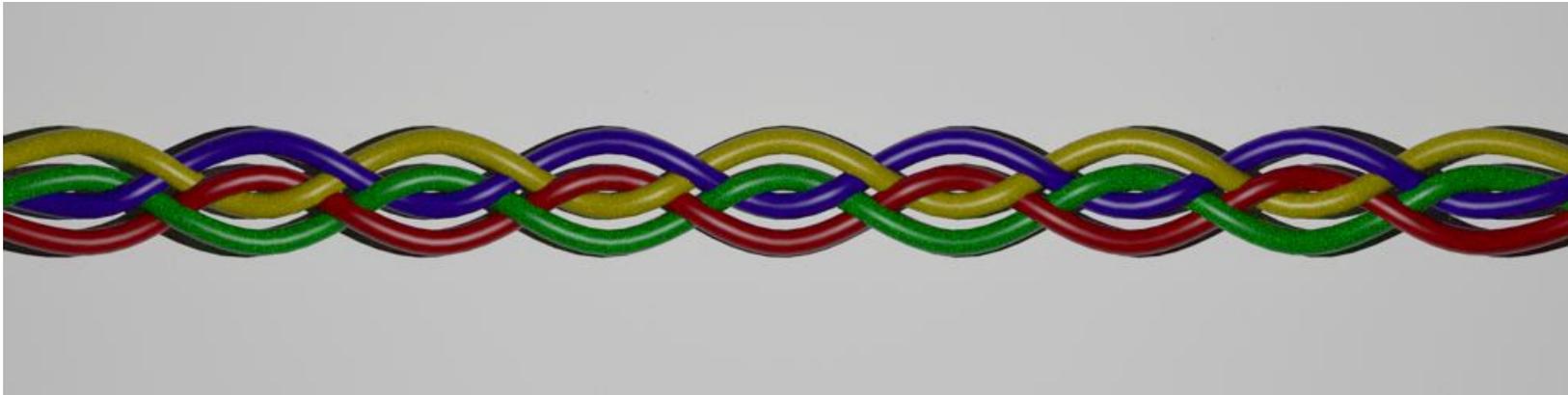


ИвГУ, ф-т МиКН, курс 2

"КОМПЬЮТЕРНАЯ АЛГЕБРА"

Тема 14.

Перестановки



Лектор: Н. И. Яцкин, 2014

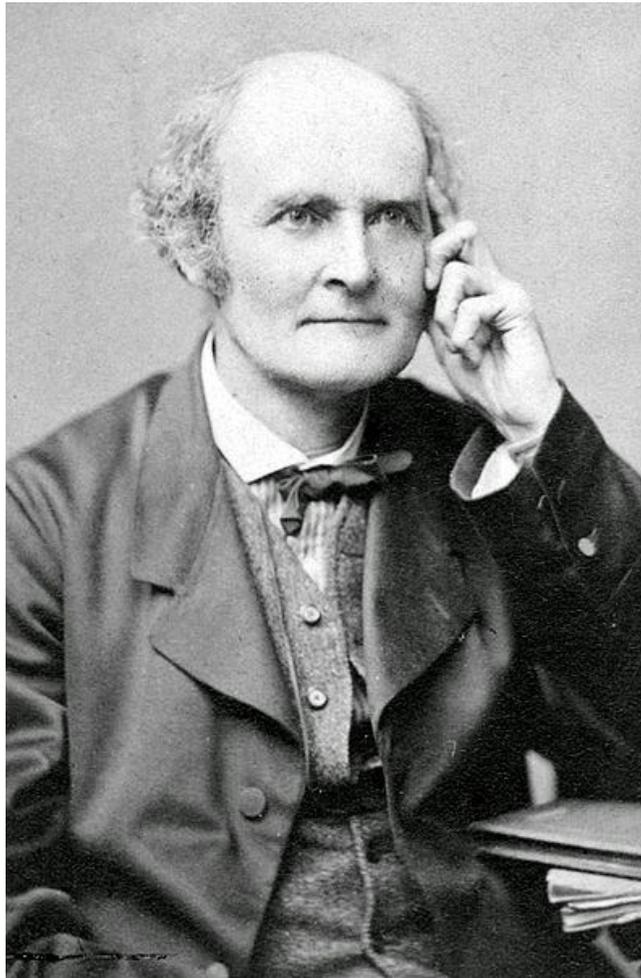
$$S_n = \left\{ \varphi = \begin{pmatrix} 1 & 2 & \dots & n \\ \varphi(1) & \varphi(2) & \dots & \varphi(n) \end{pmatrix} \right\}.$$

Симметрическая группа (группа перестановок) степени n определяется как группа **биекций** конечного множества $X = \{1, 2, \dots, n\}$ на себя.

Содержит $n!$ элементов.

Алгебраическая операция: умножение (композиция).

Теорема 1 (Кэли). *Всякая конечная группа вкладывается в некоторую группу перестановок.*



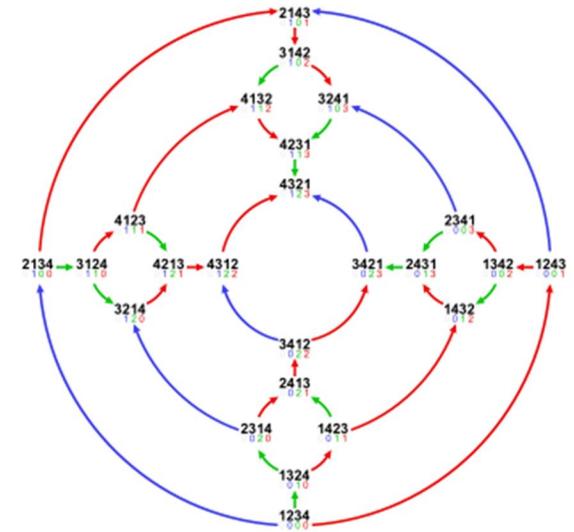
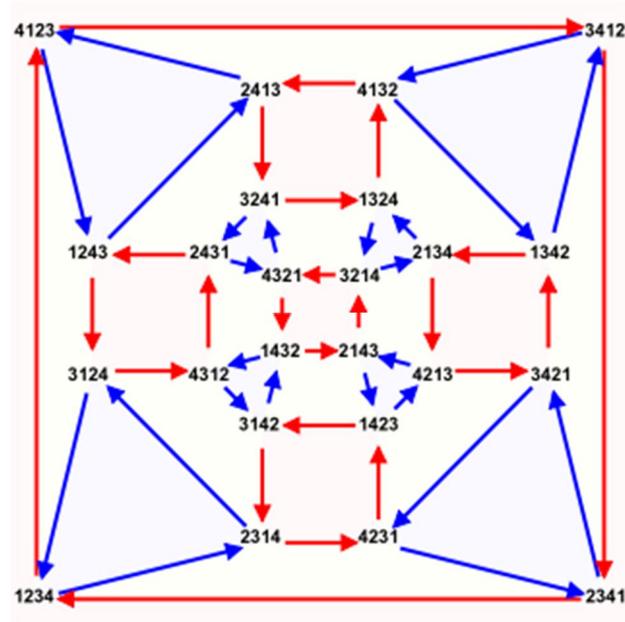
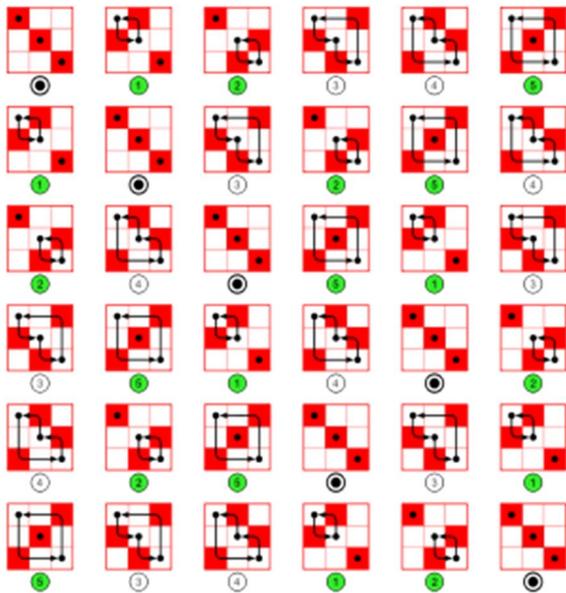
А́ртур Кэ́ли (*Arthur Cayley*; **1821 - 1895**) - английский математик.

Отец – торговец; первые восемь лет жизни Артур провел в **Санкт-Петербурге**.

