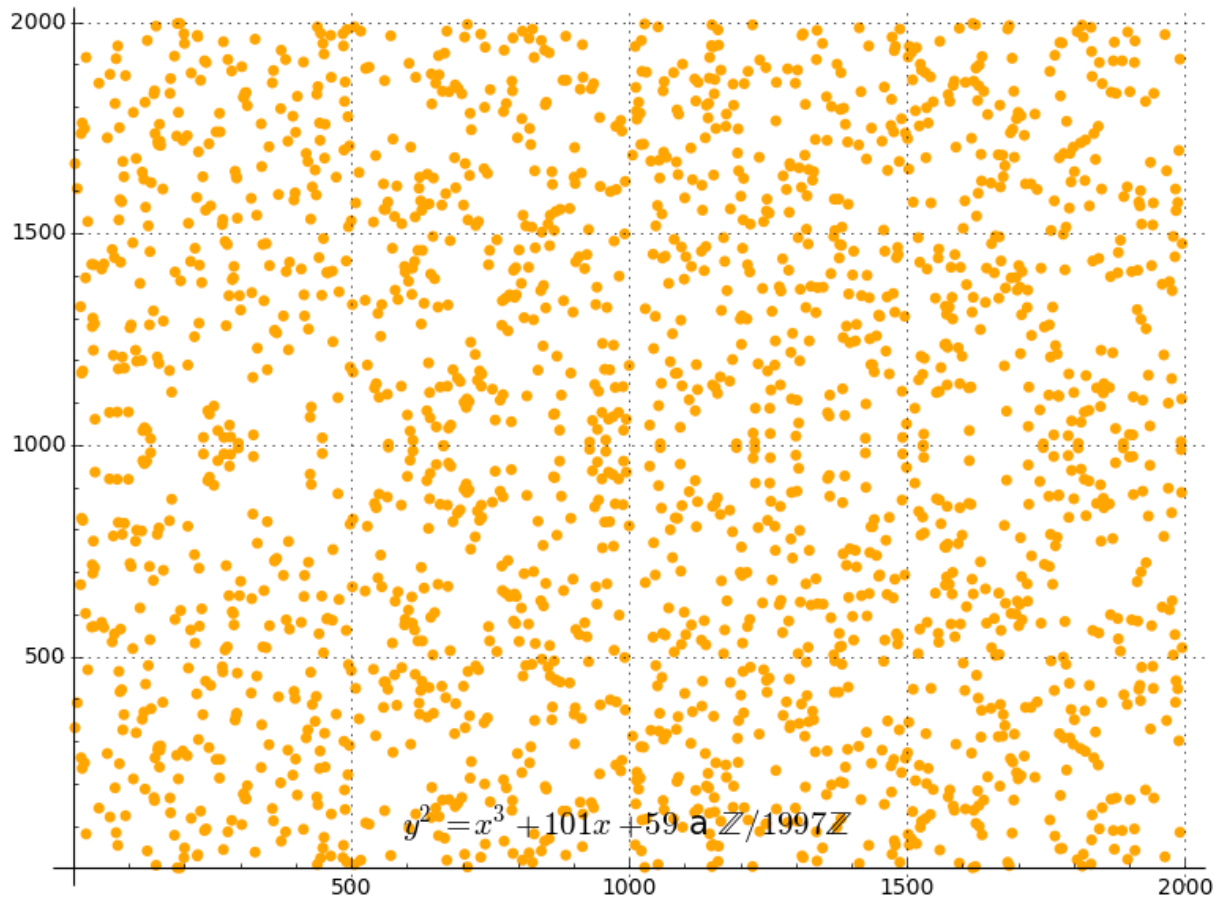
Factoritzar enters amb corbes el·líptiques modulars



Galois

Índex

1. Motivacions

Les matemàtiques són un camp extens i complex que estudia nombres, canvis, estructures i patrons. En tots els camps de les matemàtiques ens trobem conceptes que ens criden l'atenció, tant per ser complexos tot i l'aparença de ser senzills com per la seva senzillesa tot i semblar complexos.
Possiblement, el concepte que més crida l'atenció són els nombres primers.
Els nombres primers són aquells nombres enters més grans que 1 que només es poden dividir sense residu entre si mateixos, la unitat, i els seus oposats.
És a dir,

$$p \in \mathbb{P} \; si \; p \mid A, \; A = \{p, 1, -p, -1\}.$$

Els nombres primers semblen un concepte simple. Sabem que, com de nombres enters, n'hi ha infinits. Sabem també que entre un nombre enter positiu qualsevol i el seu doble, hi ha almenys un nombre primer. El Teorema Fonamental de l'Aritmètica ens diu que qualsevol nombre enter més gran que 1 pot ser expressat com el producte d'un o més nombres primers, en una combinació única on l'ordre no importa. Per exemple, 99 = 3·3·11.

Però hi ha moltes coses que no sabem dels nombres primers, i cadascuna d'elles donaria per a un treball de recerca sencer. Un d'aquests problemes és com saber si un nombre qualsevol és o no primer, i si no ho és, quina n'és la descomposició en els seus factors. Ja ho va dir Carl Friedrich Gauß al seu famós llibre *Disquisitiones Arithmeticae* l'any 1798, "se sap que el problema de distingir els nombres primers dels nombres composts i resoldre els últims en els seus factors primers és un dels problemes més importants i útils en aritmètica".

El mètode més senzill per a saber si un nombre *x* és primer és mirar que no sigui enterament divisible per cap nombre enter de l'interval tancat [2, $\sqrt{x}$]. L'1 no és dins d'aquest interval perquè tots els nombres són divisibles per 1, i aquest interval

acaba a la seva arrel quadrada perquè si un nombre no és primer, la factorització més gran que pot tenir és un primer al quadrat. És a dir, si *x* fos un quadrat perfecte, $\sqrt{x}$ en seria factor, dos cops. Per exemple, com que 25 = 5·5, l'interval en el que hauríem de buscar els divisors és [2, √25], que és el mateix que, [2, 5].

Si prenem el cas *x* = 49, hem d'anar comprovant si és enterament divisible per algun nombre de l'interval [2, √49] o el que és el mateix, [2, 7].
Comencem:

49 / 2	24.5	no divideix[1]
49 / 3	16.333….	no divideix
49 / 4	12.25	no divideix
49 / 5	9.8	no divideix
49 / 6	8.1666...	no divideix
49 / 7	7	divideix

Com que 49 és enterament divisible per algun element dins de [2, 7], no és un nombre primer: la seva factorització és 49 = 7·7. Hem hagut de fer 6 divisions per a saber si era primer o no. Si el nombre fos més gran que 49, com ara 5917, hauríem de comprovar si és divisible per algun nombre entre 2 i 79, i potser tindríem sort i trobaríem un factor de 5917 aviat, o potser no tindríem sort i 5917 és primer i hauríem de fer 77 divisions. Per un nombre més gran, per exemple 18 053, el nombre de divisions augmentaria fins a 135. Massa feina!

En criptografia, els nombres primers són molt importants, ja que les contrasenyes dels usuaris solen ser encriptades utilitzant nombres primers molt (de veritat, molt) grans, perquè, com més gran és un nombre imparell, més difícil és saber si és primer o no. És per això que el mètode que utilitzàvem per a saber si un nombre és o no primer no és gens útil.

[1] Concretament, no divideix enterament. La divisió té resultat, però aquest no és un nombre enter.

Aquest mètode que he utilitzat per a factoritzar el nombre 49 s'anomena factorització per prova de divisions, i és un algorisme: té una informació d'entrada (el nombre *x*), i seguint uns passos (provar a dividir *x* entre tots els nombres de l'interval [2, $\sqrt{x}$]) crea una informació, la sortida (si *x* és primer o si té factors, i quins són aquests). Per això, aquest algorisme és un algorisme de factorització, i alhora una prova de primalitat, perquè si un nombre més gran que 1 només té un factor, és per definició un nombre primer.

2. Hipòtesi

La hipòtesi d'aquest treball és si es pot implementar un mètode de factorització d'enters que dugui a terme la seva tasca en un temps subexponencial. Aniré precisant aquests conceptes al llarg del treball.
M'he centrat en el Mètode de Factorització per Corbes El·líptiques de Lenstra.

2.1. La importància del temps de computació

El temps de computació, concepte habitual en ciències de la computació, es refereix al que triga un algorisme a calcular la sortida, depenent de la llargada de l'entrada. Per a descriure aquest temps, se sol utilitzar la notació de la O gran. Pot ser polinòmic, logarítmic, exponencial, etcètera.

Posem per exemple una prova de primalitat amb un temps de computació $t(n) = 2^n + 2$, on *n* és el tamany de l'entrada. Per a una entrada de 3 xifres, triga 8 operacions. Per a trobar-ne la O gran, el procediment és semblant al d'avaluar el límit d'aquesta funció quan *n* tendeix a infinit,

$$\lim_{n\to\infty} 2^n + 2$$

ignorant els termes menys importants. Per tant, aquest algorisme de prova de primalitat seria de temps exponencial, concretament, $O(2^n)$.

El temps exponencial és assequible quan l'entrada no és gaire gran. Si en el programa de l'exemple anterior hi entréssim un nombre de 15 xifres, requeriria 32.768 operacions fins a obtenir un resultat. Si fos de 50 xifres, 1.125.899.906.842.624 operacions serien necessàries.

Un ordinador mitjà té una CPU que pot dur a terme uns sis mil milions d'operacions per segon (6 GFLOPS). Per tant, podem esbrinar el temps que trigaria aquest ordinador a determinar la primalitat d'un nombre de 50 xifres:

$$\frac{2^{50}+2}{6\times 10^9} = \frac{1125899906844}{6\times 10^9} = 187.65\,\mathrm{s}$$

Tenint en compte que les operacions dels nostres ordinadors són dutes a terme per impulsos elèctrics viatjant a uns 120.000.000 m/s, cent vuitanta-set segons de computació són, encara que no ho sembli, molt de temps.

És més, el nombre primer més gran conegut avui en dia, $2^{57\,885\,161}-1$, té 17.425.170 xifres. Per dir-nos si aquest nombre és primer o no, el programa de l'exemple anterior trigaria:

$$\frac{2^{17425170}+2}{6\times 10^9} = \frac{7\times 10^{5245498}}{6\times 10^9} \approx 1.1\bar{6}\times 10^{5245489}\,\mathrm{s} \approx 3.7\times 10^{5245481}\,\mathrm{anys}$$

Tenint en compte que aquest nombre està cinc milions d'ordres de magnitud per sobre de l'edat de l'Univers, aquesta no seria una prova de primalitat útil. Per a seguir trobant nombres primers sense caure en el problema plantejat per l'exemple anterior, s'han anat inventant una sèrie d'algorismes de prova de primalitat o de factorització, tant deterministics (la seva resposta és segura) com probabilístics (la seva resposta pot ser errònea), que busquen el temps subexponencial (que és per definició inferior a l'exponencial).

3. Coneixements previs

Hi ha una sèrie de conceptes matemàtics que hem d'aprendre per a entendre el funcionament de l'Algorisme de Factorització per Corbes El·líptiques de Lenstra.

3.1. Coprimalitat

Dos nombres són coprimers quan el seu màxim comú divisor és 1. A vegades es denota que dos nombres *a* i *b* són coprimers així:

$$a \perp b$$

3.2. Aritmètica modular

L'aritmètica modular és la branca de les matemàtiques encarregada de l'estudi dels nombres enters modulars i de les operacions de suma, resta, multiplicació, divisió i exponenciació en ells. Molts cops s'anomena "aritmètica de rellotge" per la seva analogia amb els nombres de l'esfera d'un rellotge.

El concepte de nombres modulars té a veure amb la divisió euclidiana. La divisió euclidiana és el procediment de divisió que aprenem a l'escola.

$$\begin{array}{r|l} dividend & divisor \\ \hline \vdots & quocient \\ residu & \end{array}$$

on es compleix que

$$dividend = divisor \times quocient + residu$$

Diem que el mòdul de dos nombres és el residu de la seva divisió euclidiana, és a dir, el mòdul del dividend i el divisor és el residu.
Es denota utilitzant l'expressió "mod",

$$dividend \bmod divisor = residu$$

com per exemple:

$$9 \bmod 5 = 4 \qquad 3 \bmod 2 = 1$$
$$10 \bmod 5 = 0 \qquad 25 \bmod 3 = 1$$

perquè 9 = 5·1 + **4**, 3 = 2·1 + **1**, 10 = 5·1 + **0**, i 25 = 3·8 + **1**. El mòdul ens diu com de lluny està un nombre *x* del múltiple d'*y* més proper a *x*.

