Una introducción al álgebra con GAP

Leandro Vendramin

Resumen. Estas notas corresponden a un minicurso de ocho horas dictado en la Universidad de Chile, Santiago de Chile, en noviembre de 2014. Compilado el 2 de diciembre de 2014 a las 07:54.

Índice

Introducción 1
1. Primeros pasos 2
2. Programación en GAP 5
3. Permutaciones y matrices 15
4. Grupos y morfismos 19
5. Aplicaciones a quandles 36
6. Ejercicios 42
Referencias 48

Introducción

Estas notas corresponden a un minicurso acerca del software matemático GAP (*Groups, Algorithms, Programming*). Este software es libre y está diseñado para cálculos algebraicos discretos, especialmente relacionados con la teoría de grupos. Para instalar GAP, para conocer la lista de autores y de gente que contribuye con el desarrollo de GAP, para manuales y ejemplos, para instalar paquetes que agreguen a GAP nuevas funcionalidades, y casi para cualquier otra cosa relacionada con este software, referimos a la página `http://www.gap-system.org`.

Estas notas están organizadas de la siguiente forma. En la sección 1 veremos algunos comandos básicos y realizaremos operaciones aritméticas elementales (sobre los racionales, sobre cuerpos ciclotómicos y sobre cuerpos finitos). En la sección 2 daremos una introducción a la programación en GAP . Esta introducción incluye variables, condicionales, funciones, listas, conjuntos, registros y bucles. En la sección 3 trabajaremos con dos conceptos fundamentales: permutaciones y matrices. La sección 4 es una de las más importantes del curso y contiene ejemplos que ilustran una buena parte de la teoría elemental de grupos. Veremos cómo construir grupos simétricos y alternados, grupos diedrales, grupos de permutaciones, grupos de matrices, grupos finitamente presentados, etc. Calcularemos subgrupos y órdenes de elementos, trabajaremos con clases de conjugación y centralizadores, listaremos los morfismos entre dos grupos, calcularemos grupos de automorfismo, etc. Veremos además cómo acceder a una base de datos con todos los grupos finitos que satisfacen ciertos criterios y utilizaremos esta base de datos para *demostrar* teoremas. La sección 5 contiene una aplicación a un problema concreto: estudiar ciertas estructuras no asociativas llamadas *quandles*, muy utilizadas en teoría de nudos para construir invariantes. La última sección contiene ejercicios y problemas.

Agradecimientos. Estas notas corresponden a un minicurso dictado en Santiago de Chile, en noviembre de 2014. Le agradezco a Andrés Navas y a Nicolás Libedinsky por la invitación y la hospitalidad.

1. Primeros pasos

Una vez ejecutado GAP veremos cierta información general sobre la distribución instalada, como por ejemplo la versión y los paquetes que se han cargado en memoria. Veremos además el cursor que indica que GAP está listo y a la espera:

```
gap>
```

Para cerrar la sesión, se utiliza el comando `quit`:

```
gap> quit;
```

Toda orden debe terminar con el símbolo ; (punto y coma). Si una orden termina con ;; (doble punto y coma), no producirá texto en la pantalla.

Para acceder a información sobre comandos o funciones, y para ver tutoriales y manuales, se utiliza el símbolo ? tal como vemos en los siguientes ejemplos:

```
gap> ?tutorial:
gap> ?sets
gap> ?help
gap> ?permutations
gap> ?Eigenvalues
gap> ?CyclicGroup
gap> ?FreeGroup
gap> ?SylowSubgroup
```

Los comentarios en GAP comienzan con el símbolo #.

Si lo que queremos ingresar en la línea de comandos ocupa mucho lugar, el uso del símbolo \ aumenta la legibilidad. Este símbolo nos permite partir la línea de comandos, tal como vemos en el ejemplo siguiente:

```
gap> # Calculamos la suma 1+2+3
gap> 1\
> +2\
> +3;
6
```

La función `LogTo` permite guardar en un archivo todo lo que veamos en pantalla. Si la función `LogTo` se llama sin argumentos, el log dejará de escribirse:

```
gap> # Guardamos la salida de pantalla en el archivo mylog
gap> LogTo("mylog");
gap> # Dejamos de escribir la salida de pantalla en un archivo
gap> LogTo();
```

Para listar las variables globales creadas por el usuario utilizamos `NamesUserGVars`. La función `MemoryUsage` devuelve la cantidad de memoria (en bytes) utilizada por una variable. Veamos un ejemplo:

```
gap> p := 2^30;;
gap> NamesUserGVars();
[ "p" ]
gap> s := "una cadena de caracteres";;
gap> NamesUserGVars();
[ "p", "s" ]
gap> MemoryUsage(p);
8
```

La función `SaveWorkspace` guarda en un archivo una imagen de la memoria. Esta imagen puede ser cargada en una nueva instancia al ejecutar GAP con un parámetro extra: `gap -L <workspace>`.

1.1. Aritmética básica. Podemos hacer operaciones aritméticas elementales con números racionales:

```
gap> 1+1;
2
gap> 2*3;
6
gap> 8/2;
4
gap> (1/3)+(2/5);
11/15
gap> 2^(-4);
1/16
gap> 2*(-6)+4;
-8
gap> (1-5^2)^2-2*(2+4*2)^2;
376
```

Tomar resto módulo un entero dado se hace gracias al comando `mod`. Ejemplos:

```
gap> 6 mod 4;
2
gap> -6 mod 5;
4
```

Ciertas operaciones se hacen mediante el uso de funciones. Por ejemplo, podemos factorizar números con `Factors` y determinar si un número es primo con `IsPrime`:

```
gap> Factors(10);
[ 2, 5 ]
gap> Factors(18);
[ 2, 3, 3 ]
gap> Factors(1800);
[ 2, 2, 2, 3, 3, 5, 5 ]
gap> IsPrime(1800);
false
gap> Factors(37);
[ 37 ]
gap> IsPrime(37);
true
```

Otras funciones de utilidad son las siguientes: `Sqrt` para calcular raíces cuadradas, `Factorial` para el factorial de un entero positivo, `Gcd` para el máximo común divisor de una lista finita de números enteros, `Lcm` para el mínimo común múltiplo, etc. Ejemplos:

```
gap> Sqrt(25);
5
gap> Factorial(15);
1307674368000
gap> Gcd(10,4);
2
gap> Lcm(10,4,2,6);
60
```

Además de poder trabajar con el cuerpo `Rationals` de números racionales, GAP nos permite trabajar con cuerpos ciclotómicos. Para crear cuerpos ciclotómicos se usa la función `CF` y para crear raíces primitivas de la unidad disponemos de la función `E`. Más precisamente,

`E(n)` devuelve la n-ésima raíz de la unidad $e^{2\pi i/n}$. Típicamente, un ciclotómico se expresará en GAP como una combinación lineal racional de raíces primitivas de la unidad. Veamos algunos ejemplos:

```
gap> E(6) in Rationals;
false
gap> E(6) in Cyclotomics;
true
gap> E(3) in CF(3);
true
gap> E(3) in CF(4);
false
gap> E(3)^2+E(3);
-1
gap> E(5)^5-E(5);
-2*E(5)-E(5)^2-E(5)^3-E(5)^4
gap> E(6);
-E(3)^2
```

La función `Inverse` (resp. `AdditiveInverse`) devuelve el inverso multiplicativo (resp. aditivo) de un elemento. La característica de un cuerpo se obtiene con `Characteristic`:

```
gap> AdditiveInverse(2/3);
-2/3
gap> Inverse(2/3);
3/2
gap> AdditiveInverse(E(7));
-E(7)
gap> Inverse(E(7));
E(7)^6
gap> Characteristic(Rationals);
0
gap> Characteristic(CF(3));
0
gap> Characteristic(CF(4));
0
```

GAP también nos da la posibilidad de trabajar sobre cuerpos finitos. Para crear el cuerpo finito $\mathbb{F}_q$, donde $q = p^n$ es una potencia de un primo, se utiliza la función `GF`. Veamos algunos ejemplos:

```
gap> GF(2);
GF(2)
gap> GF(4);
GF(2^2)
gap> GF(9);
GF(3^2)
gap> Characteristic(GF(2));
2
gap> Characteristic(GF(9));
3
```

El subconjunto $\mathbb{F}_q^\times$ de elementos no nulos de $\mathbb{F}_q$ es un grupo cíclico, digamos $\mathbb{F}_q^\times = \langle\zeta\rangle$. Obviamente $\mathbb{F}_q = \{0, \zeta, \zeta^2, \ldots, \zeta^{q-1}\}$. Los elementos no nulos de $\mathbb{F}_q$ son potencias de ζ. En GAP los elementos no nulos del cuerpo finito `GF(q)` serán potencias de `Z(q)` y el cero de `GF(q)` se escribirá como `0*Z(q)`. La función `Zero` devuelve el cero de un cuerpo. Ejemplos:

```
gap> Size(GF(4));
4
gap> Elements(GF(4));
[ 0*Z(2), Z(2)^0, Z(2^2), Z(2^2)^2 ]
```

```
gap> Z(4);
Z(2^2)
gap> Inverse(Z(4));
Z(2^2)^2
gap> Zero(GF(4));
0*Z(2)
gap> 0 in GF(4);
false
gap> Zero(Rationals);
0
```

Similarmente, la función `One` devuelve la unidad de un cuerpo:

```
gap> One(GF(4));
Z(2)^0
gap> 1 in GF(4);
false
gap> One(Rationals);
1
```

2. Programación en GAP

2.1. Objetos y variables. Para GAP, un objeto es algo que puede asignarse a una variable. Un objeto puede ser un número, una cadena de caracteres, un cuerpo, un grupo, un elemento o un subgrupo de un grupo, un morfismo entre dos grupos, un anillo, una matriz, un espacio vectorial, etc. Asignar un objeto a una variable se hará mediante el operador `:=` tal como muestra el ejemplo siguiente:

```
gap> p := 32;;
gap> p;
32
gap> p = 32;
true
gap> p := p+1;;
gap> p;
33
gap> p = 32;
false
```

2.1.1. Observación. Es muy importante remarcar que los símbolos `=` (condicional) y `:=` (asignación) son distintos.

2.1.2. Observación. En caso de que nos hayamos olvidado de guardar en una variable el valor de una expresión, podemos hacer lo siguiente:

```
gap> 2*(5+1)-6;
6
gap> n := last;
6
```

También disponemos de `last2` y `last3`.

2.2. Condicionales.

2.2.1. Tenemos tres operadores lógicos de gran importancia: `not`, `and`, `or`. Tenemos además los operadores de comparación usuales. Por ejemplo, la expresión `x<>y` devolverá `true` si las variables `x` e `y` son distintas, y `false` en caso contrario. Veamos algunos ejemplos sencillos que no necesitan explicación:

```
gap> x := 20;; y := 10;;
gap> x <> y;
true
gap> x > y;
true
gap> (x > 0) or (x < y);
true
gap> (x > 0) and (x < y);
false
gap> (2*y < x);
false
gap> (2*y <= x);
true
gap> not (x < y);
true
```

2.3. Funciones.

2.3.1. En GAP existen dos formas de definir funciones. Pueden definirse en una línea, como por ejemplo

```
gap> square := x->x^2;
function( x ) ... end
```

o bien como

```
gap> square := function(x)
> return x^2;
> end;
function( x ) ... end
```

En ambos casos se obtiene el mismo resultado. Dejamos como ejercicio para el lector verificar que con ambas funciones se obtiene:

```
gap> square(4);
16
gap> square(-4);
16
gap> square(-5);
25
```

2.3.2. En los ejemplos vistos en 2.3.1, la función `square` devuelve el cuadrado del argumento `x`. Pueden también definirse funciones sin argumento, como por ejemplo:

```
gap> hi := function()
> Display("Hola mundo");
> end;
function(  ) ... end
gap> hi();
Hola mundo
```

2.3.3. Ejemplo. Vamos a definir la función

$$f\colon n \mapsto \begin{cases} n^3 & \text{si } n \equiv 0 \text{ mód } 3, \\ n^5 & \text{si } n \equiv 1 \text{ mód } 3, \\ 0 & \text{en otro caso.} \end{cases}$$

Veamos el código y algunos ejemplos:

```
gap> f := function(n)
> if n mod 3 = 0 then
> return n^3;
> elif n mod 3 = 1 then
> return n^5;
> else
> return 0;
> fi;
> end;
function( n ) ... end
gap> f(10);
100000
gap> f(5);
0
gap> f(4);
1024
```

2.3.4. Ejemplo. La sucesión de Fibonacci es la sucesión f_n definida por $f_1 = f_2 = 1$ y $f_{n+1} = f_n + f_{n-1}$ para $n \geqslant 2$. La siguiente función recursiva calcula el n-ésimo número de Fibonacci:

```
gap> fibonacci := function(n)
> if n = 1 or n = 2 then
> return 1;
> else
> return fibonacci(n-1)+fibonacci(n-2);
> fi;
> end;
function( n ) ... end
gap> fibonacci(10);
55
```

2.4. Strings (cadenas de caracteres).

2.4.1. Una cadena de caracteres es una expresión delimitada por el símbolo " (comilla doble):

```
gap> string := "hola mundo";
hola mundo
```

Para extraer el carácter ubicado en una determinada posición de la cadena, utilizamos la expresión `string[posición]`; para extraer subcadenas de caracteres, utilizamos la expresión `string{posiciones}`. Veamos algunos ejemplos:

```
gap> string[1];
'h'
gap> string[3];
'l'
gap> string{[1,2,3,4]};
"hola"
gap> string{[6,7,8,9,10]};
"mundo"
gap> string{[10,9,8,7,6,5,4,3,2,1]};
"odnum aloh"
```

2.4.2. Hay muchas funciones importantes que nos ayudan a trabajar con cadenas de caracteres. Por ejemplo, `String` convierte un objeto cualquiera en una cadena de caracteres:

```
gap> String(1234);
"1234"
gap> String(01234);
"1234"
gap> String([1,2,3]);
"[ 1, 2, 3 ]"
gap> String(true);
"true"
```

La función `ReplacedString` devuelve una lista que se obtuvo después de haber reemplazado palabras:

```
gap> ReplacedString("Hola mundo", "mundo", "a todos");
"Hola a todos"
```

2.4.3. La función `Print` nos permite imprimir (con formato) datos en pantalla:

```
gap> string := "Hola mundo";;
gap> Print(string);
Hola mundo
```

Veamos otro ejemplo:

```
gap> n := 100;;
gap> m := 5;;
gap> Print(n, " multiplicado por ", m, " es ", n*m);
100 multiplicado por 5 es 500
```

Los caracteres precedidos por \ (como por ejemplo `\n`) son tratados de forma especial:

```
gap> Print("Hola\nmundo");
Hola
mundo
gap> Print("Los caracteres precedidos por \\...");
Los caracteres precedidos por \...
```

La función `PrintTo` funciona como `Print` excepto que los argumentos se imprimen en un archivo.