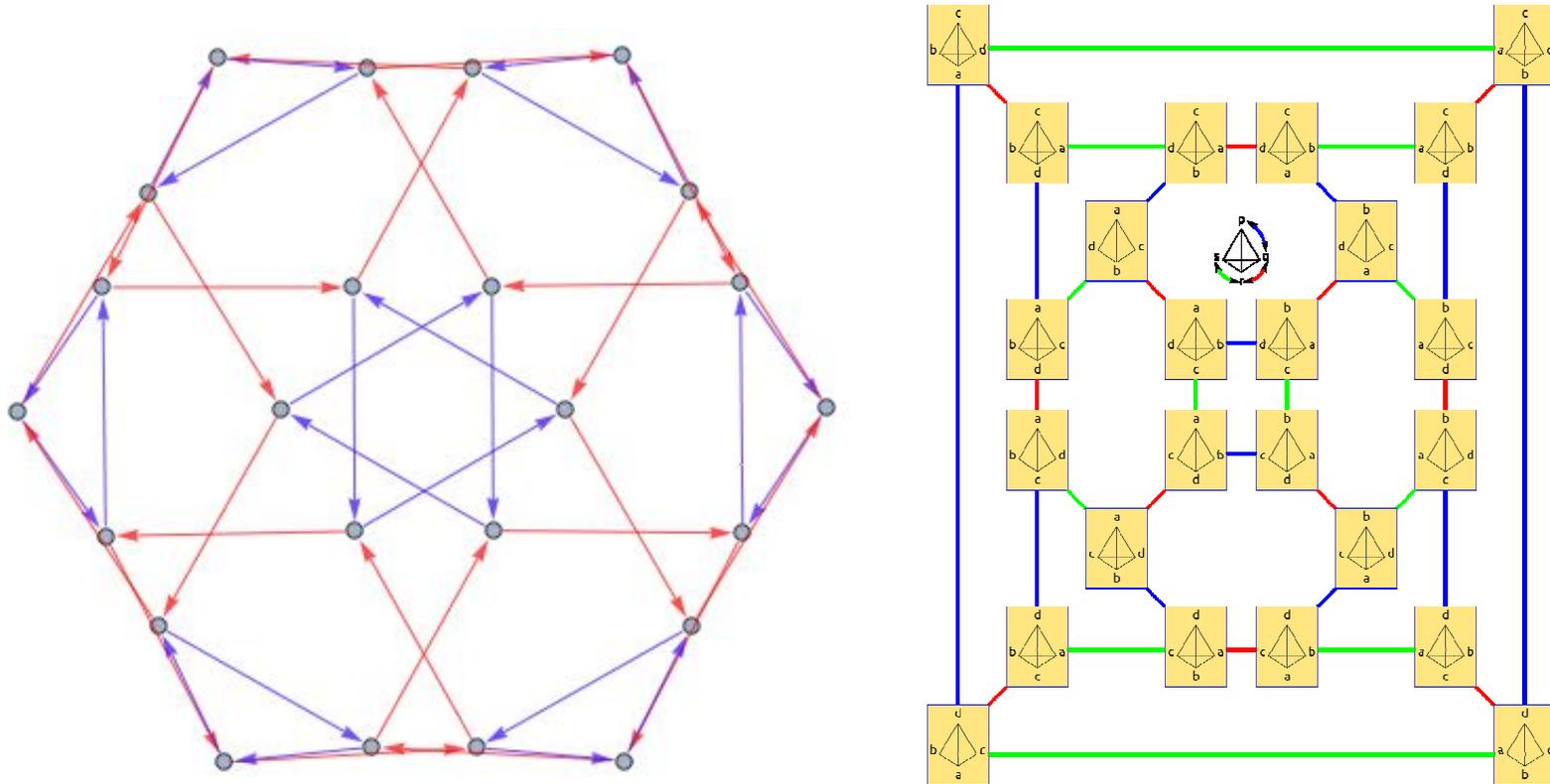
Кэли - один из плодовитейших учёных XIX века, написал более 700 работ (**линейная алгебра**, дифференциальные уравнения, **теория групп**); был первым, кто сформулировал **определение группы** в том виде, как оно дается сегодня: **множество с бинарной операцией, удовлетворяющей определённым законам**.

Группа

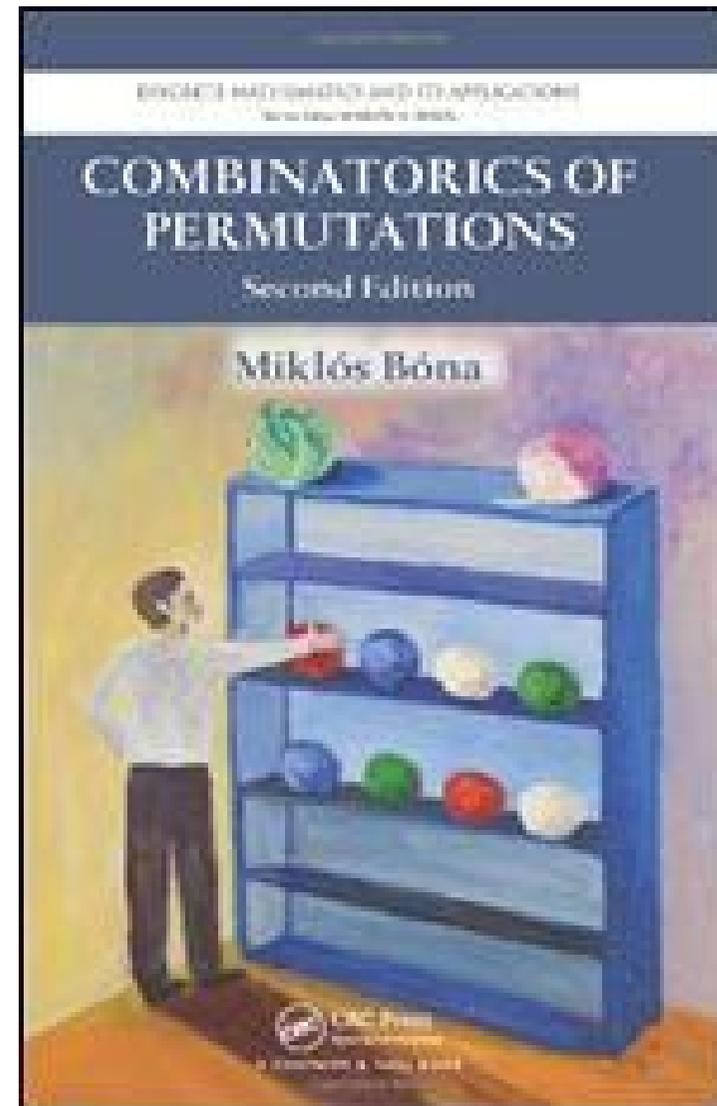
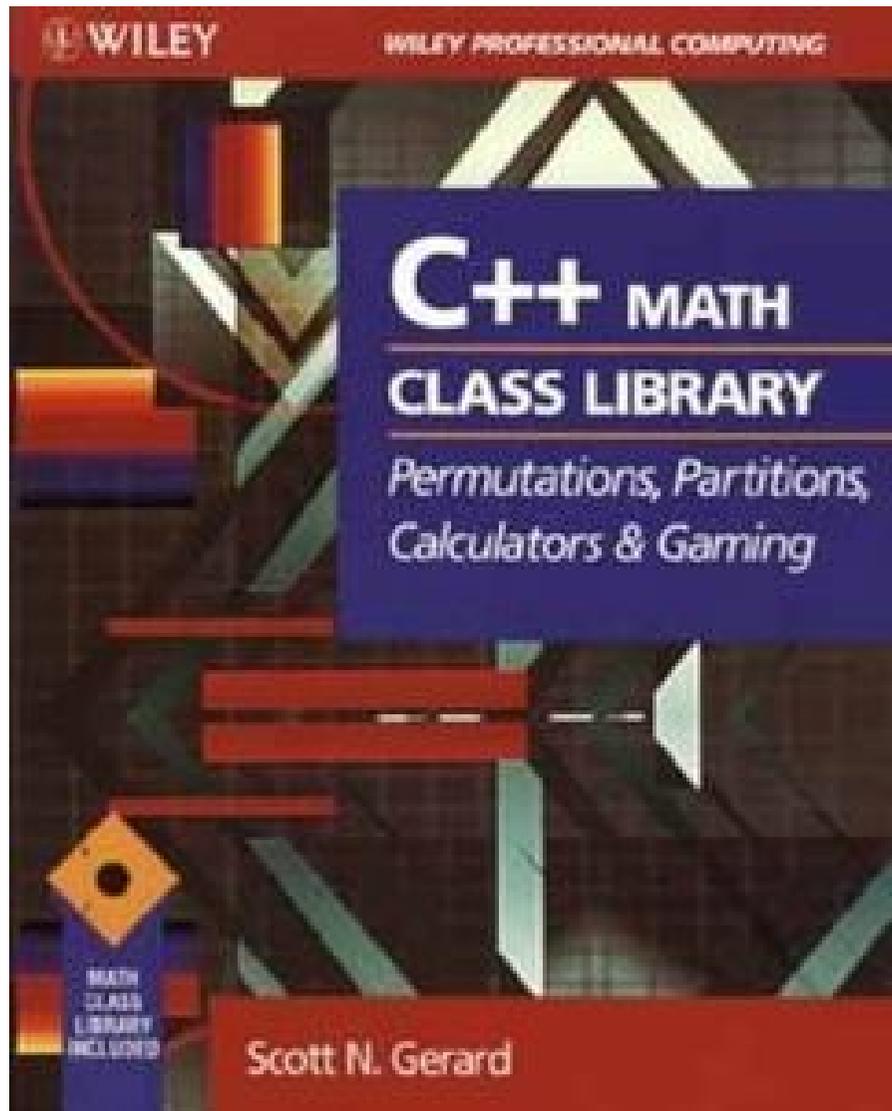
(визуализация: три облика)



