An Introduction to Computational Group Theory (GAP)

Groups, Algorthims, and Programming

Christian Poveda & David Cardozo

May 4, 2015

Universidad de los Andes
Facultad de Ciencias

What is GAP?

1. **DIEHTLYAPL**

What is GAP?

1. **DIEHTLYAPL**
2. “Do I really have to Learn Yet Another Programming Language?”

What is GAP?

1. **DIEHTLYAPL**
2. "Do I really have to Learn Yet Another Programming Language?"

Universidad de los Andes
Facultad de Ciencias

What is GAP?

1. **DIEHTLYAPL**
2. “Do I really have to Learn Yet Another Programming Language?”

GAP

Group, Algorithms, and Programming. Computer algebra system **CAS**

Terminal Tool

Read input $\rightarrow$ Evaluate $\rightarrow$ Print.

What is GAP?

1. **DIEHTLYAPL**
2. "Do I really have to Learn Yet Another Programming Language?"

GAP

Group, Algorithms, and Programming. Computer algebra system **CAS**

Terminal Tool

Read input $\rightarrow$ Evaluate $\rightarrow$ Print.

Research Tool

You shape it to your needs.

Universidad de los Andes
Facultad de Ciencias

Computational Group Theory

The study of algorithms for groups. Produce algorithms to answer questions about concrete groups: combinatorial structures.

Can we calculate the objects we define theoretically ?

Questions Concrete Groups

Complexity Theory

P=NP

Graph Isomorphism [Solved] Travel

Recall GAP is a command line tool

```
~$ mkdir ClassTutorial
~$ cd ClassTutorial/
~/ClassTutorial$ gap
 ┌───────┐   GAP, Version 4.7.5
 │  GAP  │   http://www.gap-system.org
 └───────┘   Architecture: x86_64...
Libs used:   gmp, readline
Loading the library and packages ...
Components: trans 1.0, prim 2.1, ...
Packages:    Alnuth 3.0.0, AtlasRep 1.5.0 ...
Try '?help' for help.
gap>
```

The GAP console

Internal types of data structures: Integers are built in, Boolean Values: true, false. And also Gap doesn't need to declare types of variables

```
$ gap>TeachingMode(true);
#I Teaching Mode is turned ON
$ gap>8=9;
false
gap> 1^13 + 12^3 = 9^3 + 10^3;
true
gap> FirstPerfectNumber := 6;
6
gap> 22+FirstPerfectNumber;
28
```

Universidad de los Andes
Facultad de Ciencias

Functions

GAP comes with a lot of Built-in functions that are commonly used in Group Theory, but it also provide ways to define our own functions.

Question

Is $2^{13} - 1$ a prime number ?

Universidad de los Andes
Facultad de Ciencias

Functions

GAP comes with a lot of Built-in functions that are commonly used in Group Theory, but it also provide ways to define our own functions.

Question

Is $2^{13}-1$ a prime number ?

```
gap> IsPrime(2^13 -1);
true
```

Universidad de los Andes
Facultad de Ciencias

Functions

GAP comes with a lot of Built-in functions that are commonly used in Group Theory, but it also provide ways to define our own functions.

Question

Is $2^{13} - 1$ a prime number ?

```
gap> IsPrime(2^13 -1);
true
```

Let us define our own functions:

```
gap> AddOne := function(x) return x+1; end;
function( x ) ... end
```

Shortcut: `AddOne:= (x -> x+1);`

Universidad de los Andes
Facultad de Ciencias

Lists

Sometimes we will a “container” a list structure to keep data. Built-in functions include: Test Membership, and Long List Constructor:

```
gap> L1 := [4,5,6,7,8,12];
[ 4, 5, 6, 7, 8, 12 ]
gap> Position(L1,7);
gap> List([1..10],x->x);
[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]
gap> List([5,9..45],x->x);
[ 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45 ]
gap> List([20..29],x->x^2);
[ 400, 441, 484, 529, 576, 625, 676, 729, 784, 841 ]
```

Bad Feature on GAP: **List themselves are pointers to lists**, for copying lists(vector or matrices) `ShallowCopy`

Universidad de los Andes
Facultad de Ciencias

Vectors and Matrices

```
gap> vec := [-1,2,1];
[ -1, 2, 1 ]
gap> M:=[[1,2,3],[4,5,6],[7,8,9]];
[ [ 1, 2, 3 ], [ 4, 5, 6 ], [ 7, 8, 9 ] ]
gap> Display(M);
[ [  1,  2,  3 ],
[  4,  5,  6 ],
[  7,  8,  9 ] ]
gap> vec*M;
[ 14, 16, 18 ]
gap> M*vec;
[ 6, 12, 18 ]
gap> M[3][2];
8
gap> vec*vec; #The inner product
6
```

While vectors in GAP are usually considered as row vectors, scalar products or matrix/vector products automatically consider the second factor as column vector.

