Diszkrét modellek alkalmazásai

Nagy Ádám

Számelmélet

A *SageMath* programcsomagban az egész számokat a `ZZ` objektummal kapjuk meg a „szokásos„ matematikai definíciónak megfelelén.

```
sage: type(ZZ)
     <type 'sage.rings.integer_ring.IntegerRing_class'>
```

Természetesen nem kell minden esetben használnunk a konstruktort, ha egy egész számmal szeretnénk dolgozni, a rendszer automatikusan felismeri.

```
sage: a,b = ZZ(4), 4
sage: type(a) == type(b)
```

True

```
sage: a == b
```

True

Aritmetiaki műveletek a „szokásosak„:

- Összeadás, kivonás: `+,-`;
- Szorzás, hatványozás: `*,^`;
- Egész értékű osztás és maradékképzés: `//,%`.

Megjegyzés: A `/` művelet eredménye egy racionális szám, sőt valójában a jelenléte elég, hogy innentől racionálisként tekintsen a megadott adatokra.

```
sage: 2/3
```

$$\frac{2}{3}$$

```
sage: type(2/3)
            <type 'sage.rings.rational.Rational'>
sage: 1/1
```

1

```
sage: type(1/1)
            <type 'sage.rings.rational.Rational'>
```

1. Oszthatóság

Definíció 1.1. ***(Osztó)*** Az a osztója b-nek és b többszöröse a-nak, azaz $a|b$, ha

$$\exists c : b = ac.$$

Feladat 1.1. Írd meg azt a függvény, amely edönti, hogy az első argumentuma osztható-e a másodikkal az alábbi példán kívűl még 4 különböző módon!

```
sage: def divides0(a,b):
....:     return (a/b).is_integer()
sage: divides0(5,2)
```

False

```
sage: divides0(6,3)
```

True

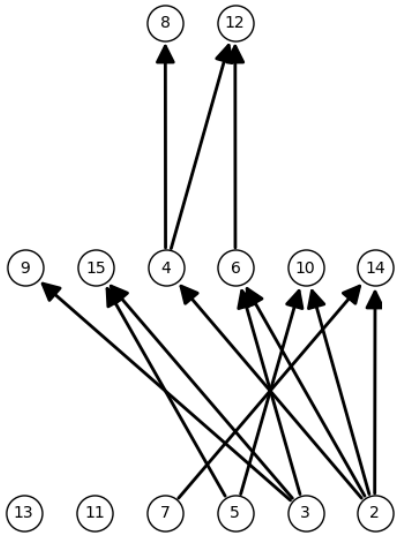
Oszthatóság tulajdonságai természetes számok esetén:

(1) Részbenrendezés, azaz
- Reflexív ($\forall a \in \mathbb{Z} : a|a$),
- Antiszimmetrikus ($\forall a, b \in \mathbb{N} : a|b \wedge b|a \Rightarrow a = b$),
- Tranzitív ($\forall a, b, c \in \mathbb{Z} : a|b \wedge b|c \Rightarrow a|c$);

(2) minden szám osztja 0-t;
(3) 1 minden számnak osztója;
(4) 0 csak saját magának osztója;
(5) $a|b \wedge c|d \Rightarrow ac|bd$;
(6) $a|b \Rightarrow \forall k \in \mathbb{Z} : ak|bk$;
(7) $k \in \mathbb{N} \setminus 0 : ak|bk \Rightarrow a|b$;
(8) $a|b \wedge a|c \Rightarrow a|b + c$;
(9) egy pozitív szám minden osztója kisebb vagy egyenlő mint a szám maga.