2.5. Listas.

2.5.1. En GAP una lista es simplemente una sucesión ordenada de objetos, donde los objetos no necesariamente son todos del mismo tipo. La lista puede además tener lugares vacíos. Las listas se escriben entre corchetes. Veamos algunos ejemplos:

```
gap> IsList([1, 2, 3]);
true
gap> IsList([1, 2, 3, "abc"]);
true
gap> IsList([1, 2,, "abc"]);
true
gap> 2 in [1, 2, 5, 4, 10];
true
gap> 3 in [0,10,"abc"];
false
```

2.5.2. Vamos a crear una lista con los primeros seis números primos y calcularemos su tamaño con la función `Size`:

```
gap> primes := [2, 3, 5, 7, 11, 13];
[ 2, 3, 5, 7, 11, 13 ]
gap> Size(primes);
6
```

Se accede a un elemento de una lista al hacer referencia a la posición:

```
gap> primes[1];
2
gap> primes[2];
3
```

Veamos cómo obtener la sublista formada por los elementos ubicados en las posiciones $2, 3, 5$:

```
gap> primes;
[ 2, 3, 5, 7, 11, 13 ]
gap> primes{[2,3,5]};
[ 3, 5, 11 ]
```

Para evitar confusión, veamos otro ejemplo:

```
gap> list := ["a", "b", "c", "d", "e", "f"];
[ "a", "b", "c", "d", "e", "f" ]
gap> list{[1,3,5]};
[ "a", "c", "e" ]
```

Para conocer la posición de un determinado elemento de una lista se utiliza la función `Position`. Si el elemento buscado no pertenece a la lista, `Position` devolverá `fail`; en caso contrario, devolverá el primer lugar donde el elemento buscado aparece. Veamos algunos ejemplos:

```
gap> Position([5, 4, 6, 3, 7, 3, 7], 5);
1
gap> Position([5, 4, 6, 3, 7, 3, 7], 1);
fail
gap> Position([5, 4, 6, 3, 7, 3, 7], 7);
5
```

Para agregar uno o más elementos al final de una lista se utilizan la funciones `Add` y `Append`. Para eliminar el elemento de una lista ubicado en una determinada posición utilizamos la función `Remove`. Veamos algunos ejemplos:

```
gap> primes;
[ 2, 3, 5, 7, 11, 13 ]
gap> # Se agrega el primo 19 al final de la lista
gap> Add(primes, 19);
gap> primes;
[ 2, 3, 5, 7, 11, 13, 19 ]
gap> # Se agrega el primo 17 en la posición 7
gap> Add(primes, 17, 7);
gap> primes;
[ 2, 3, 5, 7, 11, 13, 17, 19 ]
gap> # Se agregan los primos 23 y 29 al final
gap> Append(primes, [23, 29]);
gap> primes;
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 ]
gap> # Se borra el primer elemento de la lista
gap> Remove(primes, 1);;
gap> primes;
[ 3, 5, 7, 11, 13, 17, 19, 23, 29 ]
```

2.5.3. La concatenación de listas se hace mediante `Concatenation`. Esta función devuelve una lista formada por la concatenación de las listas que se utilizaron como argumento. Veamos un ejemplo:

```
gap> Concatenation([1,2,3],[4,5,6]);
[ 1, 2, 3, 4, 5, 6 ]
```

A diferencia de `Append`, la función `Concatenation` no modifica las listas que se utilizan como argumento.

2.5.4. La función `Collected` devuelve la lista donde cada elemento de la lista original aparece junto a su multiplicidad. Veamos un ejemplo:

```
gap> Factors(720);
[ 2, 2, 2, 2, 3, 3, 5 ]
gap> Collected(last);
[ [ 2, 4 ], [ 3, 2 ], [ 5, 1 ] ]
```

2.5.5. Si se quiere hacer una copia de una lista se utilizará `ShallowCopy`. Veamos un ejemplo que muestra la diferencia entre el operador de asignación y `ShallowCopy`:

```
gap> a := [1, 2, 3, 4];;
gap> b := a;;
gap> c := ShallowCopy(a);;
gap> Add(a, 5);
gap> a;
[ 1, 2, 3, 4, 5 ]
gap> b;
[ 1, 2, 3, 4, 5 ]
gap> c;
[ 1, 2, 3, 4 ]
```

2.5.6. La función `Reversed` devuelve una lista que se obtiene al invertir el orden de los elementos de la lista que se envía como argumento. En el ejemplo siguiente observamos que `list` no se modifica con la función `Reversed`:

```
gap> list := [2, 4, 7, 3];;
gap> Reversed(list);
[ 3, 7, 4, 2 ]
gap> list;
[ 2, 4, 7, 3 ]
```

La función `SortedList` devuelve una lista que se obtiene al ordenar en forma creciente los elementos de la lista que se envía como argumento. El ejemplo siguiente observamos que `SortedList` no modifica a `list`:

```
gap> list := [2, 4, 7, 3];;
gap> SortedList(list);
[ 2, 3, 4, 7 ]
gap> list;
[ 2, 4, 7, 3 ]
```

La función `Sort` modifica el orden de los elementos de la lista que se envía como argumento. En el ejemplo siguiente observamos que `list` se modifica con la función `Sort`:

```
gap> list := [2, 4, 7, 3];;
gap> Sort(list);
gap> list;
[ 2, 3, 4, 7 ]
```

2.5.7. Observación. En importante mencionar que para aplicar las funciones `SortedList` y `Sort` los elementos de la lista deben ser admisibles de comparación por el operador < y además deben ser todos del mismo tipo.

2.5.8. La función `Filtered` sirve para filtrar elementos de una lista que satisfacen una determinada condición. Con `Number` se obtiene la cantidad de elementos de una lista que satisfacen una determinada condición. La función `First` devuelve el primer elemento de la lista que satisface la condición. Ejemplos:

```
gap> list := [1, 2, 3, 4, 5];;
gap> Filtered(list, x->x mod 2 = 0);
[ 2, 4 ]
gap> Number(list, x->x mod 2 = 0);
2
gap> Filtered(list, x->x mod 2 = 1);
[ 1, 3, 5 ]
gap> First(list, x->x mod 2 = 0);
2
```

En el ejemplo siguiente calculamos cuántas potencias de dos dividen a 18000:

```
gap> Factors(18000);
[ 2, 2, 2, 2, 3, 3, 5, 5, 5 ]
gap> Collected(Factors(18000));
[ [ 2, 4 ], [ 3, 2 ], [ 5, 3 ] ]
gap> Number(Factors(18000), x->x=2);
4
```

2.5.9. Existen formas muy prácticas de crear listas. Veamos dos ejemplos que no necesitan mucha explicación:

```
gap> List([1, 2, 3, 4, 5], x->x^2);
[ 1, 4, 9, 16, 25 ]
gap> List([1, 2, 3, 4, 5], IsPrime);
[ false, true, true, false, true ]
```

2.6. Rangos.

2.6.1. Un tipo de lista de enteros particular son los rangos. En este tipo de listas, la diferencia entre dos enteros consecutivos es una constante. Veamos algunos ejemplos que ilustran cómo se definen los rangos:

```
gap> Elements([1,3..11]);
[ 1, 3, 5, 7, 9, 11 ]
gap> Elements([1..5]);
[ 1, 2, 3, 4, 5 ]
gap> Elements([0,-2..-8]);
[ -8, -6, -4, -2, 0 ]
gap> IsRange([1..100]);
true
gap> IsRange([1,3,5,6]);
false
```

2.6.2. Vimos que con la función `Elements` podemos listar todos los elementos de un rango. Por otro lado, si fuera posible, una lista podrá representarse como rango con la función `ConvertToRangeRep`. Veamos algunos ejemplos:

```
gap> list := [ 1, 2, 3, 4, 5 ];;
gap> ConvertToRangeRep(list);;
gap> list;
[ 1 .. 5 ]
```

```
gap> list := [ 7, 11, 15, 19, 23 ];
gap> IsRange(list);
true
gap> ConvertToRangeRep(list);
gap> list;
[ 7, 11 .. 23 ]
```

2.7. Conjuntos.

2.7.1. En GAP, un conjunto es un tipo particular de lista densa (sin agujeros) de elementos ordenados estrictamente. (En particular, un conjunto no tendrá elementos repetidos.) En GAP puede obtenerse un conjunto a partir de una lista con `Set`.

```
gap> list := [1, 2, 3, 1, 5, 6, 2];;
gap> IsSet(list);
false
gap> Set(list);
[ 1, 2, 3, 5, 6 ]
```

2.7.2. Para agregar elementos a un conjunto se usan las funciones `AddSet` y `UniteSet`; para quitar elementos, `RemoveSet`. Ejemplos:

```
gap> set := Set([1, 2, 4, 5]);;
gap> # Agregamos el 10 al conjunto
gap> AddSet(set, 10);
gap> set;
[ 1, 2, 4, 5, 10 ]
gap> # Quitamos el 4 del conjunto
gap> RemoveSet(set, 4);
gap> set;
[ 1, 2, 5, 10 ]
gap> UniteSet(set, [1, 1, 5, 6]);
gap> set;
[ 1, 2, 5, 6, 10 ]
```

Las operaciones básicas de conjuntos pueden hacerse con las funciones `Union`, `Intersection`, `Difference`, `Cartesian`. Veamos algunos ejemplos:

```
gap> S := Set([1, 2, 8, 11]);;
gap> T := Set([2, 5, 7, 8]);;
gap> Intersection(S, T);
[ 2, 8 ]
gap> Union(S, T);
[ 1, 2, 5, 7, 8, 11 ]
gap> Difference(S, T);
[ 1, 11 ]
gap> Difference(T, S);
[ 5, 7 ]
gap> Difference(S, S);
[  ]
gap> Cartesian(S, T);
[ [ 1, 2 ], [ 1, 5 ], [ 1, 7 ], [ 1, 8 ], [ 2, 2 ],
  [ 2, 5 ], [ 2, 7 ], [ 2, 8 ], [ 8, 2 ], [ 8, 5 ],
  [ 8, 7 ], [ 8, 8 ], [ 11, 2 ], [ 11, 5 ],
  [ 11, 7 ], [ 11, 8 ] ]
```

2.8. Registros.

2.8.1. Un registro es un tipo de dato de fundamental importancia ya que permite empaquetar objetos dentro de una misma estructura. En el ejemplo siguiente, guardamos el punto $(1, 2)$ en un registro:

```
gap> point := rec(x := 1, y := 2);;
gap> point.x;
1
gap> point.y;
2
gap> RecNames( point );
[ "x", "y" ]
```

Puede utilizarse la función `IsBound` para determinar si registro posee un determinado componente. La función `Unbind` se utiliza para borrar una variable de la memoria. Por ejemplo:

```
gap> point := rec( x := 1, y := 2 );;
gap> IsBound(point.z);
false
gap> point.z := 3;;
gap> IsBound(point.z);
true
gap> point;
rec( x := 1, y := 2, z := 3 )
gap> Unbind(point.z);
gap> point;
rec( x := 1, y := 2 )
```

2.9. Bucles.

2.9.1. Como primer ejemplo vamos a verificar que

$$1 + 2 + 3 + \cdots + 100 = 5050.$$

Por supuesto, podríamos utilizar la función `Sum` que suma todos los elementos de una lista:

```
gap> Sum([1..100]);
5050
```

Una alternativa con un bucle `for ... do ... od` es la siguiente:

```
gap> s := 0;;
gap> for k in [1..100] do
> s := s+k;
> od;
gap> s;
5050
```

Equivalentemente podríamos haber utilizado un bucle `while ... do ... od`:

```
gap> s := 0;;
gap> k := 1;;
gap> while k<=100 do
> s := s+k;
> k := k+1;
> od;
gap> s;
5050
```

Por último, otra versión equivalente puede hacerse con un bucle tipo `repeat ... until`:

```
gap> s := 0;;
gap> k := 1;;
gap> repeat
> s := s+k;
> k := k+1;
> until k>100;
gap> s;
5050
```

2.9.2. Ejemplo. Vamos a definir una función no recursiva (y luego más eficiente) que calcula números de Fibonacci:

```
gap> fibonacci := function(n)
> local k, x, y, tmp;
> x := 1;
> y := 1;
> for k in [3..n] do
>   tmp := y;
>   y := x+y;
>   x := tmp;
> od;
> return y;
> end;
function( n ) ... end
gap> fibonacci(100);
354224848179261915075
gap> fibonacci(1000);
43466557686937456435688527675040625802564660517 3\
71780402481729089536555417949051890403879840079 2\
55169295922593080322634775209689623239873322471 1\
61642996440906533187938298969649928516003704476 1\
37795166849228875
```

2.9.3. Los divisores de un número entero se obtienen con la función `DivisorsInt`. En el ejemplo siguiente haremos un bucle sobre los divisores de 100 e imprimiremos solamente aquellos divisores impares:

```
gap> Filtered(DivisorsInt(100), x->x mod 2 = 1);
[ 1, 5, 25 ]
```

Alternativamente:

```
gap> for d in DivisorsInt(100) do
> if d mod 2 = 1 then
> Display(d);
> fi;
> od;
1
5
25
```

Al utilizar `continue` pueden saltearse iteraciones de un bucle. Podríamos entonces haber hecho lo siguiente:

```
gap> for d in DivisorsInt(100) do
> if d mod 2 = 0 then
> continue;
> fi;
> Display(d);
> od;
```

```
1
5
25
```

2.9.4. Con `break` puede terminarse un bucle. En el ejemplo siguiente se recorren los primeros cien enteros positivos en busca del primero cuyo cuadrado sea divisible por 20.

```
gap> First([1..100], x->x^2 mod 20 = 0);
10
```

Alternativamente:

```
gap> for k in [1..100] do
> if k^2 mod 20 = 0 then
> Display(k);
> break;
> fi;
> od;
10
```

2.9.5. La función `ForAny` devuelve `true` si algún elemento de la lista satisface la condición impuesta y `false` en caso contrario. Similarmente, la función `ForAll` devuelve `true` si todos los elementos de la lista satisfacen un determinado criterio y `false` en caso contrario. Veamos algunos ejemplos:

```
gap> ForAny([2,4,6,8,10], x->x mod 2 = 0);
true
gap> ForAll([2,4,6,8,10], x->(x > 0));
true
gap> ForAny([2,3,4,5], IsPrime);
true
gap> ForAll([2,3,4,5], IsPrime);
false
```

3. Permutaciones y matrices

3.1. Permutaciones. Recordemos que dado $n \in \mathbb{N}$, una **permutación** en n letras es una función biyectiva $\sigma\colon \{1,\dots,n\} \to \{1,\dots,n\}$. En GAP las permutaciones se escriben como producto ciclos disjuntos. Por ejemplo:

$$\begin{pmatrix}1234\\2413\end{pmatrix} = (1243), \qquad \begin{pmatrix}12345\\21435\end{pmatrix} = (12)(34)(5) = (12)(34).$$

La permutación $\left(\begin{smallmatrix}12345\\21435\end{smallmatrix}\right) = (12)(34)$ en GAP se escribe como `(1,2)(3,4)`. La función `IsPerm` verifica si algo es o no una permutación. Veamos algunos ejemplos:

```
gap> IsPerm((1,2)(3,4));
true
gap> (1,2)(3,4)(5)=(1,2)(3,4);
true
gap> (1,2)(3,4)=(3,4)(2,1);
true
```

En GAP la imagen de un elemento `i` bajo la acción natural (a derecha) de una permutación `p` es `i^p`. La preimagen de un elemento `i` por la permutación `p` se recupera con `i/p`. En el ejemplo siguiente calculamos la imagen de 1 y la preimagen de 3 por la permutación (123):

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

La composición de permutaciones en GAP se hace de izquierda a derecha. Aquí tenemos un ejemplo:

```
gap> (1,2,3) * (2,3,4);
(1,3)(2,4)
```

La inversa de una permutación puede obtenerse con la función `Inverse`:

```
gap> Inverse((1,2,3));
(1,3,2)
gap> (1,2,3)^(-1);
(1,3,2)
```

Si se tiene una permutación σ escrita como producto de ciclos disjuntos, la función `ListPerm` devuelve una lista (del tamaño que uno quiera) que en el lugar i-ésimo tiene el valor $\sigma(i)$. Recíprocamente, una lista de ese tipo puede escribirse como producto de ciclos disjuntos con la función `PermList`. Veamos algunos ejemplos:

```
gap> # La permutación (12) vista en S2
gap> ListPerm((1,2));
[ 2, 1 ]
gap> # La permutación (12) vista en S4
gap> ListPerm((1,2), 4);
[ 2, 1, 3, 4 ]
gap> ListPerm((1,2,3)(4,5));
[ 2, 3, 1, 5, 4 ]
gap> ListPerm((1,3));
[ 3, 2, 1 ]
gap> PermList([1,2,3]);
()
gap> PermList([2,1]);
(1,2)
```

Se define el **signo** de una permutación σ como el número $(-1)^k$, donde $\sigma = \tau_1 \cdots \tau_k$ es alguna factorización de σ como producto de trasposiciones. Para calcular el signo de una permutación se utiliza la función `SignPerm`. Veamos algunos ejemplos:

```
gap> SignPerm(());
1
gap> SignPerm((1,2));
-1
gap> SignPerm((1,2,3,4,5));
1
gap> SignPerm((1,2)(3,4,5));
-1
gap> SignPerm((1,2)(3,4));
1
```