Еще визуализации (других групп)



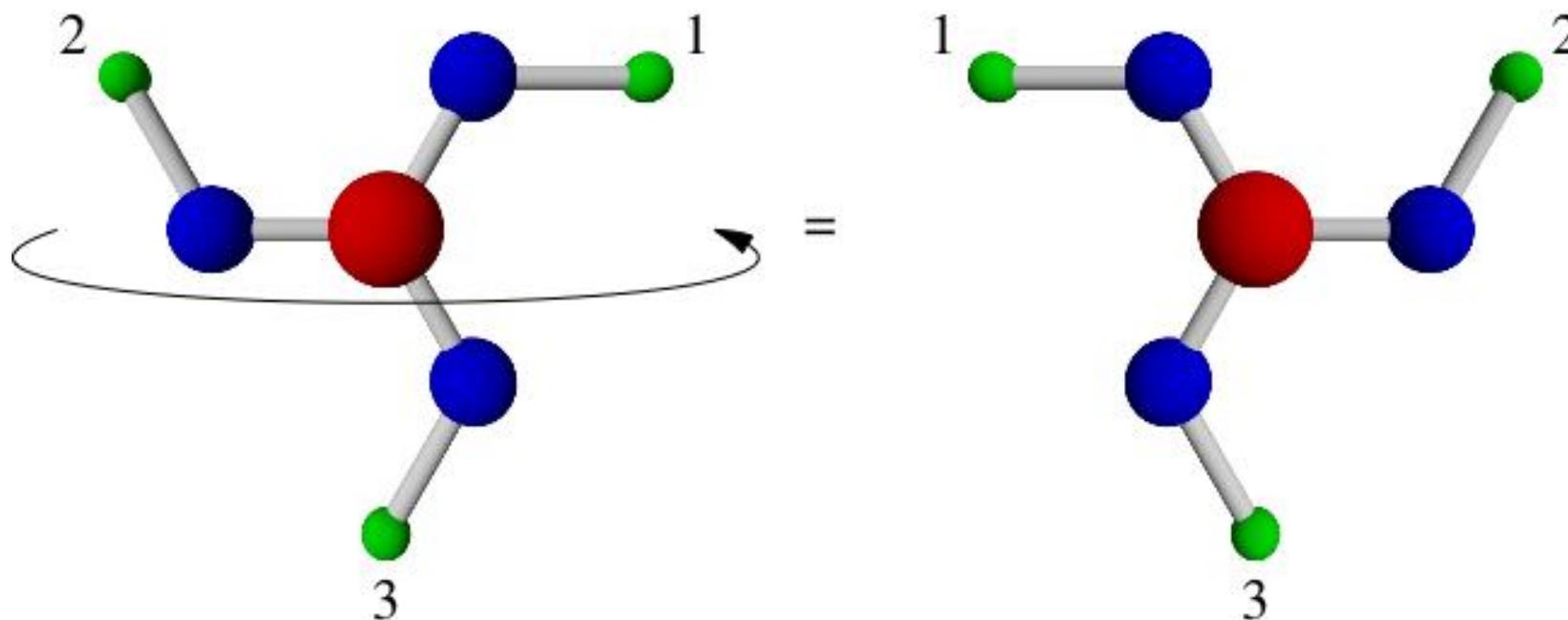
Далее: **книги** по компьютерному исследованию **перестановок** и других комбинаторных структур



Ближайшая цель: создание *пакета процедур*, обеспечивающих выполнение основных операций и решение типовых задач в группе перестановок. Детали "пакетного оформления" рассматриваться не будут. Но одна характерная особенность (определение нового, специфического для рассматриваемых задач, *типа данных*) - будет объяснена.

Замечание. В системе **Maple** есть "фирменный" пакет **group**, который умеет работать с перестановками. Он "экономнее" нашего "самодельного" пакета, ориентированного исключительно на учебные цели.

Описание процедур 00 - 15



00. Определение типа PERM (permutation).

```

> with (LinearAlgebra) ;
> TypeTools [AddType]
(PERM, a->evalb
  (type (a, Matrix) and
    RowDimension (a)=2 and
    convert (a [1, 1..ColumnDimension (a)], list) =
              [$1..ColumnDimension (a)] and
    convert (a [2, 1..ColumnDimension (a)], set) =
              {$1..ColumnDimension (a)}
  )
);

```

Чтобы матрица была признана перестановкой требуется, истинность описанной выше конъюнкции. Напоминание: \$ - знак последовательности.

Примеры. В каждой из процедур пакета принимаемые величины проверяются на принадлежность вновь введенному типу **PERM**.

> **phi:=Matrix([[1,2,3,4,5,6,7,8],[2,7,5,4,3,1,6,8]]);**

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

> **type(phi, PERM);**

true

> **psi:=Matrix([[1,2,3],[3,2,2]]);**

$$\psi := \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 2 \end{bmatrix}$$

> **type(psi, PERM);**

false

***Замечание.** Представление перестановки в виде **двустрочной матрицы** является, очевидно, **избыточным**, поскольку верхняя строка имеет фиксированное заполнение **[1, 2, ..., n]**. Было бы достаточно задавать только нижнюю строку, как **список**.*

*Именно так делается в **Maple**-пакете **group**, при этом описание соответствующего типа **'permlist'** оказывается существенно проще. Мы, однако, предпочитаем сохранять привязку к обозначениям из учебников по алгебре.*





01. Процедура вычисления степени Pdeg.

```
> Pdeg:=proc (a::PERM) ;  
RETURN (ColumnDimension (a)) ;  
end proc ;  
  
> Pdeg (phi) ; Pdeg (psi) ;
```

8

```
Error, invalid input: Pdeg expects its 1st argument, a, to be of  
type PERM, but received Matrix(2, 3, [[...],[...]], datatype =  
anything)
```

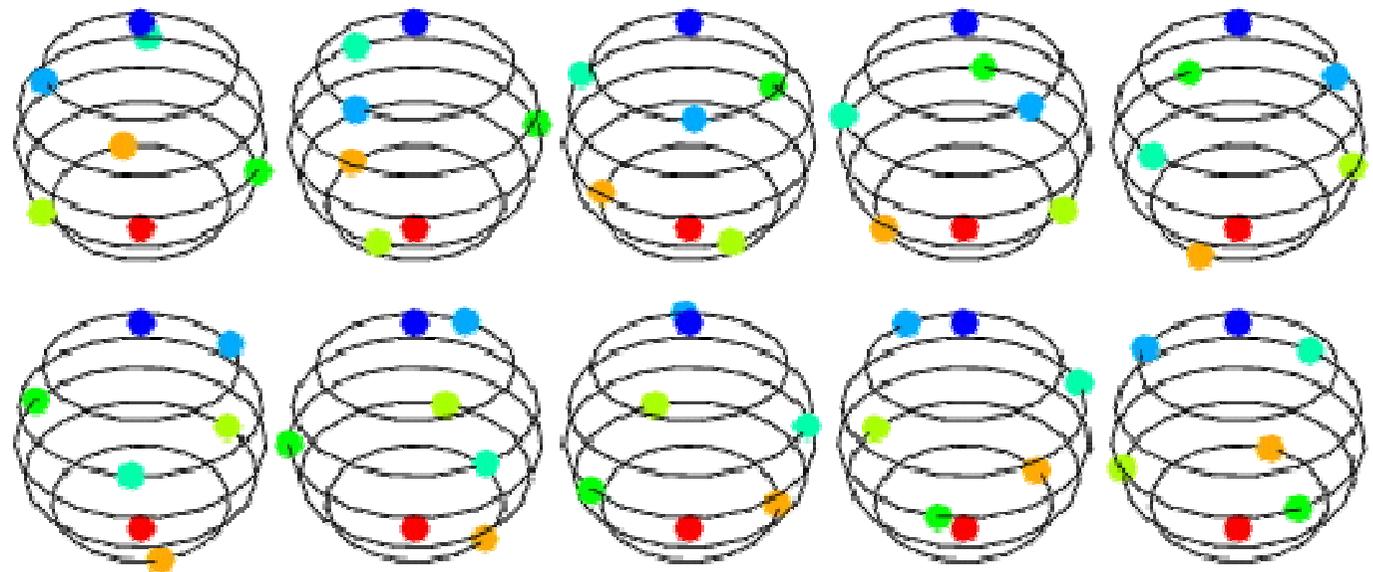
Замечание. Каждая, даже самая простейшая, функция, используемая в алгебре перестановок должна получить в пакете процедурное описание.

