xAct`xPerm`

This is the doc file xPermDoc.nb of version 1.0.1 of `xPerm``. Last update on 14 May 2008.

Author

José M. Martín−García

Instituto de Estructura de la Materia, CSIC, Spain

`jmm@iem.cfmac.csic.es`

`http://metric.iem.csic.es/Martin-Garcia/`

`xPerm`` is a scientific project, and has been published in the article

José M. Martín−García, "*xPerm*: fast index canonicalization for tensor computer algebra", submitted to Comp. Phys. Commun. 2008.

Users of `xPerm`` are requested to cite that article in any work in which the package is used. If you think your investigation would have been impossible without `xPerm``, please consider having me as an author of your publication. `xPerm`` is many hundred hours (coding, debugging, testing, documenting) of my time spent to help you.

Intro

`xPerm‘` extends *Mathematica* capabilities in computations with large groups of permutations. This package can be used independently, but it has been created as a tool for its twin package `xTensor‘` of abstract tensor manipulations. There are two main objectives in the package:

1) Computation of a Strong Generating Set (SGS) for a group of permutations, given a generating set (GS) for that group. We use the Schreier−Sims algorithm, as encoded by Butler:

G. Butler, "Fundamental Algorithms for Permutation Groups", Springer−Verlag, Berlin Heidelberg 1991.

The companion notebook `ButlerExamples.nb` contains the solutions (worked out with `xPerm‘`) of many examples in that book. Some of them have wrong answers in the book and have been corrected.

2) Computation of canonical representatives of cosets and double cosets. We use Portugal’s algorithms:

R. Portugal and B.F. Svaiter, "Group−theoretic Approach for Symbolic Tensor Manipulation: I. Free indices", math−ph/0107031

L.R.U. Manssur and R. Portugal, "Group−theoretic Approach for Symbolic Tensor Manipulation: II. Dummy indices", math−ph/0107032

L.R.U. Manssur, R. Portugal and B.F. Svaiter, "Group−theoretic Approach for Symbolic Tensor Manipulation", Int. J. Mod. Phys. C13 (2002), 859−−880.

`xPerm‘` is not, and will never be, a serious alternative to professional packages like GAP or MAGMA for algebraic computations. It just solves those two issues needed in `xTensor‘`, plus adding some commodity tools.

`xPerm‘` is directly coded up in *Mathematica* language, which is interpreted and hence rather slow. Executables called xperm.linux, xperm.mac or xperm.win (depending on your system) are given to improve significantly the speed of the hardest computations. A *MathLink* connection is set up to communicate MathKernel with that executable. Note that the Windows executable can only be linked with *Mathematica* versions 6.0 or later because the cygwin compilers were not supported by *Mathematica* in previous versions.

I thank

− Paolo Matteucci for suggesting the use of Portugal’s algorithms to do index canonicalization.

− Alfonso García−Parrado, Kasper Peeters, Renato Portugal and David Yllanes for helpful comments and suggestions, and for their help in testing the system.

Load the package

This loads the package from the default directory `$HOME/.Mathematica/Applications/xAct` for a single−user implementation under Linux, or from `/user/local/Wolfram/Mathematica/5.2/AddOns/Applications/xAct` for a system−wide installation. For installation under Windows or Mac see the Installation Notes.

```
In[1]:=  MemoryInUse[]

Out[1]=  3059504
```

```
In[2]:= <<xAct‘xPerm‘

            ------------------------------------------------------------------------------
              --

            Package xAct‘xCore‘  version 0.5.0, {2008, 5, 16}

            CopyRight (C) 2007-2008, Jose M.
              Martin-Garcia, under the General Public License.

            ------------------------------------------------------------------------------
              --

            Package ExpressionManipulation‘

            CopyRight (C) 1999-2008, David J. M. Park and Ted Ersek

            ------------------------------------------------------------------------------
              --

            Package xAct‘xPerm‘  version 1.0.1, {2008, 5, 16}

            CopyRight (C) 2003-2008, Jose M.
              Martin-Garcia, under the General Public License.

            ------------------------------------------------------------------------------
              --

            These packages come with ABSOLUTELY NO WARRANTY; for details type
              Disclaimer[]. This is free software, and you are welcome to redistribute
              it under certain conditions. See the General Public License for details.

            ------------------------------------------------------------------------------
              --

            Connecting to external linux executable...

            Connection established.
```

Comparing, we see that the packages take around 10Mbytes, most of which comes from `xCore‘`:

```
In[3]:= MemoryInUse[]

Out[3]= 13092920

In[4]:= Out[3] - Out[1]

Out[4]= 10033416
```

Note the structure of the ContextPath. There are several contexts: `xAct‘xPerm‘` contains the new reserved words. `xAct‘xCore‘` contains my added *Mathematica* functions. `xAct‘ExpressionManipulation‘` is an external package loaded to perform additional expression manipulations. `System‘` contains *Mathematica*’s reserved words. The current context `Global‘` will contain your definitions (it is now empty).

```
In[5]:= $ContextPath

Out[5]= {xAct‘xPerm‘, xAct‘xCore‘, xAct‘ExpressionManipulation‘, Global‘, System‘}

In[6]:= Context[]

Out[6]= Global‘
```

```
In[7]:= ?Global‘*

        Information::nomatch : No symbol matching Global‘* found. More…
```

We append the Doc directory of `xAct‘` to the file path to find example files below:

```
In[8]:= AppendTo[$Path, $xActDocDirectory];
```

The *Mathematica* built−in `Timing` does not include the time spent in the external executables, though `AbsoluteTiming` does it. We introduce our own timing function:

```
In[9]:= SetAttributes[myTiming, HoldFirst];
        myTiming[expr_] := Module[{time, result},
          time = SessionTime[];
          result = expr;
          Print[(SessionTime[] - time) Second];
          result]
```

Technical notes:

There are three global variables storing information on the connection to the external executable:

Whether the connection was possible or not is stored in

```
In[11]:= $xpermQ

Out[11]= True
```

The name of the executable is given by

```
In[12]:= $xpermExecutable

Out[12]= /home/jmm/.Mathematica/Applications/xAct/xPerm/mathlink/xperm.linux
```

The full description of the link is

```
In[13]:= $xpermLink

Out[13]= LinkObject[
          /home/jmm/.Mathematica/Applications/xAct/xPerm/mathlink/xperm.linux, 2, 2]
```

Basic group theory

■ 0. Mathematical notes

This is a very brief (and highly compact) recollection of definitions and results in permutation−group theory, taken from Butler 1991.

− A permutation group G acts on a set of points P on the right.

− The image of p in P under g in G is denoted p^g and (p^g)^h = p^(gh) for all p, g and h.

− The orbit of p under G is the set p^G={p^g | g in G} of images of p.

− The pointwise stabilizer of p in G is the group G_p={g in G | p^g=p} of elements that fix p.

− A base for G is a sequence B={b1, b2, ..., bk} of points such that only the identity element of G fixes all points in the base.

− Associated with a base there is a chain of subgroup stabilizers G^(i), i=1, 2, ..., k+1, where G^(i)=G_{b1, b2, ..., b_(i−1)}.

− A subset S of G is a strong generating set of G relative to B if S contains a generating set for each stabilizer G^(i) in the chain.

− If we define a total order on P, then there is an induced lexicographical order on G and on the base images B^G. If we insist that b1, b2, ..., bk are the first k points of P then the order on G corresponds to the order on the base images. In any case, the identity element is the smallest element of the group.

− Given a group G and a stabilizer G_b there is a one−to−one correspondence between the cosets of G_b in G and the points of the orbit b^G. Using the order of permutations defined in the previous point we can choose a representa− tive in each of those cosets.

− An element g of G is uniquely determined by its base image B^g = {b1^g, b2^g, ..., bk^g}. Actually there is a unique decomposition of g as a product of the representatives of the cosets identified by those images.

− This equivalence between permutations and base images is the key for a plethora of highly efficient algorithms which can answer any question on the group by using backtracking through the hierarchy of cosets and stabilizers.

■ 1. Notations for permutations

There are many ways to encode a permutation of several objects. This package uses four of them. For example suppose that the collection of six objects

♡ ⌚ □ ○ ☺ △

is reordered to

□ ⌚ ○ ♡ △ ☺

1) We can say that the first object (a heart) has gone to the fourth place (it is conventional to say that the "image" of "point" 1 is point 4), and so on. If we number the initial objects from 1 to 6, one natural way of encoding this permutation would be

```
Perm[{3, 2, 4, 1, 6, 5}]
```

Note that the point 2 is not moved by the permutation. It is called a "singleton".

2) Another natural way of encoding the permutation would be just listing the images of the six points:

```
Images[{4, 2, 1, 3, 6, 5}]
```

3) It is well known that any permutation can be understood as a set of disjoint cycles where the image of each point is written on its right and the image of the last point of the cycle is the first one:

```
Cycles[{1, 4, 3}, {5, 6}]
```

By convention, singleton cycles like {2} are removed from the cyclic notation.

4) Finally, it is particularly efficient in *Mathematica* to represent permutations using rules:

```
Rules[1->4, 4->3, 3->1, 5->6, 6->5]
```

Again, note that the singleton has not been included.

`Perm[list]`	`Images[list]`
`Cycles[cycles]`	`Rules[rules]`

Notations for permutations.

Naturally, permutations form a group under the law of multiplication. For example all permutations of n points form the so−called symmetric group S_n. However there is another important superstructure that will be very useful when working with tensor symmetries: an algebra. We shall define laws of addition of permutations and of multiplication by scalars. At the moment we only assume that permutations can be given a sign (−1 or +1). It that sense, following R. Portugal, we are working on $\{-1,1\}\times S_n$. An example of signed permutation is:

```
- Perm[{2, 3, 4, 1}]
```

We can translate from one notation to other notation:

```
In[14]:= TranslatePerm[Perm[{3, 2, 4, 1, 6, 5}], Images]

Out[14]= Images[{4, 2, 1, 3, 6, 5}]

In[15]:= TranslatePerm[%, Cycles]

Out[15]= Cycles[{1, 4, 3}, {5, 6}]
```

```
In[16]:= TranslatePerm[%, Rules]

Out[16]= Rules[1 → 4, 4 → 3, 3 → 1, 5 → 6, 6 → 5]

In[17]:= TranslatePerm[%, Perm]

Out[17]= Perm[{3, 2, 4, 1, 6, 5}]
```

The sign just remains untouched:

```
In[18]:= TranslatePerm[-Perm[{3, 2, 4, 1, 6, 5}], Images]

Out[18]= -Images[{4, 2, 1, 3, 6, 5}]
```

The notation of a permutation is given by

```
In[19]:= NotationOfPerm[Cycles[{1, 4, 3}, {5, 6}]]

Out[19]= Cycles
```

`TranslatePerm`	Change permutation to a given notation
`NotationOfPerm`	Returns the notation of a permutation

Translation and notation of permutations.

2. Operations with permutations

By default, `xPerm'` does not check at every step that the objects we are working with are actually permutations because it would take quite some time. The function `PermQ` is supplied to do that when you require it.

We can validate a permutation with `PermQ`:

```
In[20]:= PermQ[Perm[{1, 2, 3, 0}]]

Out[20]= False

In[21]:= PermQ[Perm[{1, 4, 3, 2}]]

Out[21]= True

In[22]:= PermQ[Images[{1, 2, 3, 1}]]

Out[22]= False

In[23]:= PermQ[Cycles[{1, 2.}, {4, 3}]]

Out[23]= False

In[24]:= PermQ[Rules[1 → 2, 1 → 3, 2 → 3]]

Out[24]= False
```

```
In[25]:=  PermQ[-Cycles[{2, 3}, {4, 5}]]

Out[25]=  True
```

Every permutation has a degree, defined as the largest moved point (signs are not relevant):

```
In[26]:=  PermDeg[Perm[{4, 3, 5, 2, 1}]]

Out[26]=  5

In[27]:=  PermDeg[-Images[{3, 4, 5, 6, 2, 1}]]

Out[27]=  6

In[28]:=  PermDeg[Cycles[{2, 3, 1}, {55, 78}]]

Out[28]=  78

In[29]:=  PermDeg[Perm[{2, 1, 3, 4, 5}]]

Out[29]=  2
```

The identity permutation has degree zero:

```
In[30]:=  PermDeg[Cycles[]]

Out[30]=  0

In[31]:=  PermDeg[Perm[{1, 2, 3, 4, 5}]]

Out[31]=  0
```

A different concept is that of "length" of a permutation, defined only for notations `Perm` and `Images`: it is the length of the list of points. On `Cycles` and `Rules` notations length is defined as the degree. This concept is used in internal computations to avoid recomputing the degree of a permutation many times.

```
In[32]:=  PermDeg[Perm[{2, 1, 3, 4, 5}]]

Out[32]=  2

In[33]:=  PermLength[Perm[{2, 1, 3, 4, 5}]]

Out[33]=  5
```

On notations `Perm` and `Images` the function `NotationOfPerm` returns the length (not the degree):

```
In[34]:=  NotationOfPerm[Perm[{2, 1, 3, 4, 5}]]

Out[34]=  {Perm, 5}
```

It is possible to generate random permutations, of any notation

```
In[35]:= RandomPerm[10, Cycles]

Out[35]= Cycles[{1, 3, 10, 8, 5, 2, 4, 7}, {6, 9}]

In[36]:= RandomPerm[8, Images]

Out[36]= Images[{7, 1, 8, 5, 4, 3, 2, 6}]
```

The most important operation with permutations is that of their product. I don’t like names like Times or similar. I choose a name which is associated only with permutations:

Product of two permutations in the same notation. The (noncommutative) product is from right to left. This means that the product permutation `Permute[p1, p2]` applied on a given list is equivalent to the application of p1 on that list and then p2 on the result.

```
In[37]:= Permute[Perm[{3, 2, 1}], Perm[{2, 3, 1}]]

Out[37]= Perm[{2, 1, 3}]

In[38]:= Permute[Perm[{2, 3, 1}], Perm[{3, 2, 1}]]

Out[38]= Perm[{1, 3, 2}]
```

That is the choice in GAP. This example is taken from its manual:

```
In[39]:= PermEqual[Permute[Cycles[{1, 2, 3}], Cycles[{1, 2}]], Cycles[{2, 3}]]

Out[39]= True

In[40]:= PermEqual[Permute[Cycles[{1, 2}], Cycles[{1, 2, 3}]], Cycles[{1, 3}]]

Out[40]= True
```

Inverse permutation:

```
In[41]:= InversePerm[Perm[{4, 3, 1, 2}]]

Out[41]= Perm[{3, 4, 2, 1}]

In[42]:= Permute[Perm[{4, 3, 1, 2}], %]

Out[42]= Perm[{1, 2, 3, 4}]
```

Powers of a permutation (equivalent to repeated product with itself). Negative exponents mean powers of the inverse. Note the high efficiency of this function:

```
In[43]:= myTiming[PowerPermute[Perm[{5, 1, 2, 3, 4}], 27]]

            0.000548 Second

Out[43]= Perm[{4, 5, 1, 2, 3}]
```

```
In[44]:=  myTiming[PowerPermute[Perm[{5, 1, 2, 3, 4}], -10^10 - 27]]

            0.002646 Second

Out[44]=  Perm[{3, 4, 5, 1, 2}]

In[45]:=  Permute[%, %%]

Out[45]=  Perm[{1, 2, 3, 4, 5}]

In[46]:=  PowerPermute[Perm[{5, 1, 2, 3, 4}], 0]

Out[46]=  Perm[{1, 2, 3, 4, 5}]
```

Permutations can be sorted and compared according to their lists of images:

```
In[47]:=  PermSort[{Perm[{2, 3, 1}], Perm[{3, 1, 2}], Perm[{1, 2, 3}]}]

Out[47]=  {Perm[{1, 2, 3}], Perm[{3, 1, 2}], Perm[{2, 3, 1}]}

In[48]:=  TranslatePerm[%, Images]

Out[48]=  {Images[{1, 2, 3}], Images[{2, 3, 1}], Images[{3, 1, 2}]}

In[49]:=  PermLess[Perm[{3, 1, 2}], Perm[{2, 3, 1}]]

Out[49]=  True
```

The function Perm`Equal` checks mathematical equality of permutations comparing images of points and it is thus notation independent:

```
In[50]:=  PermEqual[Cycles[{1, 2, 3}], Perm[{3, 1, 2}]]

Out[50]=  True

In[51]:=  Cycles[{1, 2, 3}] === Perm[{3, 1, 2}]

Out[51]=  False
```

All comparatives have been generalized: `PermEqual`, `PermLess`, `PermGreater`, `PermLessEqual`, `PermGreaterEqual`, `PermOrderedQ`, and we can also sort points and permutations as given by a base.

```
In[52]:=  ? PermOrderedQ

            PermOrderedQ[h[perm1, perm2]] gives True if perm1 maps integers to smaller
              points than perm2. It gives False if perm1 maps integers to larger points
              than perm2. It gives Null if perm1 and perm2 maps integers to the same
              points. PermOrderedQ[h[perm1, perm2], B] gives True if perm1 maps the points
              of base B to smaller (according to B) images than perm2. It gives False if
              perm1 maps the points of B to larger (according to B) images than perm2.
              It gives Null if perm1 and perm2 map the points of B to the same images.
```

Images of points or lists of points:

```
In[53]:=  OnPoints[4, Perm[{1, 2, 5, 3, 4}]]

Out[53]=  5

In[54]:=  OnPoints[{2, 4, 3}, Rules[1 → 3, 2 → 1, 3 → 2]]

Out[54]=  {1, 4, 2}
```

There is a simple way to generate the identity corresponding to any notation, degree and sign:

```
In[55]:=  ID[Perm[{3, 2, 4, 1}]]

Out[55]=  Perm[{1, 2, 3, 4}]

In[56]:=  ID[Cycles[{3, 2, 4}, {5, 6}]]

Out[56]=  Cycles[]

In[57]:=  ID[-Images[{4, 5, 3, 2, 1}]]

Out[57]=  Images[{1, 2, 3, 4, 5}]
```

`ID` also represents the identity:

```
In[58]:=  Permute[ID, Perm[{3, 2, 4, 1}]]

Out[58]=  Perm[{3, 2, 4, 1}]
```

We can permute any kind of list, even with repeated elements:

```
In[59]:=  PermuteList[{a, b, c, d, b}, Perm[{4, 3, 2, 5, 1}]]

Out[59]=  {d, c, b, b, a}
```

Or find a permutation linking two different lists. The result is given in `Images` notation:

```
In[60]:=  PermutationFromTo[{a, b, c, d}, {d, c, b, a}]

Out[60]=  Images[{4, 3, 2, 1}]
```

Again, note that the order of permutation is very important:

```
In[61]:=  p1 = Images[{4, 3, 2, 5, 1}];
          p2 = Images[{5, 3, 4, 1, 2}];

In[63]:=  {1, 2, 3, 4, 5} // OnPoints[#, p1] & // OnPoints[#, p2] &

Out[63]=  {1, 4, 3, 2, 5}
```

```
In[64]:= {1, 2, 3, 4, 5} // OnPoints[#, Permute[p1, p2]] &

Out[64]= {1, 4, 3, 2, 5}

In[65]:= {1, 2, 3, 4, 5} // OnPoints[#, p1] &

Out[65]= {4, 3, 2, 5, 1}

In[66]:= PermuteList[{1, 2, 3, 4, 5}, InversePerm[p1]]

Out[66]= {4, 3, 2, 5, 1}
```

`PermQ`	Validate a permutation
`PermDeg`	Give the degree of a permutation
`PermLength`	Give the length of a permutation
`RandomPerm`	Construct a random permutation of a given degree
`Permute`	Product of several permutations
`InversePerm`	Inverse of a permutation
`PowerPermute`	Repeated product of a permutation
`PermSort`	Order permutations according to their images
`OnPoints`	Compute images of points under a permutation
`ID`	Identity permutation
`PermuteList`	Apply permutation on a given list
`PermutationFromTo`	Calculate permutation changing one list into other

Basic operations with permutations.