3.2. Matrices.

3.2.1. En GAP una matriz es simplemente un arreglo rectangular de números. El tamaño de una matriz se obtiene con la función `DimensionsMat`. Para imprimir agradablemente en pantalla una matriz, usamos la función `Display`. Veamos algunos ejemplos:

```
gap> m := [[1,2,3],[4,5,6]];;
gap> Display(m);
[ [  1,  2,  3 ],
  [  4,  5,  6 ] ]
gap> m[1][1];
1
gap> m[1][2];
```

```
2
gap> m[2][1];
4
gap> DimensionsMat(m);
[ 2, 3 ]
```

3.2.2. Hagamos algunas operaciones con matrices. Sean

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \end{pmatrix}, \quad C = \begin{pmatrix} 1 & 2 \\ 6 & 1 \\ 0 & 2 \end{pmatrix}.$$

El código siguiente calcula A^3, BC, CB, $A + CB$ y $2A - 5CB$:

```
gap> A := [[1,1,1],[0,1,1],[0,0,1]];;
gap> B := [[1,4,7],[2,5,8]];;
gap> C := [[1,2],[6,1],[0,2]];;
gap> Display(A^3);
[ [  1,  3,  6 ],
  [  0,  1,  3 ],
  [  0,  0,  1 ] ]
gap> Display(B*C);
[ [  25,  20 ],
  [  32,  25 ] ]
gap> Display(C*B);
[ [   5,  14,  23 ],
  [   8,  29,  50 ],
  [   4,  10,  16 ] ]
gap> Display(A+C*B);
[ [   6,  15,  24 ],
  [   8,  30,  51 ],
  [   4,  10,  17 ] ]
gap> Display(2*A-5*C*B);
[ [   -23,   -68,  -113 ],
  [   -40,  -143,  -248 ],
  [   -20,   -50,   -78 ] ]
```

3.2.3. Para construir una matriz nula se utiliza la función `NullMat`. La identidad se construye con `IdentityMat`. Para construir matrices diagonales se usa la función `DiagonalMat`. Veamos algunos ejemplos:

```
gap> Display(NullMat(2,3));
[ [  0,  0,  0 ],
  [  0,  0,  0 ] ]
gap> Display(IdentityMat(3));
[ [  1,  0,  0 ],
  [  0,  1,  0 ],
  [  0,  0,  1 ] ]
gap> Display(DiagonalMat([1,2]));
[ [  1,  0 ],
  [  0,  2 ] ]
```

3.2.4. Ya vimos que el lugar (i, j) de una matriz se obtiene con la expresión `matriz[i][j]`. Para extraer submatrices de una matriz, se usa la expresión `matriz{filas}{columnas}` tal como muestra el ejemplo siguiente:

```
gap> m := [\
> [1, 2, 3, 4, 5],\
> [6, 7, 8, 9, 3],\
> [3, 2, 1, 2, 4],\
```

```
> [7, 5, 3, 0, 0],\
> [0, 0, 0, 0, 1]];
gap> m{[1..3]}{[1..3]};
[ [ 1, 2, 3 ], [ 6, 7, 8 ], [ 3, 2, 1 ] ]
gap> m{[2,4,5]}{[1,3]};
[ [ 6, 8 ], [ 7, 3 ], [ 0, 0 ] ]
```

3.2.5. GAP puede trabajar con matrices con elementos en cuerpos o anillos. En el ejemplo siguiente creamos una matriz en $\mathbb{F}_5$:

```
gap> m := [[1,2,3],[3,2,1],[0,0,2]]*One(GF(5));
[ [ Z(5)^0, Z(5), Z(5)^3 ],
  [ Z(5)^3, Z(5), Z(5)^0 ],
  [ 0*Z(5), 0*Z(5), Z(5) ] ]
gap> Display(m);
 1 2 3
 3 2 1
 . . 2
```

En el código siguiente, en cambio, creamos la matriz identidad del anillo de matrices de 3×3 con elementos en el anillo $\mathbb{Z}/4$:

```
gap> m := IdentityMat(3, ZmodnZ(4));;
gap> Display(m);
matrix over Integers mod 4:
[ [  1,  0,  0 ],
  [  0,  1,  0 ],
  [  0,  0,  1 ] ]
```

3.2.6. La función `IsIdentityMat` devuelve `true` si el argumento es la matriz identidad o `false` en caso contrario. En el siguiente ejemplo calculamos la inversa de una matriz con la función `Inverse` y la traspuesta de una matriz con `TransposedMat`:

```
gap> m := [[1, -2, -1], [0, 1, 0], [1, -1, 0]];;
gap> Display(Inverse(m));
[ [   0,   1,   1 ],
  [   0,   1,   0 ],
  [  -1,  -1,   1 ] ]
gap> IsIdentityMat(m*Inverse(m));
true
gap> Display(TransposedMat(m)*m);
[ [   2,  -3,  -1 ],
  [  -3,   6,   2 ],
  [  -1,   2,   1 ] ]
```

3.2.7. Ejemplo. Es fácil demostrar por inducción que si f_n es la sucesión de Fibonacci entonces

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n = \begin{pmatrix} f_{n-1} & f_n \\ f_n & f_{n+1} \end{pmatrix}$$

para todo $n \geqslant 1$.

El siguiente código utiliza esta fórmula para dar una forma alternativa (y muy eficiente) de construir números de Fibonacci:

```
gap> fibonacci := function(n)
> local m;
> m := [[0,1],[1,1]]^n;;
> return m[1][2];
> end;
function( n ) ... end
```

```
gap> fibonacci(10);
55
gap> fibonacci(100000);
<integer 259...875 (20899 digits)>
```

3.2.8. Si A es una matriz de $m \times n$ y b es una matriz de $1 \times n$ (o vector fila de n coordenadas), la función `SolutionMat` devuelve (si es posible) algún x de tamaño $1 \times m$ tal que $xA = b$. Esta función devuelve `fail` si el sistema lineal no tiene solución o alguna solución en caso contrario. Veamos algunos ejemplos:

```
gap> # Un ejemplo con una única solución
gap> SolutionMat([[1,2],[3,4]], [13,2]);
[ -23, 12 ]
gap> # Un ejemplo con infinitas soluciones
gap> SolutionMat([[1,2],[2,4]], [1,2]);
[ 1, 0 ]
gap> # Un ejemplo sin soluciones
gap> SolutionMat([[1,2],[2,4]], [1,3]);
fail
```

3.2.9. La traza de una matriz cuadrada se calcula con la función `Trace`. El determinante de una matriz cuadrada, con `Determinant`. El rango de una matriz se calcula con la función `Rank`. Ejemplos:

```
gap> m := [[1, 2, 3],[5, 4, 3],[0, 0, 2]];;
gap> Determinant(m);
-12
gap> Trace(m);
7
gap> Rank(m);
3
```

4. Grupos y morfismos

4.1. Construcciones básicas.

4.1.1. Un **grupo de matrices** es un subgrupo de $\mathbf{GL}(n, \mathbb{K})$ para algún $n \in \mathbb{N}$ y algún cuerpo $\mathbb{K}$. Un **grupo de permutaciones** en un subgrupo de $\mathbb{S}_n$. En GAP pueden construirse grupos de matrices o de permutaciones con la función `Group`, que devuelve el grupo generado por una lista de matrices o de permutaciones.

4.1.2. Ejemplo. Con la función `Order` vamos a calcular el orden del grupo generado por (12), del grupo generado por (12345) y del grupo generado por $\{(12), (12345)\}$:

```
gap> Order(Group([(1,2)]));
2
gap> Order(Group([(1,2,3,4,5)]));
5
gap> Order(Group([(1,2),(1,2,3,4,5)]));
120
```

4.1.3. Para cada $n \in \mathbb{N}$, el grupo cíclico de orden n (escrito multiplicativamente) será denotado por C_n. Para construir **grupos cíclicos** se utiliza la función `CyclicGroup`. Sin argumentos adicionales, esta función devolverá una presentación abstracta del grupo cíclico del orden pedido.

4.1.4. Ejemplo. Vamos a construir el grupo cíclico C_2 de dos elementos como grupo abstracto, como grupo de matrices y como grupo de permutaciones:

```
gap> CyclicGroup(2);
<pc group of size 2 with 1 generators>
gap> CyclicGroup(IsMatrixGroup, 2);
Group([ [ [ 0, 1 ], [ 1, 0 ] ] ])
gap> CyclicGroup(IsPermGroup, 2);
Group([ (1,2) ])
```

4.1.5. Para fijar notación recordemos que, para cada $n \in \mathbb{N}$, se define el **grupo diedral** de orden $2n$ como el grupo

$$\mathbb{D}_{2n} = \langle r, s : srs = r^{-1},\ s^2 = r^n = 1 \rangle.$$

Para construir grupos diedrales utilizamos la función `DihedralGroup`. Sin opciones adicionales, la función `DihedralGroup` devuelve una presentación abstracta del grupo diedral. Para presentar al grupo diedral como grupo de permutaciones, agregamos el filtro `IsPermGroup`, tal como se hizo en la construcción de grupos cíclicos.

4.1.6. Ejemplo. Vamos a construir el grupo diedral $\mathbb{D}_6$, vamos a calcular su orden y verificaremos no es un grupo abeliano:

```
gap> D6 := DihedralGroup(6);;
gap> Order(D6);
6
gap> IsAbelian(D6);
false
```

Para listar los elementos de un grupo podemos utilizar la función `Elements`:

```
gap> Elements(DihedralGroup(6));
[ <identity> of ..., f1, f2, f1*f2, f2^2, f1*f2^2 ]
gap> Elements(DihedralGroup(IsPermGroup, 6));
[ (), (2,3), (1,2), (1,2,3), (1,3,2), (1,3) ]
```

4.1.7. El grupo simétrico $\mathbb{S}_n$ se construye con la función `SymmetricGroup`. Las permutaciones del conjunto $\{1, \ldots, n\}$ que vimos en 3.1 son los elementos del grupo simétrico $\mathbb{S}_n$. El grupo alternado $\mathbb{A}_n$ de grado n se construye con la función `AlternatingGroup`. Los elementos del grupo alternado $\mathbb{A}_n$ son las permutaciones de $\mathbb{S}_n$ de signo uno.

4.1.8. Ejemplo. Vamos a construir los grupos $\mathbb{A}_4$ y $\mathbb{S}_4$ y listaremos sus elementos con la función `Elements`:

```
gap> S4 := SymmetricGroup(4);;
gap> A4 := AlternatingGroup(4);;
gap> Elements(A4);
[ (), (2,3,4), (2,4,3), (1,2)(3,4), (1,2,3), (1,2,4),
  (1,3,2), (1,3,4), (1,3)(2,4), (1,4,2), (1,4,3),
  (1,4)(2,3) ]
gap> Elements(S4);
[ (), (3,4), (2,3), (2,3,4), (2,4,3), (2,4), (1,2),
  (1,2)(3,4), (1,2,3), (1,2,3,4), (1,2,4,3), (1,2,4),
  (1,3,2), (1,3,4,2), (1,3), (1,3,4), (1,3)(2,4),
  (1,3,2,4), (1,4,3,2), (1,4,2), (1,4,3), (1,4),
  (1,4,2,3), (1,4)(2,3) ]
```

Podemos verificar por ejemplo que:

```
gap> (1,2,3) in A4;
true
gap> (1,2) in A4;
false
gap> (1,2,3)(4,5) in S4;
false
```

4.1.9. Ejemplo. La función `Order` también puede usarse para obtener el orden de un elemento. Comprobaremos que $\mathbb{S}_3$ tiene dos elementos de orden tres, y tres elementos de orden dos:

```
gap> S3 := SymmetricGroup(3);;
gap> List(S3, Order);
[ 1, 2, 3, 2, 3, 2 ]
gap> Collected(List(S3, Order));
[ [ 1, 1 ], [ 2, 3 ], [ 3, 2 ] ]
```

4.1.10. Ejemplo. En este ejemplo demostraremos que

$$G = \left\langle \begin{pmatrix} 0 & i \\ i & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \right\rangle$$

es un grupo no abeliano de orden ocho y que no es isomorfo a un grupo diedral. Recordemos que, en GAP, la unidad imaginaria $i = \sqrt{-1}$ se escribe como `E(4)`. Para comprobar que $G \not\cong \mathbb{D}_8$ veremos que G tiene un único elemento de orden dos mientras que $\mathbb{D}_8$ tiene cinco elementos de orden dos:

```
gap> a := [[0,E(4)],[E(4),0]];;
gap> b := [[0,1],[-1,0]];;
gap> G := Group([a,b]);;
gap> Order(G);
8
gap> IsAbelian(G);
false
gap> Number(G, x->Order(x)=2);
1
gap> Number(DihedralGroup(8), x->Order(x)=2);
5
```

4.1.11. Ejemplo. El grupo de Mathieu M_{11} es un grupo simple de orden 7920 y puede definirse como el subgrupo de $\mathbb{S}_{11}$ generado por las permutaciones

$$(1\,2\,3\,4\,5\,6\,7\,8\,9\,10\,11), \quad (3\,7\,11\,8)(4\,10\,5\,6).$$

Construiremos M_{11} y con la función `IsSimple` comprobaremos que M_{11} es un grupo simple:

```
gap> a := (1,2,3,4,5,6,7,8,9,10,11);;
gap> b := (3,7,11,8)(4,10,5,6);;
gap> M11 := Group([a,b]);;
gap> Order(M11);
7920
gap> IsSimple(M11);
true
```

4.1.12. Ejemplo. La función `Group` también puede utilizarse para construir grupos infinitos. En este ejemplo mostramos dos matrices de orden finito cuya multiplicación tiene orden infinito:

```
gap> a := [[0,-1],[1,0]];;
gap> b:= [[0,1],[-1,-1]];;
gap> Order(a);
4
gap> Order(b);
3
gap> Order(a*b);
infinity
gap> Order(Group([a,b]));
infinity
```

Es importante destacar que GAP no siempre será capaz de determinar si un elemento tiene orden infinito.

4.1.13. La función `Subgroup` nos permite construir el subgrupo de un grupo dado generado por una lista de elementos. La función `AllSubgroups` devuelve una lista con todos los subgrupos de un grupo dado. El índice de un subgrupo se calcula con `Index`.

4.1.14. Ejemplo. Vamos a verificar que el subgrupo de $\mathbb{S}_3$ generado por la trasposición (12) es $\{\mathrm{id}, (12)\}$ y tiene índice tres, y el subgrupo de $\mathbb{S}_3$ generado por (123) es $\{\mathrm{id}, (123), (132)\}$ y tiene índice dos:

```
gap> S3 := SymmetricGroup(3);;
gap> Elements(Subgroup(S3, [(1,2)]));
[ (), (1,2) ]
gap> Index(S3, Subgroup(S3, [(1,2)]));
3
gap> Elements(Subgroup(S3, [(1,2,3)]));
[ (), (1,2,3), (1,3,2) ]
gap> Index(S3, Subgroup(S3, [(1,2,3)]));
2
```

4.1.15. Recordemos que un subgrupo K de un grupo G es normal si $gKg^{-1} \subseteq K$ para todo $g \in G$. Si K es un subgrupo normal de G entonces el cociente G/K es un grupo.