A részbenrendezések, így az oszthatóság is megadható egy speciális objektummal: `Poset` (*P*artially *O*rdered *S*et). Az ilyen objektumot ábrázolva kapjuk a részberendezések szemléltetésére használt Hasse-diagramot. content/english

```
sage: k = 15
....: P = Poset((Set([2..k]), lambda a,b: b % a == 0))
```



1. ábra. Hasse-diagram oszthatóság esetén 2 és k közötti természetes számokon. (`P.plot(talk=True)`)

Feladat 1.2. Írj programot, amely egy adott számhalmaz esetén megszámolja hány él van az oszthatóság relációhoz tartozó Hasse-diagramban! Ellenőrzésre lehet használni az alábbi kódot.

```
sage: len(P.cover_relations_graph().edges())
```

13

Feladat 1.3. Írj programot, amely egy adott egész szám esetén kiírja osztóinak számát, illetve osztóinak összegét! Ellenőrzéshez használhatjuk a `sigma(n,0)` és `sigma(n,1)` parancsokat.

Feladat 1.4. Írj programot, amely a természetes számok egy adott halmazában megkeresi a tökéletes számokat (tökéletes szám: osztóinak összege megegyezik a számmal, pl. 6).

Feladat 1.5. Aliquot Természetes számok esetén definiálhatjuk a következő sorozatot: $(s_0 = n; s_{i+1} = \sigma(s_i) - s_i)$, ahol a $\sigma(n)$ az n osztóinak összege. A sorozat vagy terminál nulla értékkel vagy periódikussá válik. Készíts programot, amely egy adott természetes szám esetén kiszámolja az említett sorozatot. (Ha nem terminál, akkor csak az első periódust írja ki.)

Definíció 1.2. ***(Asszociált)*** Az $a \neq b$ elemek asszociáltak, ha $a|b$ és $b|a$ is teljesül.

Definíció 1.3. ***(Egység)*** Egy e elem egységelem, ha bármely a elemre $a = ea = ae$. Az egységelem asszociáltjait egységeknek hívjuk.

Definíció 1.4. ***(Irreducibilis)*** Egy nem egység a elemet felbonthatatlannak vagy irreducibilisnek nevezünk, ha $a = bc$ esetén b és c közül az egyik egység.

Definíció 1.5. ***(Prím)*** Egy nem egység p elemet prímnek nevezünk, ha $p|ab$ esetén a $p|a$ vagy a $p|b$ közül legalább az egyik teljesül.

Megjegyzések:

(1) Az egység és egységelem két külön fogalom, egységelem egyedi, amíg az is előfordulhat, hogy a struktúra összes eleme egység (pl.: $\mathbb{Q}$).
(2) Az egység alternatív definíciója: a egység, ha bármely b elem felírható $b = ac$ alakban.
(3) Minden prímelem egyben irreducibilis is, hiszen

$$p = ab \Rightarrow p|ab \Rightarrow p|b \Rightarrow b = qp \Rightarrow p = (aq)p \Rightarrow a \text{ egység.}$$

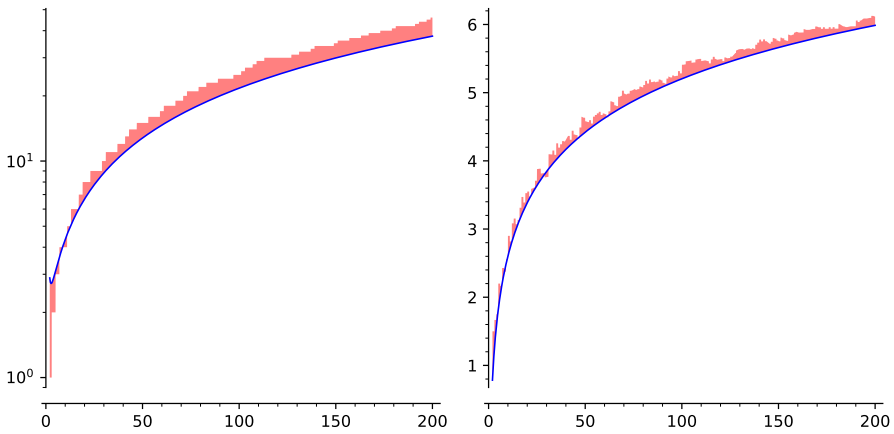
(4) Természetes számok esetén (és minden Gauss-gyűrűben), ha egy elem irreducibilis, akkor prím is.
(5) Lehet olyan struktúrát mutatni, ahol van olyan irreducibilis elem, ami nem prímelem. Például ha tekintjük az egész konstans taggal rendelkező egyváltozós polinomokat, azaz a $\mathbb{Z} + x\mathbb{R}[x]$ struktúrát, akkor az x felbonthatatlansága nyilvánvaló; ugyanakkor az $x|(x\sqrt{2})^2$ teljesül, de x nem osztja $x\sqrt{2}$-t, ui. az osztás eredményének is benne kellene lennie a struktúrában, de $\sqrt{2} \notin \mathbb{Z}$.