`SortB`	Sort points as given by a base
`MinB`	Least point of a list as given by a base
`PermEqual`	Compare permutations
`PermGreater`	Compare permutations
`PermLess`	Compare permutations
`PermGreaterqual`	Compare permutations
`PermLessEqual`	Compare permutations
`PermOrderedQ`	Check whether a list of permuations is sorted
`PermSort`	Sort permutations according to their images

Sorting and comparing permutations.

■ 3. Groups of permutations and Generating Sets

3.1. Groups by generators

Every finite group is isomorphic to some subgroup of a symmetric group of permutations (Cayley theorem). Hence, provided we can find a permutation representation, working with groups of permutations is generic.

Groups of permutations can be extremely large. The order (number of elements) of the group S_n of permutations of n points is n!. In general we will be interested in (still very large) subgroups of a given S_n. For example, the number of permutations of the 48 moving squares of a Rubik's cube is

```
In[67]:= 48! // N

Out[67]= 1.24139 × 10^61
```

On the other hand, the number of allowed permutations of a "clean" cube (forming a subgroup) turns out to be, as we will see below,

```
In[68]:=  43252003274489856000 // N

Out[68]=  4.3252 × 10^19
```

However we can describe the whole subgroup with just six permutations, corresponding to each rotation of the six faces of the cube. The subgroup can be constructed by repeated multiplication of the six "generators". We shall describe groups using generating sets, expressions with head `GenSet` where the elements are the generators. By convention, the identity permutation never belongs to a generating set. For example the group S_4 can be generated with just 2 permutations using Diminos's algorithm:

```
In[69]:=  Dimino[GenSet[Perm[{2, 1, 3, 4}], Perm[{4, 1, 2, 3}]]]

Out[69]=  Group[Perm[{1, 2, 3, 4}], Perm[{2, 1, 3, 4}], Perm[{4, 1, 2, 3}], Perm[{4, 2, 1, 3}],
           Perm[{1, 4, 2, 3}], Perm[{2, 4, 1, 3}], Perm[{3, 4, 1, 2}], Perm[{3, 4, 2, 1}],
           Perm[{3, 1, 4, 2}], Perm[{3, 2, 4, 1}], Perm[{4, 3, 1, 2}], Perm[{4, 3, 2, 1}],
           Perm[{2, 3, 4, 1}], Perm[{1, 3, 4, 2}], Perm[{2, 3, 1, 4}], Perm[{1, 3, 2, 4}],
           Perm[{2, 4, 3, 1}], Perm[{1, 4, 3, 2}], Perm[{3, 2, 1, 4}], Perm[{3, 1, 2, 4}],
           Perm[{4, 2, 3, 1}], Perm[{4, 1, 3, 2}], Perm[{1, 2, 4, 3}], Perm[{2, 1, 4, 3}]]
```

And actually any S_n can be generated with just 2 permutations. For example S_6:

```
In[70]:=  Length[Dimino[GenSet[Cycles[{1, 2}], Cycles[{1, 2, 3, 4, 5, 6}]]]]

Out[70]=  720
```

A group generated by a single permutation is called a cyclic group:

```
In[71]:=  group = Dimino[GenSet[Images[{2, 3, 4, 5, 1}]]]

Out[71]=  Group[Images[{1, 2, 3, 4, 5}], Images[{2, 3, 4, 5, 1}],
           Images[{3, 4, 5, 1, 2}], Images[{4, 5, 1, 2, 3}], Images[{5, 1, 2, 3, 4}]]
```

We can form left and right cosets:

```
In[72]:=  leftcoset = Permute[Images[{1, 3, 2, 5, 4}], group]

Out[72]=  Coset[Images[{1, 3, 2, 5, 4}], Images[{2, 4, 3, 1, 5}],
           Images[{3, 5, 4, 2, 1}], Images[{4, 1, 5, 3, 2}], Images[{5, 2, 1, 4, 3}]]

In[73]:=  rightcoset = Permute[group, Images[{1, 3, 2, 5, 4}]]

Out[73]=  Coset[Images[{1, 3, 2, 5, 4}], Images[{3, 2, 5, 4, 1}],
           Images[{2, 5, 4, 1, 3}], Images[{5, 4, 1, 3, 2}], Images[{4, 1, 3, 2, 5}]]
```

If the permutation already belongs to the group, the coset is actually the original group:

```
In[74]:=  Permute[Images[{4, 5, 1, 2, 3}], group]

Out[74]=  Group[Images[{1, 2, 3, 4, 5}], Images[{2, 3, 4, 5, 1}],
           Images[{3, 4, 5, 1, 2}], Images[{4, 5, 1, 2, 3}], Images[{5, 1, 2, 3, 4}]]
```

`GenSet`	Head for a generating set
`Group`	Head for a group of permutations
`Coset`	Head for a coset
`Dimino`	Algorithm to construct a group of permutations from one of its generating sets

Generating Sets.

3.2. Cosets and double cosets

All cosets of a given subgroup of permutations have the same order, and partition the full group. Hence the order of a subgroup divides the order of the full group (Lagrange theorem).

Suppose we start with the full group S5 of permutations:

```
In[75]:= Length[S5 = Sort@Dimino[GenSet[Images[{2, 1, 3, 4, 5}], Images[{2, 3, 4, 5, 1}]]]]

Out[75]= 120
```

One of its subgroups is S3:

```
In[76]:= Length[S3 = Sort@Dimino[GenSet[Images[{2, 1, 3, 4, 5}], Images[{2, 3, 1, 4, 5}]]]]

Out[76]= 6
```

There are therefore 20 left−cosets of S4 in S5, that we call L1...L20:

```
In[77]:= ColumnForm[S3cosets = Union[Sort /@ Outer[Permute, List @@ S5, List @@ S3]]]

Out[77]= {Images[{1, 2, 3, 4, 5}], Images[{1, 3, 2, 4, 5}], Images[{2, 1, 3, 4, 5}], Images[{2, 3,
         {Images[{1, 2, 3, 5, 4}], Images[{1, 3, 2, 5, 4}], Images[{2, 1, 3, 5, 4}], Images[{2, 3,
         {Images[{1, 2, 4, 3, 5}], Images[{1, 3, 4, 2, 5}], Images[{2, 1, 4, 3, 5}], Images[{2, 3,
         {Images[{1, 2, 4, 5, 3}], Images[{1, 3, 4, 5, 2}], Images[{2, 1, 4, 5, 3}], Images[{2, 3,
         {Images[{1, 2, 5, 3, 4}], Images[{1, 3, 5, 2, 4}], Images[{2, 1, 5, 3, 4}], Images[{2, 3,
         {Images[{1, 2, 5, 4, 3}], Images[{1, 3, 5, 4, 2}], Images[{2, 1, 5, 4, 3}], Images[{2, 3,
         {Images[{1, 4, 2, 3, 5}], Images[{1, 4, 3, 2, 5}], Images[{2, 4, 1, 3, 5}], Images[{2, 4,
         {Images[{1, 4, 2, 5, 3}], Images[{1, 4, 3, 5, 2}], Images[{2, 4, 1, 5, 3}], Images[{2, 4,
         {Images[{1, 4, 5, 2, 3}], Images[{1, 4, 5, 3, 2}], Images[{2, 4, 5, 1, 3}], Images[{2, 4,
         {Images[{1, 5, 2, 3, 4}], Images[{1, 5, 3, 2, 4}], Images[{2, 5, 1, 3, 4}], Images[{2, 5,
         {Images[{1, 5, 2, 4, 3}], Images[{1, 5, 3, 4, 2}], Images[{2, 5, 1, 4, 3}], Images[{2, 5,
         {Images[{1, 5, 4, 2, 3}], Images[{1, 5, 4, 3, 2}], Images[{2, 5, 4, 1, 3}], Images[{2, 5,
         {Images[{4, 1, 2, 3, 5}], Images[{4, 1, 3, 2, 5}], Images[{4, 2, 1, 3, 5}], Images[{4, 2,
         {Images[{4, 1, 2, 5, 3}], Images[{4, 1, 3, 5, 2}], Images[{4, 2, 1, 5, 3}], Images[{4, 2,
         {Images[{4, 1, 5, 2, 3}], Images[{4, 1, 5, 3, 2}], Images[{4, 2, 5, 1, 3}], Images[{4, 2,
         {Images[{4, 5, 1, 2, 3}], Images[{4, 5, 1, 3, 2}], Images[{4, 5, 2, 1, 3}], Images[{4, 5,
         {Images[{5, 1, 2, 3, 4}], Images[{5, 1, 3, 2, 4}], Images[{5, 2, 1, 3, 4}], Images[{5, 2,
         {Images[{5, 1, 2, 4, 3}], Images[{5, 1, 3, 4, 2}], Images[{5, 2, 1, 4, 3}], Images[{5, 2,
         {Images[{5, 1, 4, 2, 3}], Images[{5, 1, 4, 3, 2}], Images[{5, 2, 4, 1, 3}], Images[{5, 2,
         {Images[{5, 4, 1, 2, 3}], Images[{5, 4, 1, 3, 2}], Images[{5, 4, 2, 1, 3}], Images[{5, 4,

In[78]:= {L1, L2, L3, L4, L5, L6, L7, L8, L9, L10,
            L11, L12, L13, L14, L15, L16, L17, L18, L19, L20} = S3cosets;
```

And they all form the full S5 group:

```
In[79]:=  Sort[Group @@ Union @@ S3cosets] === S5

Out[79]=  True
```

The concept of double coset will be important for us later. A double coset of two subgroups H and K in the full group G is the set of permutations of the form h.g.k for all h in H, all k in K and a particular g in G. The double cosets also partition the full group G, but unfortunately they do not all have the same size, and therefore there is not an equivalent of the Lagrange theorem for double cosets. The computation of the number of double cosets in a group G is a nontrivial task.

Construct another subgroup of S5:

```
In[80]:=  Length[H = Dimino[
             GenSet[Images[{2, 1, 3, 4, 5}], Images[{1, 2, 4, 3, 5}], Images[{3, 4, 1, 2, 5}]]]]

Out[80]=  8
```

That means that there are 15 right−cosets of H in S5, that we call R1...R15:

```
In[81]:=  ColumnForm[Hcosets = Union[Sort /@ Transpose@Outer[Permute, List @@ H, List @@ S5]]]

Out[81]=  {Images[{1, 2, 3, 4, 5}], Images[{1, 2, 4, 3, 5}], Images[{2, 1, 3, 4, 5}], Images[{2, 1,
          {Images[{1, 2, 3, 5, 4}], Images[{1, 2, 5, 3, 4}], Images[{2, 1, 3, 5, 4}], Images[{2, 1,
          {Images[{1, 2, 4, 5, 3}], Images[{1, 2, 5, 4, 3}], Images[{2, 1, 4, 5, 3}], Images[{2, 1,
          {Images[{1, 3, 2, 4, 5}], Images[{1, 3, 4, 2, 5}], Images[{2, 4, 1, 3, 5}], Images[{2, 4,
          {Images[{1, 3, 2, 5, 4}], Images[{1, 3, 5, 2, 4}], Images[{2, 5, 1, 3, 4}], Images[{2, 5,
          {Images[{1, 3, 4, 5, 2}], Images[{1, 3, 5, 4, 2}], Images[{3, 1, 4, 5, 2}], Images[{3, 1,
          {Images[{1, 4, 2, 3, 5}], Images[{1, 4, 3, 2, 5}], Images[{2, 3, 1, 4, 5}], Images[{2, 3,
          {Images[{1, 4, 2, 5, 3}], Images[{1, 4, 5, 2, 3}], Images[{2, 5, 1, 4, 3}], Images[{2, 5,
          {Images[{1, 4, 3, 5, 2}], Images[{1, 4, 5, 3, 2}], Images[{3, 5, 1, 4, 2}], Images[{3, 5,
          {Images[{1, 5, 2, 3, 4}], Images[{1, 5, 3, 2, 4}], Images[{2, 3, 1, 5, 4}], Images[{2, 3,
          {Images[{1, 5, 2, 4, 3}], Images[{1, 5, 4, 2, 3}], Images[{2, 4, 1, 5, 3}], Images[{2, 4,
          {Images[{1, 5, 3, 4, 2}], Images[{1, 5, 4, 3, 2}], Images[{3, 4, 1, 5, 2}], Images[{3, 4,
          {Images[{2, 3, 4, 5, 1}], Images[{2, 3, 5, 4, 1}], Images[{3, 2, 4, 5, 1}], Images[{3, 2,
          {Images[{2, 4, 3, 5, 1}], Images[{2, 4, 5, 3, 1}], Images[{3, 5, 2, 4, 1}], Images[{3, 5,
          {Images[{2, 5, 3, 4, 1}], Images[{2, 5, 4, 3, 1}], Images[{3, 4, 2, 5, 1}], Images[{3, 4,

In[82]:=  {R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15} = Hcosets;
```

Now we want to find the number of double cosets of H and S3 in S5. The key idea is that a permutation in H acting on a left coset Li gives another (perhaps the same) left coset Lj. Therefore permutations in H now act as permutations on a set in which the points are the left cosets L1...L20. Each orbit in this set corresponds to a double coset. Note that double cosets are disjoint unions of single−cosets of each of the two subgroups.

We first have the product group, with 24 elements (it is actually S4):

```
In[83]:=  Length[doublecoset1 = List @@ Union@Flatten[Outer[Permute, H, S3]]]

Out[83]=  24
```

This double coset contains the following single cosets:

```
In[84]:=  doublecoset1 === Union[L1, L3, L7, L13]

Out[84]=  True

In[85]:=  doublecoset1 === Union[R1, R4, R7]

Out[85]=  True
```

Now we look for the double coset containing L2. It is again S4 with points 4 and 5 exchanged:

```
In[86]:=  Map[Sort, Outer[Permute, List @@ H, {L2, L5, L10, L17}], {2}] ===
```

$$\begin{pmatrix} L2 & L5 & L10 & L17 \\ L2 & L5 & L17 & L10 \\ L5 & L2 & L10 & L17 \\ L5 & L2 & L17 & L10 \\ L10 & L17 & L2 & L5 \\ L17 & L10 & L2 & L5 \\ L10 & L17 & L5 & L2 \\ L17 & L10 & L5 & L2 \end{pmatrix}$$

```
Out[86]=  True

In[87]:=  Length[doublecoset2 = Union[L2, L5, L10, L17]]

Out[87]=  24

In[88]:=  doublecoset2 === Union[R2, R5, R10]

Out[88]=  True
```

Now we look for the double coset containing L4:

```
In[89]:=  Map[Sort, Outer[Permute, List @@ H, {L4, L6, L16, L20}], {2}] ===
```

$$\begin{pmatrix} L4 & L6 & L16 & L20 \\ L4 & L6 & L20 & L16 \\ L6 & L4 & L16 & L20 \\ L6 & L4 & L20 & L16 \\ L16 & L20 & L4 & L6 \\ L20 & L16 & L4 & L6 \\ L16 & L20 & L6 & L4 \\ L20 & L16 & L6 & L4 \end{pmatrix}$$

```
Out[89]=  True

In[90]:=  Length[doublecoset3 = Union[L4, L6, L16, L20]]

Out[90]=  24

In[91]:=  doublecoset3 === Union[R3, R6, R13]

Out[91]=  True
```

Now we look for the double coset containing L8:

```
In[92]:=  Map[Sort, Outer[Permute, List @@ H, {L8, L9, L11, L12, L14, L15, L18, L19}], {2}] ===

          ( L8   L9   L11  L12  L14  L15  L18  L19 )
          ( L14  L15  L18  L19  L8   L9   L11  L12 )
          ( L9   L8   L12  L11  L15  L14  L19  L18 )
          ( L15  L14  L19  L18  L9   L8   L12  L11 )
          ( L11  L18  L8   L14  L12  L19  L9   L15 )
          ( L18  L11  L14  L8   L19  L12  L15  L9  )
          ( L12  L19  L9   L15  L11  L18  L8   L14 )
          ( L19  L12  L15  L9   L18  L11  L14  L8  )

Out[92]=  True

In[93]:=  Length[doublecoset4 = Union[L8, L9, L11, L12, L14, L15, L18, L19]]

Out[93]=  48

In[94]:=  doublecoset4 === Union[R8, R9, R11, R12, R14, R15]

Out[94]=  True
```

We conclude that there are 4 double cosets, three of them with order 24 and the last one with order 48. They all form the full group S5:

```
In[95]:=  List @@ S5 === Union[doublecoset1, doublecoset2, doublecoset3, doublecoset4]

Out[95]=  True
```

Tidy up:

```
In[96]:=  Clear[L1, L2, L3, L4, L5, L6, L7, L8, L9, L10, L11, L12, L13, L14, L15, L16, L17,
           L18, L19, L20, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15]
```

There are also commands to form transversals.

`LeftTransversal`	Transversal of left cosets
`RightTransversal`	Transversal of right cosets
`DoubleTransversal`	Transversal of double cosets

Transversals.

Strong Generating Sets

■ 4. Orbits and Schreier vectors

An essential tool in the analysis of the structure of a group of permutations is the concept of orbit of a point. It is just the set of images of that point which can be obtained by repeated action of the members of the group.