Hill Cipher

Plain text is divided into sets of *n* letters, each of which is replaced by a set of *n* cipher letters is called a **polygraphic system**.

A	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*	*P*	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	0

Theorem

A square matrix *A with entries in* $\mathcal{Z}_m$ *is invertible modulo m if and only if the residue of* $\det(A)$ *modulo m has a reciprocal modulo m*

Corollary

A *square matrix A* with entires in $\mathcal{Z}_{26}$ is invertible modulo 26 if and only if the residue of $\det(A)$ mod 26 is not divisible by 2 or 13

Algorithm for the Hill Cipher

1. Choose a 2×2 matrix with integers:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

Algorithm for the Hill Cipher

1. Choose a 2×2 matrix with integers:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

2. Group successive plaintext letter into pairs, adding an arbitrary "dummy" letter *mutatis mutandis*, and replace by its numerical value

Universidad de los Andes
Facultad de Ciencias

Algorithm for the Hill Cipher

1. Choose a 2×2 matrix with integers:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

2. Group successive plaintext letter into pairs, adding an arbitrary "dummy" letter *mutatis mutandis*, and replace by its numerical value
3. Successively convert each plaintext pair into a column vector:

$$\vec{p} = \begin{pmatrix} p_1 \\ p_2 \end{pmatrix}$$

Form the product $A\vec{p}$ the **ciphertext vector**

Universidad de los Andes
Facultad de Ciencias

Algorithm for the Hill Cipher

1. Choose a 2×2 matrix with integers:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

2. Group successive plaintext letter into pairs, adding an arbitrary "dummy" letter *mutatis mutandis*, and replace by its numerical value
3. Successively convert each plaintext pair into a column vector:

$$\vec{p} = \begin{pmatrix} p_1 \\ p_2 \end{pmatrix}$$

   Form the product $A\vec{p}$ the **ciphertext vector**

4. Convert each ciphertext vector into its alphabetic equivalent

Universidad de los Andes
Facultad de Ciencias

Message to encode: **I AM HIDING**, using the matrix:

$$\begin{pmatrix} 1 & 2 \\ 0 & 3 \end{pmatrix}$$

I A	M H	I D	I N	G G
9 1	13 8	9 4	9 14	7 7

```
gap> im:=Integers mod 26; # Represent numbers for mod 26
gap> A := [[1,2],[0,3]]*One(im);
gap> p1 := [9,1]*One(im);
gap> p2 := [13,8]*One(im);
gap> p3 := [9,4]*One(im);
gap> p4 := [9,14]*One(im);
gap> p5 := [7,7]*One(im);
```

Hill Cipher

Hell 2-cipher

```
gap> Result := List([A*p1,A*p2,A*p3,A*p4,A*p5],x->x);
gap> Display(Result);
matrix over Integers mod 26:
[ [  11,   3 ],
[   3,  24 ],
[  17,  12 ],
[  11,  16 ],
[  21,  21 ] ]
```

11 3	3 24	17 12	11 16	21 21
K C	C X	Q L	K P	U U

So we will transmit the following:

KCCXQLKPUU

Breaking a Hill Cipher

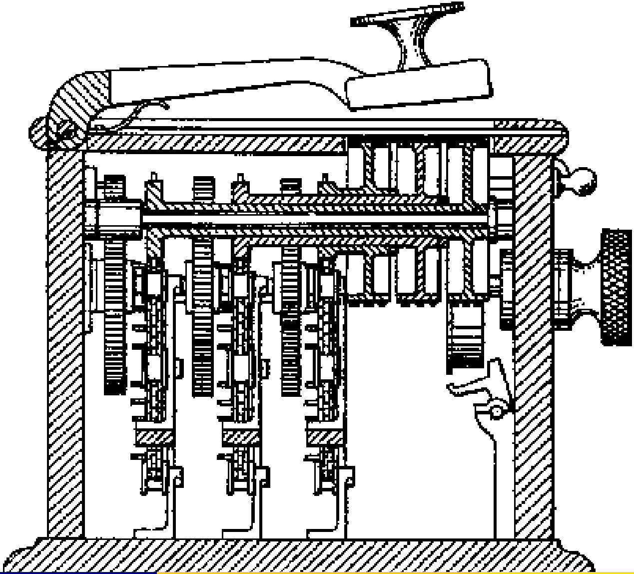


Figure: Polish Card

Creation of the Republic of Poland, was cumbersome at best. In 1934, a card was intercept from Galicia from a Polish Army officer (some sites mention Edward Rydz-Śmigły) to Warsaw.

