- le préfixe de $(1,4,5,3,2)$ est $(1,4)$,
- les permutations suivantes commencent par $(1,5)$,
- la première qui suit est $next(p) = (1,5,4,3,2)$.

Attention, il ne suffit pas de remplacer le dernier élément du préfixe par le suivant. Par exemple le préfixe $(4,3,2)$ de $(4,3,2,6,5,1)$ se termine par 2, qu'on ne peut pas remplacer par $2+1=3$ qui apparaît déjà dans le préfixe. Il faut prendre 5, qui est le plus petit élément du suffixe qui soit supérieur à 2.

Propriété Cet élément existe toujours.

En effet, le suffixe n'est pas vide, et par construction son premier élément est plus grand que le dernier du préfixe.

2.4 Une propriété des séquences décroissantes

Une propriété des séquences décroissantes simplifiera la construction de la première permutation suivante :

Propriété soit une séquence décroissante $s_1, s_2, \dots, s_n$ et t un nombre. Si il existe un k tel que s_k soit le plus petit des éléments qui soient supérieurs à t, alors la séquence obtenue en remplaçant s_k par t est décroissante.

En effet,

- $s_k \geq t$ (parce que s_k est supérieur à t) ;
- si $k > 1$, s_{k-1} existe et $s_{k-1} \geq s_k$, donc $s_{k-1} \geq t$ (parce que $s_k \geq t$) ;
- si $k < n$, s_{k+1} existe. Si t était inférieur à s_{k+1}, s_k ne serait pas le plus petit élément de la séquence qui soit supérieur à t. Donc $t \geq s_{k+1}$.

Il en résulte que la séquence obtenue est décroissante.

2.5 Conséquence et construction de la permutation suivante

Les étapes de la construction de la permutation suivante

1. Décomposer la permutation en préfixe et suffixe.
2. Si le préfixe est vide, la permutation est la dernière.
3. Dans le suffixe, trouver le plus petit élément qui soit supérieur au dernier élément du préfixe.
4. Echanger cet élément avec le dernier du préfixe
5. Retourner le suffixe pour le mettre en ordre croissant.

La remarque précédente a permis de traduire la dernière étape "compléter le suffixe par le reste des éléments en ordre croissant" par un simple retournement de tableau ("miroir").

3 Codage

Le programme ci-dessous, en Fortran 95, affiche les permutations de longueur 4.

L'enumération des permutations met en oeuvre deux fonctions qui ont un tableau et sa taille en paramètre.

- `get_first_permutation` remplit le tableau avec la permutation initiale (1,2,...)
- `get_next_permutation` transforme la permutation contenue dans le tableau en la permutation suivante

Le booléen retourné par les deux fonctions indiquent le succès de l'opération.

3.1 Code du programme principal

```
!: @file test_permutations.f95
!! test program for the permutations module
!! displays all 24 permutations of (1, 2, 3, 4)

program test_permutations
  use permutations
  implicit none
  integer :: array(1:4)
  logical :: exists
  integer :: number

  number = 0
  exists = get_first_permutation(array, size(array))
  do while (exists)
     number = number + 1
     call print_permutation
     exists = get_next_permutation(array, size(array))
  end do

contains

  subroutine print_permutation
    integer :: i
    write(*, "(I3,': ',*(I3))") number, (array(i), i = 1, size(array))

    end subroutine print_permutation

end program test_permutations
```

3.2 Le module `permutations`

Les fonctions sont fournies par le module `permutations`

```
!> @file permutations.f95
!! @brief generation of permutation in lexicographic order
```

```fortran
module permutations

  implicit none

  private
  public :: get_first_permutation, get_next_permutation

contains

!> get the first permutation (1, 2...size) into an array
!! @param array array to be filled
!! @param size length of the permutation
!! @return true if possible

  function get_first_permutation(array, size) result(found)

    integer, intent(IN)  :: size
    integer, intent(OUT) :: array(1:size)
    logical :: found

    integer :: i

    found = (size >= 1)
    if (.not. found) return

    do i = 1, size
       array(i) = i
    end do
  end function get_first_permutation

  !> change a given permutation into the next one if possible
  !! @param array array containing the permutations
  !! @param size length of the permutation
  !! @return true if possible

  function get_next_permutation(array, size) result(found)

    integer, intent(IN)     ::  size
    integer, intent(INOUT)  :: array(1:size)
    logical                 :: found

    integer                 :: prefix_length, index

    prefix_length = find_prefix_length(array, size)
    found = (prefix_length > 0)
    if (.not. found) return           ! array contains the last permutation

    index = find_next_in_prefix(array, size, array(prefix_length))
    call swap(array(prefix_length), array(index))
```

```fortran
  call reverse_suffix(array, size, prefix_length + 1)
end function get_next_permutation

!
! private part
!
function find_prefix_length(array, size) result(length)

  integer, intent(IN) ::  size
  integer, intent(IN) :: array(1:size)
  integer             :: length

  length = size - 1
  do while ((length > 0) .and. (array(length) >= array(length + 1)))
     length = length - 1
  end do
end function find_prefix_length

function find_next_in_prefix(array, size, value) result(index)

  integer,  intent(IN)      ::  size, value
  integer,  intent(IN)      :: array(1:size)

  integer                   :: index

  index = size
  do while (array(index) <= value)
     index = index - 1
  end do
end function find_next_in_prefix

subroutine reverse_suffix(array, size, suffix_start)

  integer, intent(IN)      :: size, suffix_start
  integer, intent(INOUT)   :: array(1:size)

  integer :: left, right

  left = suffix_start
  right = size
  do while (left < right)
     call swap (array(left), array(right))
     left = left + 1
     right = right - 1
  end do
end subroutine reverse_suffix

subroutine swap(a, b)

  integer, intent(inout) :: a, b
```

```fortran
    integer :: c

    c = a
    a = b
    b = c
  end subroutine swap

end module permutations
```

3.3 Compilation et exécution

```
$ gfortran -std=f95 -Wall -Wextra -pedantic    \
    test_permutations.f95 permutations.f95     \
    -o test_permutations
$ ./test_permutations
  1:   1  2  3  4
  2:   1  2  4  3
.... 20 lignes supprimées....
 23:   4  3  1  2
 24:   4  3  2  1
```