The command `Orbit` gives the orbit of a point under a group generated by a generating set:

```
In[97]:=  Orbit[2, GenSet[Perm[{4, 1, 3, 2, 6, 5}], Perm[{1, 4, 3, 2, 5, 6}]]]

Out[97]=  {2, 4, 1}
```

We can compute all orbits of a group. This can be understood as a partition of a range of integers which by default has the length of the highest degree of the permutations

```
In[98]:=  Orbits[GenSet[Perm[{4, 1, 3, 2, 6, 5}], Perm[{1, 4, 3, 2, 5, 6}]]]

Out[98]=  {{1, 2, 4}, {3}, {5, 6}}
```

but can supply the length as a second argument. Many functions in the package have an additional argument to specify the length of the underlying set of points. See the documentation of each command for details:

```
In[99]:=  Orbits[GenSet[Perm[{4, 1, 3, 2, 6, 5}], Perm[{1, 4, 3, 2, 5, 6}]], 8]

Out[99]=  {{1, 2, 4}, {3}, {5, 6}, {7}, {8}}
```

`Orbit`	Orbit of a point under a GS
`Orbits`	All orbits of a group

Compute orbits of a group.

Given an orbit of a group it is very useful to be able to find a permutation of the group linking two points of the orbit, that is, such that the latter point is the image of the former under that permutation. Schreier introduced a nice way to do that: together with the points of the orbit we store two vectors:

Same orbit, now with its Schreier vectors. Point 2 is the first point of the orbit. Point 4 can be obtained from point 2 (see second vector) usign permutation `Perm[{4,1,3,2,6,5}]` (see first vector). Point 1 can be obtained from point 4 (see second vector) using the same permutation (see first vector). Again we

```
In[100]:=
      SchreierOrbit[2, GenSet[Perm[{4, 1, 3, 2, 6, 5}], Perm[{1, 4, 3, 2, 5, 6}]]]

Out[100]=
      Schreier[{2, 4, 1},
       {Perm[{4, 1, 3, 2, 6, 5}], 0, 0, Perm[{4, 1, 3, 2, 6, 5}], 0, 0}, {4, 0, 0, 2, 0, 0}]
```

The function `TraceSchreier` gives the permutation that links the first point of the orbit with a given point:

```
In[101]:=
      TraceSchreier[1, %]

Out[101]=
      Perm[{2, 4, 3, 1, 5, 6}]

In[102]:=
      OnPoints[2, %]

Out[102]=
      1
```

`SchreierOrbit`	Calculate orbit with its Schreier vectors
`Schreier`	Head of an orbit with its Schreier vectors
`SchreierOrbits`	Calculate all orbits with a compound Schreier vector
`TraceSchreier`	Trace an orbit to find a permutation linking two points

Schreier vectors.

5. Stabilization and Strong Generating Sets

A second essential tool to analyze a group of permutations is the concept of stability.

Given a permutation, there can be points that are not moved:

```
In[103]:=
      StablePoints[Images[{3, 2, 1, 4, 6, 5, 7}], 7]

Out[103]=
      {2, 4, 7}
```

Of course, points over the degree of the permutation are not moved:

```
In[104]:=
      StablePoints[Images[{3, 2, 1, 4, 6, 5, 7}], 10]

Out[104]=
      {2, 4, 7, 8, 9, 10}
```

Given a generating set GS and a list of points, we call pointwise stabilizer of those points to the subset of GS such that none of the permutations in the subset moves none of the points in the list:

```
In[105]:=
      Stabilizer[{1, 3},
       GenSet[Perm[{1, 2, 3, 5, 4}], Perm[{1, 2, 4, 3, 5}], Perm[{3, 2, 1, 4, 5}]]]

Out[105]=
      GenSet[Perm[{1, 2, 3, 5, 4}]]
```

There is a different concept, that of set−stabilization, requiring that points do not leave the set, though they can move:

```
In[106]:=
      SetStabilizer[{1, 3},
       GenSet[Perm[{1, 2, 3, 5, 4}], Perm[{1, 2, 4, 3, 5}], Perm[{3, 2, 1, 4, 5}]]]

Out[106]=
      GenSet[Perm[{1, 2, 3, 5, 4}], Perm[{3, 2, 1, 4, 5}]]
```

The stabilizer of a group S is a subgroup. However, the stabilizer of a generating set of S is not necessarily a generating set of the subgroup. To solve that problem, Sims introduced the concept of strong generating set: we give a pair formed by a list of points B=$\{b_1, ..., b_k\}$ (called "base") and a generating set GS of the group. The generating set is strong with respect to that base if 1) the stabilizer S(b_1,...,b_m) of any sorted sublist $\{b_1, ..., b_m\}$, with m≤k, can be generated by a subset of the permutations in GS, and 2) no permutation in GS fixes all points in the base. The order of points in the base is relevant. This allows us to form a hierarchy of subgroups: S = S() ⊇ S(b_1) ⊇ S(b_1,b_2) ⊇ ... ⊇ S(b_1,...,b_k) = {ID}.

Hierarchy of subgroups associated to a simple strong generating set:

```
In[107]:=
     StabilizerChain[StrongGenSet[{1, 3},
        GenSet[Perm[{2, 1, 3, 4}], Perm[{1, 2, 4, 3}], Perm[{3, 4, 1, 2}]]]] // ColumnForm

Out[107]=
     StrongGenSet[{1, 3}, GenSet[Perm[{2, 1, 3, 4}], Perm[{1, 2, 4, 3}], Perm[{3, 4, 1, 2}]]]
     StrongGenSet[{3}, GenSet[Perm[{1, 2, 4, 3}]]]
     StrongGenSet[{}, GenSet[]]
```

Strong generating sets can be calculated from a generating set using the Schreier−Sims algorithm. Example taken from Butler's book:

```
In[108]:=
     a = Cycles[{1, 8, 9}, {2, 11, 15},
        {3, 10, 12}, {4, 14, 19}, {5, 16, 17}, {6, 21, 20}, {7, 13, 18}];
     b = Cycles[{9, 18, 20}, {12, 19, 17}];
     c = Cycles[{10, 21, 11}, {13, 16, 14}];

In[111]:=
     GS = GenSet[a, b, c];

In[112]:=
     myTiming[SGS = SchreierSims[{}, GS]]

        0.566673 Second

Out[112]=
     StrongGenSet[{9, 10, 8, 2, 1, 12},
      GenSet[Cycles[{1, 8, 9}, {2, 11, 15}, {3, 10, 12}, {4, 14, 19},
        {5, 16, 17}, {6, 21, 20}, {7, 13, 18}], Cycles[{9, 18, 20}, {12, 19, 17}],
       Cycles[{10, 21, 11}, {13, 16, 14}], Cycles[{8, 13, 21}, {10, 14, 16}],
       Cycles[{8, 16, 11}, {13, 14, 21}], Cycles[{2, 3, 6}, {4, 7, 5}],
       Cycles[{1, 7, 6}, {3, 4, 5}], Cycles[{12, 20, 15}, {17, 19, 18}]]]
```

We can give several initial points of the base. Using `MathLink` is faster, but always returns the result in Images notation:

```
In[113]:=
     myTiming[SGS2 = SchreierSims[{8}, GS, 21, MathLink → True]]

        0.004246 Second

Out[113]=
     StrongGenSet[{8, 9, 10, 2, 1, 12}, GenSet[
       Images[{8, 11, 10, 14, 16, 21, 13, 9, 1, 12, 15, 3, 18, 19, 2, 17, 5, 7, 4, 6, 20}],
       Images[{1, 2, 3, 4, 5, 6, 7, 8, 18, 10, 11, 19, 13, 14, 15, 16, 12, 20, 17, 9, 21}],
       Images[{1, 2, 3, 4, 5, 6, 7, 8, 9, 21, 10, 12, 16, 13, 15, 14, 17, 18, 19, 20, 11}],
       Images[{1, 3, 6, 7, 4, 2, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21}],
       Images[{7, 2, 4, 5, 3, 1, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21}],
       Images[{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 20, 13, 14, 12, 16, 19, 17, 18, 15, 21}]]]
```

We can also change the base points of a given SGS. This typically introduces many redundant generators and additional points in the base:

```
In[114]:=
      BaseChange[SGS2, {9, 10}]

Out[114]=
      StrongGenSet[{9, 10, 1, 12, 11, 8, 3}, GenSet[
        Images[{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 15, 13, 14, 20, 16, 18, 19, 17, 12, 21}],
        Images[{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 20, 13, 14, 12, 16, 19, 17, 18, 15, 21}],
        Images[{1, 2, 3, 4, 5, 6, 7, 8, 9, 21, 10, 12, 16, 13, 15, 14, 17, 18, 19, 20, 11}],
        Images[{1, 2, 3, 4, 5, 6, 7, 13, 9, 14, 11, 12, 21, 16, 15, 10, 17, 18, 19, 20, 8}],
        Images[{1, 2, 3, 4, 5, 6, 7, 16, 9, 10, 8, 12, 14, 21, 15, 11, 17, 18, 19, 20, 13}],
        Images[{1, 3, 6, 7, 4, 2, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21}],
        Images[{1, 6, 2, 5, 7, 3, 4, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21}],
        Images[{2, 3, 4, 6, 7, 5, 1, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21}],
        Images[{7, 2, 4, 5, 3, 1, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21}],
        Images[{8, 11, 10, 14, 16, 21, 13, 9, 1, 12, 15, 3, 18, 19, 2, 17, 5, 7, 4, 6, 20}]]]

In[115]:=
      SGS3 = DeleteRedundantGenerators[%]

Out[115]=
      StrongGenSet[{9, 10, 1, 12, 11, 8, 3}, GenSet[
        Images[{1, 3, 6, 7, 4, 2, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21}],
        Images[{1, 2, 3, 4, 5, 6, 7, 16, 9, 10, 8, 12, 14, 21, 15, 11, 17, 18, 19, 20, 13}],
        Images[{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 15, 13, 14, 20, 16, 18, 19, 17, 12, 21}],
        Images[{2, 3, 4, 6, 7, 5, 1, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21}],
        Images[{1, 2, 3, 4, 5, 6, 7, 8, 9, 21, 10, 12, 16, 13, 15, 14, 17, 18, 19, 20, 11}],
        Images[{8, 11, 10, 14, 16, 21, 13, 9, 1, 12, 15, 3, 18, 19, 2, 17, 5, 7, 4, 6, 20}]]]
```

Once we have the SGS we can solve any question related to the group:

```
In[116]:=
      StabilizerChain[SGS] // ColumnForm

Out[116]=
      StrongGenSet[{9, 10, 8, 2, 1, 12}, GenSet[Cycles[{1, 8, 9}, {2, 11, 15}, {3, 10, 12}, {4,
      StrongGenSet[{10, 8, 2, 1, 12}, GenSet[Cycles[{10, 21, 11}, {13, 16, 14}], Cycles[{8, 13,
      StrongGenSet[{8, 2, 1, 12}, GenSet[Cycles[{8, 16, 11}, {13, 14, 21}], Cycles[{2, 3, 6}, {
      StrongGenSet[{2, 1, 12}, GenSet[Cycles[{2, 3, 6}, {4, 7, 5}], Cycles[{1, 7, 6}, {3, 4, 5}
      StrongGenSet[{1, 12}, GenSet[Cycles[{1, 7, 6}, {3, 4, 5}], Cycles[{12, 20, 15}, {17, 19,
      StrongGenSet[{12}, GenSet[Cycles[{12, 20, 15}, {17, 19, 18}]]]
      StrongGenSet[{}, GenSet[]]

In[117]:=
      OrderOfGroup /@ {SGS, SGS2, SGS3}

Out[117]=
      {27783, 27783, 27783}

In[118]:=
      PermMemberQ[Permute[a, b, c], SGS]

Out[118]=
      True
```

```
In[119]:=
      PermMemberQ[Cycles[{1, 2}], SGS]

Out[119]=
      False
```

A permutation in the group can be uniquely decomposed according to the chain of stabilizers:

```
In[120]:=
      perm = Permute[a, b, c]

Out[120]=
      Cycles[{1, 8, 18, 7, 16, 12, 3, 21, 9}, {2, 10, 19, 4, 13, 20, 6, 11, 15}, {5, 14, 17}]

In[121]:=
      PermWord[perm, SGS]

Out[121]=
      {Cycles[], ID, ID, Cycles[{2, 3, 6}, {4, 7, 5}], ID,
       Cycles[{8, 13, 21}, {10, 14, 16}], Cycles[{1, 8, 9}, {2, 11, 15},
         {3, 10, 12}, {4, 14, 19}, {5, 16, 17}, {6, 21, 20}, {7, 13, 18}]}

In[122]:=
      perm === Permute @@ %

Out[122]=
      True
```

or identified from its base images:

```
In[123]:=
      OnPoints[SGS[[1]], perm]

Out[123]=
      {1, 19, 18, 10, 8, 3}

In[124]:=
      perm === FromBaseImage[%, SGS]

Out[124]=
      True
```

Not every list of images defines a permutation:

```
In[125]:=
      Catch@FromBaseImage[{1, 2, 3, 4, 5, 6}, SGS]

      FromBaseImage::noimgs : Invalid list of images {1, 2, 3, 4, 5, 6}.
```

As we said, there is a one to one correspondence between base images and permutations in a group:

```
In[126]:=
      AllBaseImages[StrongGenSet[{1, 3},
         GenSet[-Cycles[{1, 2}], -Cycles[{3, 4}], Cycles[{1, 3}, {2, 4}]]]] // ColumnForm

Out[126]=
      {1, 3} → ID
      {1, 4} → -Cycles[{3, 4}]
      {2, 3} → -Cycles[{1, 2}]
      {2, 4} → Cycles[{1, 2}, {3, 4}]
      {3, 1} → Cycles[{1, 3}, {2, 4}]
      {3, 2} → -Cycles[{1, 3, 2, 4}]
      {4, 1} → -Cycles[{1, 4, 2, 3}]
      {4, 2} → Cycles[{1, 4}, {2, 3}]
```

`StablePoints`	Points that are not moved by a permutation
`NonStablePoints`	Construct a base for a group
`Stabilizer`	Subset of permutations that do not move a list of points
`SetStabilizer`	Subset of permutations that do not move a list of points within a set
`StabilizerChain`	Chain of stabilizers of a Strong Generating Set
`StrongGenSet`	Head for a Strong Generating Set
`SchreierSims`	Algorithm to compute a SGS from a GS
`BaseChange`	Change base of a SGS
`DeleteRedundantGenerators`	Drop unnecessary permutations in a SGS
`JoinSGS`	Form SGS for product group
`OrderOfGroup`	Compute order of group described by a SGS
`PermMemberQ`	Check whether a permutation belongs to a group described by a SGS
`PermWord`	Decompose a permutation as a unique product of coset representatives in a SGS hierarchy
`FromBaseImage`	Construct the unique permutation associated to a list of base images
`AllBaseImages`	List all possible base images and their corresponding permutations

Strong Generating Sets.

The main idea behind the Schreier−Sims algorithm is that of "backtrack search". Using the one−to−one mapping between permutations and base images, it is possible to organize all permutations in a group in a tree whose nodes are the possible images of the points in the base. The Schreier vectors allow us to traverse the tree up and down, and the identification between cosets and points of orbits allows us to discard all leaves of the tree below a given node by testing just a single leave. See Butler's book for full details.

The function `Search` implements backtracking to search for a SGS of a subgroup of a given group:

```
In[127]:=
      ?Search

          Search[SGS, P, s, SGSK] returns the s-th stabilizer in the stabilizer-
            chain of the subgroup K of permutations obeying the property P
            in the group G (described by the strong generating set SGS). It
            is assumed P[perm] returns True or False on any permutation of
            the group G. The fourth argument is a strong generating set for a
            subgroup of K, possibly a deeper stabilizer in its stabilizer-chain.
```

For instance, let us suppose we start from the group S_4.

```
In[128]:=
      SGS = StrongGenSet[{1, 2, 3},
         GenSet[Cycles[{1, 2}], Cycles[{2, 3}], Cycles[{3, 4}], Cycles[{1, 4}]]];
```

and we want to find a sgs for the subgroup of permutations commuting with permutation `z` (the so−called centralizer of `z`):

```
In[129]:=
      z = Cycles[{1, 3}, {2, 4}];
```

We define the property `P` identifying the permutations commuting with `z`:

```
In[130]:=
      P[perm_] := PermEqual[Permute[z, perm], Permute[perm, z]]

In[131]:=
      P[Cycles[{1, 2}, {3, 4}]]

Out[131]=
      True

In[132]:=
      P[Cycles[{1, 2}]]

Out[132]=
      False
```

The first stabilizer in the chain coincides with the group itself and so we search for the first stabilizer of the subgroup of SGS obeying property P:

```
In[133]:=
      Search[SGS, P, 1, StrongGenSet[{}, GenSet[]], xPermVerbose → True]

          Branching over points {3, 4}

          Generate at level 3 with i=4. We have list {1, 2, 3} and permutation ID

          Generate at level 3 with i=4
           . We have list {1, 2, 4} and permutation Cycles[{3, 4}]

          Branching over points {2, 3, 4}

          Generate at level 2 with i=4
           . We have list {1, 3, 2} and permutation Cycles[{2, 3}]

          Generate at level 2 with i=4. We have list
           {1, 3, 4} and permutation Cycles[{2, 3, 4}]

          Generate at level 2 with i=4. We have list
           {1, 4, 2} and permutation Cycles[{2, 4, 3}]

          Generate at level 2 with i=4
           . We have list {1, 4, 3} and permutation Cycles[{2, 4}]

            Added permutation Cycles[{2, 4}]

          Branching over points {1, 2, 4, 3}

          Generate at level 1 with i=4
           . We have list {2, 1, 3} and permutation Cycles[{1, 2}]

          Generate at level 1 with i=4. We have list
           {2, 1, 4} and permutation Cycles[{1, 2}, {3, 4}]

            Added permutation Cycles[{1, 2}, {3, 4}]

Out[133]=
      StrongGenSet[{1, 2, 3}, GenSet[Cycles[{2, 4}], Cycles[{1, 2}, {3, 4}]]]
```

We see that we have tested 8 permutations in a group of order 24. Only 2 of those 8 form the final sgs of the centralizer, which has order 8:

```
In[134]:=
      OrderOfGroup[%]

Out[134]=
      8
```

Clean up:

```
In[135]:=
      Clear[SGS, z, P, a, b, c]
```

Search	Compute subgroup of permutations obeying a given property

Search subgroups.