4.1.16. Ejemplo. Utilizaremos la función `IsSubgroup` para verificar que $\mathbb{A}_4$ es un subgrupo de $\mathbb{S}_4$. Más aún, con la función `IsNormal`, veremos que $\mathbb{A}_4$ es un subgrupo normal de $\mathbb{S}_4$:

```
gap> S4 := SymmetricGroup(4);;
gap> A4 := AlternatingGroup(4);;
gap> IsSubgroup(S4,A4);
true
gap> IsNormal(S4,A4);
true
gap> Order(S4/A4);
2
```

Similarmente, el subgrupo de $\mathbb{S}_4$ generado por (123) no es normal en $\mathbb{S}_4$:

```
gap> IsNormal(S4, Subgroup(S4, [(1,2,3)]));
false
```

4.1.17. Ejemplo. En este ejemplo demostraremos que en $\mathbb{D}_8$ existen subgrupos H y K tales que K es normal en H, H es normal en G y K no es normal en G:

```
gap> D8 := DihedralGroup(IsPermGroup, 8);;
gap> Elements(D8);
[ (), (2,4), (1,2)(3,4), (1,2,3,4),
  (1,3), (1,3)(2,4), (1,4,3,2),
  (1,4)(2,3) ]
gap> K := Subgroup(D8, [(2,4)]);;
gap> Elements(K);
[ (), (2,4) ]
gap> H := Subgroup(D8, [(1,2,3,4)^2,(2,4)]);;
gap> Elements(H);
[ (), (2,4), (1,3), (1,3)(2,4) ]
gap> IsNormal(D8, K);
false
gap> IsNormal(D8, H);
true
gap> IsNormal(H, K);
true
```

4.1.18. Ejemplo. Vamos a construir el grupo cíclico C_4 como grupo de permutaciones y calcularemos todos sus cocientes. Como C_4 es un grupo abeliano, todo subgrupo de C_4 es normal. Usaremos entonces la función `AllSubgroups` para verificar que C_4 tiene únicamente un subgrupo propio no trivial K. Veremos además que el cociente C_4/K tiene dos elementos:

```
gap> C4 := CyclicGroup(IsPermGroup, 4);;
gap> AllSubgroups(C4);
[ Group(()), Group([ (1,3)(2,4) ]),
  Group([ (1,2,3,4) ]) ]
gap> K := last[2];;
gap> Order(C4/K);
2
```

4.1.19. Recordemos que para cada $n \in \mathbb{N}$ se define el grupo de **cuaterniones generalizados** como el grupo

$$Q_{4n} = \langle x, y \mid x^{2n} = y^4 = 1, x^n = y^2, y^{-1}xy = x^{-1} \rangle.$$

Para construir el grupo de **cuaterniones generalizados** se utiliza la función `QuaternionGroup`. Para obtener una presentación de Q_{4n} como grupo de permutaciones (resp. grupo de matrices) debe utilizarse el filtro `IsPermGroup` (resp. `IsMatrixGroup`).

4.1.20. Ejemplo. Construiremos el grupo no abeliano Q_8 de cuaterniones de orden ocho como grupo de matrices. Calcularemos todos sus subgrupos con la función `AllSubgroups`, verificaremos que todo subgrupo de Q_8 es normal y que Q_8 es no abeliano:

```
gap> Q8 := QuaternionGroup(IsMatrixGroup, 8);;
gap> IsAbelian(Q8);
false
gap> ForAll(AllSubgroups(Q8), x->IsNormal(Q8,x));
true
```

4.1.21. Recordemos que si G es un grupo, se define el **centro** de G como el grupo

$$Z(G) = \{x \in G : xy = yx \text{ para todo } y \in G\}.$$

El **conmutador** de dos elementos $x, y \in G$ se define como $[x, y] = x^{-1}y^{-1}xy$. El subgrupo de conmutadores (o conmutador de G) es el subgrupo $[G, G]$ de G generado por los conmutadores de G.

4.1.22. Ejemplo. Comprobaremos que el centro de $\mathbb{A}_4$ es trivial y que el conmutador de $\mathbb{A}_4$ es el grupo $\{\mathrm{id}, (12)(34), (13)(24), (14)(23)\}$:

```
gap> A4 := AlternatingGroup(4);;
gap> IsTrivial(Center(A4));
true
gap> Elements(DerivedSubgroup(A4));
[ (), (1,2)(3,4), (1,3)(2,4), (1,4)(2,3) ]
```

4.1.23. Ejemplo. Para construir producto directo de grupos se usa la función `DirectProduct`. En este ejemplo comprobaremos que los grupos $C_4 \times C_4$ y $C_2 \times Q_8$ tiene orden 16 y poseen tres elementos de orden dos y doce elementos de orden cuatro. Sin embargo, $C_4 \times C_4 \ncong C_2 \times Q_8$ pues $C_4 \times C_4$ es abeliano y $C_2 \times Q_8$ no lo es:

```
gap> C4 := CyclicGroup(IsPermGroup, 4);;
gap> C2 := CyclicGroup(IsPermGroup, 2);;
gap> Q8 := QuaternionGroup(8);;
gap> C4xC4 := DirectProduct(C4, C4);;
gap> C2xQ8 := DirectProduct(C2, Q8);;
gap> List(C4xC4, Order);
[ 1, 4, 2, 4, 4, 4, 4, 4, 2, 4, 2, 4, 4, 4, 4, 4 ]
gap> Collected(List(C4xC4, Order));
```

```
[ [ 1, 1 ], [ 2, 3 ], [ 4, 12 ] ]
gap> List(C2xQ8, Order);
[ 1, 4, 4, 2, 4, 4, 4, 4, 2, 4, 4, 2, 4, 4, 4, 4 ]
gap> Collected(List(C2xQ8, Order));
[ [ 1, 1 ], [ 2, 3 ], [ 4, 12 ] ]
gap> IsAbelian(C4xC4);
true
gap> IsAbelian(C2xQ8);
false
```

4.1.24. Recordemos que si G es un grupo y $g \in G$, la **clase de conjugación** de g en G es el conjunto $g^G = \{xgx^{-1} : x \in G\}$. El **centralizador** de g en G es el subgrupo

$$C_G(g) = \{x \in G : xg = gx\}.$$

Para calcular clases de conjugación, disponemos de `ConjugacyClasses` y `ConjugacyClass`. El centralizador de un elemento se calcula con `Centralizer`.

4.1.25. Ejemplo. Vamos a comprobar que $\mathbb{S}_3$ tiene tres clases de conjugación, los representantes de estas clases son id, (12) y (123). Además $(12)^{\mathbb{S}_3} = \{(12), (13), (23)\}$ y que $(123)^{\mathbb{S}_3} = \{(123), (132)\}$:

```
gap> S3 := SymmetricGroup(3);;
gap> ConjugacyClasses(S3);
[ ()^G, (1,2)^G, (1,2,3)^G ]
gap> Elements(ConjugacyClass(S3, (1,2)));
[ (2,3), (1,2), (1,3) ]
gap> Elements(ConjugacyClass(S3, (1,2,3)));
[ (1,2,3), (1,3,2) ]
```

La función `Centralizer` nos permite comprobar que $C_{\mathbb{S}_3}((123)) = \{\mathrm{id}, (123), (132)\}$:

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

4.1.26. Ejemplo. Con la función `Representative` armaremos una lista con un representante por cada clase de conjugación de $\mathbb{A}_4$:

```
gap> A4 := AlternatingGroup(4);;
gap> List(ConjugacyClasses(A4), Representative);
[ (), (1,2)(3,4), (1,2,3), (1,2,4) ]
```

4.1.27. Ejemplo. Vamos a utilizar la función `IsConjugate` para determinar si ciertos 3-ciclos de $\mathbb{A}_4$ son conjugados. En caso de que dos elementos sean conjugados, puede encontrarse un elemento que los conjuga con la función `RepresentativeAction`. Por ejemplo, las permutaciones (123) y $(132) = (123)^2$ no son conjugados en $\mathbb{A}_4$; en cambio, (123) y (134) son conjugados en $\mathbb{A}_4$ pues si $x = (234)$ entonces $(134) = x^{-1}(123)x$:

```
gap> A4 := AlternatingGroup(4);;
gap> g := (1,2,3);;
gap> h := (1,3,4);;
gap> IsConjugate(A4, g, g^2);
false
gap> IsConjugate(A4, g, h);
true
gap> x := RepresentativeAction(A4, g, h);
(2,3,4)
gap> x^(-1)*g*x=h;
true
```

4.1.28. Ejemplo. Este ejemplo está basado en [2]. En teoría de grupos, el teorema de Lagrange afirma si K es un subgrupo de un grupo finito G entonces el orden de K divide al orden de G. Es bien sabido que la afirmación recíproca no es cierta ya que, por ejemplo, $\mathbb{A}_4$ no tiene subgrupos de orden seis. A continuación, daremos varias demostraciones de este hecho.

La forma más inocente de demostrar que $\mathbb{A}_4$ no tiene subgrupos de orden seis consiste en estudiar cada uno de los $\binom{12}{6} = 924$ subconjuntos de $\mathbb{A}_4$ de seis elementos y determinar cuáles son subgrupos de $\mathbb{A}_4$. En el ejemplo siguiente primero se calculan estos 924 subconjuntos y luego, para cada subconjunto X se determina si X es subgrupo de $\mathbb{A}_4$ al contar cuántos elementos tiene el subgrupo de $\mathbb{A}_4$ generado por X:

```
gap> A4 := AlternatingGroup(4);;
gap> k := 0;;
gap> for x in Combinations(Elements(A4), 6) do
> if Size(Subgroup(A4, x))=Size(x) then
> k := k+1;
> fi;
> od;
gap> k;
0
```

Equivalentemente, esta idea de mirar cada uno de los subconjuntos de $\mathbb{A}_4$ de seis elementos puede llevarse a cabo con el siguiente código:

```
gap> ForAny(Combinations(Elements(A4), 6),\
> x->Size(Subgroup(A4, x))=Size(x));
false
```

Veamos una idea similar aunque más eficiente. Todo grupo de orden seis tiene cinco elementos no triviales. Comprobaremos que ninguno de los $\binom{11}{5} = 462$ subconjuntos de $\mathbb{A}_4$ con cinco elementos genera un subgrupo de orden seis. En el ejemplo, en vez de utilizar la función `Combinations`, generamos los subconjuntos iterativamente con `IteratorOfCombinations`:

```
gap> k := 0;;
gap> for t in IteratorOfCombinations(\
> Filtered(A4, x->not x = ()), 5) do
> if Size(Subgroup(A4, t))=Size(t)+1 then
> k := k+1;
> fi;
> od;
gap> k;
0
```

Otra forma de demostrar que no vale la recíproca al teorema de Lagrange es la siguiente: si existiera un subgrupo de $\mathbb{A}_4$ de orden seis, este subgrupo tendría índice dos en $\mathbb{A}_4$. Sin embargo, gracias a la función `SubgroupsOfIndexTwo` sabemos que en $\mathbb{A}_4$ no existen tales subgrupos:

```
gap> SubgroupsOfIndexTwo(A4);
[  ]
```

Otra demostración consiste en construir todos los subgrupos de $\mathbb{A}_4$ y verificar que ninguno de ellos tiene orden seis. En el siguiente código construimos todos los subgrupos de $\mathbb{A}_4$, calculamos sus órdenes, y verificamos que ninguno de estos subgrupos tiene orden seis:

```
gap> List(AllSubgroups(A4), Order);
[ 1, 2, 2, 2, 3, 3, 3, 3, 4, 12 ]
gap> 6 in last;
false
```

La solución anterior puede hacerse más eficiente al calcular las clases de conjugación de subgrupos de $\mathbb{A}_4$:

```
gap> c := ConjugacyClassesSubgroups(A4);;
gap> List(c, x->Order(Representative(x)));
[ 1, 2, 3, 4, 12 ]
gap> 6 in last;
false
```

Otra alternativa para demostrar que $\mathbb{A}_4$ no posee subgrupos de orden seis es utilizar clases de conjugación. Las clases de conjugación de $\mathbb{A}_4$ son:

$$\{\mathrm{id}\}, \qquad \{(243),(123),(134),(142)\},$$
$$\{(12)(34),(13)(24),(14)(23)\}, \qquad \{(234),(124),(132),(143)\}.$$

Si existiera un subgrupo K de $\mathbb{A}_4$ de orden seis, este subgrupo tendría índice dos y sería normal en $\mathbb{A}_4$. Entonces K debería estar formado por $\{\mathrm{id}\}$ y otras clases de conjugación de $\mathbb{A}_4$. Por razones de cardinalidad, esto es imposible. Con el siguiente código se construyen las clases de conjugación de $\mathbb{A}_4$:

```
gap> ConjugacyClasses(A4);
[ ()^G, (1,2)(3,4)^G, (1,2,3)^G, (1,2,4)^G ]
gap> Elements(ConjugacyClass(A4, ()));
[ () ]
gap> Elements(ConjugacyClass(A4, (1,2)(3,4)));
[ (1,2)(3,4), (1,3)(2,4), (1,4)(2,3) ]
gap> Elements(ConjugacyClass(A4, (1,2,3)));
[ (2,4,3), (1,2,3), (1,3,4), (1,4,2) ]
gap> Elements(ConjugacyClass(A4, (1,2,4)));
[ (2,3,4), (1,2,4), (1,3,2), (1,4,3) ]
```

Veamos cómo el commutador puede ayudarnos a demostrar que $\mathbb{A}_4$ no tiene subgrupos de orden seis. Si existiera un subgrupo K de orden seis, entonces K sería normal en $\mathbb{A}_4$ y el cociente $\mathbb{A}_4/K$ cíclico de orden dos y por lo tanto abeliano. Esto nos dice que el conmutador $[\mathbb{A}_4, \mathbb{A}_4]$ estaría contenido en K. Como

$$[\mathbb{A}_4, \mathbb{A}_4] = \{\mathrm{id}, (12)(34), (13)(24), (14)(23)\},$$

en particular, $[\mathbb{A}_4, \mathbb{A}_4]$ tiene cuatro elementos, y esto es una contradicción pues 4 no divide a 6. El código es:

```
gap> DerivedSubgroup(A4);
Group([ (1,4)(2,3), (1,3)(2,4) ])
gap> Elements(last);
[ (), (1,2)(3,4), (1,3)(2,4), (1,4)(2,3) ]
```

Una última variante. Si K es un subgrupo de $\mathbb{A}_4$ de orden seis, entonces es sabido que $K \simeq \mathbb{S}_3$ o $K \simeq C_6$. Veamos que $\mathbb{A}_4$ no tiene elementos de orden seis:

```
gap> Filtered(A4, x->Order(x)=6);
[  ]
```

Como $\mathbb{A}_4$ no tiene elementos de orden seis, $K \simeq \mathbb{S}_3$ y entonces K contiene tres elementos de orden dos. Esto implica que

$$\{\mathrm{id}, (12)(34), (13)(24), (14)(23)\}$$

es un subgrupo de K, una contradicción porque 4 no divide a 6.

4.1.29. Es bien sabido que si G es un grupo abeliano finito entonces existen $n_1, \ldots, n_k \in \mathbb{N}$ tales que $C_{n_1} \times \cdots \times C_{n_k}$. La construcción de **grupos abelianos** finitos se hace mediante la función `AbelianGroup`.