Un concepte molt important en el camp de l'aritmètica modular és el de congruència. Dos nombres o expressions són congruents en mòdul *n* quan el residu de tots dos al dividir-los entre *n* és el mateix. La congruència es denota amb el símbol d'equivalència $\equiv$ entre dues expressions, i al darrere d'això sol haver-hi entre parèntesis "(mod *n*)". A vegades, però, no cal posar "(mod *n*)" perquè se sobreentén pel context.
Això és

$$a \equiv b \pmod{n} \Leftrightarrow a \bmod n = b \bmod n .$$

Un altre concepte rellevant és el d'enters mòdul *n*. Es denota $\mathbb{Z}/n\mathbb{Z}$
i és convenient prendre'l com el conjunt {[0], [1], [2], …, [*n*-1]}, on cada element [*k*] representa tots els nombres que són congruents a *k* en mòdul *n*. També s'anomena "anell de classes de residus".
Per exemple, en el cas *n* = 2:

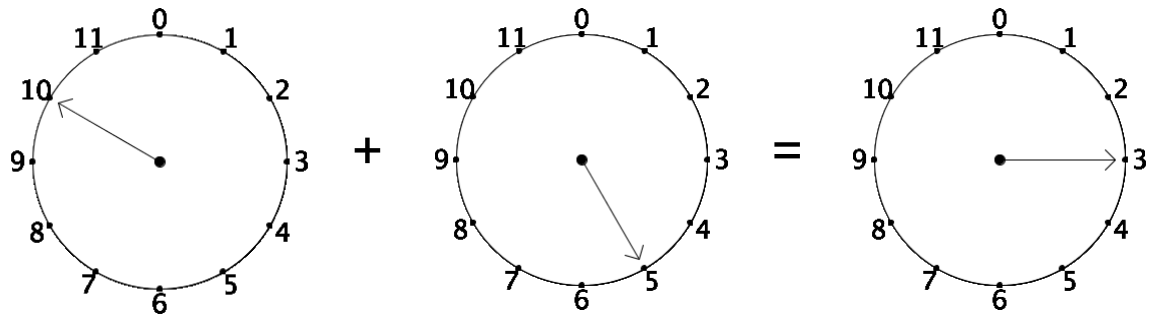
$$\mathbb{Z}/2\mathbb{Z} = \{[0], [1]\}$$

on [0] = {..., -8, -6, -4, -2, 0, 2, 4, 6, 8,…}, els nombres parells, que són congruents a 0 mod 2, i [1] = {..., -7, -5, -3, -1, 1, 3, 5, 7,...}, els nombres senars, que són congruents a 1 mod 2.

Quan estem treballant amb nombres dins d'un grup cíclic $\mathbb{Z}/n\mathbb{Z}$, les operacions aritmètiques són una mica diferents. Normalment, és suficient amb simplificar l'expressió a analitzar reduint mod *n* a cada pas.

En un rellotge, si l'agulla és a les 10 i li sumem 5 hores, seran les 3. Això significa:

$$10 + 5 \bmod 12 = 15 \bmod 12 = 3 \Leftrightarrow 15 \equiv 3 \pmod{12}$$

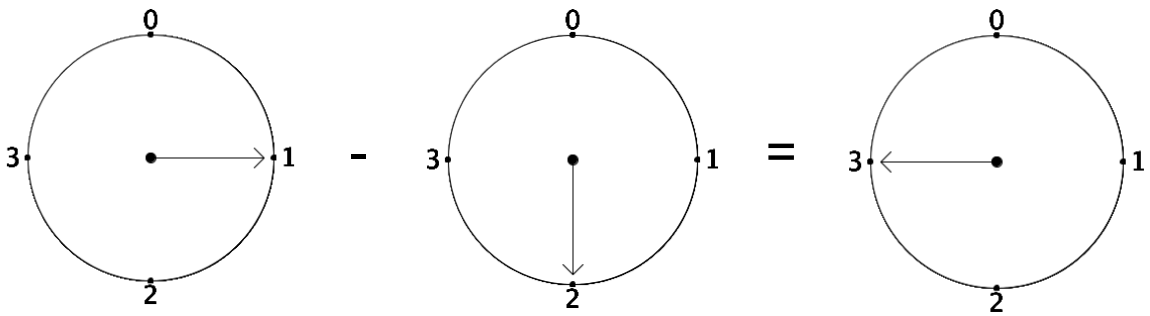


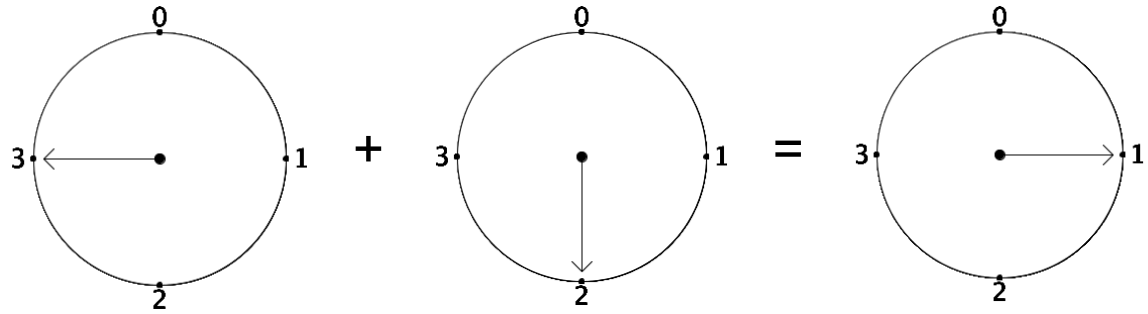
que és bàsicament l'equivalència entre el format horari de 12 hores i el de 24 hores.

Un rellotge d'agulles és la representació del grup $\mathbb{Z}/12\mathbb{Z}$.

La suma i la resta en $\mathbb{Z}/n\mathbb{Z}$ estan definides així, per a qualsevol *n* enter.

El conjunt $\mathbb{Z}/4\mathbb{Z} = \{[0], [1], [2], [3]\}$ està compost, doncs, per tots els nombres que al dividir-los entre 4 tenen residu 0 (és a dir, els múltiples de 4), els que tenen residu 1, els que tenen residu 2 i els que tenen residu 3. En un rellotge de 4 hores (mod 4), es compleix que 1 - 2 = 3 i 3 + 2 = 1.





Anem a provar ara en el conjunt anterior la següent operació possible: la multiplicació. Per exemple, si agafem [2]·[2] veurem que el resultat és [4] que és el mateix que [0]. Això significa que [2], en el conjunt dels enters mòdul 4, és un zero-divisor: a $\mathbb{Z}/4\mathbb{Z}$, dividir entre [2] és com dividir entre 0 a $\mathbb{Z}$. [3]/[2] és una indeterminació, a $\mathbb{Z}/4\mathbb{Z}$.

Per tant, a qualsevol $\mathbb{Z}/n\mathbb{Z}$ per a *n* > 0, podrem sumar, restar i multiplicar, però no sempre dividir. Per a assegurar-nos la divisió, hem de descriure el cos $(\mathbb{Z}/n\mathbb{Z})^*$. Un cos és un conjunt de coses amb dues operacions, en aquest cas addició i producte, que sigui tancat, commutatiu, associatiu, amb un element neutre per cada operació, amb propietat distributiva i finalment que tot element tingui un invers. Quan un element és operat amb el seu invers, el resultat és l'element neutre d'aquesta operació. A l'apartat 3.3 d'aquest treball hi ha més informació sobre algunes d'aquestes propietats

$(\mathbb{Z}/n\mathbb{Z})^*$ està format per totes les unitats mòdul *n,* amb operacions de suma i de producte. Aquestes unitats són tots els elements que no són zero-divisors, és a dir, els nombres coprimers a *n* i més petits que *n*, que tenen inversa multiplicativa.

$$(\mathbb{Z}/4\mathbb{Z})^* = \{1,3\}\,,\ (\mathbb{Z}/10\mathbb{Z})^* = \{1,3,7,9\}$$

És molt important tenir en compte que si *p* és un nombre primer, *p* és coprimer amb tots els enters positius més petits que *p*. Per tant, això ens dona la igualtat

$$\mathbb{Z}/p\mathbb{Z} = (\mathbb{Z}/p\mathbb{Z})^*$$

Ara si que podem definir la divisió, perquè sabem que tots els elements d'aquest cos de les unitats mòdul 4, {1, 3}, no són zero-divisors. Però no podem dividir $[1]/[3]$ així com així perquè el resultat no seria enter. Com que

$$\frac{[1]}{[3]} = [1] \times [3]^{-1}$$

tant sols hem de trobar $[3]^{-1}$. Per a fer-ho, hem de recórrer a la Identitat de Bézout, que diu que existeixen coeficients enters *x* i *y* pels quals es compleix la igualtat de la suma dels múltiples de dos nombres amb el seu màxim comú divisor. Això és:

$$\exists a, b \in \mathbb{Z},\ a, b \neq 0 : ax + by = \mathrm{mcd}(x, y)$$

Recordem que perquè dos enters siguin coprimers, el seu màxim comú divisor ha de ser 1, i que, per ser al cos $(\mathbb{Z}/n\mathbb{Z})^*$, un element ha de ser coprimer a *n*. Llavors, per a algun *n* determinat, si agafem un nombre *t* que pertanyi a $(\mathbb{Z}/n\mathbb{Z})^*$, el seu màxim comú divisor amb *n*, mcd(*n*, *t*), serà 1. Si a la identitat de Bézout hi substituïm:

$$ax + by = \mathrm{mcd}(x, y) \xrightarrow[y:=t]{x:=n} an + bt = 1$$

On *a* i *b* seran enters i no seran 0. Si reduïm la igualtat de la dreta mod *n* (cal recordar que seguim a $\mathbb{Z}/n\mathbb{Z}$), tindrem

$$an \bmod n + bt \bmod n = 1 \bmod n$$

Com que *an* és un multiple de *n*, sabem que la divisió *an*/*n* tindrà un resultat enter *a* i un residu 0. Per tant,

$$bt \bmod n = 1 \bmod n$$

o el que és el mateix,

$$bt \equiv 1 \pmod{n}$$

que significa que en mod *n*, *b* és la inversa de *t*, perquè al multiplicar-los obtenim 1.

$$bt \equiv 1 \pmod{n} \Leftrightarrow b \equiv t^{-1} \pmod{n}$$

Ens proposàvem quina era la inversa de [3] a $(\mathbb{Z}/4\mathbb{Z})^*$ per a calcular [1]/[3]. Hem de resoldre aquesta congruència:

$$3b \equiv 1 \pmod{4}$$

recorrent a la definició de congruència, aquesta expressió significa que el nombre que busquem és múltiple de 3 i val una unitat més que un múltiple de 4. El primer nombre que satisfà això és *b* = [3], perquè 3·3 = 9, i 9 val una unitat més que un múltiple de 4. En realitat, per resoldre aquest problema s'utilitzaria la Identitat de Bézout per a *x* = 4 i *y* = 3 que, recordem, són nombres coprimers, perquè es coneix un mètode molt eficaç per a fer-ho explicat a l'apartat 5.5 d'aquest treball.

Per tant, a $(\mathbb{Z}/4\mathbb{Z})^*$, [3] és la inversa de [3]. Llavors, a $(\mathbb{Z}/4\mathbb{Z})^*$

$$\frac{[1]}{[3]} = [1] \times [3]^{-1} = [1] \times [3] = [3]$$

i seguint aquest procediment, ja podem dividir dins de qualsevol cos $(\mathbb{Z}/n\mathbb{Z})^*$.

3.3. Grup abelià

Un grup abelià és un conjunt d'elements amb una operació binària que, prenent com a arguments dos elements del conjunt, dóna com a resultat un element del mateix conjunt. Aquesta operació la denotaré, per generalitzar, "#".

Perquè un grup sigui abelià, ha de complir quatre condicions:

- Hi ha d'haver un element neutre en el grup: un element *e* que si l'operem amb qualsevol altre element del mateix grup *x*, el resultat sigui *x*.
- Hi ha d'haver un element oposat *y* per a cada *x*, pel que es compleixi $x \# y = e$.
- L'operació ha de ser associativa: $(x \# y) \# z$ ha de ser igual a $x \# (y \# z)$.
- L'operació ha de ser commutativa: l'ordre dels arguments de la operació no altera el resultat, és a dir, $x \# y = y \# x$.

Per exemple, el grup dels nombres enters i l'operació suma formen un grup abelià, perquè:

- L'element neutre *e* és el 0; si sumem zero a un nombre, el resultat és aquell nombre
- Tots els elements tenen un element oposat: l'oposat del nombre *x* és -*x*, perquè $x + (-x) = 0$
- La suma és associativa, ja que $1 + (2 + 3) = (1 + 2) + 3$
- La suma és commutativa, perquè $6 + 5 = 5 + 6$

3.4. Corbes El·líptiques

Les corbes el·líptiques són corbes implícites del tipus:

$$y^2 = x^3 + Ax + B$$

Per evitar irregularitats en la corba, busquem que el seu discriminant Δ, que és un nombre característic de cada corba el·líptica (anàlogament al discriminant d'una equació de segon grau) no sigui 0, imposant la condició:

