Diszkrét modellek alkalmazásai

Nagy Ádám

Számelmélet

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

```
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

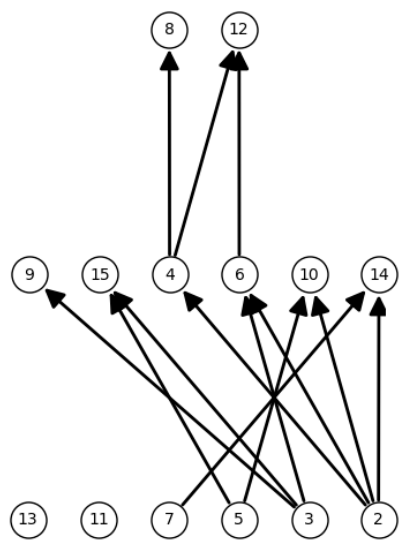
Oszthatóság tulajdonságai természetes számok esetén:

(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:

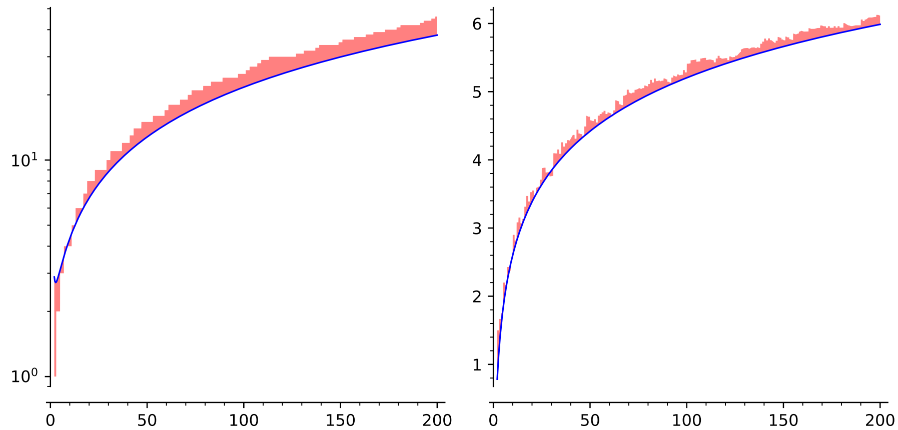
(1) Az egység és egységelem két külön fogalom, egységelem egyedi, amíg az is előfordulhat, hogy a struktúra összes eleme egység (pl.: $\mathbb{Q}$).
(2) Az egység alternatív definíciója: a egység, ha bármely b elem felírható $b = ac$ alakban.
(3) Minden prímelem egyben irreducibilis is, hiszen
$$p =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

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

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 az
$$ax \equiv b \ (m)$$
alakú kifejezéseket *lineáris kongruenciának* hívjuk.

A kongruencia és oszthatóság definíciókat használva kapjuk, hogy alkalmas y-al

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

Ez azt jelenti, hogy egy lineáris kongruencia megoldását megkaphatjuk a megfelelő lineáris diofantikus probléma megoldásával. Továbbá

- $(a, m)|b$ szükséges és elégséges feltétel a megoldás létezésére;
- $acx \equiv bc\ (cm)$ kongruencia megoldásait megkaphatjuk az $ax \equiv b\ (m)$ 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 $ax \equiv 1\ (m)$ kongruencia megoldását (ha van) az a szám *moduláris inverz*ének nevezzük modulo m.

Feladat 1.24. Írj programot, amely kiszámolja első paraméterének moduláris inverzét modulo a második paraméter! Ellenőrzéshez használható az `inverse_mod` parancs

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

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

kongruenciák összeségét *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. A fentieknek megfelelően ez azt jelenti, hogy arra alkalmas y_1 és y_2 számokkal

$$\left.\begin{array}{lcl} m_1|x - c_1 & \Leftrightarrow & x = c_1 + m_1 y_1 \\ m_2|x - c_2 & \Leftrightarrow & x = c_2 + m_2 y_2 \end{array}\right\} \Rightarrow c_1 - c_2 = m_1 y_1 - m_2 y_2,$$

ami tetszőleges c_1, c_2 esetén csak akkor lehetséges, ha m_1 és m_2 relatív prímek.