4.1.30. Ejemplo. Como ejemplo, vamos a construir el grupo abeliano $C_2 \times C_3$ y vamos a verificar que es isomorfo al grupo C_6. Para esto, veremos que en $C_2 \times C_3$ existe un elemento que tiene orden seis.

```
gap> C2xC3 := AbelianGroup(IsPermGroup, [2, 3]);;
gap> First(C2xC3, x->Order(x)=6);
(1,2)(3,4,5)
```

4.1.31. Ejemplo. Vamos a construir un grupo G tal que $G = \langle a\rangle \times \langle b\rangle$ con $C_4 \simeq \langle a\rangle$ y $C_2 \simeq \langle b\rangle$ y vamos a demostrar que

$$\langle a^2\rangle \simeq \langle b\rangle, \qquad G/\langle a^2\rangle \not\simeq G/\langle b\rangle.$$

Para estudiar isomorfismo entre grupos finitos, GAP dispone de la función `IsomorphismGroups`. Esta función devuelve `fail` si los grupos no son isomorfos, o algún isomorfismo en caso contrario.

```
gap> G := AbelianGroup(IsPermGroup, [4,2]);
Group([ (1,2,3,4), (5,6) ])
gap> K := Subgroup(G, [(5,6)]);;
gap> L := Subgroup(G, [(1,2,3,4)^2]);;
gap> IsomorphismGroups(K, L);
[ (5,6) ] -> [ (1,3)(2,4) ]
gap> IsomorphismGroups(G/K,G/L);
fail
```

Puede demostrarse que

$$\langle a^2\rangle \simeq \langle b\rangle \simeq C_2, \qquad G/\langle a^2\rangle \simeq C_4, \qquad G/\langle b\rangle \simeq C_2 \times C_2.$$

4.1.32. GAP dispone de funciones que permiten la construcción de grupos clásicos. Por ejemplo, la funcione `GL` nos permite construir los grupos $\mathbf{GL}(n,\mathbb{Z})$, $\mathbf{GL}(n,\mathbb{Z}/m)$ y $\mathbf{GL}(n,\mathbb{F}_q)$:

```
gap> Order(GL(2,Integers));
infinity
gap> Order(GL(2,ZmodnZ(4)));
96
gap> Order(GL(2,GF(4)));
180
gap> Order(GL(3,GF(4)));
181440
```

Similarmente, con la función `SL` pueden construirse grupos de matrices con determinante uno tales como $\mathbf{SL}(n,\mathbb{Z})$, $\mathbf{SL}(n,\mathbb{Z}/m)$ y $\mathbf{SL}(n,\mathbb{F}_q)$:

```
gap> Order(SL(2,GF(3)));
24
gap> Order(SL(2,Integers));
infinity
gap> Order(SL(2,ZmodnZ(4)));
48
gap> Order(SL(2,GF(4)));
60
```

4.1.33. Ejemplo. Este ejemplo fue tomado de [3]. Demostraremos que el conmutador de un grupo finito no siempre es igual al conjunto de conmutadores. Sea G el subgrupo de $\mathbb{S}_{16}$ generado por las permutaciones

$$a = (13)(24), \qquad b = (57)(6,8),$$
$$c = (911)(10,12), \qquad d = (13,15)(14,16),$$
$$e = (13)(5,7)(9,11), \qquad f = (12)(3,4)(13,15),$$
$$g = (56)(7,8)(13,14)(15,16), \qquad h = (9\ 10)(11\ 12).$$

Vamos a demostrar que $[G,G]$ tiene orden 16 y que el conjunto de conmutadores tiene 15 elementos. En particular, veremos que $cd \in [G,G]$ y sin embargo no es un conmutador:

```
gap> a := (1,3)(2,4);;
gap> b := (5,7)(6,8);;
gap> c := (9,11)(10,12);;
gap> d := (13,15)(14,16);;
gap> e := (1,3)(5,7)(9,11);;
gap> f := (1,2)(3,4)(13,15);;
gap> g := (5,6)(7,8)(13,14)(15,16);;
gap> h := (9,10)(11,12);;
gap> G := Group([a,b,c,d,e,f,g,h]);;
gap> D := DerivedSubgroup(G);;
gap> Size(D);
16
gap> Size(Set(List(Cartesian(G,G), Comm)));
15
gap> c*d in Difference(D,\
> Set(List(Cartesian(G,G), Comm)));
true
```

4.2. Morfismos.

4.2.1. GAP está preparado para que podamos trabajar con el conjunto de morfismos entre dos grupos. Existen varias formas de crear morfismos entre grupos. Por ejemplo, la función `GroupHomomorphismByImages` devuelve el morfismo de grupos que se construye a partir de una lista de generadores del dominio y el valor de sus imágenes en el codominio. Una vez que tenemos un morfismo, podemos estudiar algunas de sus propiedades con las funciones `Image`, `IsInjective`, `IsSurjective`, `Kernel`, `PreImage`, `PreImages`, etc.

4.2.2. Ejemplo. La aplicación $f\colon \mathbb{S}_4 \to \mathbb{S}_3$ que manda toda trasposición de $\mathbb{S}_4$ en (12) se extiende a un morfismo de grupos. Este morfismo f no es inyectivo pues $\ker f$ tiene doce elementos y no es sobreyectivo pues por ejemplo (123) no pertenece a la imagen de f:

```
gap> S4 := SymmetricGroup(4);;
gap> S3 := SymmetricGroup(3);;
gap> f := GroupHomomorphismByImages(S4, S3,\
> [(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)],\
> [(1,2),(1,2),(1,2),(1,2),(1,2),(1,2)]);
[ (1,2), (1,3), (1,4), (2,3), (2,4), (3,4) ] ->
[ (1,2), (1,2), (1,2), (1,2), (1,2), (1,2) ]
gap> Size(Kernel(f));
12
gap> IsInjective(f);
false
gap> Size(Image(f));
2
gap> IsSurjective(f);
false
gap> (1,2,3) in Image(f);
false
```

4.2.3. Ejemplo. Vamos a construir el grupo $C_{12} = \langle g : g^{12} = 1\rangle$ como un grupo de permutaciones, el subgrupo $K = \langle g^6\rangle$, y el cociente C_{12}/K. Con `NaturalHomomorphismByNormalSubgroup` se construye el epimorfismo canónico $C_{12} \to C_{12}/K$.

```
gap> g := (1,2,3,4,5,6,7,8,9,10,11,12);;
gap> C12 := Group(g);;
gap> K := Subgroup(C12, [g^6]);;
gap> f := NaturalHomomorphismByNormalSubgroup(C12, K);
[ (1,2,3,4,5,6,7,8,9,10,11,12) ] -> [ f1 ]
```

```
gap> Image(f, g^6);
<identity> of ...
```

4.2.4. El grupo de automorfismos de un grupo se calcula con la función `AutomorphismGroup`. Recordemos que si G es un grupo, los automorfismos de G de la forma $x \mapsto gxg^{-1}$, donde g es algún elemento de G, se llaman automorfismos interiores. La función `IsInnerAutomorphism` determina si un automorfismo es interior.

4.2.5. Ejemplo. Veamos que $\mathrm{Aut}(\mathbb{S}_3)$ es un grupo no abeliano de seis elementos:

```
gap> aut := AutomorphismGroup(SymmetricGroup(3));
<group of size 6 with 2 generators>
gap> IsAbelian(aut);
false
```

4.2.6. Ejemplo. Vamos a verificar que para $n \in \{2,3,4,5\}$ todo automorfismo de $\mathbb{S}_n$ es interior:

```
gap> for n in [2..5] do
> G := SymmetricGroup(n);;
> if ForAll(AutomorphismGroup(G),\
> x->IsInnerAutomorphism(x)) then
> Print("Todo automorfismo de S",\
> n, " es interior.\n");
> fi;
> od;
Todo automorfismo de S2 es interior.
Todo automorfismo de S3 es interior.
Todo automorfismo de S4 es interior.
Todo automorfismo de S5 es interior.
```

Se sabe que en $\mathbb{S}_6$ existen automorfismos no interiores:

```
gap> S6 := SymmetricGroup(6);;
gap> f := First(AutomorphismGroup(S6),\
> x->IsInnerAutomorphism(x)=false);
[ (1,2,3,4,5,6), (1,2) ] -> [ (2,3)(4,6,5), (1,2)(3,5)(4,6) ]
```

De hecho, el automorfismo de $\mathbb{S}_6$ que cumple $(123456) \mapsto (23)(465)$ y $(12) \mapsto (12)(35)(46)$ es un automorfismo no interior. Calculemos cuánto vale este morfismo en las trasposiciones:

```
gap> for t in ConjugacyClass(S6, (1,2)) do
> Print("f(", t, ")=", Image(f,t), "\n");
> od;
f((1,2))=(1,2)(3,5)(4,6)
f((1,3))=(1,6)(2,5)(3,4)
f((1,4))=(1,4)(2,3)(5,6)
f((1,5))=(1,5)(2,4)(3,6)
f((1,6))=(1,3)(2,6)(4,5)
f((2,3))=(1,3)(2,4)(5,6)
f((2,4))=(1,5)(2,6)(3,4)
f((2,5))=(1,6)(2,3)(4,5)
f((2,6))=(1,4)(2,5)(3,6)
f((3,4))=(1,2)(3,6)(4,5)
f((3,5))=(1,4)(2,6)(3,5)
f((3,6))=(1,5)(2,3)(4,6)
f((4,5))=(1,3)(2,5)(4,6)
f((4,6))=(1,6)(2,4)(3,5)
f((5,6))=(1,2)(3,4)(5,6)
```

4.2.7. La función `AllHomomorphisms` devuelve una lista con los morfismos entre dos grupos. La función `AllEndomorphisms` permite calcular todos los endomorfismos de un grupo.

4.2.8. Ejemplo. Demostraremos que existen diez endomorfismos de $\mathbb{S}_3$:

```
gap> S3 := SymmetricGroup(3);;
gap> Size(AllEndomorphisms(S3));
10
```

4.2.9. Ejemplo. El centro del grupo $C_2 \times \mathbb{S}_3$ no se preserva por endomorfismos de $C_2 \times \mathbb{S}_3$. A continuación, construiremos el grupo $C_2 \times \mathbb{S}_3$ con la función `DirectProduct`, verificaremos que $Z(C_2 \times \mathbb{S}_3) = \{\mathrm{id}, (12)\}$ y luego veremos existe al menos un endomorfismo de $C_2 \times \mathbb{S}_3$ que no deja fijo al generador del centro:

```
gap> C2 := CyclicGroup(IsPermGroup, 2);;
gap> S3 := SymmetricGroup(3);;
gap> C2xS3 := DirectProduct(C2, S3);;
gap> Center(C2xS3);
Group([ (1,2) ])
gap> ForAll(AllEndomorphisms(C2xS3),\
> f->Image(f,(1,2)) in [(), (1,2)]);
false
```

4.2.10. Ejemplo. Comprobaremos que $\mathrm{Aut}(\mathbb{S}_6)/\mathrm{Inn}(\mathbb{S}_6) \simeq C_2$. Para construir el grupo de automorfismos $\mathrm{Aut}(\mathbb{S}_6)$ utilizaremos la función `AutomorphismGroup`. Para construir $\mathrm{Inn}(\mathbb{S}_6)$ utilizaremos la función `InnerAutomorphismsAutomorphismGroup`:

```
gap> S6 := SymmetricGroup(6);;
gap> A := AutomorphismGroup(S6);;
gap> Size(A);
1440
gap> I := InnerAutomorphismsAutomorphismGroup(A);;
gap> Order(A/I);
2
```

4.3. SmallGroups.

4.3.1. GAP contiene una base de datos con la lista completa de grupos finitos de orden $\leqslant 2000$ con excepción de los 423164062 grupos de orden 1024. Los autores son H, Besche, B. Eick y E. O'Brien. Esta librería contiene además grupos que satisfacen otros criterios tales como tener orden p^n con $n \leqslant 6$, o tener orden libre de cuadrados, etc. `SmallGroups` hace de GAP una herramienta muy útil para analizar grupos de orden pequeño y buscar contraejemplos. Muchos de los ejemplos que mostramos a continuación provienen de [7].

4.3.2. Ejemplo. La función `SmallGroupsInformation` imprime información sobre alguno de los grupos contenidos en `SmallGroups`. Veamos por ejemplo qué puede decirnos `SmallGroups` de los grupos de orden doce:

```
gap> SmallGroupsInformation(12);

  There are 5 groups of order 12.
    1 is of type 6.2.
    2 is of type c12.
    3 is of type A4.
    4 is of type D12.
    5 is of type 2^2x3.

  The groups whose order factorises in at most 3 primes
  have been classified by O. Hoelder. This classification is
  used in the SmallGroups library.
```

```
This size belongs to layer 1 of the SmallGroups library.
IdSmallGroup is available for this size.
```

4.3.3. Ejemplo. GAP contiene una función que da información parcial acerca de la estructura de un grupo: `StructureDescription`. Veamos qué puede decirnos esta función acerca de los subgrupos de orden doce:

```
gap> List(AllSmallGroups(Size, 12),\
> StructureDescription);
[ "C3 : C4", "C12", "A4", "D12", "C6 x C2" ]
```

El grupo denotado por `C3 : C4` es un producto semidirecto $C_3 \rtimes C_4$.

4.3.4. Observación. La función `StructureDescription` no sirve para identificar grupos salvo isomorfismo. Por ejemplo, existen dos grupos de orden 20 que pueden escribirse como producto semidirecto $C_5 \rtimes C_4$. Sin embargo, `StructureDescription` no da información suficiente como para determinar la estructura de cada uno de estos grupos:

```
gap> List(AllSmallGroups(Size, 20),\
> StructureDescription);
[ "C5 : C4", "C20", "C5 : C4", "D20", "C10 x C2" ]
```

4.3.5. `SmallGroups` contiene una función que identifica grupos de la base de datos: `IdGroup`. Algunos ejemplos:

```
gap> IdGroup(SymmetricGroup(3));
[ 6, 1 ]
gap> IdGroup(SymmetricGroup(4));
[ 24, 12 ]
gap> IdGroup(AlternatingGroup(4));
[ 12, 3 ]
gap> IdGroup(DihedralGroup(8));
[ 8, 3 ]
gap> IdGroup(QuaternionGroup(8));
[ 8, 4 ]
```

4.3.6. Ejemplo. Vamos a verificar que existen grupos no abelianos de orden impar y que el menor de estos grupos es de orden 21:

```
gap> First(AllSmallGroups(Size, [1, 3..21]),\
> x->not IsAbelian(x));;
gap> Size(last);
21
```

4.3.7. Ejemplo. En una línea puede verificarse que no existen grupos simples de orden 84. Para eso, usaremos la función `IsSimple` y le aplicaremos el filtro correcto a la función `AllSmallGroups`:

```
gap> AllSmallGroups(Size, 84, IsSimple, true);
[  ]
```

4.3.8. Ejemplo. Vamos demostrar un teorema de R. Guralnick **[6]** que afirma que el menor grupo finito G tal que $\{[x,y] : x,y \in G\} \neq [G,G]$ tiene orden 96.

```
gap> G := First(AllSmallGroups(Size, [1..100]),\
> x->Order(DerivedSubgroup(x))<>Size(\
> Set(List(Cartesian(x,x), Comm))));;
gap> Order(G);
96
```

```
gap> IdGroup(G);
[ 96, 3 ]
```

Con la función `IdGroup` o con `IsomorphismGroups` podemos verificar que

$$G \simeq \langle (135)(246)(7\,11\,9)(8\,12\,10), (394\,10)(58)(67)(11\,12)\rangle.$$

4.3.9. Observación. ¿Cómo fue que encontramos el isomorfismo

$$G \simeq \langle (135)(246)(7\,11\,9)(8\,12\,10), (394\,10)(58)(67)(11\,12)\rangle$$

que mencionamos en 4.3.8?

Una vez que tenemos el grupo G de orden 96, la función `IsomorphismPermGroup` nos permite construir una representación fiel de G en algún grupo de permutaciones. La imagen del morfismo obtenido con `IsomorphismPermGroup` se calcula con `Image` y nos da un grupo de permutaciones isomorfo a G. Este grupo bien podría tener grado grande, pero, en nuestro grupo de permutaciones, puede utilizarse la función `SmallerDegreePermutationRepresentation` para encontrar una representación por permutaciones de menor grado (no necesariamente minimal). Después de algunos intentos, este procedimiento da como resultado una presentación de G como subgrupo de $\mathbb{S}_{12}$. Un conjunto (no necesariamente minimal) de generadores de un grupo se construye con la función `SmallGeneratingSet`.