$$\Delta = -16(4A^3 + 27B^2) \neq 0$$

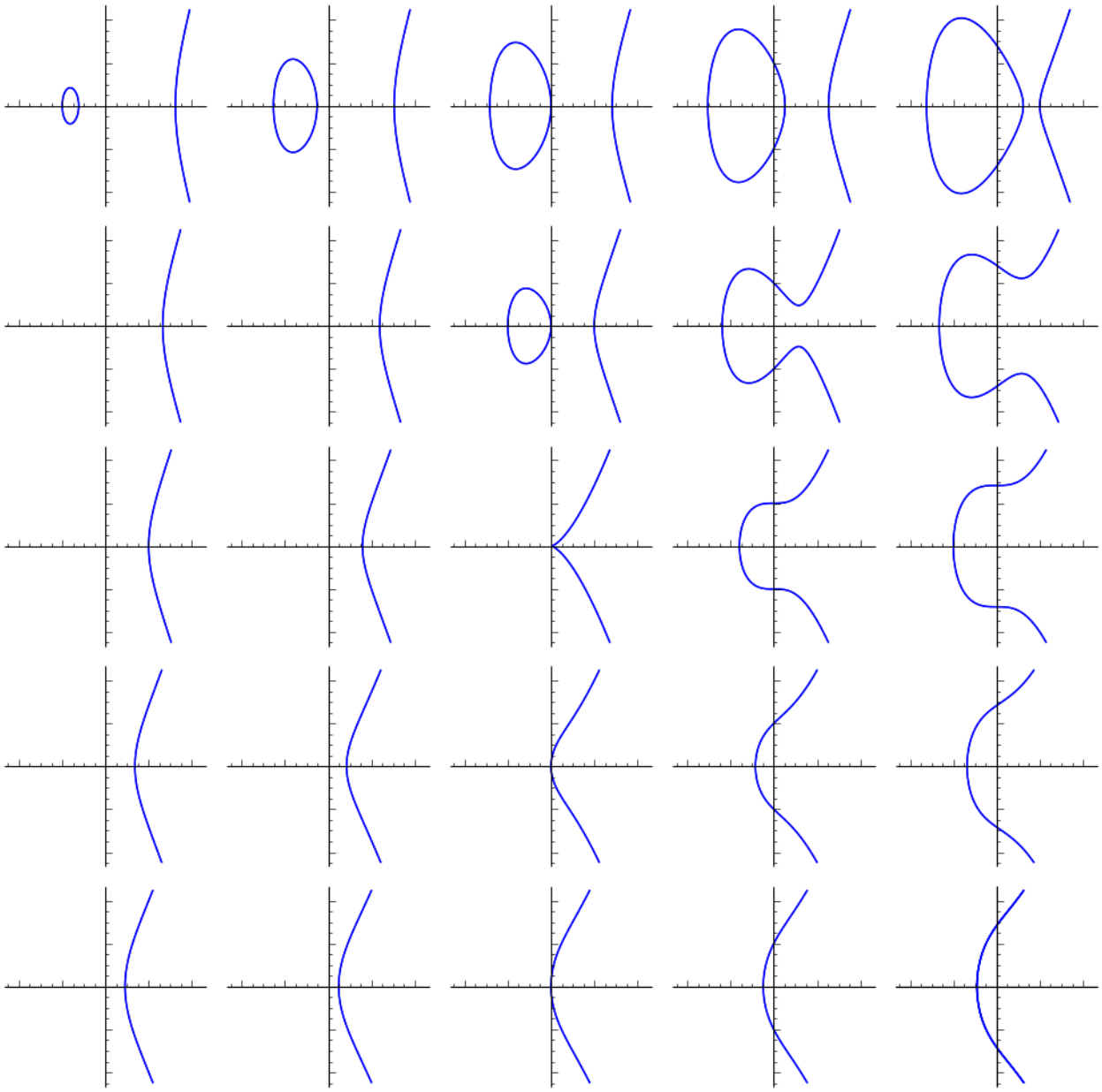
Però com que tant sols ens interessa que no sigui zero, podem simplificar desfent-nos del -16:

$$4A^3 + 27B^2 \neq 0$$

En la imatge següent, es poden veure com són les corbes el·líptiques amb diferents valors d'*A* i *B*. Aquesta imatge ha estat generada amb aquest codi de Sage[2]:

```
import matplotlib
x, y = var('x y')
T = (-4.5, 4.5)
P = []
for A in [-8,-4,..,8]:
   for B in [-8,-4,..,8]:
       P.append(implicit_plot(y^2==x^3+A*x+B, T, T))
G = graphics_array([[P[0],P[1],P[2],P[3],P[4]], [P[5],P[6],P[7],P[8],P[9]],
[P[10],P[11],P[12],P[13],P[14]], [P[15],P[16],P[17], P[18],P[19]],
[P[20],P[21],P[22],P[23],P[24]]])
G.show(tick_formatter =
(matplotlib.ticker.NullFormatter(),matplotlib.ticker.NullFormatter()),
figsize = 10)
```

[2] Pàgina web de Sage: http://www.sagemath.org/, Sage en línia: https://cloud.sagemath.com/



Considerem el conjunt

$$E = \{(x, y) : y^2 = x^3 + Ax + B\} \cup \{\mathcal{O}\}$$

és a dir, tots els punts (*x, y*) que compleixen la condició de pertànyer a la corba el·líptica, i el punt $\mathcal{O}$, que és el punt a l'infinit. Més endavant veurem d'on surt aquest punt.

La corba el·líptica pot ser reescrita com a

$$y^2 = x^3 + Ax + B \Rightarrow y = \pm\sqrt{x^3 + Ax + B}$$

i ara podem avaluar-ne els límits quan la *x* creix:

$$\lim_{x\to\infty} \sqrt{x^3 + Ax + B} = \infty$$
$$\lim_{x\to\infty} -\sqrt{x^3 + Ax + B} = -\infty$$

i quan decreix:

$$\lim_{x\to-\infty} \sqrt{x^3 + Ax + B} = i\infty$$
$$\lim_{x\to-\infty} -\sqrt{x^3 + Ax + B} = -i\infty$$

Tot i que $\mathrm{Re}(\pm i\infty) = 0$, com que els límits quan creix cap a infinit són més i menys infinit, podem dir que hi ha infinits punts que compleixen $y^2 = x^3 + Ax + B$. Per tant, el nombre d'elements del conjunt *E* mencionat abans, denotat |*E*|, és

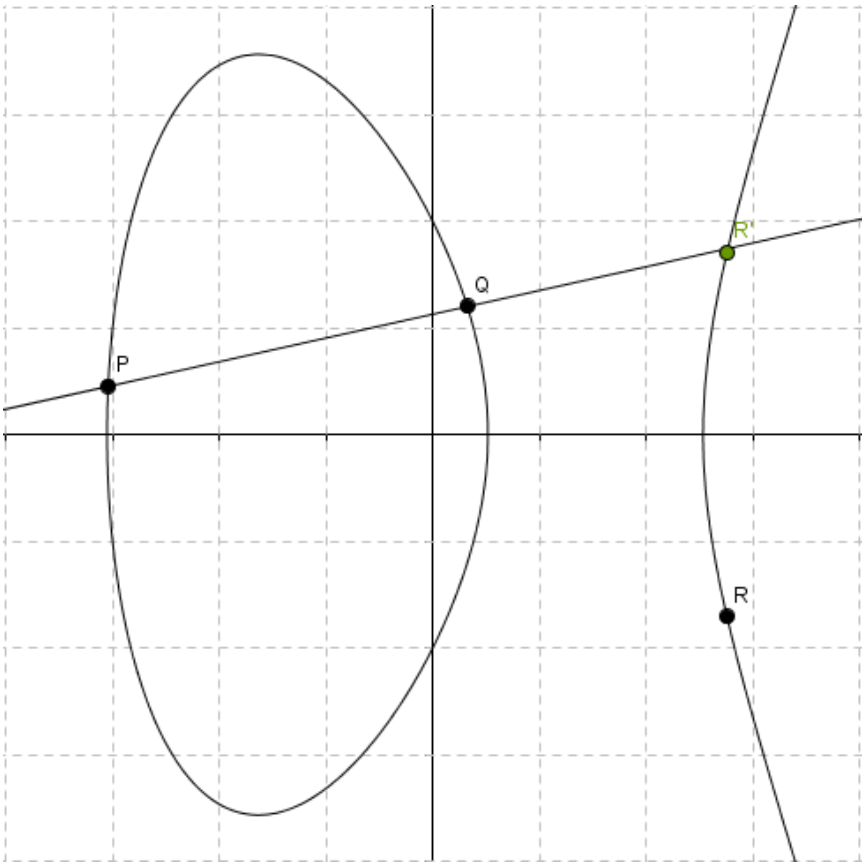
$$\left|E\right| = \infty$$

Aquest conjunt pot ser convertit en un grup abelià utilitzant la geometria. La operació que el converteix en un grup abelià té tres casos possibles:

3.4.1. Primer cas

Donats dos punts P i Q que pertanyin a la corba el·líptica, retorna el punt simètric respecte l'eix d'abscisses del punt en el que la recta que passa per P i Q talla la corba el·líptica un tercer cop.

$$P \oplus Q = R$$



El procediment per a calcular $(x_P, y_P) \oplus (x_Q, y_Q) = (x_R, y_R)$ és aquest:

Primer busquem la recta, que anomenaré L, que passa per P i Q. El seu pendent serà

$$m = \frac{y_Q - y_P}{x_Q - x_P}$$

i la recta, per tant,

$$L: \ y = mx + (y_P - mx_P)$$

Per a trobar els punts on L talla amb la corba el·líptica W, hem de resoldre el sistema d'equacions

$$\begin{cases} W : y^2 = x^3 + Ax + B \\ L : y = mx + (y_P - mx_P) \end{cases}$$

on, substituint la *y* de W per la part dreta de L, tenim:

$$(mx + (y_P - mx_P))^2 = x^3 + Ax + B$$

aplicant $(a+b)^2 = a^2 + 2ab + b^2$:

$$m^2x^2 + 2m(y_P - mx_P)x + (y_P - mx_P)^2 = x^3 + Ax + B$$

igualant a zero aquesta expressió queda:

$$x^3 - m^2x^2 + Ax - 2m(y_P - mx_P)x + B - (y_P - mx_P)^2 = 0$$

i finalment traient factor comú de la *x* de primer grau:

$$W \cap L : x^3 - m^2x^2 + (A - 2m(y_P - mx_P))x + B - (y_P - mx_P)^2 = 0$$

Sabem que el polinomi $W \cap L$ que, recordem, defineix els punts on la corba el·líptica W i la recta L que passa per P i Q es tallen, té fins a 3 solucions, perquè és de tercer grau. Però com que P i Q són punts a la corba el·líptica W i a la recta L, sabem que els seus components *x* són arrels del polinomi. L'altra arrel possible és el component *x* del punt R' que és simètric respecte OX al punt resultant, R. Com que R i R' són simètrics respecte l'eix de les abscisses, tenen el mateix component *x*. Per tant,

$$W \cap L : (x - x_P)(x - x_Q)(x - x_R) = 0$$

Multiplicant aquests tres elements obtenim:

$$x^3 - (x_Q + x_P + x_R)x^2 + (x_P x_Q + x_Q x_R + x_P x_R)x - x_P x_Q x_R = 0$$

i si ara igualem aquest polinomi amb l'altre en una taula

	coeficients polinomi del sistema	coeficients polinomi de les arrels
x^3	1	1
x^2	$-m^2$	$-(x_Q + x_P + x_R)$
x^1	$A - 2m(y_P - mx_P)$	$x_P x_Q + x_Q x_R + x_P x_R$
x^0	$B - (y_P - mx_P)^2$	$-x_P x_Q x_R$

d'on triem per simplicitat als coeficients de la *x* de segon grau, els igualem i n'aïllem el component *x* del punt resultant:

$$x_R = m^2 - x_P - x_Q$$

Substituint aquesta expressió a la recta L podem trobar el component *y* del punt R',

$$R' = (x_R, mx_R + (y_P - mx_P))$$

I com que R és simètric a R' envers l'eix de les X, tant sols li hem de canviar el signe al component *y* de R':

$$R = (x_R, -(mx_R + (y_P - mx_P)))$$

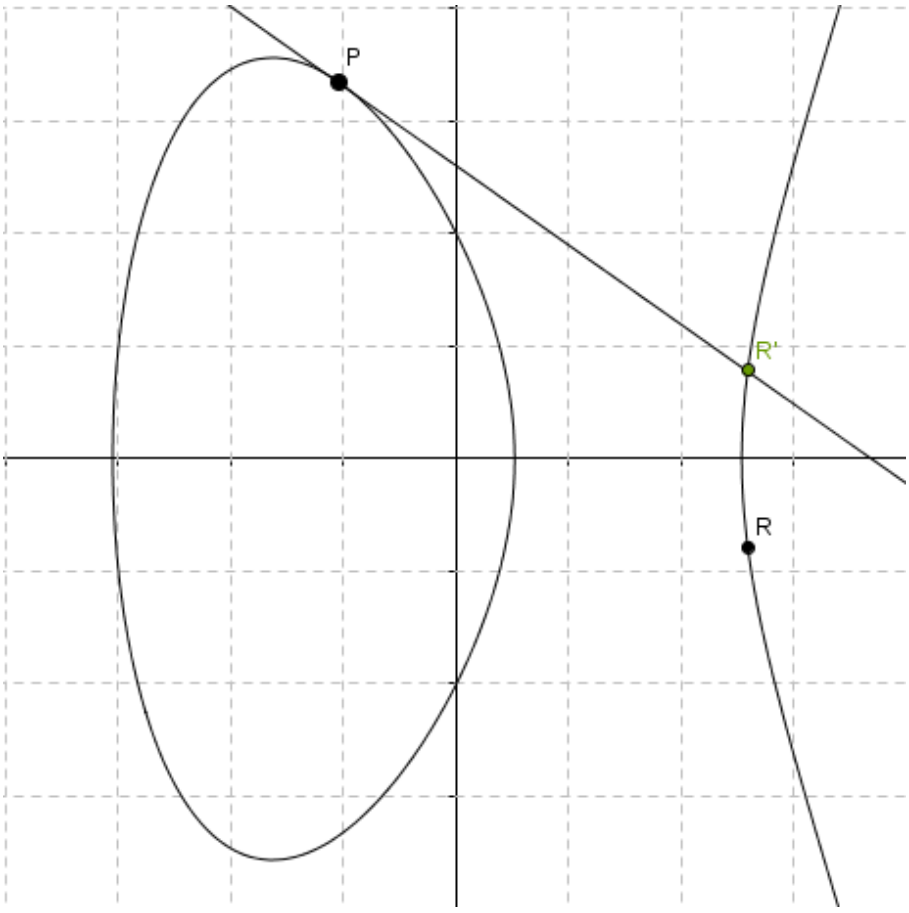
En resum: per a calcular el punt $P \oplus Q = R$, o el que és el mateix, $(x_P, y_P) \oplus (x_Q, y_Q) = (x_R, y_R)$, tant sols hem de calcular:

$$x_R = m^2 - x_P - x_Q$$
$$y_R = -(mx_R + (y_P - mx_P))$$

3.4.2. Segon cas

Operem un punt P amb si mateix. Aquest cop, el punt resultant és el simètric respecte l'eix OX al punt que interseca la corba el·líptica amb la recta tangent a P:

$$P \oplus P = 2P = R$$



Per a trobar la recta L tangent a la corba el·líptica W al punt P, s'ha d'aplicar un mètode anomenat diferenciació implícita a la corba. Primer de tot s'ha d'igualar la corba el·líptica a 0 per a tractar-la com a funció implícita *F(x, y)*:

$$\frac{\mathrm{d}}{\mathrm{d}x}(x^3 + Ax + B - y^2 = 0)$$

La derivada de la suma de funcions és la suma de les seves derivades, és a dir:

$$\frac{\mathrm{d}}{\mathrm{d}x}(x^3) + \frac{\mathrm{d}}{\mathrm{d}x}(Ax) + \frac{\mathrm{d}}{\mathrm{d}x}(B) - \frac{\mathrm{d}}{\mathrm{d}x}(y^2) = \frac{\mathrm{d}}{\mathrm{d}x}(0)$$

aplicant regles de diferenciació bàsiques:

$$3x^2 + A - \frac{\mathrm{d}}{\mathrm{d}x}(y^2) = 0$$

on la derivada d'y^2 respecte *x* ve de la regla de la cadena: dos cops *y* multiplicat per la derivada de *y* respecte de *x*.

