Worksheet 4

Version 2014-12-03

4 Permutations and groups

4.1 Permutations

Permutations of positive integers can be entered directly in GAP via cycle notation.

```
gap> pi := (1,2,3)(4,5)(7,8);
(1,2,3)(4,5)(7,8)
```

There are two natural ways to define the product of two permutations: Let X be a set, $x \in X$ and $\pi_1, \pi_2 \in \Sigma_X$. For the left action (reading from right-to-left), the multiplication is given by

$$(\pi_1 \circ \pi_2)(x) = \pi_1(\pi_2(x)) \ ,$$

while for the right action (reading from left-to-right), the multiplication is given by

$$x^{\pi_1 \pi_2} = (x^{\pi_1})^{\pi_2} \ .$$

As with matrices (see worksheet 2), GAP uses right actions, which is visible in how permutations are multiplied, or applied to elements:[1]

```
gap> (1,2) * (1,3);
(1,2,3)
gap> 1^(1,2);
2
```

The order of a permutation (or, for that matter, arbitrary group elements) can be determined using `Order`.

```
gap> Order( (1,2,3)(4,5) );
6
```

When writing code working with permutations, it can be helpful to convert a permutation π to a list where the i-th entry corresponds to i^π, and back.

```
gap> pi := (1,2,3)(4,5)(7,8);;
gap> l := ListPerm(pi);
[ 2, 3, 1, 5, 4, 6, 8, 7 ]
gap> PermList(l);
(1,2,3)(4,5)(7,8)
```

Some other useful commands for working with permutations are `SignPerm`, `CycleStructurePerm` and `MovedPoints`. As always, you can find out more about a command by entering `?cmd` in GAP, or by consulting the documentation. In particular Chapter 42 of the GAP reference manual.

[1] It is a common convention that `x^y` stands for x^y.

4.2 Symmetric groups

GAP knows about permutations and about groups, and of course also symmetric groups:

```
gap> S3 := SymmetricGroup(3);
Sym( [ 1 .. 3 ] )
```

You can use GAP to get all elements of a group (but beware of doing this with large groups):

```
gap> Elements(S3);
[ (), (2,3), (1,2), (1,2,3), (1,3,2), (1,3) ]
```

In GAP, groups (and subgroups) are more than simply lists of elements. You can query them about group theoretic information about themselves, and once something has been computed, they usually “remember” it.

```
gap> Size(S3);
6
gap> IsAbelian(S3);
false
```

Remark 4.2.1

You can find out what a group knows about itself with help of the commands `KnownAttributesOfObject(G)`, `KnownTruePropertiesOfObject` and `KnownPropertiesOfObject(G)` – try them (and remember to use TAB completion to enter these commands).

We can also construct many other groups, such as alternating groups.

```
gap> G := AlternatingGroup(6);
gap> Size(G) = Factorial(6) / 2;
true
gap> ForAll(G, g -> SignPerm(g) = 1); # alternating group is kernel of sign homomorphism
true
```

4.3 Permutation groups

A permutation group is a subgroup of Σ_n for some $n \in \mathbb{N}^*$. Given some permutations $\pi_1, \ldots, \pi_k$, they generate a permutation group $\langle \pi_1, \ldots, \pi_k \rangle$. In GAP, we can do this with the commands `Subgroup` and `Group`. Conversely, given a group G, we may obtain a generating set for it via `GeneratorsOfGroup`.

```
gap> g1:=(1,2,3,4);; g2:=(1,2)(3,4);;
gap> H:=Group(g1, g2);
Group([ (1,2,3,4), (1,2)(3,4) ])
gap> GeneratorsOfGroup(H);
[ (1,2,3,4), (1,2)(3,4) ]
```

Exercise 4.3.1

1. Suppose $G = \langle g_1, \ldots, g_n \rangle$. Show that G is abelian if and only if $g_i g_j = g_j g_i$ for $1 \leq i < j \leq n$.
2. In GAP, you can determine a set of generators for a group G via the command `GeneratorsOfGroup`. Write a function `isAbel(G)`, which, given a group G, returns `true` if G is abelian, and `false` otherwise. Do not use `IsAbelian` or `IsCommutative`.

GAP can do a lot of things with such groups. A few minor examples:

```
gap> IsAbelian(H);
false
gap> Size(H);
8
gap> Elements(H);
[ (), (2,4), (1,2)(3,4), (1,2,3,4), (1,3), (1,3)(2,4), (1,4,3,2), (1,4)(2,3) ]
```

Exercise 4.3.2

The group H is actually isomorphic to the symmetries of a square. Describe this isomorphism.