Az általános megoldás megtalálásához az előzőek alapján tegyük fel, hogy $(m_1, m_2) = 1$ és keressük x-et $x = x_1 + x_2$ alakban, ahol

$$\begin{array}{lclcllcll} x_1 & \equiv & c_1 & (m_1) & \qquad & x_1 & \equiv & 0 & (m_2) \\ x_2 & \equiv & 0 & (m_1) & & x_2 & \equiv & c_2 & (m_2). \end{array}$$

Ebből x_1-re $m_1|x_1 - c_1$ és $m_2|x_1$, azaz $m_1 u_1 = x_1 - c_1$ és $m_2 v_2 = x_1$, tehát ha $m_1 u + m_2 v = 1$, akkor

$$c_1 = m_1 u_1 - m_2 v_1 = m_1 u c_1 + m_2 v c_1.$$

Így $x_1 = c_1 - m_1 u c_1 = m_2 v c_1$ és hasonlóan $x_2 = c_2 - m_2 v c_2 = m_1 u c_2$. Ez alapján azt kaptuk, hogy a fenti két kongruenciából álló rendszer egy megoldása

$$x = c_1 m_2 v + c_2 m_1 u.$$

Az nyilvánvaló, hogy az $x + km_1m_2$ is megoldás lesz tetszőleges k egész számra, továbbá a megoldás egyértelmű is modulo m_1m_2, mivel bármely két megoldás különbsége 0 modulo m_1 és m_2 is, azaz a megoldások közötti különbség a $[m_1, m_2]$ többszöröse kell hogy legyen.

Tétel 2.1. ***(Kínai maradéktétel (KMT))*** Legyenek $m_1, m_2, \dots, m_n$ egynél nagyobb páronként relatív prím természetes számok. Ekkor az $x \equiv c_i\ (m_i)$ $(1 \leq i \leq n)$ kongruenciarendszernek van megoldása és a megoldások kongruensek modulo $m_1m_2 \dots m_n$, bármely egész $c_1, c_2, \dots c_n$ egész esetén.

Feladat 1.25. Írj eljárást, amely a kínai maradéktétel megoldását állítja elő. Az programnak két lista típusú bemenete legyen, az egyik a kínai maradéktételnél szereplő c számok a másik pedig a (páronként relatív prím) modulusok. Ellenőrzéshez használható a `crt` parancs.

Feladat 1.26. Írj eljárást amely lineáris kongruencia-rendszereket old meg! A programnak három lista típusu bemenete van: a bal oldalak együtthatóinak, a jobb oldalaknak és a modulusoknak listái.

A lineáris egyenletek mellett természetesen magasabb rendű és más típusú egyenletek is elképzelhetőek. Ezek megoldása általában más problémákat vet fel mint valós vagy akár komplex megfelelőjük, de mivel a keresett megoldás egy véges halmazban van, a legrosszabb esetben is megkaphatjuk a megoldást végigpróbálva az összes lehetséges értéket.

Feladat 1.27. ***(Kvardratikus maradékok)*** Írj programot, amely egy adott m esetén megadja azon 0 és $m-1$ közötti számok halmazát, amelyek megoldásai lehetnek a

$$x^2 \equiv b\ (m)$$

egyenletnek tetszőleges b-re vagy adott b-re attól függően, hogy a második paraméter adott-e.

Matematikában és az informatikai alkalmazások területén is fontos szerepe van a

$$a^x \equiv b\ (m)$$

típusú (logaritmushoz hasonló) egyenleteknek.

Tétel 2.2. ***((Kis) Fermat-tétel)*** Ha p prím és a tetszőleges egész szám, akkor

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

Tétel 2.3. ***(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:

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

3. Polinomok

Definíció 3.1. ***(Polinom)*** Legyen R egy olyan struktúra, amelyen van értelmezve egy additív és egy multiplikatív művelet (például egész számok vagy egy maradékrendszer). Ekkor az $f_i \in R$ $(i \in \mathbb{N})$ elemekkel, mint együtthatókkal az

$$f = (f_0, f_1, \dots)$$

sorozatot polinomnak nevezzük, ha véges sok eleme nem a nullelem.

Egy adott struktúra feletti polinomokhoz rendelhetünk változót is, amely segít a polinomok kezelésében és jelöli az adott polinomhoz tartozó struktúrát is. Például x jelölheti az egész számok fölötti polinomok változóját és ekkor az előző definicióban szereplő f polinom írható az

$$f = f(x) = f_0 + f_1 x + f_2 x^2 + \dots + f_n = \sum_{i=0}^{n} f_i x^i,$$

ha minden n-nél nagyobb indexű együttható a nullelem.

Definíció 3.2. ***(Fokszám)*** Egy f polinom esetén a legnagyobb olyan $n \in \mathbb{N}$ indexet, amelyre f_n nem nulla *fokszám*nak hívjuk. Ha nincs ilyen, azaz a poinom csak nulla elemet tartalmaz (nulla poinom), akkor a fokszám legyen $-\infty$. Jelölése: $\deg(f)$.

