Diszkrét matematika
9. előadás

MÁRTON Gyöngyvér
mgyongyi@ms.sapientia.ro

Sapientia Egyetem,
Matematika-Informatika Tanszék
Marosvásárhely, Románia

2024, őszi félév

Miről volt szó az elmúlt előadáson?

- a Hamming súly,
- az `ord`, `chr` függvények, bitműveletek,
- a `random` modul,
- bináris állományok,
- kódolási technikák: ASCII, UTF-8, Unicode,
- az `index`, `join` metódusok,
- az `str`, `bytes` típusok, átalakítások,
- a `to_bytes`, a `from_bytes` metódusok
- titkosítás (az XOR egy alkalmazása),

Miről lesz szó?

- prímszámok
- kódolási technikák: base64
- prímszámok listája: Eratosztenész szitája,
- a számelmélet alaptétele,
- prímfaktorizáció
- a prímszámtétel
- prímszámokkal kapcsolatos sejtések
- kongruenciák
- moduláris hatványozás
- maradékosztályok, maradékrendszerek

Prímszámok

- Prímszámok, azok az 1-nél nagyobb egész számok, amelyek nem oszthatóak, csak 1-el és önmagukkal: 2, 3, 5, 7, ...
- Összetett számok, azok az 1-nél nagyobb egész számok, amelyek nem prímszámok: 4, 6, 8, 9, 10, ...
- az 1 sem nem prím se nem összetett.

1. tétel

Minden 1-nél nagyobb pozitív egész számnak van egy prímosztója.

2. tétel

Végtelen sok prímszám létezik.

- Pl. Ha összeszorozzuk az első 6 prímszámot, akkor kapunk egy olyan számot, amelynek biztosan lesz egy új prímszám osztója:

$$2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 + 1 = 30031 = 59 \cdot 509$$

Prímszámok

3. tétel

Ha n egy összetett szám, akkor n-nek van egy olyan prímosztója, amelyik kisebb vagy egyenlő mint $\sqrt{n}$.

- Bizonyítása a 1. tétel alapján történik.
  - Ha n összetett, akkor $n = a \cdot b$, ahol $1 < a \leq b < n$.
  - Fenn kell álljon, hogy $a \leq \sqrt{n}$, mert másképp $\sqrt{n} < a \leq b \Rightarrow n = \sqrt{n} \cdot \sqrt{n} < a \cdot b$.
  - Az 1. tétel alapján a-nak van egy prímosztója, amelyik n-nek is prímosztója, amelyik tehát $\leq \sqrt{n}$.
- A tétel alapján algoritmusok adhatóak meg a prímszámok vizsgálatára:
  - osztási próba módszere (trial division): egy adott számról vizsgáljuk, hogy prím-e,
  - Eratosztenész szitája: meghatározza egy adott n-ig az összes prímszámot,
  - hatékonyabb algoritmusok: Miller-Rabin prímteszt, Solovay-Strassen prímteszt, AKS prímteszt, stb.

Prímtesztelő algoritmusok

Rossz hatékonyságú algoritmus: meghatározzuk egy szám osztóinak számát, ha az egyenlő kettővel akkor prímszám, másképp nem prímszám.

Osztási próba módszere, brute-force (trial division): sorra próbáljuk a páratlan számokkal való oszthatóságot. Az első osztónál leáll az algoritmus, azzal a kimenettel, hogy a szám nem prímszám. Ha a tesztelő szám négyzetgyökéig nem találunk osztót, akkor a szám prímszám. 10^{15}-ig ezt az algoritmust használják, nagyobb értékekre lassú.

Eratosztenész szitája: kizárásos módszerrel adott n-ig meghatározzuk a prímszámok listáját. Ha 10^6-ig az összes prímszámra szükség van, akkor ezt az algoritmust használják, nagyobb értékekre lassú.

Miller-Rabin prímteszt: egy adott páratlan számról biztosan megállapítja hogy az összetett, az azonban csak nagyon nagy valószínűséggel állítja, hogy a szám prím. Egy későbbi előadásban kerül bemutatásra.