Feladat 1.6. A $\mathbb{Z}_m$ struktúra alatt a $0, 1, \ldots, m-1$ számokat értjük úgy, hogy az összeadás és szorzás műveletet $\mod m$ értjük. Írj programot amely egy adott m esetén definíció alapján meghatározza az egységeket, irreducibiliseket és prímeket!

Természetes számok esetén nyilvánvaló, hogy végtelen sok prím van, hiszen ha feltennénk, hogy véges sok van, akkor azokat összeszorozva és az eredményt eggyel megnövelve olyan számot kapnánk, aminek egyik sem osztója. A prímek számára becslést az $\frac{x}{\ln x}$ formulával kaphatunk, amíg *SageMath*-ban a `prime_pi(x)` függvénnyel kaphatjuk meg a pontos számukat.

```
sage: P1 = plot(x/log(x), (2, 200), scale='semilogy', \
....:     fill=lambda x: prime_pi(x),fillcolor='red')
....: P2 = plot(1.13*log(x), (2, 200), \
....:     fill=lambda x: nth_prime(x)/floor(x), fillcolor='red')
....: P = graphics_array([P1, P2])
```

Tétel 1.1. ***(Számelmélet alaptétele)*** Minden pozitív természetes szám a sorrendtől eltekintve egyértelműen felírható prímszámok szorzataként.



2. ábra. Prímek számának és növekedésének becslése (`P1,P2`)

Feladat 1.7. ***(Erasztotenész szitája)*** Adj programot, amely megadja az összes prímet egy adott számig, azaz ugyanazt az eredmény adja mint a `primes_first_n(n)`!

Feladat 1.8. Írd meg az előző feladatot hatékonyabban úgy, hogy a páros számok ne is kerüljenek be a táblába!

Feladat 1.9. Írd meg a prímszitát úgy, hogy a 2, 3 és 5-tel osztható számok ne kerüljenek a táblába! Ehhez a számokat $30i + M[j]$ alakban tárold ($30 = 2 \cdot 3 \cdot 5$), ahol $i \in [1, \lceil n \rceil]$; $j \in [1, 8]$ és $M = [1, 7, 11, 13, 17, 19, 23, 29]$.

Feladat 1.10. ***(Ikerprímek)*** Természetes számok esetén az olyan prímeket melyeknek különbsége 2 ikerprímeknek hívjuk. Írj programot, amely megkeresi az összes ikerprímet adott a és b között.

Definíció 1.6. ***(Legnagyobb közös osztó)*** Az a és b legnagyobb közös osztója az a $c = (a, b)$, amelyre

$$c|a \wedge c|b \wedge \forall d : d|a \wedge d|b \Rightarrow d|c.$$

Ha a struktúrában van „szokásos„ rendezés (ilyen az egész számok), akkor ezek közül csak a legnagyobbat tekintjük legnagyobb közös osztónak. (Például a 12 és 18 egész számokra a 6 és -6 is megfelelő lenne, de (12,18)=6.)

Feladat 1.11. Írj programot, kiszámolja a legnagyobb közös osztót a `factor` parancs segítségével! Tesztelésre használható a `gcd(a,b)` parancs.