$$3x^2 + A - 2y\frac{\mathrm{d}y}{\mathrm{d}x} = 0$$

Reorganitzant els termes tenint en compte que d*y*/d*x* és el pendent *m* de la recta L tangent a W:

$$m = \frac{\mathrm{d}y}{\mathrm{d}x} = \frac{3x^2 + A}{2y}$$

i, com en el cas anterior, la recta és,

$$L: \ y = mx + (y_P - mx_P)$$

Per a trobar els punts on L interseca a una corba el·líptica W, hem de resoldre el sistema d'equacions

$$\begin{cases} W : y^2 = x^3 + Ax + B \\ L : y = mx + (y_P - mx_P) \end{cases}$$

i substituint la part dreta de L a la *y* de W, reorganitzant i igualant a zero:

$$W \cap L : x^3 - m^2x^2 + (A - 2m(y_P - mx_P))x + B - (y_P - mx_P)^2 = 0$$

Ara sabem que $W \cap L$ té dues arrels, i una és doble, perquè L és tangent a W al punt P:

$$W \cap L : (x - x_P)^2(x - x_R) = 0$$

i això s'arregla a:

$$W \cap L : x^3 - (x_R + 2x_P)x^2 + (x_P^2 + 2x_Px_R)x - x_P^2x_R = 0$$

fent una taula com abans:

	coeficients polinomi del sistema	coeficients polinomi de les arrels
x^3	1	1
x^2	$-m^2$	$-x_R - 2x_P$

x^1	$A - 2m(y_P - mx_P)$	$x_P^2 + 2x_P x_R$
x^0	$B - (y_P - mx_P)^2$	$-x_P^2 x_R$

d'on un altre cop igualem els coeficients de les *x* de segon grau i n'aïllem el component *x* del punt resultant

$$x_R = m^2 - 2x_P = m^2 - x_P - x_P$$

Substituint això a la recta L podem trobar el component *y* del punt R',

$$R' = (x_R, mx_R + (y_P - mx_P))$$

I com que R és simètric a R' envers l'eix d'abscisses, tant sols li hem de canviar el signe al component *y* de R':

$$R = (x_R, -(mx_R + (y_P - mx_P)))$$

En resum: per a calcular el punt $P \oplus P = R$, o el que és el mateix, $(x_P, y_P) \oplus (x_P, y_P) = (x_R, y_R)$, tant sols s'ha de calcular:

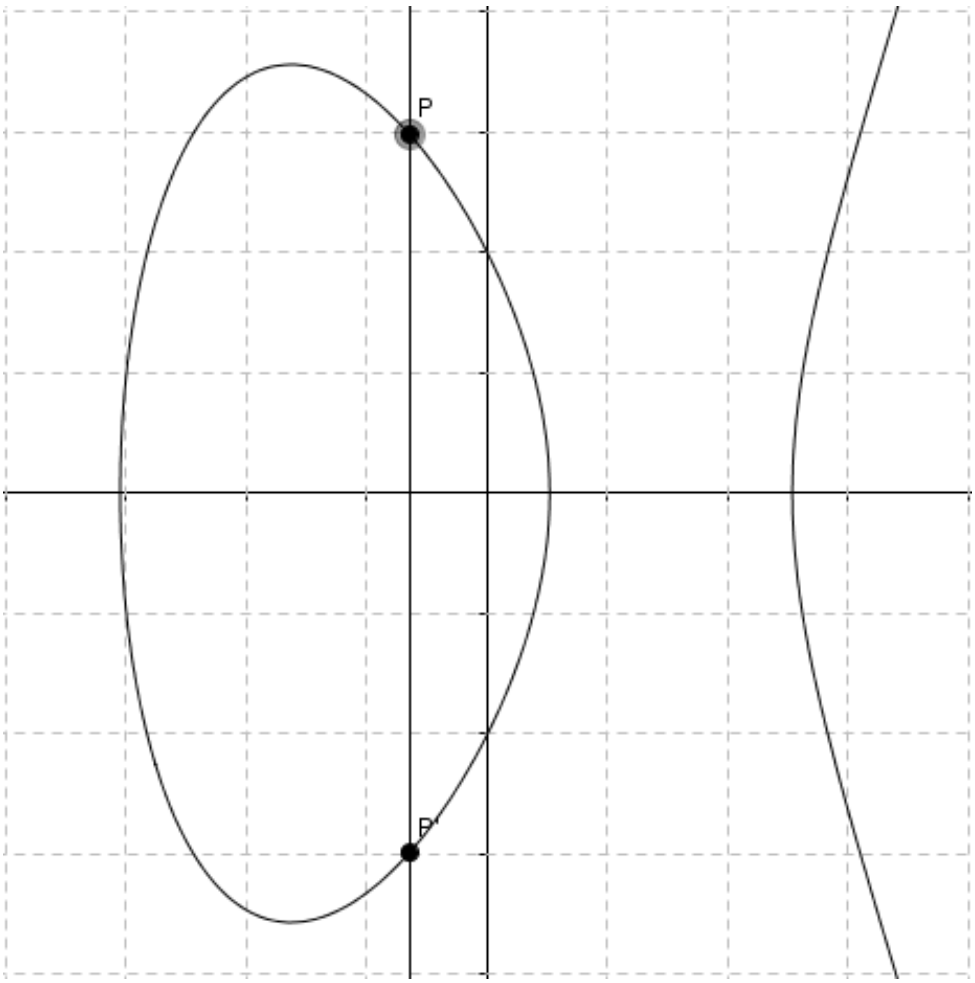
$$x_R = m^2 - 2x_P$$
$$y_R = -(mx_R + (y_P - mx_P))$$

On l'únic que canvia respecte el cas anterior és *m*.

3.4.3. Tercer cas

L'últim cas de la Llei de Grup té a veure amb la suma de dos punts simètrics, P i P'.

$$P \oplus P' = \mathcal{O}$$



Com que la línia L que passa per P i P' no torna a tocar a la corba el·líptica, la operació P⊕P' no resulta en un punt de la corba el·líptica W. Per això, diem que tota línia vertical té un punt $\mathcal{O}$, que anomenem punt a l'infinit, i així E queda com a un grup definit sota un operació i netament tancat: E és tots els punts de la corba el·líptica W i a més el punt a l'infinit $\mathcal{O}$.

Podem demostrar que (E, ⊕) és un grup abelià, perquè compleix les quatre condicions:

- L'element neutre és el punt a l'infinit, $\mathcal{O}$, perquè $P \oplus \mathcal{O} = P$.

- Cada element té el seu oposat, que correspon al seu punt simètric respecte l'eix OX, i es compleix que $P \oplus P' = \mathcal{O}$.
- L'associativitat $(P \oplus Q) \oplus R = P \oplus (Q \oplus R)$ està demostrada a l'annex perquè no és trivial.
- La commutativitat $P \oplus Q = Q \oplus P$ és bastant òbvia ja que la línia que interseca dos punts, que utilitzem per a trobar un tercer punt R', no ho fa en cap ordre determinat.

3.5. Corbes El·líptiques sobre grups d'enters mod *n*

Si un grup d'una corba el·líptica

$$E = \left\{(x, y) : y^2 = x^3 + Ax + B\right\} \cup \{\mathcal{O}\}$$

té coeficients *A* i *B* que pertanyin a $\mathbb{Z}/n\mathbb{Z}$, podem crear un subgrup d'*E*

$$E(\mathbb{Z}/n\mathbb{Z}) = \{(x,y) \in E \mid x,y \in \mathbb{Z}/n\mathbb{Z}\} \cup \{\mathcal{O}\}$$

que són tots els punts d'*E* amb coordenades que pertanyen a $\mathbb{Z}/n\mathbb{Z}$. En aquest grup, els punts els calculem no amb la igualtat $y^2 = x^3 + Ax + B$, sinó amb

$$y^2 \equiv x^3 + Ax + B \pmod{n}$$

que és una congruència. Aquesta expressió es pot llegir com que els residus de les divisions de la part esquerra entre *n* i de la part dreta entre *n* són iguals Com que està limitada mod *n*, totes les operacions que fem, per llargues i complicades que siguin, van sent reduïdes mod *n*. Com que qualsevol nombre mod *n* serà un nombre entre el zero i *n*-1. Per tant, el nombre de punts no és infinit.

$$\left|E(\mathbb{Z}/n\mathbb{Z})\right| \neq \infty$$

En haver-hi un nombre determinat de punts, podem assegurar-nos de que hi ha un nombre *k* determinat pel qual l'operació

$$kP = \underbrace{P \oplus P \cdots P \oplus P}_{k}$$

passarà per tots els punts possibles d' $E(\mathbb{Z}/n\mathbb{Z})$.

4. Algorisme de Factorització per Corbes El·líptiques de Lenstra

És un mètode de factorització d'enters (però no de qualsevol enter) que va publicar el matemàtic neerlandès Hendrik Lenstra Jr (1949) al 126è volum de la revista *Annals of Mathematics* l'any 1987.

L'algorisme, donat un enter positiu *n* diferent d'u, dos o tres, intenta factoritzar-lo probabilísticament.
El principi d'aquest algorisme és passejar-se per tants nombres relacionats amb *n* com sigui possible, i mirar ràpidament si algun d'aquests és un factor no trivial de *n*.
Per a fer-ho, s'escull una corba el·líptica amb *A* i *B* pertanyents a $\mathbb{Z}/n\mathbb{Z}$ i amb el discriminant $4A^3 + 27B^2 \in (\mathbb{Z}/n\mathbb{Z})^*$.

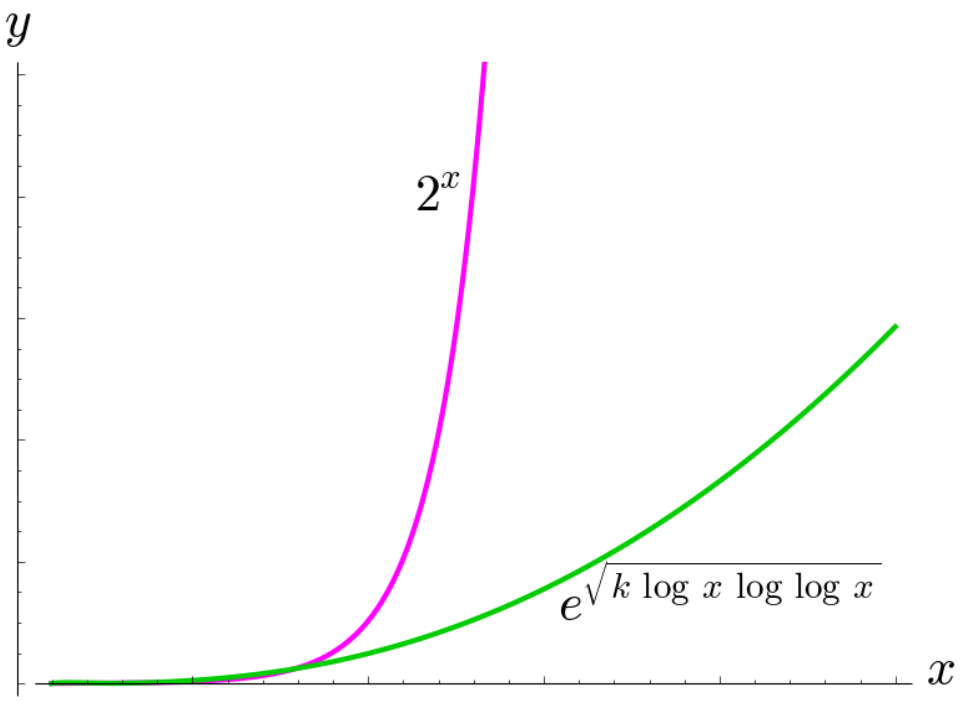
Després, es multiplica un punt P de la corba el·líptica escollida per un escalar considerablement gran *k*, perquè en fer-ho, es suma aquest punt a si mateix fins a passar per tants punts d' $E(\mathbb{Z}/n\mathbb{Z})$ com sigui possible. Com més gran sigui aquest escalar *k*, més fiable serà el resultat de l'algorisme, però alhora serà més lent. Per això és una prova de primalitat probabilística.
Quan calcula el punt resultant de la Llei de Grup, l'algorisme ha de fer operacions mod *n*. Per a trobar el pendent de la recta que passa pels dos punts que opera la Llei de Grup, una de les operacions que s'han de fer és la divisió, com ja hem vist.

Per a fer un quocient mod *n*, s'ha de calcular la inversa multiplicativa del denominador. Això es fa resolent la Identitat de Bézout. Si el màxim comú divisor de *n* i el denominador és major d'1, o dit d'una altra manera, si el denominador i *n* no són coprimers, significa que el màxim comú divisor divideix a *n* i per tant n'és un factor. Aleshores, es divideix *n* entre aquest nombre i es continua fins que s'acabin totes les operacions de Llei de Grup necessàries per a completar la multiplicació *k*P. En el cas de que *n* sigui un nombre primer, totes les operacions de divisió tenen resultat i per tant l'algorisme només troba el punt resultant de *k*P.