A gyakorlatban nagyon fontos feladat eldönteni egy nagy (több mint 300 számjegyű) számról hogy prímszám-e vagy sem.

Prímtesztelés: osztási próba / brute force

Vizsgáljuk meg az osztási próba módszerével, hogy 101 prímszám-e:

- a cél: minél kevesebb osztást végezzünk az oszthatóság eldöntéséhez,
- milyen számokkal vizsgáljuk az oszthatóságot? $\rightarrow$ csak a páratlan számokkal vizsgáljuk az oszthatóságot, mert a 2-nél nagyobb prímszámoknak nem lehet páros osztójuk,
- 101 nem osztható $3, 5, 7, 9 \Rightarrow 101$ prímszám,
- a 11-el már nem kell megvizsgálni az oszthatóságot, mert $11 \cdot 11 = 121 > 101$, vagy $\lfloor \sqrt{101} \rfloor < 11$.

Az osztási próba módszerét optimalizálhatjuk, ha a 3-al való oszthatóságot is külön teszteljük. Ekkor a 3-nál nagyobb prímszámok csak $6 \cdot i + 5$, vagy $6 \cdot i + 1$ alakúak lehetnek. Az oszthatóságot tehát csak a következő számokkal kell vizsgálni:

- $5, 11, 17, 23, 29, 35, 41, \ldots$
- $7, 13, 19, 25, 31, 37, 43, 49, \ldots$

Prímtesztelés - sympy isprime

1. feladat

Írjunk egy Python függvényt, amely a `sympy isprime` függvényét alkalmazva véletlenszerűen generál egy k-nál kisebb páratlan prímszámot.

```
import random
import sympy

def primeGen_(k):
    nr = random.randint(0, k)
    if nr & 1 == 0: nr += 1
    while True:
        if sympy.isprime(nr): return nr
        nr += 2

>>> primeGen_(10**50)
    ...
```

- Ha a generált `nr` szám páros, akkor a `nr` értékét egyel növeljük, mert az `isprime` páratlan számot vár bemenetként.
- A `while` keretén belül addig növeljük kettesével a `nr` értékét, amíg egy prímszámra nem találunk.

Prímtesztelés - sympy isprime

2. feladat

Írjunk egy Python függvényt, amely véletlenszerűen generál egy k bites, $k > 64$ páratlan prímszámot, és kiírja egy fileba a base64-es alakot.

```
...
import base64
def primeGen(k):
    while True:
        nr = random.getrandbits(k)
        if nr & 1 == 0: nr += 1
        if sympy.isprime(nr): return nr

def fileWrite(k = 128):
    p = primeGen(k)
    l = p.bit_length() // 8 + 1
    binP = p.to_bytes(l, byteorder = 'big')
    base64P = base64.b64encode(binP)
    with open('prime.txt', 'wb') as outF:
        outF.write(base64P)

>>> fileWrite()
```

A base64 kódolás

- base64 kódolás egy tetszőleges bájtsorozatot alakít át olyan bájtsorozattá, amelyben csak a következő 64 fajta szimbólumnak megfelelő bájtérték szerepel:

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=

```
>>> from string import ascii_uppercase, ascii_lowercase
>>> from string import digits
>>> ABC64 = ascii_uppercase + ascii_lowercase
>>> ABC64 += digits + '+/='
```

- az algoritmus a bementi bájtsorozat minden 3 bájtjából 4 bájtot hoz létre
- = karakternek speciális szerepe van, a kódolt adatsor végére kerül és azt jelzi, hogy a bemeneti bájtsorozat 3-al való osztási maradéka 1 vagy 2.
- Pythonban a `base64` könyvtárcsomagot kell importálni: `b64encode` lesz a kódoló, `b64decode` lesz a dekódoló függvény
- a `b64encode`, `b64decode` függvények bemenetként `bytes` típusú értéket várnak és a kimenetük is `bytes` típusú érték lesz