Definíció 1.7. ***(Legkisebb közös többszörös)*** Az a és b legkisebb közös többszöröse az a c, amelyre $c \cdot (a, b) = ab$.

Feladat 1.12. Írj programot, kiszámolja a legkisebb közös többszöröst a `factor` parancs segítségével (és gcd használata nélkül)! Tesztelésre használható a `lcm(a,b)` parancs.

Definíció 1.8. ***(Relatív prím)*** Ha $(a, b) = 1$, akkor a és b relatív prímek.

Definíció 1.9. ***(Euklideszi algoritmus)*** A legnagyobb közös osztója a-nak és b-nek kiszámolható a következő algoritmussal (amennyiben van maradékos osztás a struktúrában):

(1) Legyen $a = qb + r$, ahol $0 \leq r < b$.
(2) Ha $r = 0$, akkor a legnagyobb közös osztó a.
(3) $b \leftarrow a$
(4) $a \leftarrow r$
(5) Ugorjunk (1)-re.

Feladat 1.13. Készítsd el a fenti algoritmust és hasonlítsd össze a korábbi legnagyobb közös osztót számoló program futási idejével!

Feladat 1.14. ***(Binary GCD)*** Írj programot a legnagyobb közös osztó kiszámolására, ami csak additív és shift műveleteket használ (hatékony számítógépen) az alábbi összefüggéseket használva!

- $(2a, 2b) = 2(a, b)$,
- $(2, b) = 1 \Rightarrow (2a, b) = (a, b)$,
- $(a, b) = (a - b, b)$ és így ha a és b is páratlan, akkor $a - b$ páros.

Tétel 1.2. Létezik olyan x és y, amelyekre

$$ax + by = (a, b).$$

Definíció 1.10. ***(Bővített Euklideszi algoritmus)*** Az (a, b) és a hozzá tartozó x, y értékek ($(a, b) = ax+by$) meghatározására szolgáló algoritmus. A hagyományos algoritmushoz hasonlóan a maradékokat (r_i) fogjuk számolni az

$$r_i = r_{i-2} - q_i r_{i-1}$$

alakban, továbbá használjuk az

$$\begin{aligned} ax_i + by_i &= r_i \\ &= r_{i-2} - q_i r_{i-q} \\ &= (ax_{i-2} + bx_{i-2}) - q_i(ax_{i-1} + by_{i-1}) \\ &= a(x_{i-2} - q_i x_{i-1}) + b(y_{i-2} - q_i y_{i-1}) \end{aligned}$$

invariánst. Ennek eleget téve az algoritmus

(1) $x_0, y_0, r_0 \leftarrow 1, 0, a$;
(2) $x_1, y_1, r_1 \leftarrow 0, 1, b$;
(3) $i \leftarrow 1$
(4) Ha $r_i = 0$ akkor a megoldás (x_i, y_i, r_i), különben $i \leftarrow i + 1$;
(5) $q_i \leftarrow \lfloor r_{i-2}/r_{i-1} \rfloor$
(6) $x_i, y_i, r_i \leftarrow x_{i-2} - q_i x_{i-1}, y_{i-2} - q_i y_{i-1}, r_{i-2} - q_i r_{i-1}$
(7) Ugorjunk (4)-re.

Feladat 1.15. Írj programot, ami a bővített Euklideszi algoritmust valósítja meg természetes számokra! Ellenőrzéshez használható az `xgcd` parancs.

Definíció 1.11. ***((Lineáris) Diofantikus probléma)*** Az $a, b, c \in \mathbb{Z}$ számok esetén az $ax + by = c$ egyenletet az egész számok fölött (egész megoldásokat keresünk) lineáris Diofantikus egyenletnek hívunk.