Using the command `IsSubgroup(G,H)` one can ask GAP to test whether the group H is a subgroup of G.

```
gap> S4 := SymmetricGroup(4);;
gap> IsSubgroup(S3, H);
false
gap> IsSubgroup(S4, H);
true
```

We can also test whether a subgroup is normal, and compute various standard subgroups.

```
gap> N := Center(H); # = { g in H | forall h in h: gh = hg }
Group([ (1,3)(2,4) ])
gap> IsNormal(H,N);
true
gap> K := DerivedSubgroup(H); # = < g^-1 h^-1 g h | g,h \in H >
Group([ (1,3)(2,4) ])
gap> TrivialSubgroup(H);
Group(())
```

Exercise 4.3.3

Write a function `cent(G)` which returns a list containing precisely the elements of the center of a finite group G, but without using `Center`. Test your program with multiple groups e.g. `SymmetricGroup(n)` for various n, and also with this group:

```
1 G := Group( (1,8,14,2,9,15,3,7,13)(4,10,16)(5,11,17)(6,12,18),
2           (1,7)(2,8)(3,9)(4,11,5,12,6,10) );
```

Let's compute the quotient of H by one of its normal subgroup:

```
gap> K := H/N;
<pc group with 2 generators>
```

It turns out the group GAP computed here is not a permutation group, but rather a so-called “pc-group”. Whatever it is, we can still work with it, and its elements.

```
gap> elms := Elements(K);
[ <identity> of ..., f1, f2, f1*f2 ]
gap> elms[2] * elms[3];
f1*f2
```

By the way, conjugation of elements is also supported, again with the convention that `x^y` stands for x^y.

```
gap> elms[2] ^ elms[3];
f1
gap> (1,2,3)^(1,4);
(2,3,4)
```

Exercise 4.3.4

1. The exponent $\exp(G)$ of a finite group G is the least $n \in \mathbb{N}^*$ such that $g^n = 1_G$ for all $g \in G$. Show that $\exp(G) = \operatorname{lcm}(o(g) \mid g \in G)$.
2. Write a function `exp(G)` which returns the exponent of a given (finite) group G.

4.4 Group homomorphisms

For specific groups $U \leq G$, $N \trianglelefteq G$, we can use GAP to, for example, verify the second isomorphism theorem:

```
gap> G:=S4;; N:=Subgroup(G, [ (1,2)(3,4), (1,3)(2,4) ]);; U:=S3;
gap> UN := ClosureGroup(U,N);
Group([ (1,3), (2,3), (1,2)(3,4) ])
gap> UN = G; # UN turns out to be all of G
true
gap> A := UN/N;; B := U/Intersection(U,N);;
gap> iso := IsomorphismGroups(A, B); # construct an i
[ f1, f2 ] -> [ (2,3), (1,2,3) ]
gap> IsomorphismGroups(S4, S3); # of course S3 and S4 are not isomorphic
fail
```

We can also ask for the natural quotient epimorphism $G \to G/N$:

```
gap> epi := NaturalHomomorphismByNormalSubgroup(G,N);
[ (1,2,3,4), (1,2) ] -> [ f1*f2, f1 ]
```

The objects `iso` and `epi` are group homomorphism objects. We can ask them for their domain, range and image, and test various properties, and of course, apply them to elements. For more on group homomorphisms in GAP, see Chapter 32 of the GAP reference manual.

```
gap> Source(epi); Range(epi); # domain and range of epi are G and G/N
Sym( [ 1 .. 4 ] )
Group([ f1, f2 ])
gap> Kernel(epi) = N;      # the kernel of epi of course is N
true
gap> IsTrivial(Kernel(iso)); # iso is an isomorphism, so its kernel is trivial
true
gap> IsBijective(iso); IsBijective(epi); # iso is bijective, epi is not
true
false
gap> G.1;        # handy short cut: the first generator of G ...
(1,2,3,4)
gap> G.1 ^ epi;  # ... is mapped by epi to this element:
f1*f2
```

4.5 Homework

(s4-h1) Implement a function `ElemOrders(G)` which takes a finite group G, and returns a list of pairs of integers $[[o_1, n_1], ..., [o_k, n_k]]$ such that n_i is the number of elements of order o_i in G, for $1 \leq i \leq k$, and $\sum_{i=1}^{k} n_i = |G|$. Try to make your program efficient; computing `ElemOrders(SymmetricGroup(9))` should take less than 10 seconds.

Hint: The commands `DivisorsInt` and `Sort` may or may not be helpful to you. The command `Collected` is even more useful, but you are not allowed to use it for this exercise.

(s4-h2) Write a function `isHom(G,H,f)` whose arguments are two finite groups G and H, and an unary function f (you may assume that f maps elements of G to H without verifying it). Your program should check whether f describes a homomorphism, and if so, compute some of its properties: If f is not a homomorphism, return `fail`. If it is, return a list `[K, isInj, isSurj]`, where K should be a list with the elements of the kernel (without duplicates), and `isInj` and `isSurj` are booleans indicating whether the function is injective and / or surjective.

Note: You should do this without using the GAP functions for working with homomorphisms, such as `Kernel`, `IsInjective`. Instead, you should loop over the group elements as needed. It is possible to solve the problem using only the function `Size`, `One` and `Add`.

Hint: You can get the unit element of a group using `One`. Once you have K, it is easy to compute `isSurj` using the first isomorphism theorem, by comparing the sizes of the various lists and groups suitably. Finally, here are some simple examples:

```
gap> G := SymmetricGroup(3);;
gap> isHom(G, G, x -> One(G));
[ [ (), (1,3), (1,2,3), (2,3), (1,3,2), (1,2) ], false, false ]
gap> isHom(G, G, x -> x);
[ [ () ], true, true ]
gap> isHom(G, SymmetricGroup(4), x -> x);
[ [ () ], true, false ]
```

(s4-h3) Write a function `PermMul(g,h)` which computes the product gh of two permutations without using `*`. *Hint:* Use `PermList` and `ListPerm`.