4.4. Grupos finitamente presentados.

4.4.1. La función `FreeGroup` construye el grupo libre en un número finito de generadores. Como ejemplo vamos a crear el grupo libre en dos generadores, vamos a realizar algunas operaciones básicas con sus elementos y vamos a generar aleatoriamente un elemento con la función `Random`:

```
gap> f := FreeGroup(2);
<free group on the generators [ f1, f2 ]>
gap> f.1^2;
f1^2
gap> f.1^2*f.1;
f1^3
gap> f.1*f.1^(-1);
<identity ...>
gap> Random(f);
f1^-3
```

4.4.2. La función `Length` devuelve la longitud de una palabra del grupo libre. Generaremos al azar diez mil palabras en el grupo libre en dos generadores y listaremos la longitud de cada una de estas palabras:

```
gap> f := FreeGroup(2);;
gap> Collected(List(List([1..10000], x->Random(f)), Length));
[ [ 0, 2270 ], [ 1, 1044 ], [ 2, 1113 ],
  [ 3, 986 ], [ 4, 874 ], [ 5, 737 ],
  [ 6, 642 ], [ 7, 500 ], [ 8, 432 ],
  [ 9, 329 ], [ 10, 248 ], [ 11, 189 ],
  [ 12, 152 ], [ 13, 119 ], [ 14, 93 ],
  [ 15, 68 ], [ 16, 57 ], [ 17, 34 ],
  [ 18, 30 ], [ 19, 23 ], [ 20, 19 ],
  [ 21, 16 ], [ 22, 8 ], [ 23, 3 ], [ 24, 4 ],
  [ 25, 4 ], [ 26, 2 ], [ 27, 2 ], [ 28, 1 ],
  [ 31, 1 ] ]
```

4.4.3. Algunas de las funciones que vimos anteriormente también pueden usarse sobre el grupo libre, mencionemos por ejemplo `Normalizer`, `RepresentativeAction`, `IsConjugate`, `Intersection`, `IsSubgroup`, `Subgroup`, etc. En el ejemplo siguiente vamos a construir el grupo libre en las letras `a` y `b`, y vamos a calcular su grupo de automorfismos:

```
gap> f := FreeGroup("a", "b");;
gap> a := f.1;;
gap> b := f.2;;
gap> Random(f);
b^-1*a^-5
gap> Centralizer(f, a);
Group([ a ])
gap> Index(f, Centralizer(f, a));
infinity
gap> Subgroup(f, [a,b]);
Group([ a, b ])
gap> Order(Subgroup(f, [a,b]));
infinity
gap> AutomorphismGroup(f);
<group of size infinity with 3 generators>
gap> GeneratorsOfGroup(AutomorphismGroup(f));
[ [ a, b ] -> [ a^-1, b ],
  [ a, b ] -> [ b, a ],
  [ a, b ] -> [ a*b, b ] ]
```

Vamos a verificar que el subgrupo S generado por a^2, b y aba^{-1} tiene índice dos en el grupo libre en a, b. Vamos a calcular el grupo de automorfismos de S y vamos a verificar que no es un grupo libre:

```
gap> S := Subgroup(f, [a^2, b, a*b*a^(-1)]);
Group([ a^2, b, a*b*a^-1 ])
gap> Index(f, S);
2
gap> A := AutomorphismGroup(S);
<group of size infinity with 3 generators>
gap> IsFreeGroup(A);
false
```

4.4.4. Ejemplo. Sean $n \geqslant 3$ y $p \geqslant 2$ números enteros. Un sorprendente resultado de Coxeter de 1957 [4] sobre cocientes del grupo de trenzas establece que el grupo generado por $\sigma_1, \ldots, \sigma_{n-1}$ y las relaciones

$$\begin{aligned} &\sigma_i\sigma_{i+1}\sigma_i = \sigma_{i+1}\sigma_i\sigma_{i+1} && \text{si } i \in \{1, \ldots, n-2\}, \\ &\sigma_i\sigma_j = \sigma_j\sigma_i && \text{si } |i-j| \geqslant 2, \\ &\sigma_i^p = 1 && \text{si } i \in \{1, \ldots, n-1\}\rangle, \end{aligned}$$

es un grupo finito si y sólo si $(p-2)(n-2) < 4$. Veamos qué pasa en el caso $n = 3$. Sea

$$G = \langle a, b : aba = bab,\ a^p = b^p = 1\rangle.$$

A continuación vamos a demostrar que

$$G \simeq \begin{cases} \mathbb{S}_3 & \text{si } p = 2, \\ \mathbf{SL}(2, \mathbb{F}_3) & \text{si } p = 3, \\ \mathbf{SL}(2, \mathbb{F}_3) \rtimes C_4 & \text{si } p = 4, \\ \mathbf{SL}(2, \mathbb{F}_3) \times C_5 & \text{si } p = 5. \end{cases}$$

El siguiente código no requiere mucha explicación:

```
gap> f := FreeGroup(2);;
gap> a := f.1;;
gap> b := f.2;;
gap> p := 2;;
gap> while p-2<4 do
> G := f/[a*b*a*Inverse(b*a*b), a^p, b^p];;
```

```
> Display(StructureDescription(G));
> p := p+1;
> od;
S3
SL(2,3)
SL(2,3) : C4
C5 x SL(2,5)
```

4.4.5. Ejemplo. Dados $l, m, n \in \mathbb{N}$, se define el **grupo de von Dyck** (o grupo triangular) de tipo (l, m, n) como el grupo

$$G(l, m, n) = \langle a, b : a^l = b^m = (ab)^n = 1 \rangle.$$

Se sabe que $G(l, m, n)$ es finito si y sólo si

$$\frac{1}{l} + \frac{1}{m} + \frac{1}{n} > 1.$$

Veamos que $G(2, 3, 3) \simeq \mathbb{A}_4$, $G(2, 3, 4) \simeq \mathbb{S}_4$ y que $G(2, 3, 5) \simeq \mathbb{A}_5$:

```
gap> f := FreeGroup(2);;
gap> a := f.1;;
gap> b := f.2;;
gap> StructureDescription(f/[a^2,b^3,(a*b)^3]);
"A4"
gap> StructureDescription(f/[a^2,b^3,(a*b)^4]);
"S4"
gap> StructureDescription(f/[a^2,b^3,(a*b)^5]);
"A5"
```

4.4.6. Ejemplo. Este ejemplo fue tomado de [5]. Vamos verificar que el grupo

$$\langle a, b, c : a^3 = b^3 = c^4 = 1,\ ac = ca^{-1},\ aba^{-1} = bcb^{-1} \rangle$$

es trivial. Para eso, utilizamos la función `IsTrivial`:

```
gap> f := FreeGroup(3);;
gap> a := f.1;;
gap> b := f.2;;
gap> c := f.3;;
gap> G := f/[a^3, b^3, c^4, c^(-1)*a*c*a, \
> a*b*a^(-1)*b*c^(-1)*b^(-1)];;
gap> IsTrivial(G);
true
```

4.4.7. Ejemplo. En [8] se demuestra que, para cada $n \in \mathbb{N}$,

$$\langle a, b : a^{-1}b^n a = b^{n+1},\ a = a^{i_1}b^{j_1}a^{i_2}b^{j_2}\cdots a^{i_k}b^{j_k} \rangle,$$

es trivial si $i_1 + i_2 + \cdots i_k = 0$. Veamos como ejemplo que

$$\langle a, b : a^{-1}b^2a = b^3,\ a = a^{-1}ba \rangle$$

es una presentación del grupo trivial:

```
gap> f := FreeGroup(2);;
gap> a := f.1;;
gap> b := f.2;;
gap> G := f/[a^(-1)*b^2*a*b^(-3),a*(a^(-1)*b*a)];;
gap> IsTrivial(G);
true
```

4.4.8. Ejemplo. Se define el grupo de Burnside $B(2,n)$ como el grupo

$$B(2,n) = \langle a, b : w^n = 1 \text{ para toda palabra } w \text{ en } a \text{ y } b \rangle.$$

Se sabe que $B(2,3)$ es finito. Para verificar esta afirmación, vamos a hacer lo siguiente: dividimos al grupo libre en a, b por el subgrupo normal generado por $\{w_1^3, \ldots, w_{10000}^3\}$, donde $w_1, \ldots, w_{10000}$ son palabras en a, b generadas al azar. El código que mostramos a continuación demuestra que $B(2,3)$ tiene orden $\leqslant 27$:

```
gap> f := FreeGroup(2);;
gap> rels := Set(List([1..10000], x->Random(f)^3));;
gap> B23 := f/rels;;
gap> Order(B23);
27
```

Similarmente vamos a verificar que $B(2,4)$ es finito:

```
gap> f := FreeGroup(2);;
gap> rels := Set(List([1..10000], x->Random(f)^4));;
gap> B24 := f/rels;;
gap> Order(B24);
4096
```

4.4.9. Ejemplo. La función `IsomorphismFpGroup` construye un isomorfismo entre nuestro grupo y un grupo finitamente presentado:

```
gap> D8 := DihedralGroup(8);;
gap> G := Image(IsomorphismFpGroup(D8));;
gap> RelatorsOfFpGroup(G);
[ F1^2, F2^-1*F1^-1*F2*F1*F3^-1, F3^-1*F1^-1*F3*F1, F2^2*F3^-1,
  F3^-1*F2^-1*F3*F2, F3^2 ]
```

A veces, la presentación obtenida puede simplificarse con la función `SimplifiedFpGroup`:

```
gap> G := SimplifiedFpGroup(Image(IsomorphismFpGroup(D8)));
<fp group on the generators [ F1, F2 ]>
gap> RelatorsOfFpGroup(G);
[ F1^2, F2^4, (F1*F2)^2 ]
```

4.5. Acciones de grupos. Veamos cómo trabajar con grupos que actúan en un conjunto finito. Todo grupo de permutaciones G actúa naturalmente (digamos, a derecha) en algún subconjunto X de $\{1, \ldots, n\}$. Recordemos que la acción se dice transitiva si dados $x, y \in X$ existe $g \in G$ tal que $x \cdot g = y$. La función `IsTransitive` devuelve `true` si la acción es transitiva. La cantidad de puntos que mueve una permutación o un grupo de permutaciones se calcula con la función `NrMovedPoints`:

```
gap> G := Group([(1,3)]);;
gap> IsTransitive(G, [1, 2, 3]);
false
gap> IsTransitive(G, [1, 3]);
true
gap> NrMovedPoints(G);
2
```

Dado $x \in X$, el conjunto $x^G = \{x \cdot g : g \in G\}$ se denomina la órbita de x y se calcula con la función `Orbit`:

```
gap> Orbit(G, 1);
[ 1, 3 ]
gap> Orbit(G, 2);
[ 2 ]
gap> Orbits(G);
```

```
[ [ 1, 3 ] ]
gap> Orbits(G,[1, 2, 3]);
[ [ 1, 3 ], [ 2 ] ]
```

El estabilizador de $x \in X$ es el subgrupo $G_x = \{g \in X : x \cdot g = x\}$ y se calcula con la función `Stabilizer`:

```
gap> Stabilizer(G, 1);
Group(())
gap> Stabilizer(G, 2);
Group([ (1,3) ])
```

Una acción se dice semiregular si $G_x = 1$ para todo $x \in X$, y regular cuando es transitiva y semiregular. Para determinar si una acción es semiregular (resp. regular) se usa la función `IsSemiRegular` (resp. `IsRegular`):

```
gap> IsRegular(G, [1,2,3]);
false
gap> IsSemiRegular(G, [1,2,3]);
false
gap> IsSemiRegular(G, [1,3]);
true
gap> IsRegular(G, [1,3]);
true
```

Una acción es k-transitiva si toda k-tupla de elementos de X está conectada por algún $g \in G$ con toda otra k-upla de elementos de X. La función `Transitivity` devuelve el máximo k tal que la acción es k-transitiva:

```
gap> Transitivity(SymmetricGroup(5));
5
gap> Transitivity(AlternatingGroup(5));
3
```

4.5.1. GAP contiene una base de datos con los grupos transitivos de grado pequeño. La función `TransitiveGroups` devuelve listas de grupos transitivos. Un determinado grupo transitivo de la base de datos se obtiene con `TransitiveGroup`. La identificación de grupos transitivos se hace mediante la función `TransitiveIdentification`.

5. Aplicaciones a quandles

5.1. Definiciones y ejemplos.

5.1.1. En esta sección desarrollaremos funciones nos permitan trabajar con quandles. Los quandles son estructuras algebraicas muy utilizadas en teoría de nudos.

5.1.2. Un **quandle** es un par $(X, \triangleright)$, donde X es un conjunto (que nosotros siempre consideraremos finito) y una operación $\triangleright\colon X \times X \to X$, que denotaremos $(x, y) \mapsto x \triangleright y$ tal que:

1) para cada $x \in X$ la función $\varphi_x\colon X \to X$ dada por $\varphi_x(y) = x \triangleright y$ es biyectiva,
2) $x \triangleright (y \triangleright z) = (x \triangleright y) \triangleright (x \triangleright z)$ para todo $x, y, z \in X$,
3) $x \triangleright x = x$ para todo $x \in X$.

5.1.3. Como X es un conjunto finito, sin pérdida de generalidad, podemos suponer que $X = \{1, 2, \dots, n\}$. Podemos entonces pensar que un quandle es una sucesión de permutaciones $\varphi_1, \dots, \varphi_n$ de $\mathbb{S}_n$ tales que

1) $\varphi_x(x) = x$ para todo $x \in X$.
2) $\varphi_x \varphi_y \varphi_x^{-1} = \varphi_{\varphi_x(y)}$ para todo $x, y \in X$.

Equivalentemente, una estructura de quandle sobre $X = \{1, \dots, n\}$ puede representarse por una matriz $Q \in \mathbb{Z}^{n \times n}$ donde $Q_{ij} = i \triangleright j$ para todo $i, j \in X$. La matriz Q se denominará la **matriz del quandle** X.

5.1.4. Si X e Y son quandles, una biyección $f\colon X \to Y$ es un **isomorfismo de quandles** si $f(x \triangleright y) = f(x) \triangleright f(y)$ para todo $x, y \in X$. La siguiente función verifica (de forma no muy eficiente) si dos quandles son isomorfos por una permutación dada:

```
gap> IsQuandleIsomorphism := \
> function(p, quandle1, quandle2)
> local n;
> if Size(quandle1) <> Size(quandle2) then
>   return false;
> else
>   n := Size(quandle1);
> fi;
> return ForAll(Cartesian([1..n], [1..n]),\
> x->quandle1[x[1]][x[2]]^p=quandle2[x[1]^p][x[2]^p]);
> end;
function( p, quandle1, quandle2 ) ... end
```

5.1.5. Ejemplo. Sea X un conjunto finito y no vacío. Entonces X es un quandle con $x \triangleright y = y$ para todo $x, y \in X$. Este quandle se denomina **quandle trivial** sobre X. Vamos a construir la matriz del quandle trivial sobre el conjunto $\{1, \ldots, n\}$:

```
gap> TrivialQuandle := function(n)
> return List([1..n], x->[1..n]);
> end;
function( n ) ... end
gap> Display(TrivialQuandle(2));
[ [  1,  2 ],
  [  1,  2 ] ]
gap> Display(TrivialQuandle(3));
[ [  1,  2,  3 ],
  [  1,  2,  3 ],
  [  1,  2,  3 ] ]
gap> Display(TrivialQuandle(4));
[ [  1,  2,  3,  4 ],
  [  1,  2,  3,  4 ],
  [  1,  2,  3,  4 ],
  [  1,  2,  3,  4 ] ]
```

5.1.6. Ejemplo. Sea G un grupo y g^G una clase de conjugación finita de G. Entonces g^G es un quandle con $x \triangleright y = xyx^{-1}$ para todo $x, y \in g^G$. El quandle asociado a la clase de conjugación de g en G se llama **quandle de conjugación**. Primero escribimos la función que nos devuelve la matriz de un quandle de conjugación:

```
gap> ConjugationQuandle := function(group, g)
> local m, c, x, y, i, j;
> if g in group then
>   c := Elements(ConjugacyClass(group, g));
>   m := NullMat(Size(c), Size(c));
>   for x in c do
>     for y in c do
>       i := Position(c, x);
>       j := Position(c, y);
>       m[i][j] := Position(c, x*y*Inverse(x));
>     od;
>   od;
>   return m;
> else
>   return fail;
> fi;
```

```
> end;
function( group, g ) ... end
```

Hagamos algunos ejemplos:

```
gap> Display(ConjugationQuandle(\
> AlternatingGroup(4),(1,2,3)));
[ [  1,  3,  4,  2 ],
  [  4,  2,  1,  3 ],
  [  2,  4,  3,  1 ],
  [  3,  1,  2,  4 ] ]
gap> Display(ConjugationQuandle(\
> SymmetricGroup(3),(1,2)));
[ [  1,  3,  2 ],
  [  3,  2,  1 ],
  [  2,  1,  3 ] ]
```