A megoldások számának vizsgálatánál először észrevehető, hogy (a, b) osztja a bal oldalt, hiszen a-nak és b=nek is osztója, így a jobb oldalt is kell osztania. Ez azt jelenti, hogy csak akkor van megoldás, ha $(a, b)|c$. Viszont ebben az esetben biztosan

van megoldás hiszen a bővített Euklideszi algoritmussal kaphatunk egyet, ha annak kimenetét megszorozzuk $c/(a,b)$-vel (x_0, y_0). Ha van még további megoldás, akkor az felírható az $(x_0 + x', y_0 + y')$ alakban alkalmas x', y' számokkal. Ekkor

$$a(x_0 + x') + b(y_0 + y') = c = ax_0 + by_0,$$

azaz

$$ax' = -by'.$$

A jobb oldal osztható b-val, így a bal is, tehát

$$\begin{array}{lllll} b|ax' \Rightarrow \dfrac{b}{(a,b)}|x' & \Rightarrow & x' = & t\dfrac{b}{(a,b)} & \\ ax' = -by' \Rightarrow at\dfrac{b}{(a,b)} = -by' & \Rightarrow & y' = - & t\dfrac{a}{(a,b)} & (t \in \mathbb{Z}). \end{array}$$

Összefoglalva, ha van megoldás akkor végtelen sok van és egy tetszőleges (x_0, y_0) megoldásból a többit a

$$x_t = x_0 + t\frac{b}{(a,b)} \qquad y_t = y_0 - t\frac{a}{(a,b)} \qquad (t \in \mathbb{Z})$$

formulákkal kaphatjuk.

Megjegyzés: A lineáris Diofantikus probléma elképzelhető egy úgy is, hogy az egyenlet egy egyenes egyenlete a síkon és a kérdés az, hogy ennek az egyenesnek van-e és mennyi metszéspontja van az egész számok segítségével készített ráccsal. A fenti megoldás itt annak felel meg, hogy megpróbáljuk egész értékű eltolással elmozdítani az egyenes egy pontját az origóba. Ha sikerül az az jelenti, hogy a ráccsal közös pontok(az origón kívül) a meredekségnek $\frac{a}{b} = \frac{a/(a,b)}{b/(a,b)}$ megfelelő négyzetek megfelelő csúcsai lesznek..

Feladat 1.16. Valósítsd meg a `LinDiofantianEq` osztályt a következőeknek megfelelően!

- Konstruktorában kell megadni az a, b, c értékeket.
- Van egy `is_solvable` függvénye.
- Fel tudja sorolni a megoldásokat egy `next_solution` és egy `prev_solution` függvény segítségével.
- Az első megoldás, amivel a `next_solution` visszatér az legyen, amely esetén az x a legkisebb nemnegatív szám.
- Csak egy megoldást tároljunk az objektum használata közben.

Feladat 1.17. Hányféleképpen tudunk kifizetni 100000 pengőt 47 és 79 pengős érmékkel?

Feladat 1.18. Egy üzletben háromféle csokoládé kapható, 70, 130 és 150 forint egységárban. Hányféleképpen lehet pontosan 5000 forintért 50 darab csokoládét venni?

Feladat 1.19. Írj programot, amely a három argumentuma (a, b, c) visszatér hány megoldása van az $ax + by = c$ diofantikus problémának a természetes számok felett (nemnegatív megoldások)!

Feladat 1.20. Írd meg a

$$\texttt{multi}(\texttt{L}, \texttt{c}, \texttt{s} = 0)$$

függvényt, amelyre

- `L` lista elemei $a_0, a_1, \ldots$;
- visszatérési érték a
$$\sum_{i=0}^{\texttt{len(L)}} a_i x_i = c$$
egyenlet nemnegatív egész megoldásainak száma `s` = `0` esetén, különben
- azon megoldások száma, amelyek még teljesítik a
$$\sum_{i=0}^{\texttt{len(L)}} x_i = s$$
feltételt is.

2. Kongruencia

Definíció 2.1. ***(Kongruencia)*** Az a és b számok kongruensek modulo m ($m > 0$), azaz
$$a \equiv b \mod m, \text{ amennyiben } m|(a-b).$$

A kongruencia mint reláció reflexív, szimmetrikus és tranzitív is, azaz ekvivalenciareláció, így meghatározza az alaphalmaz egy osztályozását.