6. Some important groups of symmetry

There are some groups which are used very often. `xPerm`` defines special ways to generate their associated SGSs. By default the permutations are given in `Cycles` notation:

Strong Generating Set for the symmetric or antisymmetric groups of several points:

```
In[136]:=
      Symmetric[{x1, x2, x3}]

Out[136]=
      StrongGenSet[{x1, x2}, GenSet[Cycles[{x1, x2}], Cycles[{x2, x3}]]]

In[137]:=
      Antisymmetric[{x1, x2, x3, x4}]

Out[137]=
      StrongGenSet[{x1, x2, x3},
       GenSet[-Cycles[{x1, x2}], -Cycles[{x2, x3}], -Cycles[{x3, x4}]]]
```

Strong Generating Set for the Riemann symmetry group of four points. Note the position of the two antisymmetric pairs:

```
In[138]:=
      RiemannSymmetric[{3, 5, 2, 9}]

Out[138]=
      StrongGenSet[{2, 3, 5},
       GenSet[Cycles[{3, 2}, {5, 9}], -Cycles[{3, 5}], -Cycles[{2, 9}]]]
```

The function `PairSymmetric` is far more general. It has two switches which control symmetry and antisymmetry under exchange of pairs and of members of a given pair.

```
In[139]:=
      PairSymmetric[{{1, 2}, {3, 4}, {5, 6}, {7, 8}}, -1, 0]

Out[139]=
      StrongGenSet[{1, 3, 5, 7}, GenSet[-Cycles[{7, 1, 3, 5}, {8, 2, 4, 6}],
        Cycles[{7, 3, 5}, {8, 4, 6}], -Cycles[{5, 7}, {6, 8}]]]

In[140]:=
      PairSymmetric[{{1, 2}, {3, 4}, {5, 6}, {7, 8}}, 0, 1]

Out[140]=
      StrongGenSet[{1, 3, 5, 7},
       GenSet[Cycles[{1, 2}], Cycles[{3, 4}], Cycles[{5, 6}], Cycles[{7, 8}]]]

In[141]:=
      PairSymmetric[{{1, 2}, {3, 4}, {5, 6}, {7, 8}}, -1, 1]

Out[141]=
      StrongGenSet[{1, 3, 5, 7}, GenSet[-Cycles[{7, 1, 3, 5}, {8, 2, 4, 6}],
        Cycles[{7, 3, 5}, {8, 4, 6}], -Cycles[{5, 7}, {6, 8}],
        Cycles[{1, 2}], Cycles[{3, 4}], Cycles[{5, 6}], Cycles[{7, 8}]]]
```

The symmetry of a Riemann tensor would be given as follows, and differs from `RiemannSymmetric` in the base:

```
In[142]:=
      PairSymmetric[{{1, 2}, {3, 4}}, 1, -1]

Out[142]=
      StrongGenSet[{1, 3}, GenSet[Cycles[{1, 3}, {2, 4}], -Cycles[{1, 2}], -Cycles[{3, 4}]]]

In[143]:=
      RiemannSymmetric[{1, 2, 3, 4}]

Out[143]=
      StrongGenSet[{1, 2, 3},
       GenSet[Cycles[{1, 3}, {2, 4}], -Cycles[{1, 2}], -Cycles[{3, 4}]]]
```

`Symmetric`	Give a SGS for a symmetric group
`Antisymmetric`	Give a SGS for an alternating group
`PairSymmetric`	Give a SGS for an group of permutations of pairs and/or their elements
`RiemannSymmetric`	Give a SGS for the group of symmetries of the Riemann tensor

Some important SGSs.

■ 7. Examples of Schreier−Sims algorithm

This section contains a number of very hard examples to show the power of our implementation of the algorithms. We do not expect, however, to need these groups in tensor computer algebra applications.

Here we shall always use the external canonicalizer. It would be too slow for our *Mathematica* code.

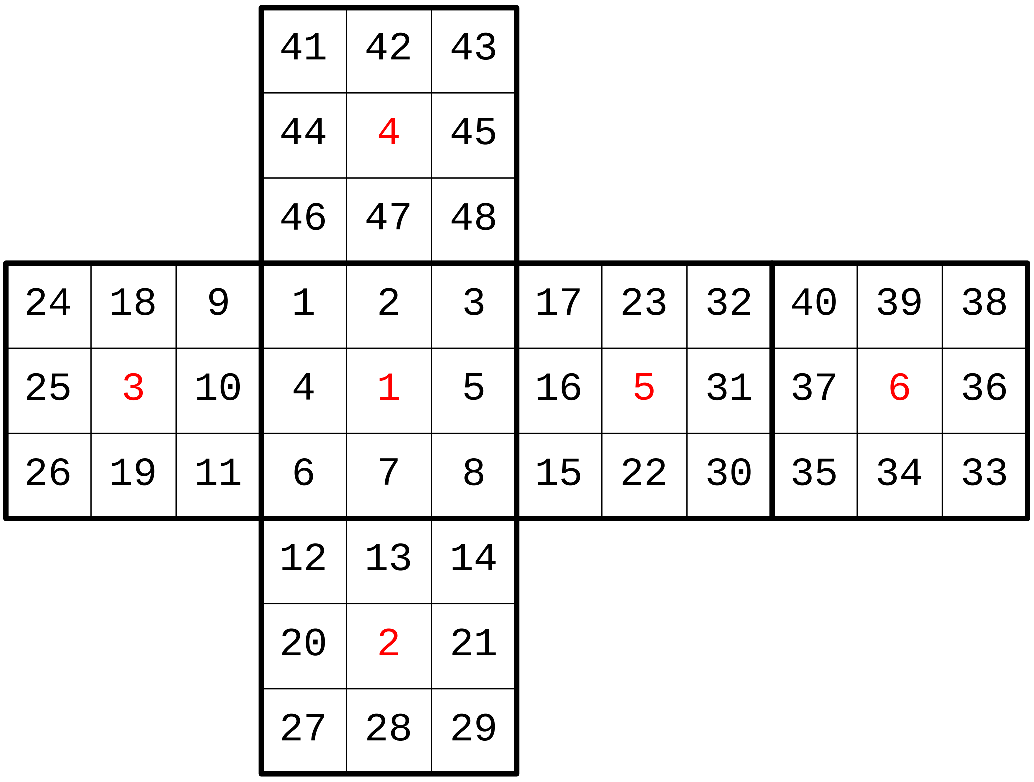
```
In[144]:=
      SetOptions[Orbit, MathLink → True];
      SetOptions[SchreierSims, MathLink → True];
```

`MathLink` executable code	Option for Orbit, SchreierSims and CanonicalPerm to use an external, faster

External executable.

7.1. Rubik's cube

Let us suppose that we number from 1 to 48 the non−central coloured squares of a Rubik's cube. There are six basic rotations, corresponding to each of the six faces:



We introduce the following notation for quarter−turns, with clockwise rotations:

```
In[146]:=
        rot1 =
          Cycles[{1, 3, 8, 6}, {2, 5, 7, 4}, {9, 48, 15, 12}, {10, 47, 16, 13}, {11, 46, 17, 14}];
        rot2 = Cycles[{6, 15, 35, 26}, {7, 22, 34, 19},
            {8, 30, 33, 11}, {12, 14, 29, 27}, {13, 21, 28, 20}];
        rot3 = Cycles[{1, 12, 33, 41}, {4, 20, 36, 44}, {6, 27, 38, 46},
            {9, 11, 26, 24}, {10, 19, 25, 18}];
        rot4 = Cycles[{1, 24, 40, 17}, {2, 18, 39, 23}, {3, 9, 38, 32},
            {41, 43, 48, 46}, {42, 45, 47, 44}];
        rot5 = Cycles[{3, 43, 35, 14}, {5, 45, 37, 21}, {8, 48, 40, 29},
            {15, 17, 32, 30}, {16, 23, 31, 22}];
        rot6 = Cycles[{24, 27, 30, 43}, {25, 28, 31, 42}, {26, 29, 32, 41},
            {33, 35, 40, 38}, {34, 37, 39, 36}];
        RubikGS = GenSet[rot1, rot2, rot3, rot4, rot5, rot6];
```

It is customary to define the following face−turns (the initials stand for Front, Down, Left, Up, Right, Back for obvious reasons):

```
In[153]:=
        faceturns = {
            F1 → rot1, F2 → Permute[rot1, rot1], Fp → InversePerm[rot1],
            D1 → rot2, D2 → Permute[rot2, rot2], Dp → InversePerm[rot2],
            L1 → rot3, L2 → Permute[rot3, rot3], Lp → InversePerm[rot3],
            U1 → rot4, U2 → Permute[rot4, rot4], Up → InversePerm[rot4],
            R1 → rot5, R2 → Permute[rot5, rot5], Rp → InversePerm[rot5],
            B1 → rot6, B2 → Permute[rot6, rot6], Bp → InversePerm[rot6]};

In[154]:=
        RubikPoints = {f, f, f, f, f, f, f, f, l, l, l, d, d, d, r, r, r, l, l, d, d, r,
            r, l, l, l, d, d, d, r, r, r, b, b, b, b, b, b, b, b, u, u, u, u, u, u, u, u};
```

With a normal Linux box we can compute a strong generating set in a few seconds:

```
In[155]:=
      myTiming[RubikSGS = SchreierSims[{}, RubikGS, 48];]

          4.070369 Second
```

We translate it to `Cycles` notation. We see that there are 18 points in the base and 99 generators (a SGS with only 19 generators is known for the Rubik’s group):

```
In[156]:=
      RubikSGS = TranslatePerm[RubikSGS, Cycles];

In[157]:=
      Length /@ RubikSGS

Out[157]=
      StrongGenSet[18, 99]
```

Reordering the base we can get this reduced SGS (this is taken from Butler’s book):

```
In[158]:=
      Rubikbase2 = {1, 6, 3, 8, 21, 23, 26, 5, 29, 19, 7, 24, 25, 28, 31, 18, 4, 2};
      RubikGS2 = GenSet[
         Cycles[{1, 3, 8, 6}, {2, 5, 7, 4}, {9, 48, 15, 12}, {10, 47, 16, 13}, {11, 46, 17, 14}],
         Cycles[{6, 15, 35, 26}, {7, 22, 34, 19},
          {8, 30, 33, 11}, {12, 14, 29, 27}, {13, 21, 28, 20}],
         Cycles[{3, 43, 35, 14}, {5, 45, 37, 21}, {8, 48, 40, 29},
          {15, 17, 32, 30}, {16, 23, 31, 22}],
         Cycles[{5, 37, 28, 21}, {8, 26, 29, 32}, {14, 33, 35, 40},
          {15, 27, 30, 43}, {16, 31, 34, 22}],
         Cycles[{5, 45, 13, 37, 21}, {7, 31, 22, 16, 23}, {26, 27, 33}, {29, 35, 30}],
         Cycles[{19, 23, 34}, {20, 45, 28}],
         Cycles[{24, 27, 30, 43}, {25, 28, 31, 42},
          {26, 29, 32, 41}, {33, 35, 40, 38}, {34, 37, 39, 36}],
         Cycles[{5, 28, 37}, {16, 34, 31}],
         Cycles[{2, 28, 47, 34}, {24, 41, 38},
          {25, 39, 31, 36, 42, 37}, {29, 43, 30, 40, 35, 32}],
         Cycles[{19, 31, 34}, {20, 37, 28}],
         Cycles[{7, 31, 34}, {13, 37, 28}],
         Cycles[{24, 41, 38}, {32, 43, 40}],
         Cycles[{24, 40}, {25, 39, 34, 37}, {28, 31, 36, 42}, {32, 38}, {41, 43}],
         Cycles[{25, 28, 31}, {34, 37, 36}],
         Cycles[{2, 31, 34}, {28, 47, 37}],
         Cycles[{2, 31, 39}, {37, 42, 47}],
         Cycles[{2, 18, 39}, {42, 47, 44}],
         Cycles[{2, 10, 39}, {4, 42, 47}],
         Cycles[{2, 47}, {39, 42}]];
```

We check that it is a strong generating set indeed:

```
In[160]:=
      myTiming[RubikSGS2 = SchreierSims[Rubikbase2, RubikGS2, 48];]

          0.939531 Second

In[161]:=
      RubikSGS2 = TranslatePerm[RubikSGS2, Cycles];
```

```
In[162]:=
     Length /@ RubikSGS2

Out[162]=
     StrongGenSet[18, 19]
```

As we said, it is a huge group:

```
In[163]:=
     order = OrderOfGroup[RubikSGS2]

Out[163]=
     43252003274489856000

In[164]:=
     FactorInteger[order]

Out[164]=
     {{2, 27}, {3, 14}, {5, 3}, {7, 2}, {11, 1}}
```

Orders of the chain of stabilizers:

```
In[165]:=
     RubikSGS2Chain = StabilizerChain[RubikSGS2];

In[166]:=
     OrderOfGroup /@ RubikSGS2Chain

Out[166]=
     {43252003274489856000, 1802166803103744000, 85817466814464000,
      4767637045248000, 317842469683200, 13243436236800, 601974374400, 50164531200,
      2508226560, 278691840, 15482880, 967680, 161280, 11520, 960, 96, 12, 2, 1}

In[167]:=
     basicindices = Drop[% / RotateLeft[%], -1]

Out[167]=
     {24, 21, 18, 15, 24, 22, 12, 20, 9, 18, 16, 6, 14, 12, 10, 8, 6, 2}
```

Now we can check potential configurations:

A rotation in a single corner or the exchange of two side−neighbours are not possible:

```
In[168]:=
     PermMemberQ[Cycles[{8, 15, 14}], RubikSGS]

Out[168]=
     False

In[169]:=
     PermMemberQ[Cycles[{10, 4}], RubikSGS]

Out[169]=
     False
```

But pairs of them are allowed, in adequate directions:

```
In[170]:=
      PermMemberQ[Cycles[{10, 4}, {7, 13}], RubikSGS]

Out[170]=
      True

In[171]:=
      PermMemberQ[Cycles[{8, 15, 14}, {24, 41, 38}], RubikSGS]

Out[171]=
      True

In[172]:=
      PermMemberQ[Cycles[{8, 15, 14}, {24, 38, 41}], RubikSGS]

Out[172]=
      False
```

A combination of both is impossible:

```
In[173]:=
      PermMemberQ[Cycles[{10, 4}, {8, 15, 14}], RubikSGS]

Out[173]=
      False
```

There is the famous question on the minimum number of moves which allows solving any configuration of the Rubik cube. Here it is very important to distinguish between face turns and quarter turns. An f−turn can be equal to one or two q−turns depending on the case. In what follows we only consider q−turns. There is a simple lower bound to the minimum number of q−turns required to solve any configuration, obtained by branching over 7 possibilities (the identity and the six rotations) that number of times: 24

24 is the first power of 7 which is larger than the order of the group:

```
In[174]:=
      order / 7^24 // N

Out[174]=
      0.225763
```

The superflip position requires 24 moves (see wikipedia):

```
In[175]:=
      superflip = Cycles[{2, 47}, {4, 10}, {7, 13}, {5, 16}, {20, 19},
         {21, 22}, {28, 34}, {18, 44}, {25, 36}, {45, 23}, {42, 39}, {31, 37}];

In[176]:=
      PermMemberQ[superflip, RubikSGS2]

Out[176]=
      True
```

Interestingly, this is the only nontrivial permutation which commutes with every other permutation in the Rubik group:

```
In[177]:=
     PermEqual[Permute[superflip, rot1, superflip], rot1]

Out[177]=
     True

In[178]:=
     PermEqual[Permute[superflip, rot2, superflip], rot2]

Out[178]=
     True

In[179]:=
     PermEqual[Permute[superflip, rot3, superflip], rot3]

Out[179]=
     True

In[180]:=
     PermEqual[Permute[superflip, rot4, superflip], rot4]

Out[180]=
     True

In[181]:=
     PermEqual[Permute[superflip, rot5, superflip], rot5]

Out[181]=
     True

In[182]:=
     PermEqual[Permute[superflip, rot6, superflip], rot6]

Out[182]=
     True
```

A position requiring 26 q−turns is known.

As of 2008, the best upper bounds known are 26 f−turns and 35 q−turns to solve arbitrary configurations of the cube.

Another frequent question whether a SGS can be used to actually solve an arbitrary configuration of the Rubik cube and the answer is of course positive. The algorithm PermWord can decompose any configuration in a product of 19 coset representatives, but then we need to decompose each of those 19 representatives in terms of the original 6 rotations. The final algorithm is, however, far from optimal, and actually very simple "human" algorithms perform better than this.