Aquest algorisme aconsegueix el seu objectiu en un temps subexponencial de

$$O\left(e^{\sqrt{(2+o(1))\log p \log\log p}}\right)$$

per a una entrada *N*, on *p* és el més petit dels factors de *N*. Per tant, com més petits siguin els factors de *N*, més ràpid serà l'algorisme. Sorprenentment, el més important no és com de gros sigui *N*. La notació "*o(1)*" significa "una funció que té límit zero quan el seu argument creix". Això vol dir que com més gran sigui *N* més petit serà el sumand *o(1)*. Aquí es pot veure clarament que és un temps subexponencial per a el pitjor cas per a aquest algorisme, quan *n* no té factors.



5. Implementació de l'algorisme en C++

Com a part pràctica d'aquest treball, he implementat l'algorisme de Factorització per Corbes El·líptiques de Lenstra en el llenguatge de programació C++. Ho he compilat en un executable adjunt a aquest treball o disponible en línia a www.bit.ly/LENSTRAEC.

El codi que he escrit s'allarga cent setanta-nou[3] línies i és aquest:

```
#include <iostream>
#include <vector>
#include <cmath>
#include <algorithm>
using namespace std;
struct punt{
    int x;
    int y;
    int z;
};
bool operator ==(const punt& A, const punt& B){
    return (A.x == B.x) && (A.y == B.y) && (A.z == B.z);
}
int mcd(int a, int b){
    if (b == 0){
        return a;
    }
    if (a == 0){
        return b;
    }
    else{
        return mcd(b, a%b);
    }
}
long long mcmdelsprimers(int n){
    long long P = 1;
    vector<int> Y;
    vector<bool> T(n + 1, true);
    for (int i = 2; i < n + 1; ++i){
        if (T[i]){
            Y.push_back(i);
            for (int j = i*i; j < n + 1; j += i){
                T[j] = false;
            }
        }
    }
    for (int q = 0; q < Y.size(); ++q){
        P *= pow(Y[q], floor(log(n) / log(Y[q])));
    }
    return P;
}
vector<int> bezout(int a, int b){
    vector<int> bez(3);
    int q, r, x, y, g;
    if (b == 0){
        bez[0] = 1, bez[1] = 0, bez[2] = 0;
        return bez;
```

[3] 179 és, "casualment", un nombre primer.

```
48      }
49      q = floor(a / b);
50      r = a % b;
51      vector<int> k = bezout(b, r);
52      x = k[0];
53      y = k[1];
54      g = k[2];
55      bez[0] = y;
56      bez[1] = x - q*y;
57      bez[2] = mcd(a, b);
58      return bez;
59  }
60  punt suma(punt P, punt Q, int A, int& N, vector<int>& L){
61      int m;
62      int num, den;
63      int invden;
64      punt R;
65      R.x = 0;
66      R.y = 0;
67      R.z = 0;
68      if (((P.x == Q.x) && (P.y != Q.y)) || ((P.y == 0) && (Q == P))){
69          R.z = 1;
70          return R;
71      }
72      if (!(P == Q)){
73          num = (Q.y - P.y) % N;
74          den = (Q.x - P.x) % N;
75      }
76      if (P == Q){
77          num = (3 * P.x*P.x + A) % N;
78          den = (2 * P.y) % N;
79      }
80      if (den < 0){
81          while (den < 0){
82              den += N;
83          }
84          den = den % N;
85      }
86      vector<int> j = bezout(den, N);
87      if (j[2] > 1){
88          if (j[2] != N && j[2] != 1){
89              N /= j[2];
90              L.push_back(j[2]);
91          }
92      }
93      invden = j[0];
94      m = num*invden;
95      R.x = (m*m - P.x - Q.x) % N;
96      R.y = - (m*R.x + (P.y - m*P.x)) % N;
97      return R;
98  }
99  punt multiplica(long long k, punt P, int A, int& N, vector<int>& L){
100     punt R;
101     R.x = 0;
102     R.y = 0;
103     R.z = 0;
104     punt Q = P;
105     while (k > 0){
106         if (k % 2 == 1){
107             R = suma(R, Q, A, N, L);
108         }
109         R = suma(Q, Q, A, N, L);
110         k = floor(k / 2);
111     }
112     return R;
```

```
}
int aleat(int N){
    int A = rand() % N;
    if (mcd(((4 * A*A*A + 27) % N), N) == 1){
        return A;
    }
    else{
        aleat(N);
    }
}
vector<int> Lenstra(int N){
    int limit = 42;
    vector<int> factors;
    punt P, R;
    P.x = 0;
    P.y = 1;
    P.z = 0;
    int A = aleat(N);
    if (round(exp(sqrt(2 * log(N)*log(log(N))) / 2)) < 42){
        limit = round(exp(sqrt(2 * log(N)*log(log(N))) / 2));
    }
    R = multiplica(mcmdelsprimers(limit), P, A, N, factors);
    cout << mcmdelsprimers(limit) << "P = (" << R.x << ", " << R.y << ", "
<< R.z << ")" << endl;
    factors.push_back(N);
    factors.erase(remove(factors.begin(), factors.end(), 1), factors.end());
    sort(factors.begin(), factors.end());
    return factors;
}
int main(){
    int entradaoriginal;
    int N;
    while (cin >> entradaoriginal){
        vector<int> L;
        vector<int> U;
        N = entradaoriginal;
        while (N >= 2 && (N % 2 == 0 || N % 3 == 0 || N ==
round(sqrt(N))*round(sqrt(N)))){
            if (N % 2 == 0){
                N /= 2;
                U.push_back(2);
            }
            else if (N % 3 == 0){
                N /= 3;
                U.push_back(3);
            }
            else if (N == round(sqrt(N))*round(sqrt(N))){
                N = sqrt(N);
                U.push_back(N);
            }
        }
        if (N>3){
                  L = Lenstra(N);
        }
        L.insert(L.begin(), U.begin(), U.end());
        sort(L.begin(), L.end());
        for (int j = 0; j < L.size(); ++j){
            if (j == 0){
                cout << entradaoriginal << " = ";
            }
            if (j == L.size() - 1){
                cout << L[j];
            }
            else{
                cout << L[j] << "*";
```

```
175                 }
176             }
177             cout << endl << endl;
178         }
179     }
```

Explicaré el codi part per part per a que s'entengui clarament què passa a cada part de l'algoritme.

5.1. Declaracions inicials

Les cinc primeres línies

```
1  #include <iostream>
2  #include <vector>
3  #include <cmath>
4  #include <algorithm>
5  using namespace std;
```

simplement inclouen les llibreries *iostream*, encarregada de l'entrada i sortida de dades del programa, *vector*, per a poder utilitzar contenidors de dades, *cmath* per les funcions matemàtiques no bàsiques com ara potències, arrodoniments i logaritmes, i finalment *algorithm* que inclou una funció que ordena els elements d'un contenidor de dades. *Using namespace std* a l'encapçalament del programa indica que les funcions que utilitzo són std::funció, i així no s'ha de repetir cada cop.

5.2. Punts tridimensionals

Les línies de la 6 a la 13

```
6   struct punt{
7       int x;
8       int y;
9       int z;
10  };
11  bool operator ==(const punt& A, const punt& B){
12      return (A.x == B.x) && (A.y == B.y) && (A.z == B.z);
13  }
```

serveixen per a crear punts cartesians de tres components *x*, *y* i *z*, i per a ensenyar al C++ com comparar-los: dos punts són iguals si els components *x*, *y* i *z* d'un són iguals als de l'altre. Com que "són iguals" o "no són iguals" és un valor booleà, la funció és de tipus *bool*.

5.3. Màxim comú divisor

La primera funció de totes és el màxim comú divisor de dos nombres:

```
14  int mcd(int a, int b){
15      if (b == 0){
16          return a;
17      }
18      if (a == 0){
19          return b;
20      }
21      else{
22          return mcd(b, a%b);
23      }
24  }
```

int al principi de tot indica que la funció *mcd* de dos nombres enters *a* i *b* és un enter també. He utilitzat l'algorisme euclidià del màxim comú denominador perquè és bastant ràpid i simple, i tot i ser recursiu no crec que causi cap problema de sobre-càrrega de memòria. Si tots dos arguments *a* i *b* de la funció són positius, el resultat del màxim comú divisor és també el màxim comú divisor de *b* i *a* mod *b*. La demostració d'això és als annexos. En C++ i molts altres llenguatges de programació, "mod" s'expressa amb el símbol del percentatge, és a dir, *a* mod *b* equival a *a*%*b*.

5.4. Mínim comú múltiple dels enters positius fins a *n*

Després hi descric la funció

```
25  long long mcmdelsprimers(int n){
26      long long P = 1;
27      vector<int> Y;
28      vector<bool> T(n + 1, true);
29      for (int i = 2; i < n + 1; ++i){
30          if (T[i]){
31              Y.push_back(i);
32              for (int j = i*i; j < n + 1; j += i){
33                  T[j] = false;
34              }
35          }
36      }
37      for (int q = 0; q < Y.size(); ++q){
38          P *= pow(Y[q], floor(log(n) / log(Y[q])));
39      }
40      return P;
41  }
```

Que donat un nombre, aquí anomenat *n*, retorna el mínim comú múltiple de tots els enters positius i més petits que *n*. La funció és de tipus *long long*, és a dir, que retorna un nombre de 64 bits. Les variables de tipus *int* són de 16 bits, i poden ser nombres més petits que 65 536. En canvi, els *long long* poden ser nombres més

petits que 18 446 744 073 709 551 616. Escullo *long long* per aquesta funció perquè els seus valors creixen molt quan *n* creix.

Primer de tot, amb el Sedàs d'Erastòtenes, es genera una llista *Y* de tots els nombres primers més petits que *n*. Hi ha mètodes una mica més ràpids que el Sedàs d'Erastòtenes, com el Sedàs d'Atkin, però com que en aquesta implementació *n* no és mai prou gran com per a que es noti la diferència de rendiment, m'he decidit pel Sedàs d'Erastòtenes, ja que és l'únic dels dos que entenc com funciona.

El Sedàs d'Erastòtenes consisteix en fer una llista *T* de valors booleans inicialitzats a "veritat", un per a cada enter més petit que *n*. Llavors, si un element d'aquesta llista és veritat, s'afegeix l'enter que aquest element representa a la llista *Y*, i es marquen tots els seus múltiples com a "mentida" a la llista *T* (línies 30 a 35). En acabar, a *Y* només hi queden els nombres que no són múltiples de cap nombre més petit que ells mateixos a part de l'u: els nombres primers.

Aleshores, el mínim comú múltiple de tots els enters fins a *n* és el producte de tots els primers diferents entre ells més petits que *n* que hi ha a *Y* elevats a algun exponent, i que $y_q^{r_q} \leq n < y_q^{r_q+1}$, cada exponent serà $r_q = \left\lfloor \log_{y_q} n \right\rfloor$. Per tant, el mínim comú múltiple és equivalent a:

$$\prod_{y_q \in Y} y_q^{\left\lfloor \log_{y_q} n \right\rfloor} = \prod_{y_q \in Y} y_q^{\left\lfloor \frac{\log n}{\log y_q} \right\rfloor}$$

que és justament el que fa el bucle *for* de les línies 37 a la 39, iterant des de *q* = 0 fins a l'últim element de *Y*. La funció *pow(arg1, arg2)* bàsicament multiplica *arg1* per si mateix *arg2* cops.

5.5. Identitat de Bézout

La funció

```
42  vector<int> bezout(int a, int b){
43      vector<int> bez(3);
44      int q, r, x, y, g;
45      if (b == 0){
46          bez[0] = 1, bez[1] = 0, bez[2] = 0;
47          return bez;
48      }
49      q = floor(a / b);
50      r = a % b;
51      vector<int> k = bezout(b, r);
52      x = k[0];
53      y = k[1];
54      g = k[2];
55      bez[0] = y;
56      bez[1] = x - q*y;
57      bez[2] = mcd(a, b);
58      return bez;
59  }
```

serveix per a resoldre la identitat de Bézout *xa+yb* = mcd(*a*, *b*): donats dos nombres *a* i *b*, troba *x*, *y*, i *mcd(a, b)*. Per a fer-ho aquesta funció agafa com a entrada dos nombres enters (*int*), *a* i *b*, i retorna una llista d'enters (*vector<int>*) amb tres elements: el primer, el coeficient d'*a* per a que es compleixi la identitat, el segon, el coeficient de *b*, i el tercer és el màxim comú divisor d'*a* i *b*. Per a fer-ho, es creen 5 variables (línia 44) que seran útils per a resoldre la identitat, i aleshores es comprova un cas trivial: si *b* és 0, la identitat de Bézout tindrà coeficient 1 per a *a*, coeficient 0 per a *b*, que és 0 (tot i que aquest coeficient podria ser qualsevol nombre perquè qualsevol nombre multiplicat per 0 és 0), i el seu màxim comú denominador serà *a* (línies 45 a 48).

Aleshores es calcula *q*, que és la divisió d'*a* i *b* arrodonida a l'enter immediatament més petit, i *r*, que és *a* mod *b*. Llavors es busca, recursivament, la funció *bezout(b, r)* que és el mateix que *bezout(b, a* mod *b*). La recursivitat aquí és semblant a l'algorisme per a esbrinar el màxim comú divisor de dos nombres que utilitzo en aquesta implementació de l'algorisme de Lenstra i que està demostrada a l'annex.
La recursivitat duu el problema a resoldre fins al un cas base, també anomenat cas trivial (línies 45 a 48). A partir d'allà, amb les ordres de la línia 52 a la línia 58, es

resol, i s'esbrina la solució del problema inicial aprofitant que per a dos nombres *a* i *b*, es compleix:

$$ya + (x - \left\lfloor \frac{a}{b} \right\rfloor y)b = xb + y(a \bmod b)$$

Per exemple, si *a* = 42 i *b* = 12:

$$42y + 12(x - \left\lfloor \frac{42}{12} \right\rfloor y) = 12x + (42 \bmod 12)y$$
$$42y + 12(x - 3y) = 12x + 6y$$
$$12x + 42y - 36y = 12x + 6y$$
$$12x + 6y = 12x + 6y$$

Per a calcular el tercer element del resultat d'aquesta funció, utilitzo la funció *mcd* definida més amunt en el programa.