5.1.7. Ejemplo. Veamos que los quandles $(123)^{\mathbb{A}_4}$ y $(132)^{\mathbb{A}_4}$ son isomorfos:

```
gap> A4 := AlternatingGroup(4);;
gap> S4 := SymmetricGroup(4);;
gap> First(S4, p->IsQuandleIsomorphism(p, \
> ConjugationQuandle(A4, (1,2,3)),\
> ConjugationQuandle(A4, (1,3,2))));
(1,4)
```

5.1.8. Con lo hecho en el ejemplo anterior es fácil escribir una función que determine si dos quandles son isomorfos:

```
gap> IsomorphismQuandles :=\
> function(quandle1, quandle2)
> local n;
> if Size(quandle1) <> Size(quandle2) then
> return fail;
> else
> n := Size(quandle1);
> fi;
> return First(SymmetricGroup(n), \
> p->IsQuandleIsomorphism(p, quandle1, quandle2));
> end;
function( quandle1, quandle2 ) ... end
```

5.1.9. Ejemplo. Sea $n \in \mathbb{N}$. El anillo $\mathbb{Z}/n$ es un quandle con la acción dada por $x \triangleright y = 2x - y$ para todo $x, y \in \mathbb{Z}/n$. Este quandle se denomina **quandle diedral** y se denota por $\mathbb{D}_n$. Para construir quandles diedrales utilizamos la función `ZmodnZ`:

```
gap> DihedralQuandle := function(n)
> local m, x, y, i, j, l;
> l := Elements(ZmodnZ(n));
> m := NullMat(Size(l), Size(l));
> for x in l do
>   for y in l do
>     i := Position(l, x);
>     j := Position(l, y);
>     m[i][j] := Position(l, 2*x-y);
>   od;
> od;
> return m;
> end;
function( n ) ... end
```

Veamos algunos ejemplos:

```
gap> Display(DihedralQuandle(3));
[ [  1,  3,  2 ],
  [  3,  2,  1 ],
  [  2,  1,  3 ] ]
gap> Display(DihedralQuandle(4));
[ [  1,  4,  3,  2 ],
  [  3,  2,  1,  4 ],
  [  1,  4,  3,  2 ],
  [  3,  2,  1,  4 ] ]
gap> Display(DihedralQuandle(5));
[ [  1,  4,  5,  2,  3 ],
  [  3,  2,  1,  5,  4 ],
  [  4,  5,  3,  1,  2 ],
  [  5,  3,  2,  4,  1 ],
  [  2,  1,  4,  3,  5 ] ]
```

5.1.10. Ejemplo. Sea $\mathbb{F}_q$ el cuerpo de q elementos, donde q es una potencia de un número primo. Para cada $\alpha \in \mathbb{F}_q \setminus \{0\}$ definimos el **quandle de Alexander** de tipo (q, α) como el quandle sobre el cuerpo $\mathbb{F}_q$ dado por $x \triangleright y = (1-\alpha)x + \alpha y$ para todo $x, y \in \mathbb{F}_q$.

```
gap> AlexanderQuandle := function(field, alpha)
> local m, x, y, i, j, l;
> l := Elements(field);
> m := NullMat(Size(l), Size(l));
> for x in l do
>   for y in l do
>     i := Position(l, x);
>     j := Position(l, y);
>     m[i][j] := Position(l, (1-alpha)*x+alpha*y);
>   od;
> od;
> return m;
> end;
```

Veamos algunos ejemplos:

```
gap> Display(AlexanderQuandle(GF(3),Z(3)));
[ [  1,  3,  2 ],
  [  3,  2,  1 ],
  [  2,  1,  3 ] ]
gap> Display(AlexanderQuandle(GF(4),Z(4)));
[ [  1,  3,  4,  2 ],
  [  4,  2,  1,  3 ],
  [  2,  4,  3,  1 ],
  [  3,  1,  2,  4 ] ]
gap> Display(AlexanderQuandle(GF(4),Z(4)^2));
[ [  1,  4,  2,  3 ],
  [  3,  2,  4,  1 ],
  [  4,  1,  3,  2 ],
  [  2,  3,  1,  4 ] ]
gap> Display(AlexanderQuandle(GF(4),Z(4)^3));
[ [  1,  2,  3,  4 ],
  [  1,  2,  3,  4 ],
  [  1,  2,  3,  4 ],
  [  1,  2,  3,  4 ] ]
gap> Display(AlexanderQuandle(GF(5),Z(5)));
[ [  1,  3,  4,  5,  2 ],
  [  4,  2,  5,  3,  1 ],
```

```
  [  5,  1,  3,  2,  4 ],
  [  2,  5,  1,  4,  3 ],
  [  3,  4,  2,  1,  5 ] ]
```

5.1.11. Ejemplo. Si G es un grupo finito entonces G es un quandle con $x \triangleright y = xy^{-1}x$ para todo $x, y \in G$. La siguiente función sirve para construir este quandle:

```
gap> CoreQuandle := function(group)
> local m, l, x, y, i, j;
> l := Elements(group);
> m := NullMat(Size(l), Size(l));
> for x in group do
>   for y in group do
>     i := Position(l, x);
>     j := Position(l, y);
>     m[i][j] := Position(l, x*Inverse(y)*x);
>   od;
> od;
> return m;
> end;
function( group ) ... end
```

Veamos algunos ejemplos:

```
gap> Display(CoreQuandle(SymmetricGroup(3)));
[ [  1,  2,  3,  5,  4,  6 ],
  [  1,  2,  6,  4,  5,  3 ],
  [  1,  6,  3,  4,  5,  2 ],
  [  5,  2,  3,  4,  1,  6 ],
  [  4,  2,  3,  1,  5,  6 ],
  [  1,  3,  2,  4,  5,  6 ] ]
gap> Display(CoreQuandle(CyclicGroup(3)));
[ [  1,  3,  2 ],
  [  3,  2,  1 ],
  [  2,  1,  3 ] ]
gap> Display(CoreQuandle(CyclicGroup(4)));
[ [  1,  4,  3,  2 ],
  [  3,  2,  1,  4 ],
  [  1,  4,  3,  2 ],
  [  3,  2,  1,  4 ] ]
gap> Display(CoreQuandle(CyclicGroup(5)));
[ [  1,  5,  4,  3,  2 ],
  [  3,  2,  1,  5,  4 ],
  [  5,  4,  3,  2,  1 ],
  [  2,  1,  5,  4,  3 ],
  [  4,  3,  2,  1,  5 ] ]
```

5.1.12. Vimos que un quandle finito de tamaño n puede presentarse como la sucesión de permutaciones $\varphi_1, \ldots, \varphi_n$ o como una matriz de $n \times n$ cuyas filas son las imágenes de las permutaciones. La siguiente función devuelve la matriz de un quandle dado por una sucesión de permutaciones:

```
gap> Quandle := function(perms)
> return List(perms, x->ListPerm(x));
> end;
function( perms ) ... end
```

La siguiente, en cambio, devuelve la sucesión de permutaciones que define un quandle:

```
gap> Permutations := function(quandle)
> return List(quandle, PermList);
> end;
function( quandle ) ... end
```

5.1.13. Observación. No todo quandle es un quandle de conjugación, ver ejercicio 6.4.13.

5.2. Grupos asociados a quandles.

5.2.1. Si X es un quandle se define el **grupo interior** de X como el grupo generado por las permutaciones φ_x, es decir: $\mathrm{Inn}(X) = \langle \varphi_x : x \in X \rangle$. El grupo $\mathrm{Inn}(X)$ actúa naturalmente (por evaluación) en X. El quandle X se dice **conexo** si $\mathrm{Inn}(X)$ actúa transitivamente en X. Para calcular el grupo interior de un quandle finito se tiene la siguiente función:

```
gap> InnerGroup := function(quandle)
> return Group(Permutations(quandle));
> end;
function( quandle ) ... end
```

Para determinar si un quandle es conexo tenemos la siguiente función:

```
gap> IsConnected := function(quandle)
> return IsTransitive(InnerGroup(quandle),\
> [1..Size(quandle)]);
> end;
function( quandle ) ... end
```

5.2.2. Ejemplos. Veamos cómo utilizar las funciones escritas en 5.2.1.

quandle	grupo interior	comentarios
$\mathbb{D}_3$	$\mathbb{S}_3$	conexo
$\mathbb{D}_4$	$C_2 \times C_2$	disconexo
$\mathbb{D}_5$	$\mathbb{D}_{10}$	conexo
$(123)^{A_4}$	$\mathbb{A}_4$	conexo

Cuadro 1. Algunos grupos interiores de quandles.

El código para obtener el cuadro 1 es el siguiente:

```
gap> D3 := DihedralQuandle(3);;
gap> IsConnected(D3);
true
gap> StructureDescription(InnerGroup(D3));
"S3"
gap> D4 := DihedralQuandle(4);;
gap> IsConnected(D4);
false
gap> StructureDescription(InnerGroup(D4));
"C2 x C2"
gap> D5 := DihedralQuandle(5);;
gap> IsConnected(D5);
true
gap> StructureDescription(InnerGroup(D5));
"D10"
gap> T := ConjugationQuandle(\
> AlternatingGroup(4), (1,2,3));;
gap> IsConnected(T);
true
gap> StructureDescription(InnerGroup(T));
"A4"
```

5.2.3. Si X es un quandle se define el **grupo envolvente** G_X de X como el grupo con generadores $x \in X$ y relaciones $xy = (x \triangleright y)x$ para todo $x, y \in X$. En la siguiente función utilizamos `FreeGroup` y `GeneratorsOfGroup` para construir el grupo envolvente de un quandle finito:

```
gap> EnvelopingGroup := function(quandle)
> local rels, gens, f, x, y;
> f := FreeGroup(Size(quandle));
> rels := [];
> gens := GeneratorsOfGroup(f);
> for x in [1..Size(quandle)] do
>   for y in [1..Size(quandle)] do
>     if x <> y then
>       Add(rels,\
>       gens[x]*gens[y]*\
>       Inverse(gens[quandle[x][y]]*gens[x]));
>     fi;
>   od;
> od;
> return f/rels;
> end;
function( quandle ) ... end
```

Veamos algunos ejemplos:

```
gap> EnvelopingGroup(DihedralQuandle(3));
<fp group on the generators [ f1, f2, f3 ]>
gap> RelatorsOfFpGroup(last);
[ f1*f2*f1^-1*f3^-1, f1*f3*f1^-1*f2^-1,
  f2*f1*f2^-1*f3^-1, f2*f3*f2^-1*f1^-1,
  f3*f1*f3^-1*f2^-1, f3*f2*f3^-1*f1^-1 ]
gap> EnvelopingGroup(DihedralQuandle(4));
<fp group on the generators
[ f1, f2, f3, f4 ]>
gap> RelatorsOfFpGroup(last);
[ f1*f2*f1^-1*f4^-1, f1*f3*f1^-1*f3^-1,
  f1*f4*f1^-1*f2^-1, f2*f1*f2^-1*f3^-1,
  f2*f3*f2^-1*f1^-1, f2*f4*f2^-1*f4^-1,
  f3*f1*f3^-1*f1^-1, f3*f2*f3^-1*f4^-1,
  f3*f4*f3^-1*f2^-1, f4*f1*f4^-1*f3^-1,
  f4*f2*f4^-1*f2^-1, f4*f3*f4^-1*f1^-1 ]
```

5.2.4. En [**1**, Lemma 1.9] se demuestra que si X es un quandle de conjugación, entonces $\mathrm{Inn}X \simeq G_X/Z(G_X)$. Veamos un ejemplo:

```
gap> D3 := DihedralQuandle(3);;
gap> G := EnvelopingGroup(D3);;
gap> IsomorphismGroups(G/Center(G),InnerGroup(D3));
[ (1,2)(3,6)(4,5), (1,3)(2,5)(4,6),
  (1,4)(2,6)(3,5) ] -> [ (2,3), (1,2), (1,3) ]
```

6. Ejercicios

6.1. Primeros pasos.

6.1.1. Use la función `Product` para calcular $2 \cdot 4 \cdot 6 \cdots 20$. ¿De qué otra forma puede realizar este cálculo?

6.1.2. Utilice la función `MinimalPolynomial` y calcule el polinomio minimal sobre $\mathbb{Q}$ de $3 + \sqrt{5}$.

6.1.3. Calcule los primeros 100 términos de la sucesión de Fibonacci.

6.1.4. Dado $k \in \mathbb{N}$ se define la sucesión a_n por $a_1 = \cdots = a_{k+1} = 1$ y $a_n = a_{n-k} + a_{n-k-1}$ para todo $n > k+1$. Escriba una función que, dado $k \in \mathbb{N}$, devuelva el m-ésimo término de la sucesión a_n. Para más información sobre esta sucesión ver `http://oeis.org/A103379`.

6.1.5. Escriba una función que devuelva el n-ésimo término de la sucesión a_n definida por $a_0 = a_1 = a_2 = a_3 = 1$ y

$$a_n = \frac{a_{n-1}a_{n-3} + a_{n-2}^2}{a_{n-4}}$$

para $n \geqslant 4$. Esta sucesión es una de las sucesiones de Somos. Para más información ver `http://oeis.org/A006720`.

6.1.6. Escriba una función que, dada una lista de palabras y una letra `letter`, devuelva una sublista donde cada palabra empiece con la letra `letter`.

6.1.7. Escriba una función que, dado n, devuelva la cantidad de números primos $\leqslant n$.

6.1.8. Escriba una función que muestre todos los anagramas de una palabra dada. Sugerencia: utilice la función `Permuted`.

6.1.9. Escriba una función que, dada una lista de enteros no negativos, imprima en pantalla el histograma asociado a la lista. Por ejemplo, si el argumento es la lista `[1,4,2]` entonces la función deberá imprimir

```
X
XXXX
XX
```

6.1.10. Escriba una función que, dada una lista de palabras, devuelva la de mayor longitud.

6.1.11. Escriba una función que, dado un número de segundos, devuelva su equivalente en días, horas, minutos y segundos.

6.1.12. Escriba una función que calcule el promedio de una lista de números.

6.1.13. Escriba una función que, dada una letra, devuelva `true` si esa letra es una consonante o `false` en caso contrario.

6.2. Permutaciones y matrices.

6.2.1. Escriba a las permutaciones

$$\begin{pmatrix} 123456 \\ 253461 \end{pmatrix}, \qquad \begin{pmatrix} 123456789 \\ 234517896 \end{pmatrix}, \qquad \begin{pmatrix} 12345 \\ 32451 \end{pmatrix},$$

como producto de ciclos disjuntos.

6.2.2. Escriba a las permutaciones

$$(123)(45)(1625)(341), \quad (12)(245)(12)$$

como producto de ciclos disjuntos.

6.2.3. Encuentre una permutación τ tal que

1) $\tau(12)(34)\tau^{-1} = (56)(13)$.
2) $\tau(123)(78)\tau^{-1} = (257)(13)$.
3) $\tau(12)(34)(567)\tau^{-1} = (18)(23)(456)$.

6.2.4. Calcule $\tau\sigma\tau^{-1}$ en los siguientes casos:

1) $\sigma = (123)$ y $\tau = (34)$.
2) $\sigma = (567)$ y $\tau = (12)(34)$.

6.2.5. Sea $\sigma \in \mathbb{S}_9$ definida por $\sigma(i) = 10 - i$ para todo $i \in \{1, \ldots, 9\}$. Escriba a σ como producto de ciclos disjuntos.

6.2.6. Encuentre (si es posible) tres permutaciones $\alpha, \beta, \gamma \in \mathbb{S}_5$ tales que $\alpha\beta = \beta\alpha$, $\beta\gamma = \gamma\beta$ y $\alpha\gamma \neq \gamma\alpha$.

6.2.7. Utilice la función `PermutationMat` y escriba todas los elementos del grupo simétrico $\mathbb{S}_3$ como matrices de 3×3.