IOSBTHXESPXHOPDE

It was customary in Poland to star a letter with DEAR

It is a basic result in linear algebra that a linear transformation is completely determined by its values at a basis. Thus suggests that if we have a Hill n-cipher, and if $\vec{p}_1, \ldots \vec{p}_n$ are linearly independent plaintext vectors whose corresponding cipher vectors: $A\vec{p}_1, \ldots, A\vec{p}_n$ are known, then there is enough information available to determine the matrix A and hence $A^{-1} \mod 26$

Theorem

Let $\vec{p}_1, \ldots \vec{p}_n$ be linearly independent plaintext vectors, and let $\vec{c}_1, \ldots, \vec{c}_n$ be the corresponding ciphertext vectors in a Hill n-cipher if:

$$P = \begin{pmatrix} \vec{p}_1^T \\ \vdots \\ \vec{p}_n^T \end{pmatrix} \quad C = \begin{pmatrix} \vec{c}_1^T \\ \vdots \\ \vec{c}_n^T \end{pmatrix}$$

Where both are $n \times n$, then the sequence of elementary row operations that reduces C to I transforms P to $(A^{-1})T$

Solution to Problem using Gap

```
gap> G:= CyclicGroup(6);
<fp group of size 6 on the generators [ a ]>
gap> Elements(G);
[ <identity ...>, a, a^2, a^3, a^4, a^5 ]
gap> List(Elements(G),Order);
[ 1, 6, 3, 2, 3, 6 ]
gap> ShowMultiplicationTable(G);
*     | <id> a    a^2  a^3  a^4  a^5
-----+------------------------------
<id> | <id> a    a^2  a^3  a^4  a^5
a    | a    a^2  a^3  a^4  a^5  <id>
a^2  | a^2  a^3  a^4  a^5  <id> a
a^3  | a^3  a^4  a^5  <id> a    a^2
a^4  | a^4  a^5  <id> a    a^2  a^3
a^5  | a^5  <id> a    a^2  a^3  a^4
```

Abelian Group

The Fundamental Theorem Of Finite Abelian Groups

A finite Abelian group is isomorphic to a direct product of cyclic groups of prime-power order

```
gap> G:=AbelianGroup([2,4,5]);
<fp group of size 40 on the generators [ f1, f2, f3 ]>
gap> gens:= GeneratorsOfGroup(G);
[ f1, f2, f3 ]
gap> List(gens,Order);
[ 2, 4, 5 ]
```

Universidad de los Andes
Facultad de Ciencias

Working with: $\left(\frac{\mathbb{Z}}{n\mathbb{Z}}\right)^{\times}$

```
gap> G:= Units(Integers mod 21);
<group of size 12 with 2 generators>
gap> e:=Elements(G);
[ ZmodnZObj( 1, 21 ), ZmodnZObj( 2, 21 ), ZmodnZObj( 4, 21
ZmodnZObj( 5, 21 ), ZmodnZObj( 8, 21 ), ZmodnZObj( 10, 21 )
ZmodnZObj( 11, 21 ), ZmodnZObj( 13, 21 ), ZmodnZObj( 16, 21
ZmodnZObj( 17, 21 ), ZmodnZObj( 19, 21 ), ZmodnZObj( 20, 21
gap> List(e,Order);
[ 1, 6, 3, 6, 2, 6, 6, 2, 3, 6, 6, 2 ]
```

Universidad de los Andes
Facultad de Ciencias

Working with: $\left(\frac{\mathbb{Z}}{n\mathbb{Z}}\right)^{\times}$

```
gap> G:= Units(Integers mod 21);
<group of size 12 with 2 generators>
gap> e:=Elements(G);
[ ZmodnZObj( 1, 21 ), ZmodnZObj( 2, 21 ), ZmodnZObj( 4, 21
ZmodnZObj( 5, 21 ), ZmodnZObj( 8, 21 ), ZmodnZObj( 10, 21 )
ZmodnZObj( 11, 21 ), ZmodnZObj( 13, 21 ), ZmodnZObj( 16, 21
ZmodnZObj( 17, 21 ), ZmodnZObj( 19, 21 ), ZmodnZObj( 20, 21
gap> List(e,Order);
[ 1, 6, 3, 6, 2, 6, 6, 2, 3, 6, 6, 2 ]
```

Interpret:

```
List(e,x->Position(e,Inverse(x)));
```

Finally: `gap> ShowGcd(5,9)`

Universidad de los Andes
Facultad de Ciencias

Dihedral Group and Advanced functions

This will be short, just to show a few advanced commands:

```
gap> D3 := DihedralGroup(6);
<fp group of size 6 on the generators [ r, s ]>
gap> D4:=DihedralGroup(8);
<fp group of size 8 on the generators [ r, s ]>
gap> List([D3,D4],x->IsSolvable(x));
[ true, true ]
gap> List([D3,D4],x->IsNilpotent(x));
[ false, true ]
```