Feladat 1.21. Írj programot, amely egy egész számokat tartalmazó halmaz elemeit osztályozza modulo m, ahol az m a második paraméter.

Definíció 2.2. ***(Maradékrendszer)*** Egész számok esetén a kongruencia mint ekvivalenciareláció által meghatározott osztályokat *maradékosztály*nak, míg rendszerüket *maradékrendszer*nek nevezzük.

Számolás során a maradékosztályokat egy-egy reprezentánsukkal szoktuk jelölni, például m esetén gyakori a $0, 1, \ldots, m-1$ (legkisebb nem negatív reprezentások) vagy egész számok esetén a $-\lfloor \frac{m-1}{2} \rfloor, \ldots, 0, \ldots, \lceil \frac{m-1}{2} \rceil$ (legkisebb abszolút értékű reprezentások) használata.

Definíció 2.3. ***(Redukált maradékrendszer)*** Ha a maradékrendszerből elhagyjuk az összes olyan maradékosztályt melyek elemei nem relatív prímek a modulushoz, akkor megkapjuk a *redukált maradékrendszer*t.

Definíció 2.4. ***(Euler-féle φ függvény)*** A $\varphi : \mathbb{N} \to \mathbb{N}$ függvényt az Euler-féle φ függvénynek nevezzűk, ha $\varphi(m)$ a modulo m redukált maradékrendszerek száma, azaz
$$\varphi(m) = |\{k \in \mathbb{Z} : 1 \leq k < m \wedge (k, m) = 1\}|.$$

Ha p egy prím és n tetszőleges természetes szám, akkor a $\varphi(p^n) = p^n - p^{n-1}$ könnyen kapható, hiszen pontosan minden p-edik maradékosztály tartalmaz p-vel osztható számokat, a többiben relatív prímek vannak p-hez és így p^n-hez is. Összetett számokkal való számoláshoz elég észrevenni, hogy a φ számelméleti függvény multiplikatív, azaz relatív prím a, b számokra $\varphi(ab) = \varphi(a)\varphi(b)$.

Feladat 1.22. Írj programotfüggvényt, amely az Euler-féle φ függvény értékét számolja ki! Ellenőrzéshez használható az `euler_phi` parancs.

Definíció 2.5. ***(Lineáris kongruenciák)*** Az a, b egész és m pozitív egész számok esetén a
$$ax \equiv b \ (m)$$
alakú kongruenciákat *lineáris kongruenciának* hívjuk.

A kongruencia és oszthatóság definíciót használva kapjuk, hogy

$$ax \equiv b\ (m) \Leftrightarrow m|ax-b \Leftrightarrow ax-b=my \Leftrightarrow ax-my=b.$$

Tehát a lineáris kongruencia megoldásait megkaphatjuk a megfelelő lineáris diofantikus probléma megoldásával, ami alapján

- $(a,m)|b$ szükséges és elégséges feltétel a megoldás létezésére;
- $acx \equiv bc\ (cm)$ kongruencia megoldásait megkaphatjuk a $ax \equiv b\ (m)$ kongruenca megoldásával;
- $(a,m)=1$ esetén mindkét oldalt oszthatjuk (a,b)-vel;
- $(a,m)=1$ és $(b,m)=c$ esetén a $ax \equiv b\ (m)$ kongruencia megoldásait kaphatjuk a $ax \equiv b/c\ (m/c)$ kongruencia megoldásával.

Feladat 1.23. Írj eljárást lineáris kongruenciák megoldására! Ellenőrzéshez használható a `solve_mod` parancs.

Definíció 2.6. ***(Moduláris inverz)*** Az $ax \equiv 1\ (m)$ kongruenciának megoldását (ha van) az a szám *moduláris inverz*ének nevezzük modulo m.