Definíció 3.3. ***(Műveletek polinomokkal)*** Legyen f az f_i és g a g_i $(i \in \mathbb{N})$együtthatókkal érelmezett polinomok. Ekkor a struktúrán értelmezett műveletek segítségével a polinomk fölött is értelmezhetünk aritmetikai műveleteket:

- *összeadás* elemenként történik, tehát

  $$(f + g)_i = f_i + g_i$$

  és az eredmény fokszámára $\deg(f + g) \leq \max\{\deg(f), \deg(g)\}$

- *szorzás*nál minend együtthatót minden együtthatóval össze kell szorozni és az eredményt az együtthatók összegével megfelelő indexhez kell adni, azaz
$$(fg)_i = \sum_{i=j+k} f_j g_k = \sum_{j=0}^{i} f_j g_{i-j}$$
és az eredmény fokszámára $\deg(fg) = \deg(f) + \deg(g)$ (ha nincs olyan két elem R-ben, melyek szorzata nulla).

Feladat 1.28. Írj polinom osztályt, ahol definiálva van a fokszám és az aritmetika!

Definíció 3.4. ***(Polinomfüggvény)*** Egy R fölött értelmezett polinomfüggvényen az $\widehat{f} : C \to C$ leképezést értjük, ha $R \subseteq C$ és $\widehat{f}(c) = f(c)$ a polinom kiértékelése c helyen.

Definíció 3.5. ***(Horner-elrendezés)*** Egy n-edfokú polinom defnició szerinti kiértékelése $n-1$ összeadással és $n(n+1)/2$ szorzással jár. A szorzások számára ennél jóval jobb ($n-1$ db) eljárást kapunk a *Horner elrendezés*t haszálva:
$$f(x) = \sum_{i=0}^{n} f_i x^i = f_0 + x(f_1 + x(f_2 + \cdots + x(f_{n-1} + xf_n)\dots)).$$

Feladat 1.29. Egészítsd ki az előző feladatban adott polinomosztályt egy kiértékelő függvényargumentummal, ami a kiértékelést Horner-elrendezésnek megfelelően készíti el!

Egy polinom és annak kiértékelési helyei között szoros kapcsolat áll. Nyilvánvalóan egy polinom egyértelmüen meghatározza, hogy milyen értéket vesz az fel egy adott helyen. Kevésbé nyilvánvaló, hogy egy n-edfokú polinomot egyértelműen meghatároz annak $n+1$ különböző helyen felvett értéke. Legyenek ezek a különböző helyek x_i-vel és a felvett értékek $y_i = f(x_i)$-vel jelöve $(0 \le i \le n)$. Ha sikerülne minden x_i helyhez külön-külön egy-egy olyan n-edfokú $p_i()$ polinomot konstruálni, amely az i-edik helyen y_i értéket a többi x_j $(j \ne i)$ helyen pedig nullát vesz fel akkor
$$f(x) = \sum_{i=0}^{n} p_i(x).$$
Egy polinom akkor vesz fel egy adott x_j helyen nullát, ha az felírható a $p_i(x) = (x - x_j) r_{ij}(x)$ alakban arra alkalmas r_{ij} polinommal. Ez alapján a
$$\prod_{i \ne j = 0}^{n} (x - x_j)$$
polinom minden x_i-től különböző helyen nullát vesz fel. Ahhoz hogy x_i helyen y_i-t vegyen fel osszuk el a jelenleg felvett értékével x_i helyen majd szorozzuk y_i-vel, azaz
$$p_i(x) = y_i \frac{\prod_{i \ne j=0}^{n} (x - x_j)}{\prod_{i \ne j=0}^{n} (x_i - x_j)} = y_i \prod_{i \ne j=0}^{n} \frac{x - x_j}{x_i - x_j}.$$

Tétel 3.1. ***(Lagrange-interpoláció)*** Egy f n-edfokú polinomot egyértelműen meghatároz annak $n+1$ páronként különböző helyen felvett értéke és ha (x_i, y_i) $(0 \le i \le n)$ a hely-érték párok, akkor a polinom felírható a
$$f(x) = \sum_{i=0}^{n} y_i \prod_{i \ne j=0}^{n} \frac{x - x_j}{x_i - x_j}$$

alakban.

Feladat 1.30. Írj programot, amely megvalósítja a Lagrange-interpolációt egész számokra, azaz egy n elemű egész számokból alkotott párokból (x_i, y_i) álló lista esetén visszaadja az egész együtthatós interpolációs polinomot (ha van ilyen az egész számok felett)! Ellenőrzéshez használható a

```
PolynomialRing(QQ).lagrange_polynomial(L)
```

függvény.