02. Процедура-тест `Equal` равенства двух перестановок.

Написать самостоятельно! Какие *перестановки* называются *равными*?

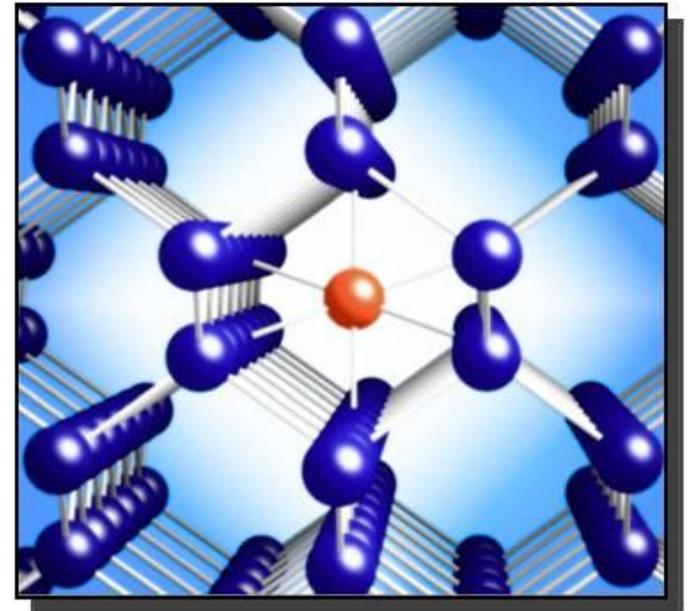
Можно использовать стандартную команду-тест из пакета

LinearAlgebra: функция `Equal(A, B)` возвращает *true*, тогда и только тогда, когда **A=B**. Какие *матрицы* называются *равными*?



Код и примеры.

```
> Pequal := proc (alpha :: PERM, beta :: PERM) ;  
if Equal(  
    alpha [2, 1 .. Pdeg (alpha) ], beta [2, 1 .. Pdeg (beta) ]) then  
    RETURN (true) ;  
else  
    RETURN (false) ;  
end if ;  
end proc ;
```



```
> alpha:=Matrix([[1,2,3,4,5,6,7,8],[3,8,5,4,2,1,6,7]]);  
beta:=Matrix([[1,2,3,4,5,6,7,8],[7,2,4,5,8,1,6,3]]);
```

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

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

```
> Pequal(alpha,beta);
```

false



03. Процедура **Pid**, генерирующая тождественную (единичную) перестановку заданной степени.

```
> Pid:=proc (n::posint) ;
RETURN (Matrix ([ [$1..n], [$1..n] ])) ;
end proc ;
```

Пример.

```
> epsilon:=Pid(8) ; type (epsilon, PERM) ; Pdeg (epsilon) ;
```

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

true

8

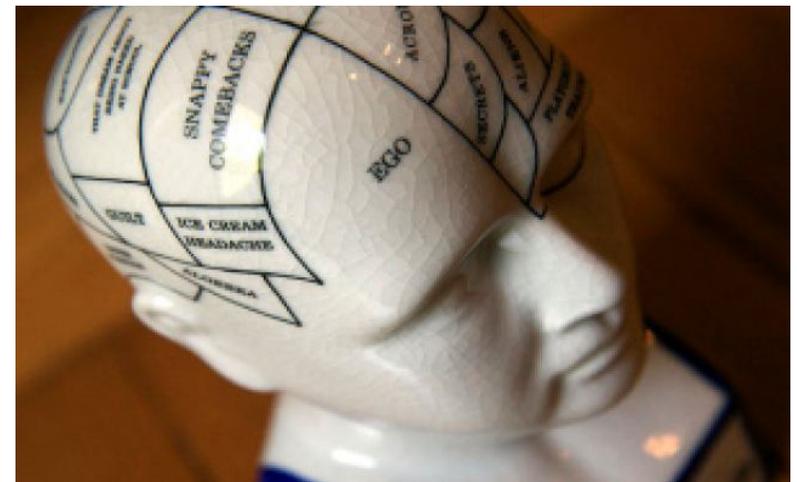
04. Процедура `Pmul` умножения (композиции) двух перестановок.

Начало работы:

```
> Pmul := proc (alpha :: PERM, beta :: PERM)
  local n, i, gamma;
# Внимание! Умножение (композиция) ведется
# справа налево.
```

.....

Продолжить!



Код и примеры.

```
> Pmul := proc (alpha :: PERM, beta :: PERM)
    local n, i, gamma;
n := Pdeg(alpha);
if Pdeg(beta) <> n then
    ERROR();
else
    gamma := Matrix(2, n);
    for i from 1 to n do
        gamma[1, i] := i;
        gamma[2, i] := alpha[2, beta[2, i]];
    end do;
end if;
RETURN(gamma);
end proc;
```

```
> alpha:=Matrix([[1,2,3,4,5,6,7,8],[3,8,5,4,2,1,6,7]]);
beta:=Matrix([[1,2,3,4,5,6,7,8],[7,2,4,5,8,1,6,3]]);
```

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

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

```
> xi:=Pmul(alpha,beta);eta:=Pmul(beta,alpha);
```

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

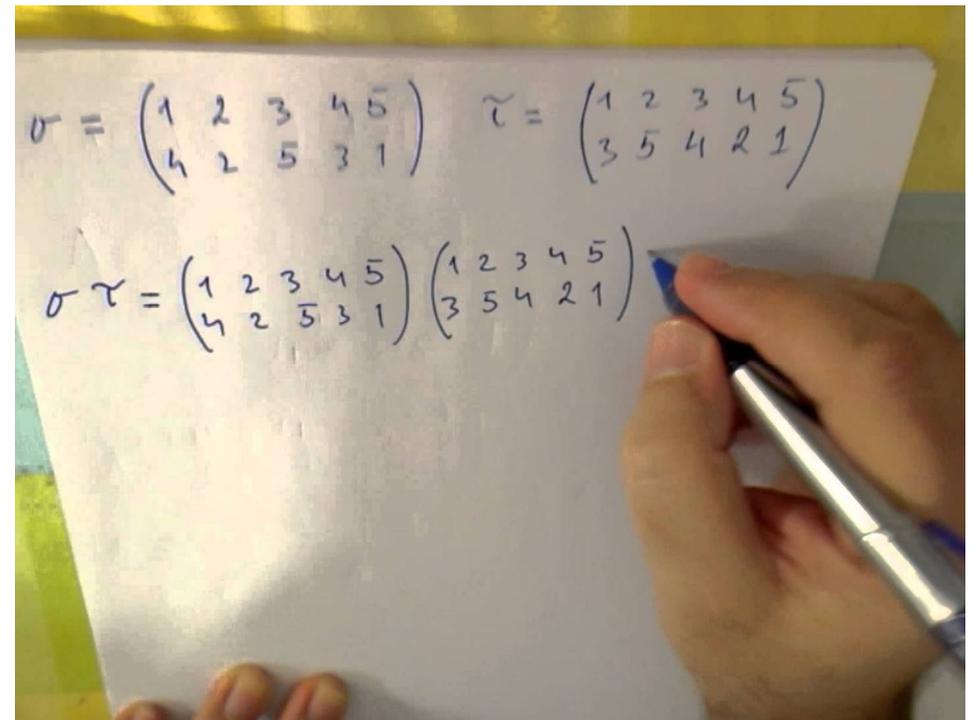
$$\eta := \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 4 & 3 & 8 & 5 & 2 & 7 & 1 & 6 \end{bmatrix}$$

> Pmul (alpha, epsilon) ;