There are 239 possible nontrivial coset representatives:

```
In[183]:=
     Plus @@ basicindices - 18

Out[183]=
     239
```

Let us construct the canonical representatives:

```
In[184]:=
      reprs = {};
      Do[
        orbit = Orbit[Rubikbase2[[-count]], RubikSGS2Chain[[-count - 1, 2]]];
        sorbit = SchreierOrbit[Rubikbase2[[-count]], RubikSGS2Chain[[-count - 1, 2]]];
        AppendTo[reprs, Drop[TraceSchreier[#, sorbit] & /@ orbit, 1]],
        {count, 1, 18}];
      reprs = Flatten[reprs];
```

Those moves can be expressed in terms of face turns. For simplicity, we use an external solver to do it (we use kociemba.org). This function transforms a permutation into a chain of facelets that can be fed to that solver:

```
In[187]:=
      tochain[list_] := ToSymbol @@ {list[[41]], list[[42]], list[[43]],
          list[[44]], u, list[[45]], list[[46]], list[[47]], list[[48]],
          list[[17]], list[[23]], list[[32]], list[[16]], r,
          list[[31]], list[[15]], list[[22]], list[[30]],
          list[[1]], list[[2]], list[[3]], list[[4]], f,
          list[[5]], list[[6]], list[[7]], list[[8]],
          list[[12]], list[[13]], list[[14]], list[[20]], d,
          list[[21]], list[[27]], list[[28]], list[[29]],
          list[[24]], list[[18]], list[[9]], list[[25]], l,
          list[[10]], list[[26]], list[[19]], list[[11]],
          list[[40]], list[[39]], list[[38]], list[[37]], b,
          list[[36]], list[[35]], list[[34]], list[[33]]}
```

For example:

```
In[188]:=
      tochain[PermuteList[RubikPoints, InversePerm@reprs[[235]]]]

Out[188]=
      uuduudurrubbdrrdrdrbbffffffllldddrflllflluflurbbubbbrd
```

and relate them with the collections of face−turns that we have previously stored in a file:

```
In[189]:=
      (RubikFaceTurns = << "data/RubikFaceTurns") // ColumnForm
```

Check that the relations are correct:

```
In[190]:=
      Apply[And, RubikFaceTurns /. permute → Permute /. faceturns /. Rule → PermEqual]

Out[190]=
      True
```

7.2. Alternating groups

The alternating group A(n>3), of order n!/2, can be generated using two permutaions of degree n. They are always

```
In[191]:=
      AltGS[n_] := GenSet[Cycles[{1, 2, 3}], Cycles[Range[2, n]]]
```

The largest alternating group explicitly described in the Atlas is A(14):

```
In[192]:=
      myTiming[A14SGS = SchreierSims[{}, AltGS[14], 14];]

          0.013386 Second

In[193]:=
      Length /@ A14SGS

Out[193]=
      StrongGenSet[12, 22]

In[194]:=
      OrderOfGroup[A14SGS]

Out[194]=
      43589145600
```

With xPerm` we can manipulate within seconds groups which are much larger

```
In[195]:=
      myTiming[A30SGS = SchreierSims[{}, AltGS[30], 30];]

          0.922208 Second

In[196]:=
      Length /@ A30SGS

Out[196]=
      StrongGenSet[28, 54]

In[197]:=
      OrderOfGroup[A30SGS]

Out[197]=
      132626429906095529318154240000000
```

Going further takes almost a minute:

```
In[198]:=
      myTiming[A50SGS = SchreierSims[{}, AltGS[50], 50];]

          31.952438 Second

In[199]:=
      Length /@ A50SGS

Out[199]=
      StrongGenSet[48, 94]

In[200]:=
      OrderOfGroup[A50SGS]

Out[200]=
      15207046600856689021806304083032384422188820784480256000000000000
```

```
In[201]:=
      % // N

Out[201]=
      1.5207 × 10^64
```

7.2. Some "small" sporadic simple groups

As further examples of manipulation of nontrivial finite groups, here you find the construction of strong generating sets for some of the 26 sporadic simple groups:

M24 is the largest of the five sporadic simple Mathieu groups, with a permutation representation of degree 24. It was found by Mathieu in 1873. Frobenious showed that the other four simple Mathieu groups (M11, M12, M22, M23) are subgroups of M24.

```
In[202]:=
      g1 = Images[
          {4, 7, 17, 1, 13, 9, 2, 15, 6, 19, 18, 21, 5, 16, 8, 14, 3, 11, 10, 24, 12, 23, 22, 20}];
      g2 = Images[{4, 21, 9, 6, 18, 1, 7, 8, 15, 5, 11, 12, 17,
           2, 3, 13, 16, 10, 24, 20, 14, 22, 19, 23}];

In[204]:=
      PermDeg /@ {g1, g2}

Out[204]=
      {24, 24}

In[205]:=
      myTiming[Mathieu24SGS = SchreierSims[{}, GenSet[g1, g2], 24];]

          0.011070 Second

In[206]:=
      Length /@ Mathieu24SGS

Out[206]=
      StrongGenSet[7, 13]

In[207]:=
      OrderOfGroup[Mathieu24SGS]

Out[207]=
      244823040
```

The sixth sporadic simple group was discovered by Z. Janko in 1965 and it is called Janko1. It can be given in terms of a permutation representation of degree 266:

```
In[208]:=
      g1 = Cycles[{1, 262}, {2, 107}, {3, 21}, {4, 213}, {5, 191}, {6, 22}, {7, 133}, {8, 234},
         {9, 232}, {10, 151}, {11, 139}, {12, 176}, {13, 202}, {14, 253}, {15, 222},
         {17, 195}, {18, 206}, {19, 68}, {20, 55}, {23, 179}, {24, 217}, {25, 216}, {26, 256},
         {27, 87}, {28, 70}, {29, 131}, {30, 44}, {31, 105}, {32, 170}, {33, 77}, {34, 104},
         {35, 198}, {36, 137}, {37, 243}, {38, 56}, {39, 124}, {40, 223}, {41, 134},
         {43, 174}, {46, 51}, {47, 128}, {48, 94}, {49, 250}, {50, 264}, {52, 183},
         {53, 231}, {54, 115}, {57, 85}, {58, 233}, {59, 261}, {60, 95}, {61, 235},
         {62, 177}, {63, 249}, {64, 91}, {65, 247}, {66, 155}, {69, 219}, {71, 237},
         {72, 211}, {73, 84}, {74, 192}, {75, 130}, {76, 251}, {79, 260}, {80, 112},
         {81, 193}, {82, 156}, {83, 242}, {86, 238}, {88, 143}, {89, 168}, {90, 148},
         {92, 119}, {93, 212}, {96, 150}, {97, 199}, {98, 140}, {99, 189}, {100, 180},
         {101, 147}, {102, 111}, {103, 159}, {106, 162}, {108, 194}, {109, 166}, {110, 200},
         {113, 120}, {114, 141}, {116, 182}, {117, 181}, {118, 225}, {121, 254}, {122, 125},
         {123, 146}, {126, 208}, {127, 221}, {129, 210}, {132, 255}, {136, 175}, {138, 207},
         {142, 240}, {144, 172}, {145, 185}, {149, 224}, {152, 169}, {153, 241}, {154, 190},
         {157, 214}, {158, 161}, {160, 236}, {163, 239}, {164, 229}, {165, 230}, {167, 188},
         {171, 258}, {173, 186}, {178, 245}, {184, 205}, {187, 228}, {197, 203}, {201, 252},
         {209, 248}, {215, 259}, {218, 246}, {220, 227}, {257, 263}, {265, 266}];
      g2 = Cycles[{1, 146, 21}, {2, 132, 82}, {4, 156, 166}, {5, 242, 253}, {6, 107, 28},
         {7, 125, 76}, {8, 245, 130}, {9, 174, 42}, {10, 241, 244}, {11, 264, 63},
         {12, 248, 234}, {13, 36, 44}, {14, 116, 128}, {15, 47, 25}, {16, 178, 112},
         {17, 170, 110}, {18, 197, 74}, {19, 233, 180}, {20, 121, 96}, {22, 228, 155},
         {23, 48, 173}, {24, 201, 187}, {26, 136, 190}, {27, 212, 94}, {29, 175, 52},
         {30, 77, 32}, {31, 237, 34}, {33, 226, 90}, {35, 129, 54}, {37, 161, 114},
         {38, 232, 87}, {39, 219, 192}, {40, 78, 159}, {41, 139, 71}, {43, 211, 251},
         {45, 222, 240}, {46, 97, 135}, {49, 70, 131}, {50, 153, 200}, {51, 186, 209},
         {53, 203, 216}, {55, 169, 64}, {56, 140, 230}, {57, 260, 118}, {58, 91, 243},
         {59, 199, 227}, {60, 108, 164}, {61, 208, 101}, {62, 206, 106}, {65, 103, 66},
         {67, 95, 205}, {68, 73, 225}, {69, 151, 113}, {72, 221, 152}, {75, 143, 202},
         {79, 217, 254}, {80, 93, 122}, {81, 181, 252}, {83, 258, 126}, {84, 163, 177},
         {85, 154, 213}, {86, 182, 196}, {88, 133, 215}, {89, 117, 247}, {92, 191, 160},
         {99, 229, 263}, {100, 138, 188}, {102, 194, 157}, {105, 149, 184}, {109, 123, 193},
         {111, 137, 183}, {115, 238, 235}, {119, 167, 147}, {120, 134, 189}, {124, 185, 265},
         {127, 218, 261}, {141, 231, 210}, {142, 239, 236}, {144, 224, 249},
         {145, 158, 220}, {148, 214, 172}, {150, 250, 259}, {162, 257, 256},
         {165, 179, 246}, {176, 195, 266}, {198, 204, 207}, {223, 262, 255}];

In[210]:=
      PermDeg /@ {g1, g2}

Out[210]=
      {266, 266}

In[211]:=
      myTiming[Janko1SGS = SchreierSims[{}, GenSet[g1, g2], 266];]

         0.162497 Second

In[212]:=
      Length /@ Janko1SGS

Out[212]=
      StrongGenSet[3, 7]

In[213]:=
      OrderOfGroup[Janko1SGS]

Out[213]=
      175560
```

The Janko2 group can be given in terms of a permutation representation of degree 100:

```
In[214]:=
     g1 = Cycles[{1, 84}, {2, 20}, {3, 48}, {4, 56}, {5, 82}, {6, 67}, {7, 55},
        {8, 41}, {9, 35}, {10, 40}, {11, 78}, {12, 100}, {13, 49}, {14, 37}, {15, 94},
        {16, 76}, {17, 19}, {18, 44}, {21, 34}, {22, 85}, {23, 92}, {24, 57},
        {25, 75}, {26, 28}, {27, 64}, {29, 90}, {30, 97}, {31, 38}, {32, 68},
        {33, 69}, {36, 53}, {39, 61}, {42, 73}, {43, 91}, {45, 86}, {46, 81},
        {47, 89}, {50, 93}, {51, 96}, {52, 72}, {54, 74}, {58, 99}, {59, 95},
        {60, 63}, {62, 83}, {65, 70}, {66, 88}, {71, 87}, {77, 98}, {79, 80}];
     g2 = Cycles[{1, 80, 22}, {2, 9, 11}, {3, 53, 87}, {4, 23, 78}, {5, 51, 18},
        {6, 37, 24}, {8, 27, 60}, {10, 62, 47}, {12, 65, 31}, {13, 64, 19}, {14, 61, 52},
        {15, 98, 25}, {16, 73, 32}, {17, 39, 33}, {20, 97, 58}, {21, 96, 67}, {26, 93, 99},
        {28, 57, 35}, {29, 71, 55}, {30, 69, 45}, {34, 86, 82}, {38, 59, 94},
        {40, 43, 91}, {42, 68, 44}, {46, 85, 89}, {48, 76, 90}, {49, 92, 77},
        {50, 66, 88}, {54, 95, 56}, {63, 74, 72}, {70, 81, 75}, {79, 100, 83}];

In[216]:=
     PermDeg /@ {g1, g2}

Out[216]=
     {100, 100}

In[217]:=
     myTiming[Janko2SGS = SchreierSims[{}, GenSet[g1, g2], 100];]

        0.026291 Second

In[218]:=
     Length /@ Janko2SGS

Out[218]=
     StrongGenSet[4, 8]

In[219]:=
     OrderOfGroup[Janko2SGS]

Out[219]=
     604800
```

This is the smallest of the three sporadic simple Conway groups, with a permutation representation of degree 276:

```
In[220]:=
      g1 = Images[{245, 42, 112, 15, 131, 7, 188, 75, 132, 10, 11, 187, 186, 265, 22, 159, 256,
           43, 101, 123, 134, 4, 32, 209, 238, 35, 45, 235, 126, 5, 19, 60, 66, 80, 154,
           251, 117, 206, 71, 118, 93, 87, 167, 271, 221, 261, 182, 155, 47, 230, 172, 236,
           109, 191, 76, 156, 73, 116, 147, 23, 127, 231, 38, 53, 122, 210, 24, 68, 86, 255,
           196, 139, 149, 21, 111, 203, 252, 72, 262, 114, 214, 9, 181, 174, 85, 95, 2, 250,
           257, 243, 90, 158, 170, 148, 69, 105, 249, 263, 16, 54, 31, 115, 51, 104, 125,
           219, 92, 46, 64, 204, 8, 266, 225, 34, 175, 145, 161, 180, 237, 241, 224, 169,
           269, 12, 96, 129, 189, 190, 29, 17, 30, 82, 143, 74, 168, 13, 227, 217, 78, 258,
           220, 178, 228, 146, 58, 254, 273, 215, 57, 106, 77, 110, 50, 26, 248, 260, 274,
           107, 99, 253, 37, 25, 272, 44, 52, 119, 18, 201, 65, 41, 233, 103, 246, 200,
           102, 160, 198, 207, 157, 40, 223, 49, 267, 79, 1, 136, 124, 6, 61, 268, 100, 70,
           98, 171, 121, 39, 62, 211, 208, 84, 135, 97, 55, 152, 141, 63, 142, 259, 67, 33,
           177, 173, 14, 242, 94, 113, 240, 264, 150, 205, 27, 183, 83, 195, 216, 163, 247,
           133, 36, 153, 197, 140, 194, 120, 270, 165, 166, 162, 218, 138, 234, 81, 91,
           89, 185, 212, 137, 48, 202, 276, 229, 151, 176, 144, 192, 130, 244, 232, 199,
           56, 108, 184, 193, 239, 213, 3, 222, 128, 20, 28, 164, 226, 59, 179, 275, 88}];
      g2 = Images[{204, 203, 33, 236, 5, 172, 77, 76, 47, 146, 133, 224, 229, 53, 84, 16,
           223, 228, 130, 131, 252, 190, 13, 263, 242, 10, 32, 196, 199, 65, 246, 209, 40,
           99, 241, 198, 269, 251, 75, 118, 176, 271, 183, 116, 197, 238, 22, 29, 178, 26,
           174, 129, 2, 153, 272, 257, 41, 12, 59, 20, 27, 175, 106, 159, 218, 259, 137, 258,
           261, 164, 262, 189, 45, 177, 260, 85, 25, 15, 226, 96, 24, 1, 274, 148, 264, 132,
           48, 117, 36, 60, 171, 201, 101, 253, 95, 120, 142, 213, 165, 51, 115, 44, 103,
           167, 243, 66, 141, 108, 88, 97, 276, 30, 139, 222, 166, 173, 231, 3, 73, 239,
           56, 170, 82, 162, 163, 207, 145, 128, 52, 104, 90, 216, 220, 155, 74, 237, 28,
           4, 113, 273, 230, 270, 248, 180, 206, 50, 250, 78, 127, 150, 54, 232, 217, 121,
           69, 156, 6, 125, 210, 86, 89, 46, 184, 211, 265, 93, 19, 138, 23, 126, 43, 188,
           102, 244, 219, 192, 256, 83, 58, 144, 181, 187, 91, 158, 205, 235, 147, 157, 114,
           9, 152, 57, 39, 64, 143, 67, 119, 161, 87, 200, 111, 79, 14, 123, 21, 149, 122,
           191, 61, 194, 266, 225, 31, 81, 62, 160, 151, 112, 215, 254, 234, 72, 17, 179,
           105, 267, 227, 18, 169, 249, 109, 208, 275, 68, 233, 168, 55, 124, 80, 240, 35,
           7, 212, 100, 245, 98, 195, 247, 107, 182, 42, 185, 94, 11, 255, 135, 154, 221,
           63, 193, 134, 71, 214, 8, 34, 70, 202, 268, 37, 110, 38, 136, 140, 49, 186, 92}];

In[222]:=
      PermDeg /@ {g1, g2}

Out[222]=
      {276, 276}

In[223]:=
      myTiming[Conway3SGS = SchreierSims[{}, GenSet[g1, g2], 276];]

         2.768254 Second

In[224]:=
      Length /@ Conway3SGS

Out[224]=
      StrongGenSet[6, 13]

In[225]:=
      OrderOfGroup[Conway3SGS]

Out[225]=
      495766656000
```

`xPerm'` can also handle bigger sporadic groups, but it takes very long, or much memory:

This is the smallest of the four sporadic simple Fischer groups. It can be given in terms of a permutation representation of degree 3510. The computation of a SGS requires 400 Mbyte RAM. (The SchreierSims command is deactivated; we read the result from a file.)

```
In[226]:=
      g1 = Images[<< "data/F22p3510.1"];
      g2 = Images[<< "data/F22p3510.2"];

In[228]:=
      PermDeg /@ {g1, g2}

Out[228]=
      {3510, 3510}

In[228]:=
      myTiming[Fischer22SGS = SchreierSims[{}, GenSet[g1, g2], 3510];]

          3607.283267 Second

In[230]:=
      base = {1, 2, 3, 5, 4};
      GS = << "data/Fischer22SGS";
      Fischer22SGS = StrongGenSet[base, GS];

In[233]:=
      Length /@ Fischer22SGS

Out[233]=
      StrongGenSet[5, 18]

In[234]:=
      OrderOfGroup[Fischer22SGS]

Out[234]=
      64561751654400
```

The Janko3 group can be given in terms of a permutation representation of degree 6156. This computation requires almost 900 Mbyte of memory:

```
In[235]:=
      g1 = << "data/J3G1-p6156B0.g1";
      g2 = << "data/J3G1-p6156B0.g2";

In[237]:=
      PermDeg /@ {g1, g2}

Out[237]=
      {6156, 6156}

In[238]:=
      myTiming[Janko3SGS = SchreierSims[{}, GenSet[g1, g2], 6156];]

          894.421476 Second

In[239]:=
      Length /@ Janko3SGS

Out[239]=
      StrongGenSet[3, 7]
```

```
In[240]:=
      OrderOfGroup[Janko3SGS]

Out[240]=
      50232960
```

7.3. Other groups

These are groups taken from the Atlas of Finite Group Representations (`http://for.mat.bham.ac.uk/atlas/`), in the section Miscellaneous Groups.