5.6. Operar dos punts

Per a operar dos punts d'una corba el·líptica he escrit la funció

```
60  punt suma(punt P, punt Q, int A, int& N, vector<int>& L){
61      int m;
62      int num, den, invden;
63      punt R;
64      R.x = 0;
65      R.y = 0;
66      R.z = 0;
67      if (((P.x == Q.x) && (P.y != Q.y)) || ((P.y == 0) && (Q == P))){
68          R.z = 1;
69          return R;
70      }
71      if (!(P == Q)){
72          num = (Q.y - P.y) % N;
73          den = (Q.x - P.x) % N;
74      }
75      if (P == Q){
76          num = (3 * P.x*P.x + A) % N;
77          den = (2 * P.y) % N;
78      }
79      if (den < 0){
80          while (den < 0){
81              den += N;
```

```
82          }
83          den = den % N;
84      }
85      vector<int> j = bezout(den, N);
86      if (j[2] > 1){
87          if (j[2] != N && j[2] != 1){
88              N /= j[2];
89              L.push_back(j[2]);
90          }
91      }
92      invden = j[0];
93      m = num*invden % N;
94      R.x = (m*m - P.x - Q.x) % N;
95      R.y = - (m*R.x + (P.y - m*P.x)) % N;
96      return R;
97  }
```

Aquesta funció retorna un punt. Com a arguments, té:

- Un punt *P* d'una corba el·líptica determinada
- Un punt *Q* d'una corba el·líptica determinada
- Un nombre enter *A*, que és el primer coeficient de la corba el·líptica escollida on es fa la operació $P \oplus Q$
- Un nombre enter *N* en el que la funció *punt()* hi té dret a canvi (*int&* significa que és un enter que es pot modificar), que determina el conjunt dels enters mod *N* sobre el qual està definit el grup de la corba el·líptica
- Un contenidor d'enters (*vector<int>&,* on & significa que es pot canviar) *L*, on la funció anirà posant els factors no trivials del nombre *N* a factoritzar.

Llavors a la funció s'hi declaren les variables *m*, *num*, *den*, i *invden*, que estan relacionades així:

$$m = \frac{num}{den} = num \times invden$$

i també s'hi declara un punt *R*, el punt resultant, amb les tres coordenades cartesianes igualades a zero.

Aleshores, el condicional de les línies 67 a 70, busca si els dos punts a operar tenen la mateixa coordenada *x* i diferent coordenada *y* (és a dir, són punts oposats) o si els dos punts són el mateix i la seva coordenada *y* és zero. En termes booleans:

$$((x_P = x_Q) \wedge (y_P \neq y_Q)) \vee ((y_P = 0) \wedge (P = Q))$$

En tots dos casos, la línia que passa pels punts és una línia vertical, i per tant, el punt resultant de l'operació és el punt a l'infinit. Això és el tercer cas de l'operació que converteix els punts d'una corba el·líptica en un grup.

El punt a l'infinit $\mathcal{O}$, en la meva implementació de l'algorisme, és representat com a un punt tridimensional que està fora del pla de la corba el·líptica. El punt a l'infinit és $\mathcal{O} = (0, 0, 1)$. Per això a la línia 68 s'assigna el component *z* del punt resultant *R* a 1, i es retorna aquest punt, acabant la funció.

El següent cas que es busca, en el condicional de les línies 71 a 74, és el primer cas possible, en el que el punt P i el punt Q són diferents entre ells. Si es dóna aquest cas, el pendent *m* de la recta que passa pels dos punts serà

$$m = \frac{y_Q - y_P}{x_Q - x_P}$$

i per això *num* passa a ser la diferència dels components *y* dels punts, i *den* la diferència dels components *x* dels punts. Totes les operacions aritmètiques en aquesta funció es duen a terme mod *N* perquè la corba el·líptica és sobre els enters mod *N*.

L'únic cas possible que roman és quan els punts *P* i *Q* són el mateix punt (línies 75 a 78). El pendent *m* serà

$$m = \frac{3x_P^2 + A}{2y_P}$$

on *num* és el numerador i *den* el denominador.

A continuació, de la línia 79 a la 84, s'arreglen els casos en els que el denominador *den* pot sortir negatiu, i amb un bucle *while* se li suma *N* fins que *den* deixi de ser negatiu, perquè així tindrà la mateixa equivalència modular.

Aleshores es crea un contenidor de nombres enters *j*, que és el resultat de la funció *bezout()* declarada a les línies de la 42 a la 59, prenent com a arguments *den* i *N*. Això és per a trobar el valor de la inversa del denominador *den*.

Si el màxim comú divisor del denominador *den* i de *N*, que és el tercer element de *j* (és a dir, *j[2]*), és major que 1 i no és *N*, significa que s'ha trobat un divisor no trivial del nombre que es vol factoritzar. Aleshores, s'assigna *N* a la divisió de *N* entre aquest divisor *j[2]*, i s'afegeix a la llista *L* dels factors de *N* (línies 86 a 91).

La inversa del denominador *den* és el primer nombre de la llista *j*, és a dir, *j[0]*. A la línia 92 s'assigna *invden = j[0]*.
Finalment, es calculen els components *x* i *y* del punt resultant R utilitzant

$$x_R = (m^2 - x_P - x_Q) \bmod N$$
$$y_R = -(mx_R + (y_P - mx_P)) \bmod N$$

i es retorna el punt R (línies 94 a 96).

5.7. Dobla-i-suma

Per a augmentar les possibilitats de trobar factors no trivials de *N* operant punts d'una corba el·líptica, el mètode de Lenstra computa un nombre *k* molt gran i el "multiplica" per un punt de la corba, *P*.

$$kP = \underbrace{P \oplus P \cdots P \oplus P}_{k}$$

Aquesta operació és bastant pesada: per a calcular *kP* es necessiten *k* operacions.

En el mètode dobla-i-suma, s'expressa *k* com a una suma de potències de dos (qualsevol nombre positiu es pot expressar així, demostració als annexos). Per exemple, si *k* = 107

$$107 = 2^0 + 2^1 + 2^3 + 2^5 + 2^6 = 1 + 2^1 + 2^3 + 2^5 + 2^6$$

aleshores, 107*P* és:

$$107P = 1P + 2^1P + 2^3P + 2^5P + 2^6P$$

on hem de doblar *P* quinze (1+3+5+6) cops i sumar 4 punts: en total, 19 operacions, en comptes de 107. A la pràctica s'ha trobat que de mitjana, amb aquest mètode, es necessiten $\frac{3}{2}\log_2 k$ operacions.

La implementació que he fet d'aquest mètode és la funció

```
98  punt multiplica(long long k, punt P, int A, int& N, vector<int>& L){
99      punt R;
100     R.x = 0;
101     R.y = 0;
102     R.z = 0;
103     punt Q = P;
104     while (k > 0){
105         if (k % 2 == 1){
106             R = suma(R, Q, A, N, L);
107         }
108         Q = suma(Q, Q, A, N, L);
109         k = floor(k / 2);
110     }
111     return R;
112 }
```

que pren com a arguments:

- Un nombre de 64 bits *k*, que és el nombre pel qual es "multiplica"
- Un punt *P* d'una corba el·líptica determinada
- Un nombre enter *A*, que és el primer coeficient de la corba el·líptica escollida on es fa la operació *kP*

- Un nombre enter *N* en el que la funció *multiplica()* hi té dret a canvi que determina el conjunt dels enters mod *N* sobre el qual està definit el grup de la corba el·líptica
- Un contenidor d'enters *L*

Primer de tot es declara *R*, el punt resultant, iniciat a (0, 0, 0). També assigna un punt *Q* a *P*. El bucle *while* de les línies 104 a la 110, mentre *k* sigui més gran que zero, busca si és imparell, si ho és crida a la funció *suma()* per a què *R* passi a ser el resultat d'operar *Q* i *R*, i després assigna *Q* al seu doble, i finalment divideix *k* entre dos, arrodonint cap avall. Quan acaba, és a dir, quan *k* és 1, retorna el punt *R*.

5.8. El coeficient *A* de la Corba El·líptica

La funció

```
113  int aleat(int N){
114      int A = rand() % N;
115      if (mcd(((4 * A*A*A + 27) % N), N) == 1){
116          return A;
117      }
118      else{
119          aleat(N);
120      }
121  }
```

pren com a argument un enter *N* que determina el conjunt dels enters mod *N* sobre el qual està definit el grup de la corba el·líptica i retorna un enter *A* que és el coeficient decisiu de la corba $y^2 = x^3 + Ax + 1$, amb *B* = 1 per simplicitat. Recordem que perquè la corba el·líptica no tingui irregularitats, hem d'imposar la condició

$$4A^3 + 27B^2 \neq 0$$

i que per a que l'algorisme funcioni, aquest discriminant ha de ser un membre del conjunt $(\mathbb{Z}/N\mathbb{Z})^*$.

Per això, a la línia 114 s'assigna *A* a un nombre aleatori més petit que *N*, i al condicional de les línies 115 a 117, es mira si el discriminant és coprimer a *N*. Si ho és, aquesta *A* és vàlida, i la funció la retorna com a resultat. Si no, la funció es crida

a si mateixa un altre cop. Ara *A* serà algun altre nombre aleatori, i es repeteix el procés.

5.9. Trobar els factors de *N*

La darrera funció secundària és

```
122  vector<int> Lenstra(int N){
123      int limit = 42;
124      vector<int> factors;
125      punt P, R;
126      P.x = 0;
127      P.y = 1;
128      P.z = 0;
129      int A = aleat(N);
130      if (round(exp(sqrt(2 * log(N)*log(log(N))) / 2)) < 42){
131          limit = round(exp(sqrt(2 * log(N)*log(log(N))) / 2));
132      }
133      R = multiplica(mcmdelsprimers(limit), P, A, N, factors);
134      cout << mcmdelsprimers(limit) << "P = (" << R.x << ", " << R.y << ", "
     << R.z << ")" << endl;
135      factors.push_back(N);
136      factors.erase(remove(factors.begin(), factors.end(), 1),
     factors.end());
137      sort(factors.begin(), factors.end());
138      return factors;
139  }
```

Que pren com a argument un enter positiu *N* (que ha de ser més gran que 3), i retorna una llista d'enters. Crea una variable *limit* que és el nombre del qual se'n traurà el mínim comú múltiple de tots els nombres entre *limit* i 1, aquests inclosos. Es creen dos punts, *P* i *R*, i *P* s'inicialitza a (0, 1, 0). Després es declara un enter *A* que és el coeficient decisiu de la corba el·líptica $y^2 = x^3 + Ax + 1$, utilitzant la funció anterior, *aleat(N)*.

Hendrik Lenstra, en la seva publicació original, ofereix una fórmula complicada per a determinar quin és el valor adient de *limit* per a cada *N*. Aquesta fórmula és

$$limit = e^{\frac{\sqrt{2\ \ln N\ \ln\ln N}}{2}}$$

però vaig trobar que la funció *mcmdelsprimers(limit)* no funcionava per a *limit* > 42 perquè el resultat és massa gran per a C++. Per això de la línia 130 a la 132, busco amb un condicional si el *limit* donat per la fórmula és menor que 42.

Si és el cas, *limit* tindrà el valor que resulti de la fórmula, que serà òptim per a *N*. En cas contrari, *limit* = 42.

Després es computa *R* = *kP* amb *k* = *mcmdelsprimers(limit)* a la línia 133, i a la funció *multiplica* se li dona la llista *factors* per a què hi guardi tots els factors no trivials de *N* trobats aplicant la Llei de Grup. El punt resultant *R* no és rellevant, l'únic que ens interessa és el contingut de la llista *factors*.

A la línia 134 utilitzant *cout* el programa ensenya a l'usuari quina multiplicació *kP* ha realitzat i qui n'és el resultat. Ho fa mostrant el valor de *mcmdelsprimers(limit)* seguit dels caràcters "P = (", després els components *x*, *y* i *z* del punt resultant *R*, i finalment ")". L'ordre *endl* li diu a C++ que acabi la línia de text.

Tot seguit, a la llista *factors* s'hi afegeix *N*, que si tot ha anat bé ara serà l'últim factor no trivial del valor de *N* que se li ha donat a la funció *multiplica* a la línia 133. A la línia 136 s'esborren tots els uns que hi hagi a la llista *factors* perquè l'1 no és un nombre primer. Finalment la funció *sort* ordena els elements de la llista *factors* de més gran a més petit i es retorna aquesta llista com a resultat de la funció *Lenstra(N)*.