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

> Pequal (alpha, Pmul (alpha, epsilon)) ;

true



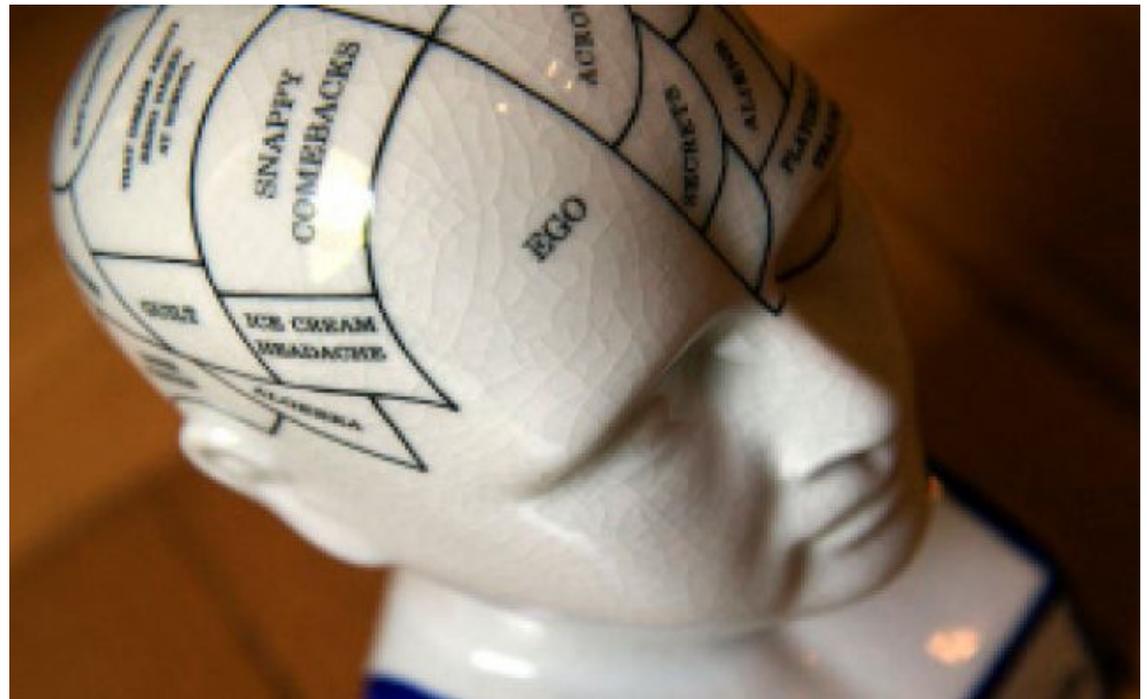
05. Процедура `Pinv` обращения перестановки.

Начало работы:

```
> Pinv:=proc(alpha::PERM)
  local .....
```

.....

Продолжить!



Код и примеры.

```
> Pinv:=proc(alpha::PERM)
  local beta,i,j,n;
n:=Pdeg(alpha);
beta:=Matrix(2,n);
for i from 1 to n do
  beta[1,i]:=i;
  for j from 1 to n do
    if alpha[2,j]=i then
      beta[2,i]:=j;
      break;
    end if;
  end do;
end do;
RETURN(beta);
end proc;
```

```
> xi := Pinv(phi) ;
```

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

```
> Pmul(phi, xi) ; Pequal(% , epsilon) ;
```

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

true

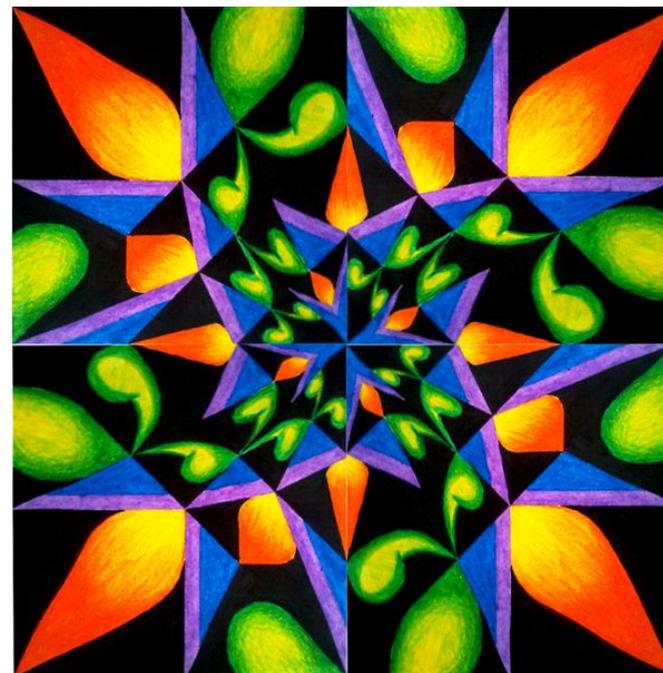


06. Процедура **Ррор** возведения перестановки в целую степень.

Написать самостоятельно!

Указания. Можно сначала определить нулевую, первую и минус первую степени: $\alpha^0, \alpha^1, \alpha^{-1}$; затем, с помощью цикла, - α^k при $k > 1$.

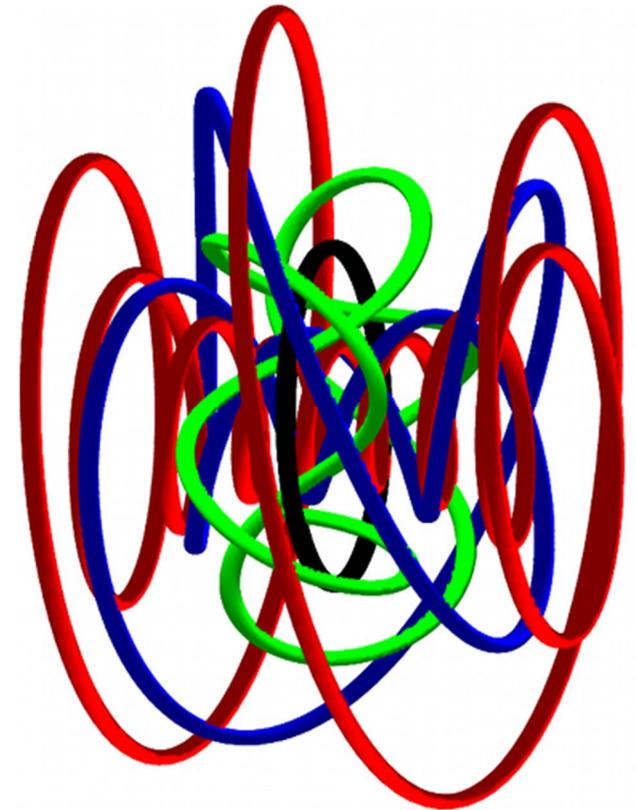
Ваши предложения по определению α^k при $k < -1$?



Код и примеры.

```
> Ppow:=proc(alpha::PERM,k::integer)
  local n,beta,i;
n:=Pdeg(alpha);
if k=0 then
  RETURN(Pid(n));
elif k=1 then
  RETURN(alpha);
elif k=-1 then
  RETURN(Pinv(alpha));
elif k>1 then
  beta:=alpha;
  for i from 1 to k-1 do
    beta:=Pmul(beta,alpha);
  end do;
  RETURN(beta);
```

```
elif k<-1 then
  beta:=Pinv(alpha);
  for i from 1 to abs(k)-1 do
    beta:=Pmul(beta,Pinv(alpha));
  end do;
  RETURN(beta);
end if;
end proc;
```



```
> rho := Ppow(alpha, 5);  
sigma := Ppow(alpha, -5);  
Pmul(rho, sigma);
```