6.2.8. Para $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 0 & 2 & 3 \end{pmatrix}$ calcule

$$I + A + \frac{1}{2!}A^2 + \frac{1}{3!}A^3 + \frac{1}{4!}A^4.$$

6.2.9. Escriba una función que, dados $n \in \mathbb{N}$ y una matriz cuadrada A, devuelva la matriz

$$I + A + \frac{1}{2!}A^2 + \frac{1}{3!}A^3 + \cdots + \frac{1}{n!}A^n.$$

6.2.10. Para $n \in \mathbb{N}$ se define la **matriz de Hilbert** H_n por

$$(H_n)_{ij} = \frac{1}{i + j - 1}$$

para todo i, j. Escriba una función que, dado $n \in \mathbb{N}$, devuelva la matriz de Hilbert H_n.

6.2.11. Las **matrices de Walsh** $H(2^k)$, $k \geqslant 1$, se definen recursivamente de la siguiente forma:

$$H(2) = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad H(2^k) = H(2) \otimes H(2^{k-1}), \quad k \geqslant 1,$$

donde $\otimes$ denota el producto de Kronecker de matrices. Utilice la función `KroneckerProduct` y escriba una función que, dado $n \in \mathbb{N}$, devuelva la matriz de Walsh $H(2^n)$.

6.2.12. Utilice las funciones `Eigenvalues` y `Eigenvalues` y calcule los autovalores y los autovectores de la matriz

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 6 & 7 & 8 \end{pmatrix} \in \mathbb{Q}^{3\times 3}.$$

Cuidado: `Eigenvectors` devuelve generadores de los autoespacios, donde $v \neq 0$ es autovector de A de autovalor λ si y sólo si $vA = \lambda v$.

6.2.13. Utilice la función `NullspaceMat` y determine el espacio nulo de la matriz A del ejercicio 6.2.12. Observe que el espacio nulo de una matriz A se define como el conjunto de vectores v tales que $vA = 0$.

6.3. Grupos.

6.3.1. Determine el orden del grupo generado por las matrices

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}, \qquad \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

6.3.2. El grupo de Klein es el único grupo no cíclico de orden cuatro. Construya el grupo de Klein como grupo de permutaciones.

6.3.3. Verifique que todo subgrupo de $C_4 \times Q_8$ es normal.

6.3.4. Sea G el conjunto de matrices de la forma

$$\begin{pmatrix} 1 & c \\ 0 & d \end{pmatrix}, \quad c, d \in \mathbb{F}_4,\ c \neq 0.$$

Verificar que G es un grupo y calcular su orden.

6.3.5. Encuentre todos los subgrupos de $\mathbb{S}_4$ que actúan transitivamente en $\{1,2,3,4\}$ y analice la transitividad múltiple.

6.3.6. Utilice la función `IsomorphicSubgroups` y verifique que $\mathbb{A}_6$ no contiene un subgrupo isomorfo a $\mathbb{S}_5$ y que $\mathbb{A}_7$ sí contiene un subgrupo isomorfo a $\mathbb{S}_5$.

6.3.7. Verifique que $\mathbb{A}_6$ no tiene subgrupos de índice primo.

6.3.8. Utilice las funciones `SylowSubgroup` y `ConjugacyClassSubgroups` y construya todos los subgrupos de Sylow de $\mathbb{A}_4$ y de $\mathbb{S}_4$.

6.3.9. Verifique que $\mathbb{S}_5$ tiene un 2-subgrupo de Sylow isomorfo a $\mathbb{D}_8$ y que $\mathbb{S}_6$ tiene un 2-subgrupo de Sylow isomorfo a $\mathbb{D}_8 \times C_2$.

6.3.10. Utilice la función `Normalizer` y determine la cantidad de conjugados que tiene un 2-subgrupo de Sylow de $\mathbb{A}_5$.

6.3.11. Encuentre los subgrupos de Sylow para los grupos C_{27}, $\mathbf{SL}(2,\mathbb{F}_5)$, $\mathbb{S}_7$, $\mathbb{S}_3 \times \mathbb{A}_4$ y $\mathbb{S}_3 \times C_{20}$.

6.3.12. Construya las clases de conjugación de subgrupos de $\mathbb{S}_3 \times \mathbb{S}_3$ y encuentre tres p-subgrupos de Sylow, digamos A, B, C, tales que $A \cap B = 1$ y $A \cap C \neq 1$.

6.3.13. Construya el grupo

$$\left\{ \begin{pmatrix} 1 & b \\ 0 & d \end{pmatrix} : b, d \in \mathbb{F}_{19},\ d \neq 0 \right\}$$

y demuestre que no es simple.

6.3.14. Verifique que el grupo

$$G = \langle (12)(6\,11)(8\,12)(9\,13), (5\,13\,9)(6\,10\,11)(7\,8\,12), (2,4,3)(5,8,9)(6\,10\,13)(7\,11\,12) \rangle$$

cumple que $[G,G] \neq \{[x,y] : x, y \in G\}$.

6.3.15. Verifique que $\mathbb{A}_4 \times C_7$ no tiene subgrupos de índice dos.

6.3.16. Verifique que $\mathbb{A}_5$ no tiene subgrupos de orden $8, 15, 20, 24, 30, 40$.

6.3.17. Es sabido que si A es un subgrupo abeliano de $\mathbb{S}_n$ entonces A tiene orden $\leqslant 3^{\lfloor n/3 \rfloor}$ y que esta cota es óptima. Para $n \in \{5,6,7,8\}$ encuentre un subgrupo abeliano A de $\mathbb{S}_n$ de orden $3^{\lfloor n/3 \rfloor}$.

6.3.18. Verifique que $\mathbf{SL}(2,\mathbb{F}_5)$ no contiene subgrupos isomorfos a $\mathbb{A}_5$.

6.3.19. Verifique que para cada d que divida a 24 existe un subgrupo de $\mathbb{S}_4$ de orden d.

6.3.20. Verifique que el grupo $\mathbf{SL}(2,\mathbb{F}_3)$ tiene un único elemento de orden dos y que no tiene subgrupos de orden 12.

6.3.21. Verifique que el conmutador de $\mathbf{SL}(2,\mathbb{F}_3)$ es isomorfo a Q_8.

6.3.22. Determine la estructura del grupo $\mathbf{SL}(2,\mathbb{F}_3)/Z(\mathbf{SL}(2,\mathbb{F}_3))$.

6.3.23. Determine si $\mathbb{S}_5$ y $\mathbf{SL}(2,\mathbb{F}_5)$ son isomorfos.

6.3.24. Consideremos al grupo diedral $\mathbb{D}_8 = \langle r^4 = s^2 = 1,\ srs = r^{-1} \rangle$. Encuentre todos los subgrupos de $\mathbb{D}_8$ que contengan al subgrupo $\langle 1, r^2 \rangle$.

6.3.25. Encuentre todos los morfismos $\mathbb{S}_3 \to \mathbf{SL}(2,\mathbb{F}_3)$.

6.3.26. Determine si existen epimorfismos $\mathbb{D}_{16} \to \mathbb{D}_8$ y $\mathbb{D}_{16} \to C_2$.

6.3.27. Verifique que $\mathrm{Aut}(\mathbb{A}_4) \simeq \mathbb{S}_4$.

6.3.28. Verifique que $\mathrm{Aut}(\mathbb{D}_8) \simeq \mathbb{D}_8$ y que $\mathrm{Aut}(\mathbb{D}_{16}) \not\simeq \mathbb{D}_{16}$.

6.3.29. Determine el orden de $\mathrm{Aut}(C_{11} \times C_2 \times C_3)$.

6.3.30. Verifique que $\mathbb{D}_{12} \simeq \mathbb{S}_3 \times C_2$.

6.3.31. Verifique que todo grupo de orden < 60 es soluble.

6.3.32. Verifique que todo grupo de orden 12 no isomorfo a $\mathbb{A}_4$ contiene un elemento de orden seis.

6.3.33. Verifique que todo grupo de orden 455 es cíclico.

6.3.34. Determine la cantidad de grupos de Sylow que tienen los grupos no abelianos de orden 34.

6.3.35. Determine la cantidad de elementos de orden siete que tiene un grupo simple de orden 168.

6.3.36. Verifique que no existen grupos simples de orden 2540 y 9075.

6.3.37. Encuentre un grupo G de orden 3^6 tal que $\{[x,y] : x,y \in G\}$ y $[G,G]$ sean distintos.

6.3.38. Encuentre un grupo G de orden 2^7 tal que $\{[x,y] : x,y \in G\}$ y $[G,G]$ sean distintos.

6.3.39. Verifique que los grupos de orden 15, 35 y 77 son cíclicos.

6.3.40. Verifique que si G es un grupo simple y $|G| = 60$ entonces $G \simeq \mathbb{A}_5$.

6.3.41. Verifique que el único grupo simple no abeliano de orden < 100 es $\mathbb{A}_5$.

6.3.42. Sea G un grupo finito. Encuentre un contraejemplo para la siguiente afirmación: el conjunto $\{x^2 : x \in G\}$ de cuadrados siempre es un subgrupo de G.

6.3.43. Verifique el siguiente teorema de R. Guralnick **[6]**: Existe un grupo G de orden $n \leqslant 200$ tal que $[G,G] \neq \{[x,y] : x,y \in G\}$ si y sólo si $n \in \{96, 128, 144, 162, 168, 192\}$.

6.3.44. Extienda el resultado de R. Guralnick visto en 6.3.43 y verifique que existe un grupo G de orden $n < 1024$ tal que $[G,G] \neq \{[x,y] : x,y \in G\}$ si y sólo si n es alguno de los siguientes números: 96, 128, 144, 162, 168, 192, 216, 240, 256, 270, 288, 312, 320, 324, 336, 360, 378, 384, 400, 432, 448, 450, 456, 480, 486, 504, 512, 528, 540, 560, 576, 594, 600, 624, 640, 648, 672, 702, 704, 720, 729, 744, 750, 756, 768, 784, 792, 800, 810, 816, 832, 840, 864, 880, 882, 888, 896, 900, 912, 918, 936, 960, 972, 1000, 1008.

6.3.45. Verifique que el grupo $\langle (123\cdots 7), (26)(34)\rangle$ es simple, tiene orden 168 y actúa transitivamente en $\{1,\ldots,7\}$. onjuga

6.3.46. Determine el orden del grupo $\langle a,b : a^2 = b^2 = (bab^{-1})^3 = 1\rangle$.

6.3.47. Sea $G = \langle a,b : a^2 = abab^{-1} = 1\rangle$. Verifique que G tiene infinitos elementos.

6.3.48. Encuentre el orden del grupo

$$\langle a,b : a^8 = b^2a^4 = ab^{-1}ab = 1\rangle.$$

6.3.49. Verifique que el grupo

$$\langle a,b : a^5 = 1, b^2 = (ab)^3, (a^3ba^4b)^2 = 1\rangle$$

es isomorfo a $\mathbb{A}_5$.

6.3.50. Verifique que el grupo

$$\langle a,b : a^2 = b^3 = a^{-1}b^{-1}ab = 1\rangle$$

es cíclico y finito.

6.3.51. Verifique que el grupo

$$\langle a,b : a^2 = b^3 = 1\rangle$$

es no abeliano.

6.3.52. Determine el orden del grupo

$$\langle a, b, c : a^3 = b^3 = c^3 = 1,\ aba = bab,\ cbc = bcb,\ ac = ca\rangle.$$

6.3.53. Verifique que

$$B(3,3) = \langle a, b, c : w^3 = 1 \text{ para toda palabra } w \text{ en } a, b\rangle$$

es un grupo finito de orden $\leqslant 2187$.

6.4. Quandles.

6.4.1. Escriba una función que determine si una matriz de $n \times n$ y con valores en $\{1, \dots, n\}$ es la matriz de un quandle.

6.4.2. Escriba una función que determine si una sucesión de n permutaciones define un quandle de tamaño n.

6.4.3. Utilice la función `CyclicGroup` y escriba una función que devuelva la matriz de un quandle diedral finito.

6.4.4. Escriba una función que devuelva la matriz del quandle de conjugación asociado a una unión finita de clases de conjugación finitas de un grupo dado.

6.4.5. Determine para qué valores de $\alpha \in \mathbb{F}_5 \setminus \{0\}$ el quandle de Alexander de tipo $(5, \alpha)$ es conexo.

6.4.6. Un quandle X se dice **fiel** si la función $X \to \mathrm{Inn}(X)$, $x \mapsto \varphi_x$, es inyectiva. Escriba una función que determine si un quandle es fiel.

6.4.7. Si X es un quandle se define el **grupo de automorfismos** de X como el grupo

$$\mathrm{Aut}(X) = \{f \in \mathbb{S}_X : f(x \triangleright y) = f(x) \triangleright f(y) \text{ para todo } x, y \in X\},$$

donde $\mathbb{S}_X$ denota el conjunto de biyecciones $X \to X$. Escriba una función que calcule el grupo de automorfismos de un quandle.

6.4.8. ¿Es cierto que para todo quandle X se tiene que $\mathrm{Inn}(X) = \mathrm{Aut}(X)$?

6.4.9. Determine si el quandle $(123)^{\mathbb{A}_4}$ es isomorfo a algún quandle afín sobre $\mathbb{F}_4$.

6.4.10. Verifique que el quandle diedral $\mathbb{D}_7$ es isomorfo al quandle de conjugación de las involuciones del grupo diedral $\mathbb{D}_{14}$ de 14 elementos.

6.4.11. Si A es un grupo abeliano finito y $g \in \mathrm{Aut}(A)$ se define el **quandle afín** de tipo (A, g) como el quandle sobre A dado por

$$x \triangleright y = (1 - g)(x) + g(y)$$

para todo $x, y \in A$. Escriba una función que, dados un grupo abeliano A y un automorfismo de A, devuelva la matriz del quandle afín de tipo (A, g).

6.4.12. Un quandle X se dice **involutivo** si para cada $x \in X$ se tiene $\varphi_x^2 = \mathrm{id}$. Escriba una función que determine si una matriz es la matriz de un quandle involutivo.

6.4.13. Es un ejercicio sencillo demostrar que en un quandle de conjugación necesariamente se verifica que $x \triangleright y = y =$ si y sólo si $y \triangleright x = x$. Demuestre que el quandle de tres elementos dado por la sucesión de permutaciones $\varphi_1 = (23)$, $\varphi_2 = \varphi_3 = \mathrm{id}$ no es un quandle de conjugación.

6.4.14. Construya el quandle dado por las permutaciones

$$\varphi_1 = (376)(485), \quad \varphi_2 = (376)(485), \quad \varphi_3 = (168)(257), \quad \varphi_4 = (168)(257),$$
$$\varphi_5 = (174)(283), \quad \varphi_6 = (174)(283), \quad \varphi_7 = (135)(246), \quad \varphi_8 = (135)(246),$$

calcule su grupo interior, compruebe que es conexo y verifique que no es un quandle de conjugación.

Referencias

[1] N. Andruskiewitsch and M. Graña. From racks to pointed Hopf algebras. *Adv. Math.*, 178(2):177–243, 2003.

[2] M. Brennan and D. Machale. Variations on a Theme: A_4 Definitely Has no Subgroup of Order Six! *Math. Mag.*, 73(1):36–40, 2000.

[3] R. D. Carmichael. *Introduction to the theory of groups of finite order*. Dover Publications, Inc., New York, 1956.

[4] H. S. M. Coxeter. *Kaleidoscopes*. Canadian Mathematical Society Series of Monographs and Advanced Texts. John Wiley & Sons, Inc., New York, 1995. Selected writings of H. S. M. Coxeter, Edited by F. Arthur Sherk, Peter McMullen, Anthony C. Thompson and Asia Ivić Weiss, A Wiley-Interscience Publication.

[5] P. de la Harpe. *Topics in geometric group theory*. Chicago Lectures in Mathematics. University of Chicago Press, Chicago, IL, 2000.

[6] R. M. Guralnick. Commutators and commutator subgroups. *Adv. in Math.*, 45(3):319–330, 1982.

[7] D. MacHale. Minimum counterexamples in group theory. *Math. Mag.*, 54(1):23–28, 1981.

[8] C. F. Miller, III and P. E. Schupp. Some presentations of the trivial group. In *Groups, languages and geometry (South Hadley, MA, 1998)*, volume 250 of *Contemp. Math.*, pages 113–115. Amer. Math. Soc., Providence, RI, 1999.