This is the exceptional twisted group Sz8 (the Suzuki group), which can be given in terms of permutations on 65 points.

```
In[241]:=
      g1 = Images[{2, 1, 4, 3, 7, 9, 5, 12, 6, 13, 15, 8, 10, 19, 11, 21, 23, 25, 14, 28, 16, 31,
           17, 33, 18, 35, 32, 20, 37, 39, 22, 27, 24, 43, 26, 46, 29, 48, 30, 40, 51, 44, 34, 42,
           55, 36, 50, 38, 58, 47, 41, 60, 61, 59, 45, 62, 63, 49, 54, 52, 53, 56, 57, 65, 64}];
      g2 = Images[{3, 2, 5, 6, 8, 10, 11, 1, 12, 14, 16, 17, 18, 4, 20, 22, 24, 26, 27, 29, 30, 7,
           32, 9, 34, 36, 31, 19, 38, 40, 28, 41, 42, 44, 45, 13, 47, 15, 49, 50, 52, 53, 33, 54,
           56, 37, 51, 57, 59, 21, 46, 23, 43, 25, 58, 63, 55, 48, 60, 39, 64, 65, 35, 62, 61}];

In[243]:=
      PermDeg /@ {g1, g2}

Out[243]=
      {65, 65}

In[244]:=
      myTiming[Sz8SGS = SchreierSims[{}, GenSet[g1, g2], 65];]

          0.009850 Second

In[245]:=
      Length /@ Sz8SGS

Out[245]=
      StrongGenSet[3, 5]

In[246]:=
      OrderOfGroup[Sz8SGS]

Out[246]=
      29120
```

This is the exceptional untwisted group G2(4), which can be given in terms of permutations on 416 points.

```
In[247]:=
      g1 =
        Images[{2, 1, 5, 7, 3, 10, 4, 13, 12, 6, 11, 9, 8, 20, 22, 24, 26, 27, 29, 14, 32, 15, 23,
          16, 36, 17, 18, 40, 19, 43, 31, 21, 47, 49, 50, 25, 53, 55, 57, 28, 60, 42, 30, 62, 64,
          66, 33, 69, 34, 35, 72, 74, 37, 77, 38, 79, 39, 82, 84, 41, 87, 44, 90, 45, 92, 46, 95,
          97, 48, 100, 101, 51, 104, 52, 107, 76, 54, 111, 56, 109, 114, 58, 117, 59, 120, 122,
          61, 125, 89, 63, 128, 65, 131, 132, 67, 135, 68, 138, 99, 70, 71, 140, 142, 73, 143,
          144, 75, 147, 80, 150, 78, 153, 155, 81, 156, 158, 83, 133, 161, 85, 164, 86, 166,
          167, 88, 170, 171, 91, 174, 176, 93, 94, 118, 181, 96, 184, 137, 98, 186, 102, 188,
          103, 105, 106, 145, 192, 108, 195, 196, 110, 179, 200, 112, 203, 113, 115, 157, 116,
          208, 209, 119, 212, 211, 121, 165, 123, 124, 217, 219, 126, 127, 223, 225, 129, 228,
          130, 230, 232, 151, 235, 134, 238, 183, 136, 241, 139, 244, 141, 193, 248, 250, 146,
          189, 252, 148, 149, 254, 255, 257, 152, 220, 259, 154, 262, 214, 240, 264, 159,
          160, 267, 163, 162, 221, 205, 263, 274, 168, 276, 169, 201, 213, 224, 172, 222,
          173, 280, 282, 175, 247, 177, 266, 178, 287, 289, 180, 277, 291, 182, 294, 206,
          185, 284, 278, 187, 301, 302, 229, 190, 304, 191, 307, 194, 308, 197, 198, 311,
          199, 313, 202, 275, 261, 204, 215, 207, 319, 231, 210, 322, 324, 326, 327, 328,
          330, 216, 260, 218, 236, 243, 333, 226, 281, 227, 336, 242, 339, 340, 233, 299,
          234, 343, 237, 345, 347, 239, 349, 296, 351, 352, 288, 354, 245, 246, 356, 249,
          359, 306, 251, 253, 362, 310, 256, 365, 258, 314, 315, 316, 368, 370, 265, 373,
          350, 268, 375, 269, 325, 270, 271, 272, 377, 273, 338, 376, 279, 379, 363, 283,
          348, 331, 285, 286, 384, 383, 290, 353, 292, 387, 293, 337, 295, 321, 297, 298,
          344, 300, 392, 303, 357, 395, 305, 396, 389, 309, 335, 398, 312, 381, 397, 317,
          369, 318, 399, 380, 320, 401, 323, 332, 329, 403, 334, 372, 366, 382, 342, 341,
          405, 393, 346, 388, 361, 390, 394, 355, 386, 391, 358, 360, 367, 364, 371, 400,
          374, 407, 378, 411, 385, 408, 402, 406, 409, 410, 404, 412, 414, 413, 415, 416}];
      g2 = Images[{3, 4, 6, 8, 9, 11, 12, 14, 15, 16, 17, 18, 19, 21, 23, 25, 1, 28, 30, 31,
          2, 33, 34, 35, 37, 38, 39, 41, 42, 44, 45, 46, 48, 5, 51, 52, 54, 56, 58, 59, 7,
          61, 22, 63, 65, 67, 68, 70, 71, 50, 73, 75, 76, 10, 78, 80, 81, 83, 85, 86, 88,
          89, 13, 91, 93, 94, 96, 98, 99, 43, 102, 103, 105, 106, 108, 109, 110, 60, 112,
          113, 115, 116, 118, 119, 121, 123, 124, 126, 82, 127, 129, 130, 20, 133, 134,
          136, 137, 139, 140, 64, 47, 141, 107, 104, 24, 145, 146, 148, 149, 151, 152, 154,
          26, 74, 157, 159, 160, 27, 162, 163, 165, 90, 55, 168, 169, 29, 172, 173, 175,
          177, 178, 179, 180, 182, 183, 32, 185, 132, 101, 187, 143, 189, 49, 190, 191,
          193, 194, 36, 197, 198, 199, 201, 202, 204, 77, 205, 206, 207, 62, 210, 211,
          213, 214, 79, 40, 215, 216, 218, 220, 221, 222, 224, 226, 227, 100, 229, 231,
          233, 234, 236, 237, 239, 166, 240, 242, 243, 245, 246, 247, 249, 114, 251, 72,
          238, 230, 253, 53, 256, 155, 195, 258, 260, 261, 164, 263, 57, 265, 266, 181,
          268, 269, 270, 271, 272, 273, 228, 275, 267, 219, 241, 277, 278, 279, 122, 262,
          281, 283, 250, 284, 235, 285, 286, 288, 290, 287, 66, 292, 293, 295, 296, 297,
          298, 299, 300, 69, 252, 223, 303, 305, 306, 188, 192, 309, 208, 310, 212, 312,
          314, 315, 316, 276, 317, 244, 318, 320, 321, 87, 323, 325, 150, 84, 329, 135,
          259, 313, 331, 328, 332, 142, 304, 334, 335, 337, 338, 92, 341, 200, 342, 282,
          138, 344, 346, 322, 348, 95, 350, 125, 97, 353, 156, 355, 170, 357, 358, 360,
          167, 361, 161, 363, 280, 364, 366, 294, 111, 347, 367, 369, 371, 372, 365, 374,
          147, 117, 339, 308, 171, 376, 302, 120, 352, 378, 326, 377, 128, 380, 381, 382,
          362, 383, 340, 327, 131, 379, 385, 386, 209, 388, 217, 319, 389, 390, 373, 391,
          264, 393, 225, 394, 255, 307, 144, 397, 176, 392, 291, 158, 351, 153, 289, 356,
          248, 400, 398, 387, 254, 349, 232, 402, 203, 345, 368, 333, 174, 404, 384, 311,
          403, 406, 274, 184, 257, 186, 196, 407, 370, 408, 396, 409, 375, 405, 354, 410,
          336, 343, 412, 413, 330, 411, 401, 359, 414, 301, 324, 415, 395, 416, 399}];
```

```
In[249]:=
      PermDeg /@ {g1, g2}

Out[249]=
      {414, 416}
```

```
In[250]:=
      myTiming[G24SGS = SchreierSims[{}, GenSet[g1, g2], 416];]

          3.204673 Second
```

```
In[251]:=
     Length /@ G24SGS

Out[251]=
     StrongGenSet[4, 11]

In[252]:=
     OrderOfGroup[G24SGS]

Out[252]=
     251596800
```

The largest symplectic group given in the Atlas is S10. It can be represented using permutations of 496 points.

```
In[253]:=
      g1 =
        Images[{1, 4, 3, 2, 5, 6, 7, 8, 11, 10, 9, 12, 15, 17, 13, 16, 14, 18, 25, 20, 21, 22, 30,
          24, 19, 26, 35, 28, 38, 23, 31, 32, 33, 34, 27, 36, 37, 29, 39, 40, 41, 42, 43, 44, 54,
          48, 47, 46, 49, 50, 51, 61, 60, 45, 55, 66, 57, 68, 69, 53, 52, 62, 63, 64, 74, 56, 77,
          58, 59, 70, 80, 81, 83, 65, 87, 76, 67, 78, 93, 71, 72, 82, 73, 84, 101, 86, 75, 88,
          89, 90, 108, 92, 79, 94, 113, 115, 105, 118, 112, 121, 85, 116, 126, 104, 97, 106,
          114, 91, 109, 110, 136, 99, 95, 107, 96, 102, 145, 98, 119, 149, 100, 122, 123, 124,
          155, 103, 153, 128, 129, 130, 163, 132, 133, 134, 135, 111, 166, 138, 139, 140, 141,
          142, 143, 144, 117, 146, 182, 184, 120, 150, 189, 191, 127, 154, 125, 156, 157, 158,
          159, 160, 161, 162, 131, 164, 165, 137, 167, 168, 169, 170, 171, 172, 173, 210, 175,
          213, 177, 178, 217, 180, 181, 147, 183, 148, 190, 186, 224, 188, 151, 185, 152, 192,
          231, 211, 195, 234, 197, 198, 199, 200, 240, 202, 214, 243, 205, 206, 207, 208,
          209, 174, 194, 212, 176, 203, 215, 216, 179, 218, 259, 220, 262, 222, 223, 187,
          225, 226, 227, 228, 271, 273, 193, 232, 276, 196, 235, 236, 237, 283, 284, 201,
          241, 288, 204, 244, 292, 246, 247, 295, 249, 296, 251, 252, 253, 254, 255, 301,
          257, 258, 219, 279, 307, 221, 263, 311, 265, 266, 314, 268, 316, 318, 229, 285,
          230, 325, 291, 233, 329, 278, 260, 280, 333, 282, 238, 239, 272, 286, 287, 242,
          342, 343, 275, 245, 293, 348, 248, 250, 297, 353, 299, 355, 256, 302, 303, 304,
          305, 306, 261, 358, 324, 366, 264, 312, 313, 267, 315, 269, 317, 270, 369, 320,
          321, 322, 379, 309, 274, 382, 327, 328, 277, 330, 331, 387, 281, 334, 388, 346,
          337, 338, 339, 340, 341, 289, 290, 384, 345, 336, 397, 294, 349, 350, 351, 400,
          298, 402, 300, 404, 357, 308, 359, 360, 361, 411, 363, 364, 365, 310, 416, 417,
          319, 370, 420, 422, 414, 374, 375, 376, 377, 378, 323, 380, 381, 326, 430, 344,
          385, 386, 332, 335, 389, 390, 391, 392, 393, 439, 395, 396, 347, 398, 399, 352,
          401, 354, 403, 356, 448, 406, 418, 408, 451, 410, 362, 412, 413, 373, 415, 367,
          368, 407, 419, 371, 459, 372, 423, 424, 462, 426, 427, 428, 429, 383, 442, 432,
          437, 434, 435, 436, 433, 438, 394, 440, 441, 431, 443, 444, 445, 446, 447, 405,
          449, 450, 409, 452, 453, 454, 455, 456, 457, 458, 421, 460, 461, 425, 463, 464,
          465, 466, 467, 468, 481, 483, 471, 485, 486, 474, 488, 476, 477, 489, 479, 480,
          469, 482, 470, 484, 472, 473, 490, 475, 478, 487, 491, 492, 493, 494, 496, 495}];
      g2 = Images[{2, 4, 1, 6, 3, 8, 5, 10, 7, 12, 13, 9, 16, 11, 19, 21, 22, 14, 26, 15,
          28, 29, 17, 18, 30, 34, 20, 37, 39, 40, 23, 24, 25, 44, 45, 27, 42, 48, 49, 50,
          31, 32, 33, 53, 38, 35, 36, 57, 58, 59, 41, 43, 56, 63, 46, 47, 67, 60, 70, 51,
          65, 52, 72, 54, 55, 75, 69, 78, 61, 71, 62, 82, 64, 85, 88, 66, 90, 92, 68, 95,
          97, 98, 91, 73, 81, 74, 103, 105, 76, 107, 77, 109, 110, 79, 114, 80, 117, 119,
          83, 84, 123, 86, 127, 87, 129, 89, 131, 132, 134, 135, 93, 94, 139, 141, 142,
          96, 146, 147, 122, 99, 151, 100, 153, 101, 102, 157, 159, 104, 161, 106, 164,
          165, 108, 167, 168, 169, 111, 112, 173, 113, 175, 176, 115, 116, 179, 181, 183,
          118, 186, 120, 190, 121, 193, 124, 195, 125, 197, 126, 199, 128, 162, 130, 201,
          188, 203, 133, 172, 205, 206, 136, 137, 138, 209, 140, 212, 214, 143, 144, 218,
          145, 156, 219, 221, 222, 148, 185, 149, 150, 226, 180, 228, 152, 232, 154, 202,
          155, 236, 158, 238, 160, 241, 163, 242, 166, 245, 246, 170, 171, 249, 250, 174,
          216, 253, 254, 177, 178, 256, 258, 260, 182, 263, 264, 184, 266, 187, 268, 189,
          270, 191, 192, 274, 275, 194, 278, 196, 281, 198, 239, 200, 285, 287, 289, 265,
          204, 248, 293, 207, 208, 295, 213, 210, 211, 299, 300, 215, 302, 217, 230, 304,
          306, 220, 309, 225, 312, 223, 313, 224, 231, 227, 319, 320, 229, 323, 326, 277,
          327, 233, 330, 234, 235, 334, 237, 255, 336, 338, 240, 332, 340, 291, 243, 244,
          345, 347, 247, 298, 350, 251, 252, 354, 283, 356, 343, 257, 359, 259, 361, 362,
          261, 365, 262, 367, 368, 369, 370, 267, 372, 269, 374, 375, 376, 271, 272, 310,
          273, 315, 383, 339, 276, 385, 341, 279, 280, 348, 335, 282, 389, 284, 366, 286,
          382, 288, 391, 393, 290, 395, 292, 349, 352, 294, 398, 296, 297, 397, 333, 403,
          405, 301, 303, 408, 305, 410, 311, 307, 308, 414, 415, 316, 394, 418, 419, 314,
          318, 317, 423, 424, 425, 321, 322, 331, 324, 325, 428, 373, 328, 431, 329, 417,
          355, 434, 337, 436, 342, 438, 344, 440, 346, 442, 443, 351, 402, 353, 446, 426,
          435, 432, 357, 358, 401, 360, 413, 452, 363, 364, 387, 384, 454, 380, 456, 457,
          458, 371, 460, 412, 427, 433, 377, 378, 379, 381, 463, 449, 386, 388, 465, 390,
          421, 392, 450, 466, 420, 396, 409, 469, 399, 400, 451, 404, 459, 406, 407, 472,
          473, 411, 475, 416, 429, 437, 476, 464, 477, 422, 445, 479, 430, 471, 448,
          439, 441, 482, 444, 447, 455, 487, 453, 462, 468, 474, 461, 490, 467, 486,
          484, 488, 470, 481, 492, 478, 493, 483, 480, 485, 495, 491, 489, 494, 496}];

In[255]:=
      PermDeg /@ {g1, g2}

Out[255]=
      {496, 495}
```

```
In[256]:=
     myTiming[S10SGS = SchreierSims[{}, GenSet[g1, g2], 496];]

        21.573289 Second

In[257]:=
     Length /@ S10SGS

Out[257]=
     StrongGenSet[10, 20]

In[258]:=
     OrderOfGroup[S10SGS]

Out[258]=
     24815256521932800
```