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

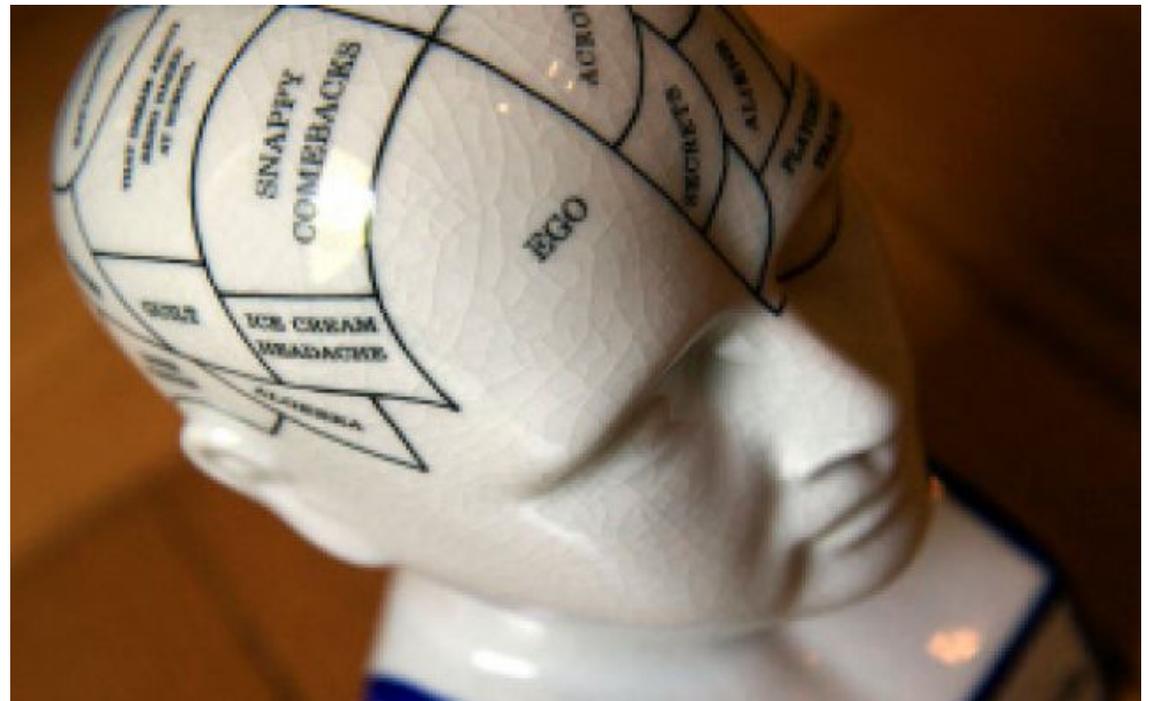
$$\sigma := \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 5 & 7 & 2 & 4 & 8 & 3 & 1 & 6 \end{bmatrix}$$

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

07. Процедура **Pord** вычисления порядка перестановки.

Написать самостоятельно!

Указание. Вспомнить определение *порядка* перестановки.



Определение (в произвольной мультипликативной группе $(G; \cdot, e)$):

$$\mathbf{o(a) = \min\{k \in \mathbb{N} : a^k = e\}.$$

Из *теоремы Лагранжа* вытекает: порядок любого элемента делит порядок группы: $\mathbf{o(a) \mid |G|}$.

Определение (в симметрической группе $(S_n; \circ, \varepsilon)$):

$$\mathbf{o(\alpha) = \min\{k \in \mathbb{N} : \alpha^k = \varepsilon\}.$$

Из *теоремы Лагранжа*: $\mathbf{o(\alpha) \mid n!}$.

Код и примеры.

```
> Pord:=proc(alpha::PERM)
  local n,k;
n:=Pdeg(alpha);
k:=1;
while not Pequal(Ppow(alpha,k),Pid(n)) do
  k:=k+1;
end do;
RETURN(k);
end proc;
```



> **Pord(epsilon) ;**

1

Ранее были введены или вычислены перестановки:

> **alpha, beta, xi, eta, phi ;**

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

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

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

> **map(Pord, [alpha, beta, xi, eta, phi]) ;**

[7, 12, 4, 8, 4]

08. Процедура **Porb** вычисления орбиты (цикла) элемента под действием перестановки.

Всё множество $X = \{1, 2, \dots, n\}$ действием перестановки $\varphi \in S_n$ (и ее степеней) разбивается на попарно не пересекающиеся классы эквивалентности (орбиты).

Определение. *Орбитой* элемента $i \in X$ под действием перестановки φ называется множество элементов

$$\mathbf{orb}_{\varphi}(i) = \{\varphi^k(i) : k \in \mathbb{N}\}.$$

Известно, что значения $\varphi^k(i)$ *периодически повторяются*, причем *период* их повторения *делит* $\mathbf{o}(\varphi)$ (*порядок* данной перестановки).

Мы будем искать орбиты в упорядоченном (по степеням k) виде, т. е. как *списки* вида:

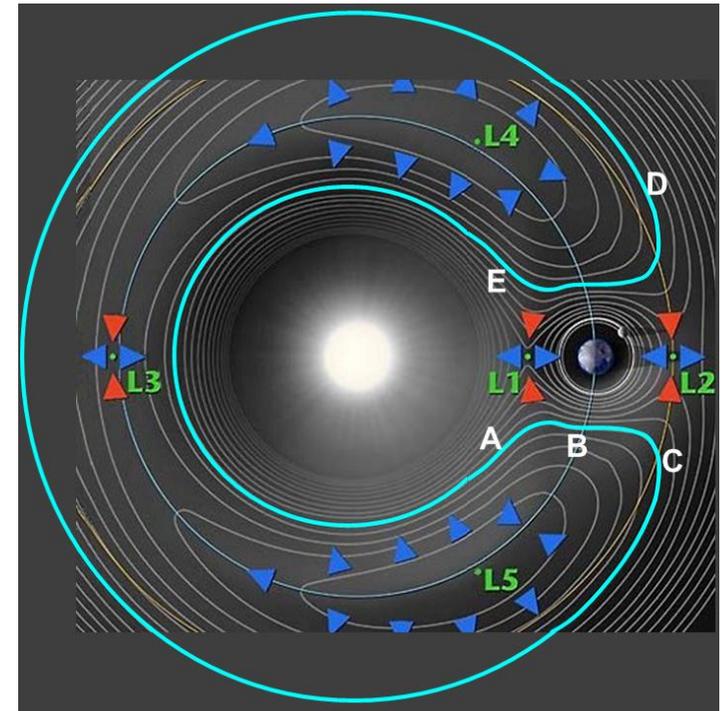
$$[i, \varphi(i), \varphi^2(i), \dots, \varphi^{s-1}(i)],$$

где s – *длина* (мощность) *орбиты* (цикла), т. е. *период* "заикливания": $\varphi^s(i) = i$; все $\varphi^k(i)$ при $0 \leq k \leq s - 1$ попарно различны.

Код и примеры.

```
> Porb:=proc(i::posint,phi::PERM)
  local n,orb;
n:=Pdeg(phi);
if not i in [$1..n] then
  ERROR();
else
  orb:=[i];
  while phi[2,orb[nops(orb)]]<>i do
    orb:=[orb[],phi[2,orb[nops(orb)]]];
```

```
# nops(orb) - длина текущего участка орбиты;  
# orb[nops(orb)] - последний найденный элемент;  
# если значение phi[2,orb[nops(orb)]]  
# перестановки phi на этом элементе  
# все еще отлично от i, (т. е. если пока  
# не произошло зацикливания), то указанное  
# значение приписывается к списку.
```



```

end do;
RETURN (orb) ;
end if;
end proc;

```

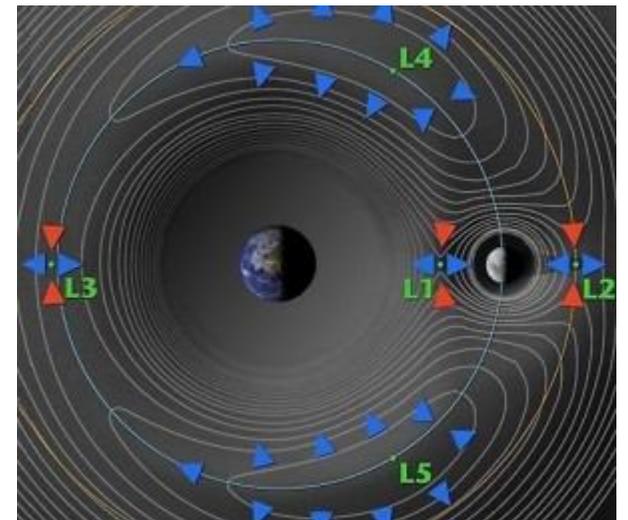
```
> Porb (1 , alpha) ; Porb (2 , alpha) ; Porb (4 , alpha) ;
```

```
[1, 3, 5, 2, 8, 7, 6]
```

```
[2, 8, 7, 6, 1, 3, 5]
```

```
[4]
```

Заметьте, что орбиты **1** и **2** отличаются *как списки*, но не отличаются *как множества*. Фактически это — одна и та же орбита!



09. Процедура **Pdecomp** разложения (декомпозиции) перестановки в произведение независимых циклов.

Если $C = \{i_1, i_2, \dots, i_s\}$ - некоторое (произвольным образом упорядоченное) подмножество в множестве $X = \{1, 2, \dots, n\}$, то символом (i_1, i_2, \dots, i_s) обозначается перестановка из группы S_n , циклически переставляющая элементы C и не перемещающая элементы дополнения $X \setminus C$.

Перестановки такого вида называются (частичными) *циклами*.

Циклы называются *независимыми*, если их области действия не пересекаются (*дизъюнкты*).

Дизъюнктные циклы *коммутируют*.

Циклы (i_1) длины единица тривиальны (совпадают с тождественной перестановкой) и могут быть выброшены из разложения.