We can then compute the multiplication table:

```
gap> ShowMultiplicationTable(DihedralGroup(6));
*    | <id> r^-1 r    s    r*s  s*r
-----+------------------------------
<id> | <id> r^-1 r    s    r*s  s*r
r^-1 | r^-1 r    <id> s*r  s    r*s
r    | r    <id> r^-1 r*s  s*r  s
s    | s    r*s  s*r  <id> r^-1 r
r*s  | r*s  s*r  s    r    <id> r^-1
s*r  | s*r  s    r*s  r^-1 r    <id>
```

Subgroups

Subgroups in GAP are created and stored by giving generators of the subgroup. The function `Subgroup(group,generators)` constructs a subgroup with given generators. Compared with `Group`, GAP tests whether the generators are actually in the group.

The commands `Normalizer(group,sub)` and `Centralizer(group,sub)` return associated subgroups.

The command `AllSubgroups` should be used with caution:

```
gap> D6:=DihedralGroup(IsPermGroup,6);
Group([ (1,2,3), (2,3) ])
gap> AllSubgroups(D6);
[ Group(()), Group([ (2,3) ]), Group([ (1,2) ]),
  Group([ (1,3) ]),
  Group([ (1,2,3) ]),
  Group([ (1,2,3), (2,3) ]) ]
```

Universidad de los Andes
Facultad de Ciencias

Subgroup Lattice

GAP provides very general functionality to determine the subgroup structure of a group. To reduce storage this is typically done up to conjugacy. `ConjugacyClassesSubgroups(G)` returns a list of conjugacy classes of subgroups of *G*. For each class *C* in this list `Representative(C)` returns one subgroup in this class. `Stabilizer(C)` returns the normalizer of this Representative in *G*. `Size(C)` returns the number of conjugate subgroups in the class (the index of the normalizer). Last but not least, `NormalSubgroups(G)` returns a list of all normal subgroups.

Subgroup Lattice II

`LatticeSubgroups(G)` determines an object L that represents the lattice of subgroups of G .(The classes of subgroups can be also obtained from this object as `ConjugacyClassesSubgroups(L)`.) For such a lattice object, `MaximalSubgroupsLattice(L)` returns a list *M* that describes maximality inclusion.

Universidad de los Andes
Facultad de Ciencias

Subgroup Lattice II

`LatticeSubgroups(G)` determines an object L that represents the lattice of subgroups of G .(The classes of subgroups can be also obtained from this object as `ConjugacyClassesSubgroups(L)`.) For such a lattice object, `MaximalSubgroupsLattice(L)` returns a list *M* that describes maximality inclusion.

Activity

Let's draw the lattice of S_4 using GAP

```
gap> L:=LatticeSubgroups(G);
DotFileLatticeSubgroups(L,"tester.dot");
```

This will produce a dot file that can be drawn with:

```
dot -Teps tester.dot > output.eps
```

Universidad de los Andes
Facultad de Ciencias

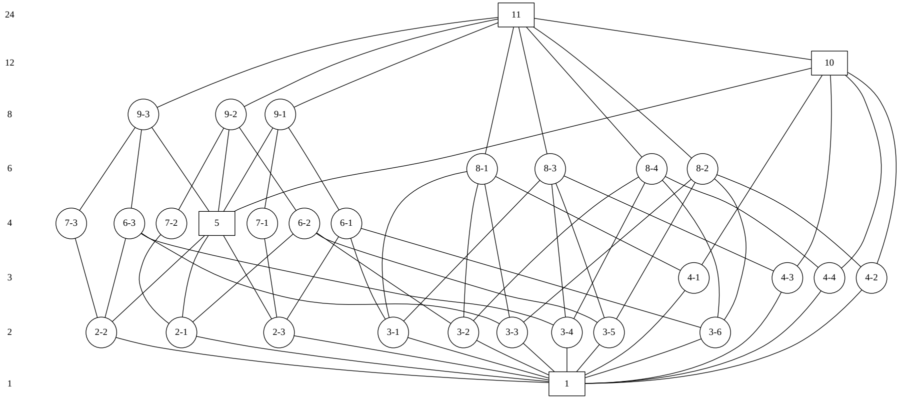
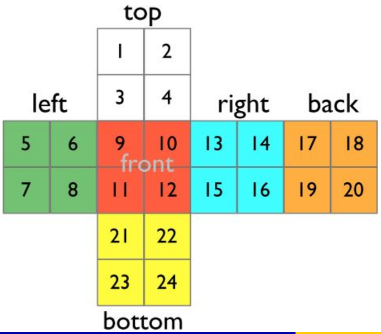


Figure: Lattice of S_4



Many puzzles can be described in this way: Each state of the puzzle corresponds to a permutation, the task of solving the puzzle then corresponds to expressing the permutation as a product of generators.