This is the exceptional twisted group T3D4, represented with permutations on 819 points.

```
In[259]:=
        g1 =
          Images[{2, 1, 5, 7, 3, 10, 4, 13, 15, 6, 18, 20, 8, 23, 9, 26, 28, 11, 31, 12, 34, 35, 14,
            38, 33, 16, 42, 17, 45, 47, 19, 50, 25, 21, 22, 55, 57, 24, 60, 62, 64, 27, 67, 54, 29,
            71, 30, 74, 49, 32, 78, 80, 82, 44, 36, 86, 37, 89, 91, 39, 94, 40, 96, 41, 99, 98, 43,
            103, 104, 106, 46, 109, 111, 48, 114, 116, 118, 51, 121, 52, 124, 53, 127, 129,
            131, 56, 87, 135, 58, 138, 59, 141, 143, 61, 146, 63, 148, 66, 65, 152, 154,
            156, 68, 69, 159, 70, 162, 164, 72, 165, 73, 167, 169, 75, 171, 76, 174, 77,
            177, 179, 79, 182, 183, 81, 186, 188, 83, 190, 84, 193, 85, 196, 198, 200, 88,
            203, 205, 90, 207, 209, 92, 212, 93, 215, 217, 95, 220, 97, 222, 224, 226, 100,
            229, 101, 232, 102, 235, 236, 105, 239, 241, 107, 244, 108, 110, 247, 112, 250,
            113, 253, 115, 255, 257, 117, 260, 262, 119, 265, 120, 180, 269, 122, 123, 273,
            275, 125, 277, 126, 189, 128, 281, 283, 130, 194, 242, 132, 288, 133, 290, 134,
            293, 295, 136, 297, 137, 300, 139, 303, 140, 305, 307, 142, 310, 311, 144, 274,
            145, 316, 318, 147, 321, 149, 285, 150, 325, 151, 328, 330, 153, 280, 334, 155,
            336, 338, 157, 158, 339, 341, 160, 344, 161, 195, 346, 163, 349, 351, 166, 354,
            356, 168, 359, 361, 170, 364, 172, 367, 173, 263, 371, 175, 373, 176, 258, 376,
            178, 317, 379, 375, 181, 383, 385, 387, 184, 216, 185, 391, 187, 358, 393, 230,
            191, 397, 192, 399, 223, 401, 403, 197, 406, 199, 408, 410, 201, 413, 202, 416,
            204, 419, 421, 206, 340, 302, 208, 426, 210, 429, 211, 312, 432, 213, 214, 308,
            436, 437, 439, 218, 266, 219, 444, 446, 221, 448, 323, 450, 225, 402, 452, 227,
            455, 228, 457, 332, 333, 231, 462, 233, 463, 234, 237, 301, 238, 465, 467, 240,
            380, 243, 352, 471, 245, 474, 246, 347, 476, 248, 478, 249, 480, 278, 251, 422,
            252, 386, 484, 254, 425, 423, 256, 487, 488, 370, 259, 491, 261, 492, 268, 264,
            495, 496, 267, 345, 497, 499, 270, 502, 271, 362, 272, 506, 508, 510, 276, 513,
            279, 516, 459, 396, 282, 518, 284, 521, 286, 326, 287, 524, 525, 289, 515, 291,
            529, 292, 532, 534, 294, 536, 537, 296, 540, 539, 298, 544, 299, 360, 366, 546,
            365, 304, 549, 551, 306, 553, 550, 309, 557, 559, 561, 313, 314, 565, 315, 568,
            569, 571, 573, 319, 576, 320, 578, 322, 579, 324, 581, 327, 584, 586, 329, 589,
            331, 591, 395, 594, 545, 335, 337, 596, 342, 466, 343, 585, 483, 601, 348, 602,
            604, 350, 606, 353, 608, 355, 609, 357, 612, 614, 469, 363, 485, 616, 368, 369,
            595, 619, 372, 374, 621, 519, 377, 378, 381, 625, 382, 620, 628, 384, 503, 629,
            631, 388, 632, 389, 634, 390, 600, 638, 392, 641, 407, 394, 535, 398, 494, 592,
            400, 647, 640, 404, 405, 651, 563, 654, 409, 657, 603, 411, 643, 412, 517, 414,
            415, 661, 418, 417, 659, 653, 543, 420, 461, 424, 574, 635, 427, 431, 428, 618,
            430, 665, 673, 660, 433, 676, 434, 679, 435, 682, 527, 633, 438, 658, 613, 440,
            441, 688, 442, 666, 443, 547, 693, 445, 694, 447, 449, 697, 451, 698, 639, 453,
            468, 454, 683, 588, 456, 704, 458, 520, 663, 460, 489, 464, 691, 702, 709, 511,
            470, 472, 531, 473, 715, 475, 717, 477, 479, 687, 720, 481, 567, 482, 723, 486,
            669, 552, 490, 500, 493, 727, 623, 728, 498, 675, 730, 501, 504, 732, 505, 507,
            564, 509, 548, 648, 736, 512, 583, 523, 514, 664, 533, 655, 722, 672, 522, 636,
            745, 746, 526, 716, 542, 528, 644, 749, 530, 566, 541, 556, 538, 701, 593, 642,
            554, 572, 708, 754, 617, 707, 695, 646, 555, 719, 626, 558, 755, 737, 560, 726,
            681, 562, 587, 706, 765, 767, 610, 570, 770, 690, 597, 773, 575, 577, 671, 775,
            580, 582, 712, 739, 662, 598, 778, 590, 763, 684, 670, 667, 599, 782, 783, 699,
            784, 785, 605, 652, 607, 786, 674, 611, 789, 645, 615, 791, 757, 680, 622, 624,
            793, 627, 794, 630, 760, 795, 787, 637, 678, 798, 700, 799, 801, 753, 744, 743,
            649, 650, 796, 756, 656, 772, 804, 805, 742, 668, 677, 748, 725, 780, 777,
            733, 766, 807, 705, 790, 685, 761, 686, 808, 809, 689, 810, 750, 692, 797,
            696, 800, 759, 703, 788, 758, 812, 710, 711, 713, 714, 718, 735, 779, 721,
            764, 724, 816, 729, 731, 734, 747, 774, 738, 740, 776, 741, 817, 811, 751,
            752, 815, 762, 768, 769, 771, 803, 781, 818, 819, 806, 792, 802, 813, 814}];
        g2 = Images[{3, 4, 6, 8, 9, 11, 12, 14, 16, 17, 19, 21, 22, 24, 25, 27, 29, 30, 32, 33,
            7, 36, 37, 39, 40, 41, 43, 44, 46, 48, 49, 51, 52, 53, 54, 56, 58, 59, 61, 63, 65,
            66, 68, 69, 70, 72, 73, 75, 76, 77, 79, 81, 83, 84, 85, 87, 88, 90, 92, 93, 95, 31,
            97, 98, 100, 101, 102, 67, 105, 107, 108, 110, 112, 113, 115, 117, 119, 120, 122,
            123, 125, 126, 128, 130, 132, 133, 134, 136, 137, 139, 140, 142, 144, 145, 147,
            121, 149, 150, 151, 153, 155, 157, 99, 158, 160, 161, 163, 28, 15, 166, 131, 168,
            138, 170, 172, 173, 175, 176, 178, 180, 181, 1, 184, 185, 187, 189, 177, 191,
            192, 194, 195, 197, 199, 201, 202, 204, 20, 206, 208, 210, 211, 213, 214, 216,
            218, 219, 2, 221, 223, 225, 227, 228, 230, 231, 233, 234, 5, 237, 238, 240, 242,
            243, 245, 246, 135, 248, 249, 251, 252, 164, 254, 256, 258, 259, 261, 263, 264,
            266, 267, 268, 270, 271, 272, 274, 276, 143, 278, 279, 280, 193, 282, 174, 284,
            285, 286, 287, 224, 289, 291, 292, 294, 229, 296, 298, 299, 301, 302, 179, 304,
            306, 308, 309, 167, 312, 313, 314, 315, 317, 319, 320, 322, 323, 324, 156, 326,
            327, 329, 331, 332, 333, 335, 207, 337, 300, 82, 118, 340, 342, 343, 260, 345,
            42, 347, 348, 350, 352, 353, 355, 357, 358, 360, 362, 363, 365, 366, 368, 369,
            370, 372, 71, 361, 374, 375, 377, 378, 159, 380, 381, 382, 384, 386, 310, 388,
            389, 390, 104, 392, 89, 394, 395, 396, 257, 398, 400, 182, 402, 404, 405, 239,
            407, 409, 411, 412, 414, 415, 417, 418, 420, 422, 423, 424, 425, 148, 427, 428,
            269, 430, 431, 433, 86, 434, 435, 341, 438, 440, 441, 442, 443, 445, 351, 447,
            449, 198, 109, 235, 451, 453, 454, 200, 456, 458, 459, 460, 461, 339, 305, 106,
            336, 367, 464, 290, 466, 265, 468, 469, 273, 470, 472, 473, 475, 387, 471, 477,
            38, 10, 479, 354, 247, 462, 481, 482, 483, 80, 485, 262, 45, 486, 446, 489, 379,
            490, 436, 205, 493, 346, 494, 406, 492, 196, 23, 498, 500, 501, 503, 504, 35,
            505, 507, 509, 511, 512, 514, 515, 283, 517, 397, 518, 519, 520, 522, 338, 523,
            215, 502, 526, 527, 496, 528, 530, 531, 533, 535, 448, 13, 538, 539, 541, 542,
            543, 437, 545, 457, 55, 547, 190, 548, 550, 552, 450, 554, 555, 556, 558, 560,
            562, 563, 564, 566, 567, 444, 570, 572, 574, 575, 577, 18, 288, 91, 580, 484,
            582, 583, 585, 587, 588, 590, 455, 592, 593, 595, 408, 78, 373, 597, 598, 50,
            599, 600, 601, 429, 452, 603, 605, 169, 255, 607, 209, 165, 610, 611, 613, 62,
            349, 615, 303, 617, 385, 618, 34, 620, 506, 222, 622, 623, 465, 488, 624, 626,
            627, 413, 293, 116, 399, 630, 416, 410, 236, 633, 635, 636, 637, 639, 640, 642,
            569, 281, 643, 644, 594, 645, 646, 232, 648, 649, 650, 652, 653, 655, 656, 183,
            658, 557, 103, 659, 660, 421, 508, 540, 662, 663, 664, 665, 666, 661, 516, 667,
            608, 668, 669, 250, 670, 671, 672, 94, 674, 480, 675, 677, 678, 680, 681, 467,
            683, 684, 685, 686, 383, 687, 657, 689, 690, 307, 691, 692, 679, 241, 621, 695,
            696, 651, 474, 604, 699, 371, 700, 701, 203, 702, 703, 596, 705, 499, 706, 707,
            26, 708, 124, 426, 710, 146, 711, 712, 713, 714, 581, 716, 718, 719, 573, 576,
            546, 721, 722, 723, 724, 561, 497, 403, 725, 726, 344, 586, 682, 154, 729, 359,
            731, 632, 188, 733, 734, 571, 491, 735, 253, 334, 737, 738, 532, 739, 740, 741,
            325, 536, 96, 742, 743, 744, 612, 747, 364, 748, 647, 495, 578, 432, 750, 751,
            752, 753, 419, 754, 619, 755, 277, 756, 757, 758, 244, 614, 759, 609, 141, 760,
            513, 761, 602, 762, 534, 763, 544, 727, 764, 57, 766, 768, 769, 728, 771, 393,
            772, 311, 774, 321, 510, 226, 776, 698, 114, 777, 171, 463, 553, 779, 780, 478,
            629, 730, 781, 186, 356, 606, 775, 64, 641, 212, 152, 787, 74, 788, 790, 47, 487,
            162, 792, 521, 591, 525, 676, 782, 717, 111, 551, 565, 796, 797, 778, 794, 391,
            800, 297, 802, 127, 732, 589, 803, 704, 791, 631, 628, 537, 217, 439, 220, 328,
            616, 529, 806, 785, 783, 799, 275, 318, 129, 801, 805, 715, 60, 476, 767, 784,
            745, 524, 688, 786, 316, 808, 559, 693, 811, 625, 813, 673, 814, 401, 549, 815,
            809, 295, 638, 694, 789, 793, 773, 634, 807, 749, 812, 746, 817, 795, 584, 709,
            330, 736, 654, 818, 579, 720, 765, 376, 770, 804, 697, 816, 568, 798, 810, 819}];
```

In[261]:=

```
PermDeg /@ {g1, g2}
```

Out[261]=

```
{819, 818}
```

In[262]:=

```
myTiming[T3D4SGS = SchreierSims[{}, GenSet[g1, g2], 819];]

   13.412142 Second
```

In[263]:=

```
Length /@ T3D4SGS
```

Out[263]=

```
StrongGenSet[5, 14]
```

In[264]:=

```
OrderOfGroup[T3D4SGS]
```

Out[264]=

```
211341312
```

Go back to the original settings:

In[265]:=

```
SetOptions[Orbit, MathLink → False];
SetOptions[SchreierSims, MathLink → False];
```

Canonicalization

The meaning of canonicalization. Tensor indices and slots

Before describing the algorithms for "canonicalization" of a permutation we want to introduce what we mean by it. Given some equivalence relation which decomposes a group of permutations into disjoint classes we want to identify uniquely a representative per class. That representative will be called the "canonical representative" of the class, and the process of finding the representative of the class of a given permutation *perm* will be referred to as the "canonicalization" of *perm*. The main criterium for canonicalization is that the identity permutation must always be the representative of its class; the secondary criterium will choose the permutation which is "closest" to the identity. This will be done by defining a total ordering among the permutations of the group and choosing the smallest permutation of the class. There are many ways to define that ordering, with the only restriction that the identity must be always the first one in any set containing it. As you can imagine, we will choose an ordering based on the base of an SGS describing the full group.

The special role of the identity poses a small but important problem, because the identity is its own inverse. So, given a sorted set of permutations, are their inverses automatically sorted with respect to the same ordering? The answer is no, and therefore we must always know whether we are sorting the permutations themselves or their inverses. Here we shall follow the convention in the papers by R. Portugal et al, which are the main source for the algorithms implemented here.

In relation to tensors, this amounts to the following: for example, given the collection of sorted indices a,b,c,d,... (renamed as 1,2,3,4,...) Renato says that permutation Cycles[{1,3,2,4}] represents the tensor T^{cdba} (at first slot we find third index; at third slot we find second index; etc.). The opposite view would represent that tensor with the inverse permutation Cycles[{1,4,2,3}] (the first index goes to the fourth slot; the fourth index goes to the second slot; etc).

■ 8. Canonical representative of a right coset

Given a permutation g and a group S of permutations we can construct the right coset S.g, which is not a group in general. Here we want to choose a canonical representant of S.g. ‘Canonical´ means the first according to a given ordering of the points of the Ω set. That ordering is defined through the base of the SGS.

Suppose a group generated by two permutations. We construct a SGS. Note that we can give the first points of the base of the SGS or not:

```
In[267]:=
      SGS1 = SchreierSims[{1, 2}, GenSet[Perm[{2, 1, 3, 4, 6, 5}], Perm[{3, 4, 1, 2, 5, 6}]], 6]

Out[267]=
      StrongGenSet[{1, 2, 3},
       GenSet[Perm[{2, 1, 3, 4, 6, 5}], Perm[{3, 4, 1, 2, 5, 6}], Perm[{1, 2, 4, 3, 6, 5}]]]

In[268]:=
      SGS2 = SchreierSims[{}, GenSet[Perm[{2, 1, 3, 4, 6, 5}], Perm[{3, 4, 1, 2, 5, 6}]], 6]

Out[268]=
      StrongGenSet[{1, 3},
       GenSet[Perm[{2, 1, 3, 4, 6, 5}], Perm[{3, 4, 1, 2, 5, 6}], Perm[{1, 2, 4, 3, 6, 5}]]]
```

The group generated by those SGSs has order 8 and therefore there are 90 cosets of that group in S6:

```
In[269]:=
      OrderOfGroup[SGS1]

Out[269]=
      8
```

```
In[270]:=
      group = Dimino[SGS1[[2]]]

Out[270]=
      Group[Perm[{1, 2, 3, 4, 5, 6}], Perm[{2, 1, 3, 4, 6, 5}],
       Perm[{3, 4, 1, 2, 5, 6}], Perm[{3, 4, 2, 1, 6, 5}], Perm[{4, 3, 1, 2, 6, 5}],
       Perm[{4, 3, 2, 1, 5, 6}], Perm[{1, 2, 4, 3, 6, 5}], Perm[{2, 1, 4, 3, 5, 6}]]

In[271]:=
      6 ! / 8

Out[271]=
      90
```

Now we take one permutation which does not belong to the group and construct the corresponding right coset, sorting the permutations:

```
In[272]:=
      perm = Perm[{2, 3, 1, 4, 5, 6}];

In[273]:=
      PermMemberQ[perm, SGS1]

Out[273]=
      False

In[274]:=
      coset = Permute[group, perm]

Out[274]=
      Coset[Perm[{2, 3, 1, 4, 5, 6}], Perm[{1, 3, 2, 4, 6, 5}],
       Perm[{4, 1, 3, 2, 5, 6}], Perm[{4, 2, 3, 1, 6, 5}], Perm[{3, 1, 4, 2, 6, 5}],
       Perm[{3, 2, 4, 1, 5, 6}], Perm[{2, 4, 1, 3, 6, 5}], Perm[{1, 4, 2, 3, 5, 6}]]

In[275]:=
      PermSort[coset]

Out[275]=
      Coset[Perm[{1, 3, 2, 4, 6, 5}], Perm[{1, 4, 2, 3, 5, 6}],
       Perm[{3, 1, 4, 2, 6, 5}], Perm[{4, 1, 3, 2, 5, 6}], Perm[{2, 3, 1, 4, 5, 6}],
       Perm[{2, 4, 1, 3, 6, 5}], Perm[{3, 2, 4, 1, 5, 6}], Perm[{4, 2, 3, 1, 6, 5}]]
```

The sorting is only evident in `Images` notation. The first permutation is our canonical representative:

```
In[276]:=
      TranslatePerm[#, Images] & /@ %

Out[276]=
      Coset[Images[{1, 3, 2, 4, 6, 5}], Images[{1, 3, 4, 2, 5, 6}],
       Images[{2, 4, 1, 3, 6, 5}], Images[{2, 4, 3, 1, 5, 6}], Images[{3, 1, 2, 4, 5, 6}],
       Images[{3, 1, 4, 2, 6, 5}], Images[{4, 2, 1, 3, 5, 6}], Images[{4, 2, 3, 1, 6, 5}]]
```

It can also be computed using Portugal’s algorithms.

```
In[277]:=
      RightCosetRepresentative[perm, 6, SGS1]

Out[277]=
      {Perm[{1, 3, 2, 4, 6, 5}], StrongGenSet[{}, GenSet[]], {2, 1, 3, 4, 6, 5}}
```

There is an additional argument which assigns special priority to some points ("free slots" below). The default is Range[deg], assigning that special priority to all points ("all slots are free"):

```
In[278]:=
      RightCosetRepresentative[perm, 6, SGS1, {1, 4, 5}]

Out[278]=
      {Perm[{2, 4, 1, 3, 6, 5}], StrongGenSet[{2}, GenSet[]], {1, 3, 6}}
```

Together with the canonical representative, the algorithm returns a SGS for the stabilizer of the special points, and the new positions of the special points under the change of coset element (performed by the permutation h in G taking the old permutation to the new permutation):

```
In[279]:=
      RightCosetRepresentative[perm, 6, SGS1, {4, 3}]

Out[279]=
      {Perm[{4, 1, 3, 2, 5, 6}], StrongGenSet[{3}, GenSet[Perm[{1, 2, 4, 3, 6, 5}]]], {2, 1}}

In[280]:=
      h = Permute[First[%], InversePerm[perm]]

Out[280]=
      Perm[{3, 4, 1, 2, 5, 6}]

In[281]:=
      OnPoints[{4, 3}, h]

Out[281]=
      {2, 1}

In[282]:=
      Clear[h]
```

It is important to stress that `PermSort` uses by default the trivial ordering of points {1, 2, 3,...}, which in this case coincide with the base of SGS1. However, if we want to use other bases, then we need a second argument:

```
In[283]:=
      PermSort[coset, SGS2[[1]]]

Out[283]=
      Coset[Perm[{1, 3, 2, 4, 6, 5}], Perm[{1, 4, 2, 3, 5, 6}],
       Perm[{2, 3, 1, 4, 5, 6}], Perm[{2, 4, 1, 3, 6, 5}], Perm[{3, 1, 4, 2, 6, 5}],
       Perm[{4, 1, 3, 2, 5, 6}], Perm[{3, 2, 4, 1, 5, 6}], Perm[{4, 2, 3, 1, 6, 5}]]

In[284]:=
      RightCosetRepresentative[perm, 6, SGS2]

Out[284]=
      {Perm[{1, 3, 2, 4, 6, 5}], StrongGenSet[{}, GenSet[]], {2, 1, 3, 4, 6, 5}}
```

`PermSort` with a base uses the ordering given by `PermOrderedQ`. In any case the identity must always be the least permutation. Here we check that all permutations of degree four with all bases come after the identity:

```
In[285]:=
      And @@ Flatten[Outer[PermOrderedQ[{ID, Perm[#1]}, #2] &,
         Drop[Permutations@Range[4], 1], Permutations@Range[4], 1]]

Out[285]=
      True
```

It is possible to follow the internals of `CosetRepresentative` using the option `xPermVerbose`. Note that we use the terminology of tensor computations (tensor, slots, indices, etc.):

```
In[286]:=
     RightCosetRepresentative[perm, 6, SGS2, Range[6], xPermVerbose → True]

         RIGHT-COSET-REPRESENTATIVE ALGORITHM for Perm[{2, 3, 1, 4, 5, 6}]

         which corresponds to the index list: Perm[{3, 1, 2, 4, 5, 6}]

         base: {1, 3}

         ****** Analysing element i=1 of base: slot 1 ******

         Symmetry orbit Delta of slots: {1, 2, 3, 4}

         Free slots: {1, 2, 3, 4, 5, 6}

         Free slots that can go to that slot: {1, 2, 3, 4}

         At those slots we respectively find indices Deltap: {3, 1, 2, 4}

         The least index is 1, found at position pk: 2 of Deltap

         That index is found in tensor at slot pp: 2

         We can move slot 2 to slot 1 using permutation om: Perm[{2, 1, 3, 4, 6, 5}] in S

         New indices list: Perm[{1, 3, 2, 4, 6, 5}]

         Computing stabilizer in S of slot 1

         newbase before change: {1, 3}

         newbase after change: {3}

         ****** Analysing element i=2 of base: slot 3 ******

         Symmetry orbit Delta of slots: {3, 4}

         Free slots: {2, 1, 3, 4, 6, 5}

         Free slots that can go to that slot: {3, 4}

         At those slots we respectively find indices Deltap: {2, 4}

         The least index is 2, found at position pk: 1 of Deltap

         That index is found in tensor at slot pp: 3

         We can move slot 3 to slot 3 using permutation om: ID in S

         New indices list: Perm[{1, 3, 2, 4, 6, 5}]

         Computing stabilizer in S of slot 3

         newbase before change: {3}

         newbase after change: {}

Out[286]=
     {Perm[{1, 3, 2, 4, 6, 5}], StrongGenSet[{}, GenSet[]], {2, 1, 3, 4, 6, 5}}
```

`RightCosetRepresentative`	Compute a canonical representative of a coset

Canonicalization of free indices.