Definíció 3.6. ***(Shamir-féle titokmegosztás)*** Az S titok és $D = \{S_1, S_2, \ldots, S_n\}$ látszólag véletlen és látszólag S-től független adat megfelel a *Shamir-féle titokmegosztás*i sémának (n, k) paraméterekkel, ha

- Bármely legfeljebb $k - 1$ elemű részhalmaza D-nek alkalmatlan S-re vonatkozó információ megszerzésére.
- Bármely legalább k elemű részhalmaza D-nek alkalmas S helyreállítására.

Feladat 1.31. Lagrange-interpoláció segítségével valósítsd meg a Shamir-féle titokmegosztást! (Segítség: Titok lehet egy amúgy véletlen $n - 1$-edfokú polinom konstans tagja.)

Kódoláselmélet

A kódoláselmélet kódok tulajdonságait vizsgálja abból a szempontból, hogy mennyire felelnek meg a különféle alkalmazási területeken. Ezen vizsgálat során felmerülő feladatok általánosíthatók a következő modellel: egy küldő egy vagy több üzenetet probál meg eljuttatni egy fogadóhoz valamilyen tulajdonságokkal rendelkező csatornán keresztül.

1. Forráskódolás

A forráskódolás a kódok hosszával foglalkozik, vagyis azzal a kérdéssel, hogy az adott mennyiségű információt mekkora mennyiségű adattal tudjuk tárolni. Ehhez szükségünk van az információ alapegységére r, amely bináris esetben 2.

Definíció 1.1. ***(Entrópia)*** A kódolt és kódolatlan üzenetek (m) karakterekre bonthatóak, melyek önmagukban is, de leginkább az üzenetben elfoglalt helyük segítségével információt tárolnak. Arra hogy mennyi információt hordoz egy üzenet a karakterek rendezetlensége utal. Például egy egyetlen karaktert ismételgető forrás által küldött forrás információmennyisége kisebb mint egy olyané ami a karaktereket valamilyen bonyolutabb szabály szerint fűzi egymás után (szöveg). Erre a rendezetlenségre és így a relatív információmennyiségre utal az *entrópia*, amely ha az egyes karakterek előfordulásának valószínűsége $p_1, p_2, \ldots, p_n$ m-ben, akkor értéke

$$H_r(m) = -\sum_{i=1}^{n} p_i \log_r p_i.$$

Az entrópia értéke akkor a legkisebb (0), ha az üzenet csak egy karaktert tartalmaz; és akkor a legnagyobb ($\log_r n$), ha minden üzenet azonos valószínüséggel szerepel. Ebből közvetlenül tudunk következtetni, hogy a szöveg mennyire „tömör”, mivel az entrópia megadja hogy a karaktereket mennyire „jól” alkalmazzuk.
A kódolás során legfontosabb szempont a dekódolhatóság. Egy kódhalmaz (kódszavakat tartalmazó halmaz) azon tulajdonsága, hogy bármely belőle készített kódolt egyértelműen dekódolható-e azonban nem minden esetben könnyű, így szokás a kódhalmazra (kódra) vonatkozó következő fogalmakat definiálni.

Definíció 1.2. ***(Felbontható, egyenletes, vesszős és prefix kód)*** Legyen a kódszavak ábécéje B és $\alpha, \beta, \gamma \in B*$ az ábécé feletti szavak (nem feltétlenül kódszavak). A kód ekkor

- *felbontható*, ha bármely szöveg egyértelműen dekódolható;
- *egyenletes*, ha minden kódszó azonos számú karaktert tartalmaz;
- *vesszős*, ha minden kódszó felírható az $\alpha\gamma$ alakban és ha $\alpha\gamma\beta$ kódszó, akkor a $\beta = \varepsilon$, ahol ε az üres szó és $\gamma \neq \varepsilon$;
- *prefix*, ha a kódszavak halmaza prefixmentes, azaz ha az $\alpha \neq \varepsilon$és $\alpha\beta$ is kódszó, akkor $\beta = \varepsilon$;

Definíció 1.3. ***(Betűnkénti kódolás)*** A kódolás betűnként történik, ha a szöveg A ábécéje és a kódszavak B ábécéje között létezik egy $\varphi \in A \to B$ injektív (minden értéket felvesz pontosan egyszer) leképezés.

A továbbiakban csak betűnkénti kódolásról fogunk beszélni.

2. Hibajelző és hibajavító kódolás

Megoldások

3. Számelmélet

3.1. 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
```