Definíció 2.7. ***(Lineáris kongruencia-rendszer)*** Legyen $1 < n \in \mathbb{N}$, $a_i, b_i \in \mathbb{Z}$ és $1 < m_i \in \mathbb{N}$ $(1 \le i \le n)$. Ekkor a

$$a_i x \equiv b_i\ (m_i) \quad (1 \le i \le n)$$

kongruenciákat *lineáris kongruencia-rendszer*nek hívunk és csak olyan x egész számot tekintünk megoldásnak, amely mindegyiknek külön-külön is megoldása.

A kongruenciarendszerek megoldásának megkereséséhez tekintsünk csak két kongruenciát és első lépésként oldjuk meg őket külön-külön. Ezek után a feladat az

$$x \equiv c_1(m_1) \text{ és } x \equiv c_2(m_2),$$

kongruenciarendszer megoldásainak megtalálása.

Tétel 2.1. ***((Kis) Fermat-tétel)*** Ha p prím és a tetszőleges egész szám, akkor

$$a^{p-1} \equiv 1\ (p).$$

Tétel 2.2. ***(Euler-Fermat-tétel)*** Ha a és m relatív prímek, akkor

$$a^{\varphi(m)} \equiv 1\ (m),$$

ahol φ az Euler-féle φ függvény.

Definíció 2.8. ***(RSA asszimetrikus titkosítás)*** Általában egy asszimetrikus titkosítási sémánál két kulcs áll rendelkezésre (egy publikus és egy privát) és a két kucsot egymás után használva visszakapjuk az üzenetet. *RSA* séma esetén

- választunk két elég nagy és megfelelő formájú p, q prímet,
- egy $e > 1$ kitevőt és
- számoljuk ki $n = pq$-t, illetve
- egy d egész számot, melyre $ed \equiv 1\ (\varphi(n) = (p-1)(q-1))$.

A publikus kulcs (n,e), a privát kulcs (n,d) lesz és egy $m < n$ szám mint üzenet titkosított formáját kapjuk az $s = m^d \mod n$ kiszámolásával. A visszafejtés az Euler-Fermat-tétel használatával

$$s^d \equiv (m^d)^e \equiv m^{ed} \equiv m^{\varphi(m)q+1} \equiv m\ (n).$$

Definíció 2.9. ***(Diszkrét logarimus probléma)*** Vegyünk egy p prímet és egy olyan g számot, amely hatványaival modulo p előállítja az összes p-nél kisebb pozitív számot. Ekkor egy a esetén a $g^a \mod p$ értékből a meghatározását diszkrét logaritmus problémának hívjuk.

Definíció 2.10. ***(Diffie-Hellman kulcscsere)*** A diszkrét logaritmus problémánál használt p és g publikus paramétereket használva két kommunikációs fél (Alice és Bob) tud közös értékben (kulcs) megállapodni Diffie-Hellman sémát használva. A séma során mindkét fél választ egy-egy véletlen értéket (titok) és számolják a g^a és g^b publikus értékeket. Ezek alapján mindketten ki tudják számolni a közös kulcsot:

$$g^{ab} = (g^a)^b = (g^b)^a.$$

0. rész

Megoldások

1. Számelmélet

1.1. Sok-sok megoldás elképzelhető, például:

```
sage: def divides0(a,b):
....:     return (a/b).is_integer()
sage: def divides1(a,b):
....:     return a % b == 0
sage: def divides2(a,b):
....:     return (a//b)*b == a
sage: def divides3(a,b):
....:     return (a/b).denom() == 1
sage: def divides4(a,b): #there is room to improve
....:     if a == 0:
....:         return True
....:     b *= sign(b)
....:     if b == 1:
....:         return True
....:     q = b
....:     a *= sign(a)
....:     while q <= a:
....:         q <<= 1
....:     while a > b:
....:         q >>= 1
....:         a -= q
....:         a *= sign(a)
....:     return a == 0 or a == b
```