5.10. Funció principal del programa

El *main()* de la meva implementació és

```
140  int main(){
141      int entradaoriginal;
142      int N;
143      while (cin >> entradaoriginal){
144          vector<int> L;
145          vector<int> U;
146          N = entradaoriginal;
147          while (N >= 2 && (N % 2 == 0 || N % 3 == 0 || N ==
     round(sqrt(N))*round(sqrt(N)))){
148              if (N % 2 == 0){
149                  N /= 2;
150                  U.push_back(2);
151              }
152              else if (N % 3 == 0){
153                  N /= 3;
154                  U.push_back(3);
155              }
156              else if (N == round(sqrt(N))*round(sqrt(N))){
157                  N = sqrt(N);
```

```
158                 U.push_back(N);
159             }
160         }
161         if (N>3){
162                     L = Lenstra(N);
163         }
164         L.insert(L.begin(), U.begin(), U.end());
165         sort(L.begin(), L.end());
166         for (int j = 0; j < L.size(); ++j){
167             if (j == 0){
168                 cout << entradaoriginal << " = ";
169             }
170             if (j == L.size() - 1){
171                 cout << L[j];
172             }
173             else{
174                 cout << L[j] << "*";
175             }
176         }
177         cout << endl << endl;
178     }
179 }
```

Primer de tot es creen dues variables, totes dues nombres enters (*int*), anomenats *entradaoriginal* i *N*. Aleshores comença el bucle *while* que continuarà mentre l'usuari entri nombres. Si l'usuari entrés una lletra o una paraula, el bucle es trencaria i el programa acabaria.

Quan entra un nombre, es creen dues llistes de nombres, *L* i *U*, i s'assigna *N* al valor d'*entradaoriginal*. Així es guarda el valor original, perquè *N* serà modificat al llarg del programa. Després hi ha un altre bucle *while* per a trobar els factors més bàsics: això augmenta bastant la velocitat del programa, ja que la meitat dels nombres enters són parells, per exemple. La forma booleana de l'expressió que condiciona aquest segon bucle és

$$(N \geq 2) \wedge (N \geq 3) \wedge ((N \bmod 2 = 0) \vee (N \bmod 3 = 0) \vee (N = [\sqrt{N}][\sqrt{N}]))$$

on *[x]* significa *x* arrodonida a l'enter més proper.

El condicional de les línies 148 a 151 busca si el nombre *N* és un múltiple de dos (és congruent a zero mòdul dos) i si ho és, *N* passa a ser *N*/2, i com que el dos n'era un divisor, és afegit a la llista *U*. El mateix passa amb el nombre 3 a les línies 152 a 155.

Al tercer condicional (línies 156 a 159) busca si *N* és un quadrat perfecte (com 4, 9, 16, 25, 36, 49, 64, 81, etcètera) i si ho és *N* passa a ser la seva arrel quadrada i s'afegeix a la llista *U*.

Després de tots aquests casos senzills, si *N* queda més gran que 3, s'hi aplica el mètode de factorització per corbes el·líptiques de Lenstra amb la funció *Lenstra(N)* i els factors no trivials que trobi aquest mètode són guardats a la llista d'enters *L*. Després a la línia 164 s'uneixen les llistes *U* (factors trobats amb els casos senzills) i *L* (factors trobats amb el mètode de Lenstra) a la llista *L*, i a la 165 s'ordenen els elements de més petit a més gran.

Les línies de la 166 a la 176 s'encarreguen de mostrar la factorització del nombre que ha entrat l'usuari, *entradaoriginal*, que serà el producte de tots els elements de la llista *L*. Ho he fet de tal manera que la sortida sigui "*entradaoriginal = L[0] * L[1] * L[2]*...", i per últim, la la línia 177 s'encarrega que hi hagi dues línies de separació entre aquest resultat i l'entrada següent.

6. Un exemple: 2001

Per a millorar la comprensió del funcionament de l'Algorisme de Factorització per Corbes El·líptiques de Lenstra, utilitzarem un exemple.
Descompondrem el nombre 2001 en els seus factors, seguint el procediment de l'algorisme però simplificant les parts auxiliars, com ara trobar el màxim comú divisor de dos nombres, trobar els coeficients de la Igualtat de Bézout o buscar per a quin nombre s'ha de "multiplicar" el punt inicial d'una corba el·líptica modular.
En aquest apartat, cada cop que trobem un factor de 2001, serà ressaltat en negreta.

Primer de tot mirem si 2001 és parell, múltiple de 3 o un quadrat perfecte. Com que acaba en 1 no és parell, però la suma dels seus dígits és igual a 3 (2+0+0+1 = 3),

per tant el **3** n'és un divisor. 2001 entre 3 és igual a 667, i tornem a fer el mateix que havíem fet amb 2001.

El nombre 667 no és parell perquè acaba en 7, i no és un múltiple de 3 perquè 6 + 6 + 7 = 19, que tampoc ho és. L'arrel quadrada de 667 és 25'8263…, per tant tampoc és un quadrat perfecte.

El nombre de vegades que s'ha d'operar el punt *P* = (0, 1) en aquest cas és 27 720. Escollim un nombre aleatori i coprimer a 667, el 243. La corba el·líptica que utilitzarem és

$$y^2 \equiv x^3 + 243x + 1 \ \ (\mathbb{Z}/667\mathbb{Z})$$

Un petit codi de Python ens dirà quines operacions hem de fer al punt *P*. Cada cop que comenci una d'aquestes operacions ho marcaré subratllant-ho:

```
1  k = 27720
2  llista = []
3  while (k > 0):
4     if (k % 2 == 1):
5         llista.append("suma")
6     llista.append("dobla")
7     k = k//2
8  print(llista)
```

```
Python:  ['dobla', 'dobla', 'dobla', 'suma', 'dobla', 'dobla', 'dobla', 'suma',
         'dobla', 'dobla', 'dobla', 'dobla', 'suma', 'dobla', 'suma', 'dobla',
         'dobla', 'suma', 'dobla', 'suma', 'dobla']
```

Doblem el punt (0, 1) per començar. El pendent en aquest cas serà

$$\frac{3 \times 0^2 + 243}{2 \times 1} \mod 667$$

La inversa multiplicativa de 2 en mòdul 667 és -333 o el que és el mateix[4], 334.

243 · 334 mod 667 és 455.

El component *x* del doble de (0, 1) serà

$$455^2 - 2 \times 0 \mod 667 = 255$$

[4] No hi ha cap diferència utilitzar -333 o 334, perquè en mòdul 667, són congruents

i el component *y*:

$$-(455 \times 255 + (1 - 455 \times 0)) \mod 667 = 32$$

Tot seguit hem de doblar el punt (255, 32), que tindrà pendent

$$\frac{3 \times 255^2 + 243}{2 \times 32} \mod 667$$

La inversa multiplicativa del denominador, 64, en mòdul 667 és -198 o 469, per tant el pendent és (3 · 255 · 255 + 243) · 469 mod 667 = 363.

El component *x* és:

$$363^2 - 2 \times 255 \mod 667 = 520$$

i el component *y*:

$$-(363 \times 520 + (32 - 363 \times 255)) \mod 667 = 488$$

Hem de doblar el punt (520, 488), que tindrà pendent

$$\frac{3 \times 520^2 + 243}{2 \times 488} \mod 667$$

La inversa multiplicativa del denominador, 976, en mòdul 667 és 259. Aleshores el pendent és (3 · 520 · 520 + 243) · 259 mod 667 = 41.

El component *x* és:

$$41^2 - 2 \times 520 \mod 667 = 641$$

i el component *y*:

$$-(41 \times 641 + (488 - 41 \times 520)) \mod 667 = 554$$

Operem els punts (641, 554) $\oplus$ (0, 1). El pendent en aquest cas serà:

$$\frac{1 - 554}{0 - 641} \mod 667$$

El numerador és -553 o 114, i la inversa multiplicativa de -641 (o el que és el mateix, 26) és 77. Per tant, el pendent és 114 · 77 mod 667 = 107.

El component *x* és:

$$(107^2 - 641 - 0) \mod 667 = 136$$

El component *y*:

$$-(107 \times 136 + (1 - 107 \times 0)) \mod 667 = 121$$

Doblem el punt (136, 121). El pendent serà:

$$\frac{3 \times 136 + 243}{2 \times 121} \mod 667$$

La inversa multiplicativa en mòdul 667 del denominador és -113 o 554. Per tant el pendent és (3 · 136 + 243) · 554 mod 667 = 474.

El component *x* del punt resultant és:

$$(474^2 - 2 \times 136) \mod 667 = 292$$

i el component *y* és:

$$-(474 \times 292 + (121 - 474 \times 136)) \mod 667 = 639$$

Doblem el punt (292, 639). El pendent serà:

$$\frac{3 \times 292 + 243}{2 \times 639} \mod 667$$

La inversa multiplicativa en mòdul 667 del denominador és -251 o 416, i el pendent és (3 · 292 + 243) · 416 mod 667 = 605.

El component *x* és:

$$605^2 - 2 \times 292 \mod 667 = 592$$

i el component *y*:

$$-(605 \times 592 + (639 - 605 \times 292)) \mod 667 = 619$$

Doblem el punt (592, 619). El pendent en aquest cas serà:

$$\frac{3 \times 592 + 243}{2 \times 619} \mod 667$$

La inversa multiplicativa del denominador mòdul 667 és -245 o 422. El pendent aleshores és (3 · 592 + 243) · 422 mod 667 = 259.

El component *x* del punt resultant és:

$$259^2 - 2 \times 592 \mod 667 = 531$$

El component *y*:

$$-(259 \times 532 + (619 - 259 \times 592)) \mod 667 = 247$$

Operem els punts (531, 247) ⊕ (0, 1). El pendent serà:

$$\frac{1 - 247}{0 - 531} \mod 667$$

El numerador és -247 o 420. La inversa multiplicativa de -531 o 136 és 103. El pendent és 420 · 103 mod 667 = 572.

El component *x* del punt resultant és:

$$572^2 - 2 \times 531 \mod 667 = 626$$

El component *y*:

$$-(572 \times 626 + (247 - 572 \times 531)) \mod 667 = 107$$

Doblem el punt (626, 107). El pendent serà:

$$\frac{3 \times 626 + 243}{2 \times 107} \mod 667$$

La inversa multiplicativa del denominador mòdul 667 és 240. Per tant, el pendent és (3 · 626 + 243) · 240 mod 667 = 119.

El component *x* és:

$$119^2 - 2 \times 626 \mod 667 = 236$$

i el component *y* és:

$$-(119 \times 236 + (107 - 119 \times 626)) \mod 667 = 280$$

Doblem el punt (236, 280). El pendent serà:

$$\frac{3 \times 236 + 243}{2 \times 280} \mod 667$$

La inversa multiplicativa del denominador mòdul 667 és 187. Per tant, el pendent és (3 · 236 + 243) · 187 mod 667 = 415.

El component *x* és:

$$415^2 - 2 \times 236 \bmod 667 = 334$$

i el component *y* és:

$$-(415 \times 334 + (280 - 415 \times 236)) \bmod 667 = 404$$

Doblem el punt (334, 404). El pendent serà:

$$\frac{3 \times 334 + 243}{2 \times 404} \bmod 667$$

La inversa multiplicativa del denominador mòdul 667 és 123. Per tant, el pendent és (3 · 334 + 243) · 123 mod 667 = 392.

El component *x* és:

$$392^2 - 2 \times 334 \bmod 667 = 253$$

El component *y* és:

$$-(392 \times 253 + (404 - 392 \times 334)) \bmod 667 = 666$$

Operem els punts (253, 666) ⊕ (0, 1). El pendent serà:

$$\frac{1 - 666}{0 - 253} \bmod 667$$

El numerador és -665 o 2. Al intentar calcular la inversa multiplicativa de -253 o 414 ens trobem que el màxim comú divisor de 414 i 667 és 23, per tant, 414 no té inversa multiplicativa en els enters mòdul 667. Això significa que **23** és un factor de 667.

Dividim 667 entre 23 i el resultat és 29. Ara hauríem d'escollir un nombre primer més petit que 29 com a coeficient d'una corba el·líptica, on hi faríem operacions amb el punt (0, 1). No ho farem perquè ja que 29 és un nombre primer, no en trobaríem cap factor. Per tant, **29** és el tercer i últim factor de 2001.

7. Conclusions

Després de moltes hores de programar, de depurar el programa, i de proves amb el programa resultant, he de dir que m'ha sorprès la rapidesa d'aquest algorisme tot i que tan sols és el tercer més ràpid que existeix.

Ja havia implementat alguns altres algorismes de factorització o tests de primalitat, molt més senzills d'entendre i alhora molt més lents. És sorprenent que, a la implementació que he fet del mètode de Factorització per Corbes El·líptiques de Lenstra, la resposta sigui instantània, encara que el nombre a factoritzar sigui el més gran que accepta C++ sense penjar-se i tancar-se de cop.

També cal dir que aquest algorisme és probabilístic i que si dos o més dels factors de l'entrada *N* són bastant grans, és més probable que l'algorisme retorni erròniament el producte d'aquests factors grans com a un dels factors. I també pot ser que tot i *N* tingui factors petits, la corba el·líptica escollida no sigui una "bona[5]" corba i l'algorisme falli en factoritzar *N*, prenent-lo com a nombre primer, però això és molt poc probable.

En conclusió, la pregunta plantejada al principi d'aquest Treball de Recerca:

> "Es pot implementar un mètode de factorització d'enters que dugui a terme la seva tasca en un temps subexponencial?"

té com a resposta:

> "Sí, però el mètode ha de ser probabilístic."