Запись цикла $\sigma = (i_1, i_2, \dots, i_{s-1}, i_s)$ может быть начата с любого элемента цикла, например: $\sigma = (i_2, i_3, \dots, i_s, i_1)$.

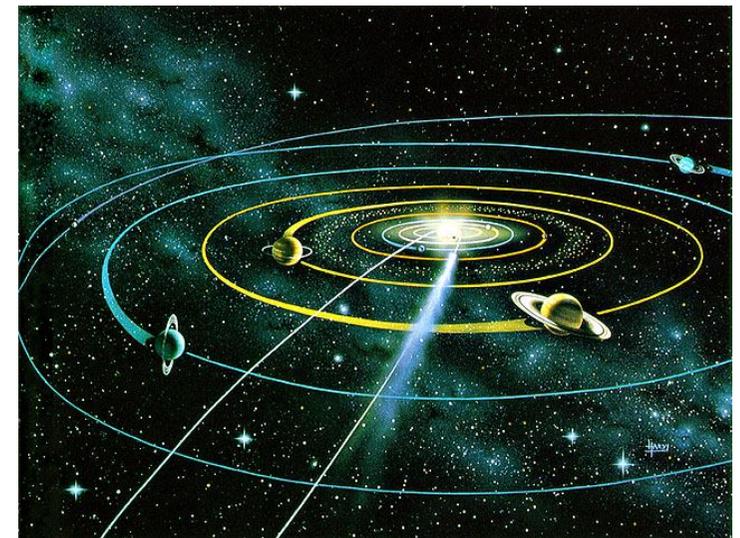
Теорема 2. *Всякая перестановка однозначно (с точностью до порядка сомножителей) может быть разложена в произведение независимых циклов.*

Код и примеры.

Разложение ищется в виде *списка списков* **LL**, элементы которого (списки) можно трактовать как *орбиты* данной перестановки или же как *циклы*, на которые эта перестановка разлагается.

```
> Pdecomp:=proc (phi :: PERM)
  local n, cumm, LL, rest, j;
n:=Pdeg (phi) ;
LL:=[Porb (1, phi) ] ;
cumm:=convert (Porb (1, phi) , set) ;
# Первая орбита находится с помощью функции Porb
# и заносится в список списков LL;
# одновременно начинается накопление
# "отработанных" элементов в множестве cumm.
```

```
while not {$1..n} minus cumm = {} do
# Пока дополнение множества cumm не пусто
# конвертируем его в список rest и сортируем,
# после чего выбираем первый элемент дополнения
# и начинаем новый цикл.
  rest:=sort(convert({$1..n} minus cumm,list));
  j:=rest[1];
  LL:=[LL[],Porb(j,phi)];
  cumm:=cumm union convert(Porb(j,phi),set);
end do;
RETURN(LL);
end proc;
```



> **alpha,beta,xi,eta,phi;**

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

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

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

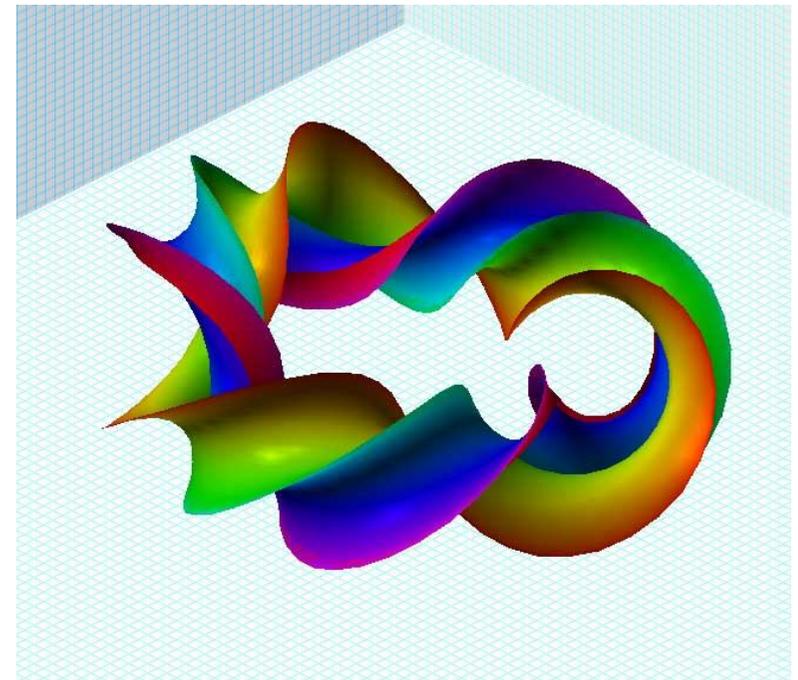
> **map(Pdecomp, [alpha,beta,xi,eta,phi,epsilon]);**

```
[[[1, 3, 5, 2, 8, 7, 6], [4]], [[1, 7, 6], [2], [3, 4, 5, 8]], [[1, 6, 7, 2], [3, 5], [4], [8]],
[[1, 4, 5, 2, 3, 8, 6, 7]], [[1, 2, 7, 6], [3, 5], [4], [8]],
[[1], [2], [3], [4], [5], [6], [7], [8]]]
```

10. Процедура **Pdecompred** разложения перестановки в произведение нетривиальных циклов .

Написать самостоятельно!

Указание. Выбросить в списке списков, который возвращается предыдущей процедурой тривиальные циклы.



Код и примеры.

Применим функцию `remove (a->f (a) , L)` удаления из списка **L** всех элементов, для которых принимает значение **true** указанная *булевозначная* функция `a->f (a)`.

```
> Pdecompred:=proc (alpha::PERM) ;
RETURN (remove (a->evalb (nops (a)=1) , Pdecomp (alpha) ) ) ;
end proc;
```

```
> map (Pdecompred, [alpha,beta,xi,eta,phi,epsilon] ) ;
```

```
[[[1, 3, 5, 2, 8, 7, 6]], [[1, 7, 6], [3, 4, 5, 8]], [[1, 6, 7, 2], [3, 5]],
[[1, 4, 5, 2, 3, 8, 6, 7]], [[1, 2, 7, 6], [3, 5]], [ ]]
```

11. Определение типа `DisjCyc` (`DisjointCycles`).

К типу `DisjCyc` мы будем относить

- списки списков натуральных чисел,*
члены которых (*списки натуральных чисел*)
- (1) *не содержат повторений;*
 - (2) попарно *не имеют общих элементов.*

Сначала определим *процедуру-тест DC*, проверяющую выполнение всех перечисленных выше условий.

Замечание. В `Maple`-пакете `group` также предусмотрен "рабочий" тип `'disjcy'`, аналогичный нашему `DisjCyc`.

```
> DC:=proc(a)
  local n,i,j;
n:=nops(a);
if not type(a,list(list(posint))) then
  # Проверяется первое условие (на тип).
  RETURN(false);
else
  for i from 1 to n do
    if nops(a[i])<>nops(convert(a[i],set)) then
      # Проверяется второе условие: нет повторений.
      RETURN(false);
    end if;;
  end do;
end do;
```

```
for i from 1 to n do
  for j from i+1 to n do
    # Проверяется третье условие: нет пересечений.
    if convert(a[i],set) intersect
      convert(a[j],set)<>{} then
      RETURN(false);
    end if;
  end do;
end do;
RETURN(true);
end if;
end proc;
```



Примеры на процедуру.

```
> DC ([[1, -2], [3]]),
      DC ([[1, 2, 4], [5, 3, 3]]),
      DC ([[1, 2, 4], [5, 2, 3]]),
      DC ([[2, 3], [5, 1, 4], [7, 8], [9]]);
```

false, false, false, true

Разберитесь, какое из условий нарушается в каждом из первых трех примеров.

Располагая процедурой-тестом **DC** определим новый тип **DisjCyc** (см. выше определение типа **PERM**).

Без этого можно и обойтись, но с типами работать удобнее.

> **TypeTools [AddType] (DisjCyc, DC) ;**

Примеры. Процедуры **Pdecomp** и **Pdecompred** фактически *конвертируют* объекты типа **PERM** в объекты типа **DisjCyc**.

> **sigma :=**
Matrix ([[1, 2, 3, 4, 5, 6, 7, 8, 9, 10], [1, 10, 4, 3, 2, 6, 9, 7, 8, 5]]) ;

$$\sigma := \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 10 & 4 & 3 & 2 & 6 & 9 & 7 & 8 & 5 \end{bmatrix}$$

> **type (sigma, PERM) ;**

true

> **L1 := Pdecomp (sigma) ; L2 := Pdecompred (sigma) ;**

L1 := [[1], [2, 10, 5], [3, 4], [6], [7, 9, 8]]

L2 := [[2, 10, 5], [3, 4], [7, 9, 8]]

> **type (L1, DisjCyc) , type (L2, DisjCyc) ;**

true, true

11. Процедура конвертации списка типа **DisjCyc** в перестановку.