■ 9. Canonical representative of a double coset

In this and the next section we shall use tensor terminology: the points will be either indices or slots, with the convention that the permutation g acts on slots and its images are indices. That is, 3^g = 7 will now be read "the third slot contains the seventh index".

When including dummy indices in a tensor there are two different kinds of symmetries: apart from those coming from the intrinsic symmetries of the tensor slots (the group S), we have more symmetries due to exchange of names of dummy indices and to the exchange of up/down indices of a pair when a metric is present (we shall call D the group of those new symmetries). Now we need to work with the double coset S.g.D, and again we need to find a canonical representative. It is important to note that this code is prepared to work only with a special type of group D: that needed to canonicalize tensors, and that the S group acts from the left of g while the D group acts from the right of g.

The most general situation we need to handle for the D group is the following:

− We can have dummies coming from several different vbundles. The D symmetries cannot mix them and so the D group can be reduced as a direct product of smaller groups. Therefore the strong generating sets can be easily con−structed independently and then joined together.

− Inside each vbundle there are two types of "dummies": pair−dummies, each pair having a S_2 or A_2 group or nothing, and then repeated indices, each type of repeated index having a S_n symmetry group, with n being the number of times that a given index is repeated. All pair−dummies in a vbundle can be exchanged. Therefore the D symmetry group for a vbundle is also the product of symmetric and pair−symmetric groups. (Note: sometimes I call "drummies" the combined type of dummies and "repes").

This is an important simplification because we do not need to worry about what is happenning to the generating set of the D group, but only to its base. That is, given the base of the D group we know how to construct an adequate generat−ing set. This is done using the concepts of `DummySet` and `RepeatedSet`.

Suppose a tensor with 12 indices where there are 5 pairs of dummies of the same vbundle TM. We encode the situation in a `Dummy−Set` expression. The first pair {7,11} means that the seventh index in the canonical configuration is an up−index paired with the eleventh index (which is hence a down−index). The number 0 at the end means that there is no metric:

```
In[287]:=
      dummyset = DummySet[TM, {{7, 11}, {4, 5}, {10, 6}, {8, 9}, {12, 1}}, 0];

In[288]:=
      SGSOfDummySet[dummyset]

Out[288]=
      StrongGenSet[{7, 4, 10, 8, 12}, GenSet[Cycles[{7, 4}, {11, 5}],
        Cycles[{4, 10}, {5, 6}], Cycles[{10, 8}, {6, 9}], Cycles[{8, 12}, {9, 1}]]]
```

We include a symmetric metric with a 1 at the end (−1 would give an antisymmetric metric):

```
In[289]:=
      dummyset = DummySet[TM, {{7, 11}, {4, 5}, {10, 6}, {8, 9}, {12, 1}}, 1];

In[290]:=
      SGSD = SGSOfDummySet[dummyset]

Out[290]=
      StrongGenSet[{7, 4, 10, 8, 12}, GenSet[Cycles[{7, 11}], Cycles[{4, 5}],
        Cycles[{10, 6}], Cycles[{8, 9}], Cycles[{12, 1}], Cycles[{7, 4}, {11, 5}],
        Cycles[{4, 10}, {5, 6}], Cycles[{10, 8}, {6, 9}], Cycles[{8, 12}, {9, 1}]]]

In[291]:=
      groupD = Dimino[TranslatePerm[SGSD[[2]], {Images, 12}]];
```

```
In[292]:=
      Length[groupD]

Out[292]=
      3840
```

Now suppose this configuration, where the free indices 2 and 3 are at the third and fifth slots. The canonicalization process (assuming no additional symmetry in the tensor) exchanges the pair {4,5} and changes the pair {10,6} with the pair {8,9}. The pair {11,7} is also exchanged. Compare the timing of the "brute−force" method and that of Portugal’s algorithm.

```
In[293]:=
      perm = Images[{1, 5, 2, 12, 3, 9, 6, 8, 4, 11, 7, 10}];

In[294]:=
      coset = Permute[perm, groupD];

In[295]:=
      Timing@Module[
        {p = First[coset], i},
        Do[
         If[PermLess[coset[[i]], p], p = coset[[i]]; Print["Change at i=", i, " to ", p]],
         {i, 2, Length@coset}];
        p]
         Change at i=2 to Images[{1, 5, 2, 12, 3, 9, 6, 8, 4, 7, 11, 10}]

         Change at i=3 to Images[{1, 4, 2, 12, 3, 9, 6, 8, 5, 11, 7, 10}]

         Change at i=4 to Images[{1, 4, 2, 12, 3, 9, 6, 8, 5, 7, 11, 10}]

         Change at i=11 to Images[{1, 4, 2, 12, 3, 8, 6, 9, 5, 11, 7, 10}]

         Change at i=12 to Images[{1, 4, 2, 12, 3, 8, 6, 9, 5, 7, 11, 10}]

         Change at i=195 to Images[{1, 4, 2, 12, 3, 6, 9, 10, 5, 11, 7, 8}]

         Change at i=196 to Images[{1, 4, 2, 12, 3, 6, 9, 10, 5, 7, 11, 8}]

         Change at i=199 to Images[{1, 4, 2, 12, 3, 6, 8, 10, 5, 11, 7, 9}]

         Change at i=200 to Images[{1, 4, 2, 12, 3, 6, 8, 10, 5, 7, 11, 9}]

         Change at i=359 to Images[{1, 4, 2, 12, 3, 6, 7, 10, 5, 9, 8, 11}]

         Change at i=360 to Images[{1, 4, 2, 12, 3, 6, 7, 10, 5, 8, 9, 11}]

Out[295]=
      {1.5401 Second, Images[{1, 4, 2, 12, 3, 6, 7, 10, 5, 8, 9, 11}]}
```

The base for sorting of permutations is taken from the SGS of group S. If the base does not include all positions of dummies, then it is expanded to cover them all, sorting the missing dummies.

```
In[296]:=
      Timing@DoubleCosetRepresentative[perm, 12, StrongGenSet[{}, GenSet[]], {dummyset}]

Out[296]=
      {0.072004 Second, Images[{1, 4, 2, 12, 3, 6, 7, 10, 5, 8, 9, 11}]}
```

The external C code would be even faster.

`DummySet`	Head for a set of dummy indices
`RepeatedSet`	Head for a set of repeated indices
`SGSOfDummySet`	Compute SGS for the group associated to a set of dummy indices
`DoubleCosetRepresentative`	Compute a canonical representative of a double coset

Canonicalization of dummy and repeated indices.

10. Canonical permutation

Finally, we give an algorithm that canonicalize simultaneously the free and the dummy/repeated indices. The algorithm always canonicalizes first the free indices and then the dummy/repeated indices.

Portugal’s example in paper II. Suppose a tensor with 12 indices whose index−configuration is described by the following permutation:

```
In[297]:=
      perm = Cycles[{1, 12, 11, 4, 5, 6, 8, 9, 10, 7, 3}];

In[298]:=
      TranslatePerm[perm, Images]

Out[298]=
      Images[{12, 2, 1, 5, 6, 8, 3, 9, 10, 7, 4, 11}]
```

The symmetries of the tensor are described by this GS:

```
In[299]:=
      GS = GenSet[-Cycles[{1, 2}], -Cycles[{3, 4}],
         -Cycles[{5, 6}], -Cycles[{7, 8}], -Cycles[{9, 10}], -Cycles[{11, 12}],
         Cycles[{1, 3}, {2, 4}], Cycles[{5, 7}, {6, 8}], Cycles[{9, 11}, {10, 12}],
         Cycles[{5, 9}, {6, 10}, {7, 11}, {8, 12}]];
```

1) Canonical configuration assuming that all indices are free (empty `DummySet/RepeatedSet` expressions):

```
In[300]:=
      CanonicalPerm[perm, 12, GS, Range[12], {}, TimeVerbose → True, MathLink → False]

          SGS for group of order 1024 computed in 0.660042 secs.

          Free algorithm applied in 0.016001 secs.

Out[300]=
      Cycles[{2, 5, 3}, {4, 12, 10, 11, 7, 6, 9}]

In[301]:=
      TranslatePerm[%, Images]

Out[301]=
      Images[{1, 5, 2, 12, 3, 9, 6, 8, 4, 11, 7, 10}]
```

```
In[302]:=
      CanonicalPerm[perm, 12, GS, Range[12], {}, MathLink → True]

Out[302]=
      Images[{1, 5, 2, 12, 3, 9, 6, 8, 4, 11, 7, 10}]
```

Let us check that this is the smallest permutation in the right coset S.g. We work in `Images` notation because only then the sorting process is apparent:

```
In[303]:=
      Length[groupS = Dimino[TranslatePerm[GS, {Images, 12}]]]

Out[303]=
      1024

In[304]:=
      Module[{g = TranslatePerm[perm, Images], ming, gg},
        ming = g;
        Do[
         gg = Permute[groupS[[i]], g];
         If[PermLess[gg, ming], ming = gg; Print["Change at i=", i, " to ", ming]],
         {i, Length[groupS]}];
        ming] // Timing

         Change at i=2 to -Images[{2, 12, 1, 5, 6, 8, 3, 9, 10, 7, 4, 11}]

         Change at i=18 to Images[{2, 12, 1, 5, 6, 8, 3, 9, 7, 10, 4, 11}]

         Change at i=65 to Images[{1, 5, 12, 2, 6, 8, 3, 9, 10, 7, 4, 11}]

         Change at i=67 to -Images[{1, 5, 2, 12, 6, 8, 3, 9, 10, 7, 4, 11}]

         Change at i=83 to Images[{1, 5, 2, 12, 6, 8, 3, 9, 7, 10, 4, 11}]

         Change at i=195 to -Images[{1, 5, 2, 12, 3, 9, 6, 8, 10, 7, 4, 11}]

         Change at i=211 to Images[{1, 5, 2, 12, 3, 9, 6, 8, 7, 10, 4, 11}]

         Change at i=451 to -Images[{1, 5, 2, 12, 3, 9, 6, 8, 4, 11, 10, 7}]

         Change at i=483 to Images[{1, 5, 2, 12, 3, 9, 6, 8, 4, 11, 7, 10}]

Out[304]=
      {0.208013 Second, Images[{1, 5, 2, 12, 3, 9, 6, 8, 4, 11, 7, 10}]}
```

2) Now suppose that indices 1,2 (in the canonical list) are free while the others are contracted as given by the list of pairs or slots {{3,4},{5,6},{7,8},{9,10},{11,12}}. The canonical configuration is (note that this is the result quoted by Portugal et al.):

```
In[305]:=
      CanonicalPerm[perm, 12, GS, {1, 2},
       {DummySet[M, {{3, 4}, {5, 6}, {7, 8}, {9, 10}, {11, 12}}, 1]},
       TimeVerbose → True, MathLink → False]

         SGS for group of order 1024 computed in 0.656041 secs.

         Free algorithm applied in 0.008001 secs.

         Dummy algorithm applied in 0.128008 secs.

Out[305]=
      -Cycles[{2, 3}, {4, 5}, {6, 7, 9}, {8, 11}]
```

```
In[306]:=
      TranslatePerm[%, {Images, 12}]

Out[306]=
      -Images[{1, 3, 2, 5, 4, 7, 9, 11, 6, 10, 8, 12}]

In[307]:=
      CanonicalPerm[perm, 12, GS, {1, 2},
       {DummySet[M, {{3, 4}, {5, 6}, {7, 8}, {9, 10}, {11, 12}}, 1]}, MathLink → True]

Out[307]=
      -Images[{1, 3, 2, 5, 4, 7, 9, 11, 6, 10, 8, 12}]
```

Note that this permutation is smaller than the one obtained using only the right coset S.g. This is obvious because the double coset S.g.D contains the right coset S.g. Again, we check that this is the right answer by brute force. We construct the SGS of the group D and search for the least of the permutations of the double coset S.g.D. Do not evaluate the Module below because it takes very long (the cell has been made nonevaluatable):

```
In[308]:=
      SGSD = TranslatePerm[SGSOfDummySet[
         DummySet[M, {{3, 4}, {5, 6}, {7, 8}, {9, 10}, {11, 12}}, 1]], {Images, 12}]

Out[308]=
      StrongGenSet[{3, 5, 7, 9, 11}, GenSet[Images[{1, 2, 4, 3, 5, 6, 7, 8, 9, 10, 11, 12}],
        Images[{1, 2, 3, 4, 6, 5, 7, 8, 9, 10, 11, 12}],
        Images[{1, 2, 3, 4, 5, 6, 8, 7, 9, 10, 11, 12}],
        Images[{1, 2, 3, 4, 5, 6, 7, 8, 10, 9, 11, 12}],
        Images[{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 11}],
        Images[{1, 2, 5, 6, 3, 4, 7, 8, 9, 10, 11, 12}],
        Images[{1, 2, 3, 4, 7, 8, 5, 6, 9, 10, 11, 12}],
        Images[{1, 2, 3, 4, 5, 6, 9, 10, 7, 8, 11, 12}],
        Images[{1, 2, 3, 4, 5, 6, 7, 8, 11, 12, 9, 10}]]]

In[309]:=
      Length[groupD = Dimino[SGSD[[2]]]]

Out[309]=
      3840

In[310]:=
      Module[{g = TranslatePerm[perm, Images], ming, gg},
        ming = g;
        Do[
         Do[
          gg = Permute[groupS[[i]], g, groupD[[j]]];
          If[PermLess[gg, ming], ming = gg;
           Print["Change at i=", i, ", j=", j, " to ", ming]],
          {j, Length[groupD]}],
         {i, Length[groupS]}];
        ming] // Timing
```

3) With OrderedBase→False we can get a different result, because we are sorting indices with a different base:

```
In[311]:=
     CanonicalPerm[perm, 12, GS, {1, 2},
      {DummySet[M, {{3, 4}, {5, 6}, {7, 8}, {9, 10}, {11, 12}}, 1]},
      OrderedBase → False, TimeVerbose → True, MathLink → False]

        SGS for group of order 1024 computed in 0.400025 secs.

        Free algorithm applied in 0.004001 secs.

        Dummy algorithm applied in 0.616038 secs.

Out[311]=
     -Cycles[{2, 3}, {4, 6, 10, 8, 12}]
```

```
In[312]:=
     TranslatePerm[%, {Images, 12}]

Out[312]=
     -Images[{1, 3, 2, 6, 5, 10, 7, 12, 9, 8, 11, 4}]
```

```
In[313]:=
     CanonicalPerm[perm, 12, GS, {1, 2},
      {DummySet[M, {{3, 4}, {5, 6}, {7, 8}, {9, 10}, {11, 12}}, 1]},
      OrderedBase → False, MathLink → True]

Out[313]=
     -Images[{1, 3, 2, 6, 5, 10, 7, 12, 9, 8, 11, 4}]
```

That result can also be interpreted in terms of an ordering of permutations with respect to a particular base. In this case the dummy indices are sorted differently (again, this cell has been made nonevaluatable):

```
In[314]:=
     Module[{g = TranslatePerm[perm, Images], ming, gg},
       ming = g;
       Do[
        Do[
         gg = Permute[groupS[[i]], g, groupD[[j]]];
         If[PermOrderedQ[{gg, ming}, {1, 2, 3, 5, 7, 9, 11, 4, 6, 8, 10, 12}],
          ming = gg; Print["Change at i=", i, ", j=", j, " to ", ming]],
         {j, Length[groupD]}],
        {i, Length[groupS]}];
       ming] // Timing
```

`CanonicalPerm`	Compute a canonical representative given the free and dummy indices of an object
`TimeVerbose`	Timings information
`xPermVerbose`	Detailed information of the internal processes
`OrderedBase`	Fill missing points in base with sorted points

Index canonicalization

Final comments

■ Statistics

This is the time used inside the Mathematica session, not counting the time in the external executable:

```
In[315]:=
      TimeUsed[]

Out[315]=
      17.1051
```

and again, this is the maximum memory used in the session, not counting the external executable:

```
In[316]:=
      MaxMemoryUsed[] / 1000 / 1024.

Out[316]=
      11.1003
```

The package has 78 public reserved words:

```
In[317]:=
      ? xAct‘xPerm‘*
```

xAct‘xPerm‘

```
AllBaseImages                 LeftTransversal   PermOrderedQ               Search
Antisymmetric                 MathLink          PermQ                      SetStabilizer
BaseChange                    MinB              PermSignature              SGSOfDummySet
BaseChangeCheck               NonStablePoints   PermSort                   SortB
CanonicalPerm                 NotationOfPerm    PermutationFromTo          Stabilizer
Coset                         OnPoints          Permute                    StabilizerChain
Cycles                        Orbit             PermuteList                StablePoints
DeleteRedundantGenerators     Orbits            PermWord                   StrongGenSet
Dimino                        OrderedBase       PowerPermute               Symmetric
Disclaimer                    OrderOfGroup      RandomPerm                 TimeVerbose
DoubleCosetRepresentative     PairSymmetric     RepeatedSet                TraceSchreier
DoubleTransversal             Perm              RiemannSymmetric           TranslatePerm
DummySet                      PermDeg           RiemannSymmetry            UseRules
FromBaseImage                 PermEqual         RightCosetRepresentative   xPermVerbose
GenSet                        PermGreater       RightTransversal           $Version
Group                         PermGreaterEqual  Rules                      $xpermExecutable
ID                            PermLength        Schreier                   $xpermLink
Images                        PermLess          SchreierOrbit              $xpermQ
InversePerm                   PermLessEqual     SchreierOrbits
JoinSGS                       PermMemberQ       SchreierSims
```