[5] Es desconeix com diferenciar les bones i les males corbes: simplement algunes no serveixen

Annexos

Annex I: Generador de rellotges modulars

Per a generar les imatges de les pàgines 6 i 7, vaig escriure un programa en Processing (un llenguatge de programació) al qual li proporciones un nombre d'hores *n* i una hora a marcar *f* i fa una imatge jpg amb el rellotge.

```
1  void setup(){
2   size(350, 350);
3   background(255,255,255);
4   textSize(26);
5  }
6  void arrow(int x1, int y1, float x2, float y2) {
7   line(x1, y1, x2, y2);
8   pushMatrix();
9   translate(x2, y2);
10  float a = atan2(x1-x2, y2-y1);
11  rotate(a);
12  line(0, 0, -10, -10);
13  line(0, 0, 10, -10);
14  popMatrix();
15 }
16 void draw(){
17  int n=12;
18  int f=3;
19  textAlign(CENTER);
20  f=f%n;
21  fill(255, 255, 255);
22  ellipse(width/2, height/2, 250, 250);
23  fill(0, 0, 0);
24  ellipse(width/2, height/2, 10, 10);
25  for(int i=0; i < n; i++){
26    fill(0, 0, 0);
27    ellipse(cos(2*PI*i/n-PI/2)*125+width/2, sin(2*PI*i/n-PI/2)*125+height/2,
    5, 5);
28    text(i, cos(2*PI*i/n-PI/2)*140+width/2,
    sin(2*PI*i/n-PI/2)*140+height/2+9);
29    if(i == f){
30      arrow(width/2, height/2, cos(2*PI*i/n-PI/2)*115+width/2,
    sin(2*PI*i/n-PI/2)*115+height/2);
31    }
32  }
33  String S="rellotge_de_"+n+"_hores_a_les_"+f+".jpg";
34  save(S);
35 }
```

Aquest programa és bastant senzill. Cal tenir en compte que l'origen de coordenades, en Processing, és a dalt a l'esquerra.

Primer de tot, a *void setup()*, es declara el tamany de la finestra gràfica a 350 x 350 píxels amb *size*, el color dels fons a blanc (255-255-255 en codi RGB de 8 bits) amb *background* i el tamany del text a 26 píxels amb *textSize*.

La funció que dibuixa fletxes, *arrow()*, la vaig treure d'una publicació al fòrum de Processing[6].

A *void draw()* es declaren les variables *n* i *f*, on *n* és el nombre d'hores que tindrà el rellotge i *f* l'hora a la que assenyalarà la fletxa. En aquest cas, serà un rellotge de 12 hores amb la fletxa a les 3. L'ordre de la línia 19 alinea el text al mig amb *textAlign(CENTER)*. Això farà que quan se li digui a Processing que posi un text determinat al punt *(x, y)*, el text tindrà el centre a aquest punt, en comptes de prendre el punt com a la cantonada superior esquerra del requadre de text, que és el que faria Processing per defecte. A la línia 20 es redueix *f* mod *n*. Després es declara que totes les figures que es facin a partir de la línia 22 siguin de color blanc per dins amb *fill()*. Després es fa el cercle que fa d'esfera del rellotge. Després es canvia el color de *fill()* a negre i es fa el centre del rellotge. Tot seguit s'utilitza un bucle *for* per a repartir uniformement *n* punts en la circumferència. Això ho faig amb trigonometria senzilla (línies 27, 28 i 30). Finalment es crea un nom útil per a la imatge que s'ha generat, per exemple *rellotge_de_12_hores_a_les_3.jpg*, i es guarda.

Annex II: Demostració de l'associativitat de la Llei de Grup

Donat un conjunt

$$E = \left\{(x, y) : y^2 = x^3 + Ax + B\right\} \cup \{\mathrm{O}\}$$

es vol demostrar que

$$\boxed{P, Q, R \in E : (P \oplus Q) \oplus R = P \oplus (Q \oplus R)}$$

Sent P, Q i R punts diferents entre ells.

[6] http://processing.org/discourse/beta/num_1219607845.html

Aquesta demostració és una adaptació de la demostració de l'associativitat de la Llei de Grup en corbes cúbiques del llibre *Algebraic Curves* de William Fulton.

Recordem que l'element neutre del grup abelià (E, $\oplus$) és el punt a l'infinit $\mathcal{O}$, i que l'oposat d'un punt P és denotat com a P', de manera que

$$P \oplus P' = \mathcal{O}$$

També cal tenir present el Teorema de Cayley-Bacharach, que diu que si dues corbes cúbiques (com ara corbes el·líptiques) es troben en 9 punts del pla cartesià, qualsevol altre corba cúbica que passi per 8 d'aquests punts passa pel novè també.

Per últim abans de començar la demostració hem de saber que qualsevols dos punts P i Q que pertanyin a una corba el·líptica, existeix una única recta L que passa per aquests dos punts i talla la corba el·líptica en un únic tercer punt. En el cas de que P i Q siguin el mateix punt, L és tangent a aquest punt i talla la corba en un altre únic punt. Això és:

$$(\forall P, Q \in E), \exists! L \mid L \cap E = \{P, Q, R\}, R \in E$$

Per començar la demostració, hem de definir una funció λ: E×E → E, és a dir, una funció que pren com a arguments dos punts d'E i en resulta un, de la següent manera: per cada

$$L \cap E = \{P, Q, R\}$$

sent L la recta que passa pels punts P, Q i R de la corba el·líptica E,

$$\lambda(P, Q) = R$$

És a dir, λ completa el trio de punts on es tallen una recta L i una corba el·líptica E. La funció λ és quasibé com l'operació $\oplus$ sobre E però sense element neutre. La relació entre $\oplus$ i λ és:

$$P \oplus Q = \lambda(\mathcal{O}, \lambda(P, Q))$$

Tenint una corba el·líptica E, definirem amb lletres de la P a la U com a punts d'aquesta corba, i L o M seran rectes. Per exemple, diguem que la recta L_1 talla amb E en els punts P, Q i S'. Similarment tindrem:

$$L_1 \cap E = \{P, Q, S'\}, \quad M_1 \cap E = \{\mathcal{O}, S, S'\}$$
$$L_2 \cap E = \{S, R, T'\}, \quad M_2 \cap E = \{Q, R, U'\}$$
$$L_3 \cap E = \{\mathcal{O}, U, U'\}, \quad M_3 \cap E = \{P, U, T''\}$$

Busquem $P \oplus (Q \oplus R)$ en termes de λ. Ja que $Q \oplus R = \lambda(\mathcal{O}, \lambda(Q, R))$,

$$P \oplus (Q \oplus R) = P \oplus \lambda(\mathcal{O}, \lambda(Q, R))$$

$\lambda(Q, R)$ té com a resultat l'element que completa el trio. Si ens fixem en $M_2 \cap E$, veurem que $\lambda(Q, R) = U'$. Cada cop que ens trobem amb λ recorrerem als trios de punts que hem creat abans.

Tornant a $P \oplus (Q \oplus R)$:

$$P \oplus (Q \oplus R) = P \oplus \lambda(\mathcal{O}, \lambda(Q, R)) =$$
$$P \oplus \lambda(\mathcal{O}, U') = P \oplus U =$$
$$\lambda(\mathcal{O}, \lambda(P, U)) = \lambda(\mathcal{O}, T'')$$

Similarment, es pot desenvolupar $(P \oplus Q) \oplus R$:

$$(P \oplus Q) \oplus R = R \oplus (P \oplus Q) = \\ R \oplus \lambda(\mathcal{O}, \lambda(P, Q)) = R \oplus \lambda(\mathcal{O}, S') = \\ R \oplus S = \lambda(\mathcal{O}, \lambda(R, S)) = \\ \lambda(\mathcal{O}, T')$$

Per demostrar que $(P \oplus Q) \oplus R = P \oplus (Q \oplus R)$ és suficient demostrar que T' = T'', perquè l'associativitat de la Llei de Grup implica que $\lambda(\mathcal{O}, T'') = \lambda(\mathcal{O}, T')$, com acabem de veure.

Prenguem

$$\bigcup_{i=1}^{3}(L_i \cap E) \cap \bigcup_{j=1}^{3}(M_j \cap E) =$$

$$\{P, Q, S', S, R, T', \mathcal{O}, U, U'\} \cap \{\mathcal{O}, S, S', Q, R, U', P, U, T''\}$$

Cada un dels dos elements d'aquesta unió és una col·lecció de 9 punts. Si imaginem que per cada un dels dos grups de 9 punts hi passa una corba cúbica, ja que els dues cúbiques tindrien 8 punts en comú (P, Q, R, S, S', U, U', $\mathcal{O}$), segons el Teorema de Cayley-Bacharach han de compartir un novè punt. Això significa que tots dos novens punts són el mateix, és a dir, T' = T''. Per tant, $\lambda(\mathcal{O}, T'') = \lambda(\mathcal{O}, T')$ i queda demostrat que $(P \oplus Q) \oplus R = P \oplus (Q \oplus R)$.

Annex III: Algorisme euclidià del màxim comú divisor

El principi d'aquest algorisme és que el màxim comú divisor de dos nombres enters és el màxim comú divisor del segon i el mòdul de la divisió del primer i el segon:

$$\boxed{\mathrm{mcd}(a, b) = \mathrm{mcd}(b, a \bmod b)}$$

Com que *a* i *b* són tots dos divisibles pel seu màxim comú divisor, qualsevol combinació entera d'aquests dos, com per exemple *a-qb*, també és divisible per mcd(*a*, *b*). Si recordem que

$$dividend = divisor \times quocient + residu$$

podem suposar que *a* és el dividend, *b* el divisor i *q* el quocient i llavors el residu és *a-qb*, o el que és el mateix, *a* mod *b*. Per tant, el residu *a* mod *b* és divisible per mcd(*a*, *b*), i per consegüent, mcd(*a*, *b*) = mcd(*b*, *a* mod *b*), QED.

El problema es simplifica a cada pas, i s'ha d'afegir un cas base per a que la recursivitat pari. El cas base és que quan un dels dos arguments de mcd(*a*, *b*) és 0, el màxim comú divisor és l'altre argument.

Per exemple, per a trobar el màxim comú divisor de 54 i 24:

$$\begin{aligned}&\mathrm{mcd}(54, 24) = \mathrm{mcd}(24, 54 \bmod 24) = \mathrm{mcd}(24, 6) = \mathrm{mcd}(6, 24 \bmod 6)\\ &= \mathrm{mcd}(6, 0) = 6\end{aligned}$$

Annex IV: Qualsevol enter positiu pot ser expressat com a suma de potències de dos

Es vol demostrar que qualsevol enter positiu *k*, pot ser escrit com a una suma de potències de dos. Ho demostraré per prova exhaustiva.

$$k = 2^{e_1} + 2^{e_2} + \cdots + 2^{e_f} = \sum_{1}^{f} 2^{e_i}, e_i \geq 0$$

Hi ha dos casos possibles per a *k*: pot ser un enter parell o un enter senar, és a dir,

$$k = 2x$$
$$k = 2x + 1$$

i ja que

$$1 = 2^0$$

i *k* > *x*, per inducció forta un enter *x* també podrà ser expressat com a suma de potències de dos, sigui parell (per què la multiplicació té propietat distributiva amb la suma) o senar (perquè la suma té propietat associativa amb si mateixa). Q.E.D.

Bibliografia

CREMONA, J.E., “Algorithms for Modular Elliptic Curves”, *s.l.*, Cambridge University Press, 1992, pp. 52-61
També disponible en línia a:
<http://homepages.warwick.ac.uk/~masgaj/book/fulltext/index.html> [Consulta: juny 2014]

HANKERSON, D. *et al.*, “Guide to Elliptic Curve Cryptography”, *s.l.*, Springer, 2004, pp. 75-77, 89-93.
També disponible en línia a:
<http://cdn.preterhuman.net/texts/cryptography/Hankerson,%20Menezes,%20Vanstone.%20Guide%20to%20elliptic%20curve%20cryptography%20(Springer,%202004)(ISBN%20038795273X)(332s)_CsCr_.pdf> [Consulta: juliol 2014]

HOFFSTEIN, J. *et al.*, “An Introduction to Mathematical Cryptography”, *s.l.*, Springer, 2008, pp. 281-283.
També disponible en línia a:
<http://books.google.es/books?id=XLY9AnfDhsYC&printsec=frontcover#v=onepage&q&f=true> [Consulta: juny 2014]

LENSTRA JR, H., “Factoring Integers with Elliptic Curves”, *Annals of Mathematics* vol. *126*, 1987, núm. 3, pp. 649-673.
També disponible en línia a:
<http://www.math.leidenuniv.nl/~hwl/PUBLICATIONS/1987c/art.pdf> [Consulta: juliol 2014]

FULTON, W., “Algebraic Curves”, *s.l.*, W. A. Benjamin, 1969, pp. 63-64.
També disponible en línia a:
<http://www.math.lsa.umich.edu/~wfulton/CurveBook.pdf>

Webgrafia

Certicom (empresa de criptografia propietat de BlackBerry): Tutorial sobre Criptografia de Corbes El·líptiques, apartats 1, 2 i 3, *ca*. 2010. <https://www.certicom.com/index.php/ecc-tutorial> [Consulta: juny 2014]

CHAREST, A.-S., "Pollard's p-1 and Lenstra's factoring algorithms", 2005, pp. 11-20 <http://www.math.mcgill.ca/darmon/courses/05-06/usra/charest.pdf> [Consulta: agost 2014]

Coprimers: article sobre nombres coprimers a la Wikipedia en anglès, 2014. <http://en.wikipedia.org/wiki/Coprime_integers> [Consulta: juny 2014]

Departament de Matemàtiques de la Universitat de Brown: "Teoria de Nombres Computacional i Aplicacions en Criptografia" a la Universitat de Wyoming, 2006. <http://www.math.brown.edu/~jhs/Presentations/WyomingEllipticCurve.pdf> [Consulta: juny 2014]

Reddit: post que vaig fer al *subreddit* "r/learnmath" demanant aclaracions sobre alguns conceptes que no entenia aleshores, i que em va ajudar molt. 2014. <http://www.reddit.com/r/learnmath/comments/2880wz/number_theory_lenstras_elliptic_curve/> [Consulta: juny 2014]

Tutorial sobre Corbes El·líptiques i Hiperel·líptiques: classes prèvies a la conferència anual de Criptografia amb Corbes El·líptiques. 2007. <http://www.hyperelliptic.org/tanja/conf/summerschool07> [Consulta: agost 2014]

Yahoo! Answers: explicació clara i senzilla (i poc formal) dels conjunts d'enters mòdul *n*. <https://answers.yahoo.com/question/index?qid=20101008044808AAIoNtG> [Consulta: maig 2014]