Необходима процедура, решающая *обратную* задачу конвертации типа **DisjCyc** в тип **PERM**.

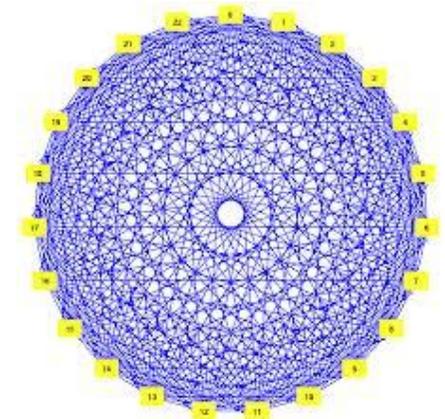
Ясно, что для *однозначности восстановления* перестановки **sigma** по ее разложению на циклы **L** (с, возможно, выброшенными *тривиальными* циклами) необходимо указывать *степень* этой перестановки, причем значение степени не должно быть меньше, чем *наибольшее* из натуральных чисел, входящих в циклы.

Комментированный код и пример.

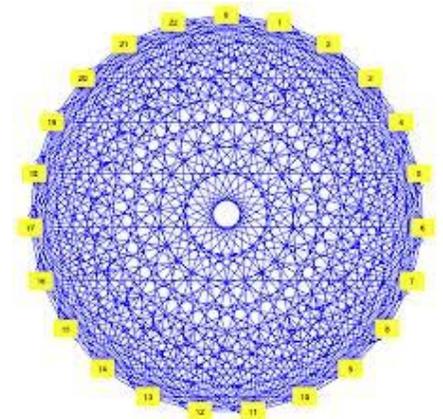
```
> DisjCyc_to_PERM:=proc(L::DisjCyc,n::posint)
  local s,elems,M,sigma,i,j,k,t;
s:=nops(L);
elems:={};
for i from 1 to s do
  elems:=elems union convert(L[i],set);
end do;
elems:=sort(elems);
M:=elems[nops(elems)];
# Найдено множество elems всех натуральных чисел,
# встречающихся в списке списков L,
# а также наибольшее из них - число M.
```

```
if M>n then
  # Обработка ошибочной ситуации: число n недостаточно
  # для того, чтобы быть степенью искомой перестановки.
  ERROR();
else
  sigma:=Matrix(2,n);
  for i from 1 to n do
    sigma[1,i]:=i;
    # Заготовка для перестановки sigma
    # (заполнена первая строка).
    if not i in elems then
      sigma[2,i]:=i;
      # Если номер i отсутствует в множестве elems,
      # то sigma переводит i в i.
```

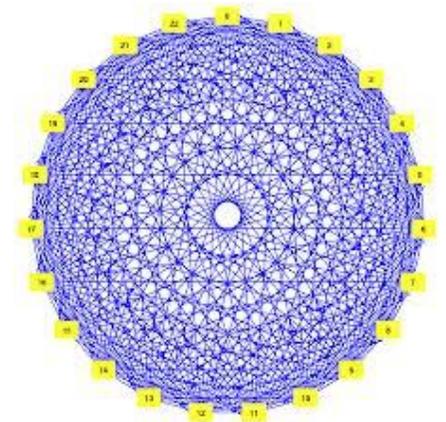
```
else
  for j from 1 to s do
    # Определяем, в какой из списков (циклов) L[j]
    # попадает номер i.
    k:=nops(L[j]);
    if i in L[j] and k=1 then
      sigma[2,i]:=i;
      # Если номер i попал в тривиальный
      # цикл L[j], то под действием sigma
      # он переходит сам в себя.
      break;
```



```
elif i in L[j] and k>1 then
  t:=ListTools[Search](i,L[j]);
  if t<k then
    # Если номер i попал в нетривиальный
    # цикл L[j], под номером t, причем -
    # не последним, то под действием sigma
    # он переходит в элемент цикла L[j]
    # с номером t+1.
    sigma[2,i]:=L[j][t+1];
    break;
```



```
elif t=k then
# Если номер i попал в L[j]
# под последним номером, то
# под действием sigma он переходит
# в первый элемент этого цикла.
    sigma[2,i]:=L[j][1];
    break;
end if;
end if;
end do;
end if
end do;
end if;
RETURN(sigma);
end proc;
```



```

> L2;
  tau:=DisjCyc_to_PERM(L2,10);
  Pequal(tau,sigma);
  DisjCyc_to_PERM(L2,15);

```

$$L2 := [[2, 10, 5], [3, 4], [7, 9, 8]]$$

$$\tau := \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 10 & 4 & 3 & 2 & 6 & 9 & 7 & 8 & 5 \end{bmatrix}$$

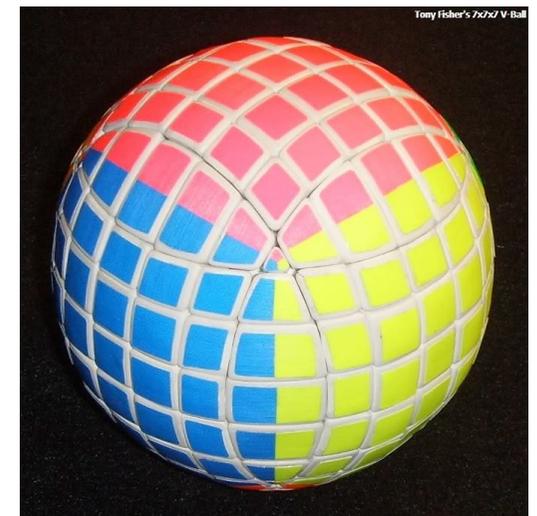
true

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 1 & 10 & 4 & 3 & 2 & 6 & 9 & 7 & 8 & 5 & 11 & 12 & 13 & 14 & 15 \end{bmatrix}$$

12. Процедура `Pdecomp` разложения перестановки в произведение транспозиций.

Написать самостоятельно!

Указание. Транспозиции – это циклы *длины два*. Всякий нетривиальный цикл (длины s) можно разложить в произведение $s - 1$ транспозиций. Вспомните соответствующую формулу. Процедура должна возвращать список списков. Помните также, что, в отличие от разложения на независимые циклы, разложение на транспозиции определено *не однозначно*.



13. Процедура **Pdesc** вычисления декремента перестановки.

Написать самостоятельно!

Указание. Вспомнить определение *декремента*.



Код и примеры.

```
> Pdescr:=proc(phi::PERM)
  local n,m;
n:=Pdeg(phi);
m:=nops(Pdecomp(phi));
# Число (всех) независимых циклов в разложении.
RETURN(n-m);
end proc;

> map(Pdescr,[alpha,beta,xi,eta,phi]);
      [6, 5, 4, 7, 4]
```

14. Процедура Psign вычисления знака перестановки.

Вспомним три способа определения знака:

(1) с помощью разложения перестановки на *транспозиции*;

(2) с помощью *декремента*;

(3) с помощью подсчета количества *инверсий* (беспорядков)

в перестановке.

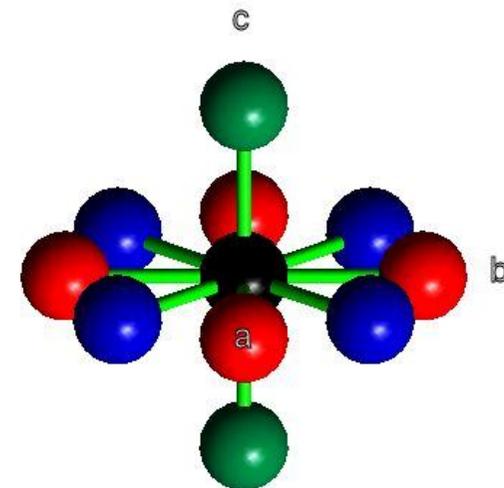
В каждом из способов -1 возводится в степень (равную количеству транспозиции, декременту или количеству инверсий соответственно).

*Написать процедуру (на основе **второго** способа) самостоятельно!*

Код и примеры.

```
> Psign:=proc(phi::PERM) ;  
RETURN((-1)^Pdecr(phi)) ;  
end proc;
```

```
> map(Psign, [alpha, beta, xi, eta, phi]) ;  
[1, -1, 1, -1, 1]
```



15. Процедура **Pdisord** вычисления количества инверсий в перестановке.

Инверсия (беспорядок, **disorder**) в перестановке φ - это пара номеров i, j ($i < j$) таких, что $\varphi(i) > \varphi(j)$.

Написать процедуру самостоятельно!