A base64 kódolás

Python példák:

```
>>> from base64 import b64encode, b64decode

>>> b64encode(b'Sapientia')
    b'U2FwaWVudGlh='

>>> type(b64encode(b'Sapientia'))
    <class 'bytes'>

>>> b64encode('Sapientia')
    ...TypeError: a bytes-like object is required, not 'str'

>>> b64encode('Sapientia'.encode())
    b'U2FwaWVudGlh'
>>> b64encode('Sapientia'.encode()).decode()
    'U2FwaWVudGlh'

>>> b64decode(b'U2FwaWVudGlh')
    b'Sapientia'
>>> b64decode(b'U2FwaWVudGlh').decode()
    'Sapientia'
```

A Sapientia szó base64 kódolása

A 'Sapientia' szó lépésenkénti base64 kódolását a következő Python kód végrehajtásával tudjuk nyomon követni:

```
def nyomonkovetesBase64(myStr):
    lStr = len(myStr)
    for i in range(0, lStr):
        codedStr = b64encode(myStr[:i+1].encode())
        print('%10s\t%s' % (myStr[:i+1], codedStr.decode()))

>>> nyomonkovetesBase64('Sapientia')
             S        Uw==
            Sa        U2E=
           Sap        U2Fw
          Sapi        U2FwaQ==
          ...
```

A Sapi szó base64 kódolása

szöveg	S	a	p	i		
karakterkód	83	97	112	105	0	0
bitminta	01010011	01100001	01110000	01101001	00000000	00000000

bitminta	010100	110110	000101	110000	011010	010000	000000	000000
kódindex	20	54	5	48	26	16	0	0
base64 kód	U	2	F	w	a	Q	=	=

```
>>> ABC64='ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/='
>>> print(ord('S'), ord('a'), ord('p'), ord('i'))
    83 97 112 105
>>> print(ABC64[20], ABC64[54], ABC64[5], ABC64[48])
    U 2 F w
>>> print(ABC64[26], ABC64[16])
    a Q
```

Prímtesztelés - sympy isprime

3. feladat

Írjunk egy Python függvényt, amely beolvassa egy fileba kiírt szám base64-es alakját és meghatározza a tízes számrendszerbeli értéket.

```
import base64
def fileRead():
    with open('prime.txt', 'rb') as inF:
        base64P = inF.read()
    binP = base64.b64decode(base64P)
    p = int.from_bytes(binP, byteorder='big')
    print(p)

def main():
    fileWrite(2048)
    fileRead()

>>> main()
```

Prímszámok listája - Eratosztenész szitája

Eratosztenész szitájával határozzuk meg 100-ig a prímszámokat:

- a 3-al kezdődő páratlan számokat tartalmazó listában az első szám a 3-as prímszám, az összes többszöröse összetett szám lesz
- az algoritmus szerint `False`-ra fogjuk állítani az ilyen sorszámú elemek értékét
- a következő táblázatban ezeket az értékeket áthúztuk. 3 első többszöröse, ami a listában szerepel az a $3 \cdot 3 = 9$-es érték:

3	5	7	~~9~~	11	13	~~15~~	17	19	~~21~~	23	25	~~27~~
29	31	~~33~~	35	37	~~39~~	41	43	~~45~~	47	49	~~51~~	53
55	~~57~~	59	61	~~63~~	65	67	~~69~~	71	73	~~75~~	77	79
~~81~~	83	85	~~87~~	89	91	~~93~~	95	97	~~99~~			

a maradék 5-el kezdődő számokat tartalmazó listában az 5 első többszöröse, ami a listában szerepel az a $5 \cdot 5 = 25$-ös érték:

5	7	11	13	17	19	23	~~25~~	29	31	~~35~~	37	41
43	47	49	53	~~55~~	59	61	~~65~~	67	71	73	77	79
83	~~85~~	89	91	~~95~~	97							

Prímszámok listája - Eratosztenész szitája

a maradék 7-el kezdődő számokat tartalmazó listában 7 első többszöröse, ami a listában szerepel az a $7 \cdot 7 = 49$-es érték.

7	11	13	17	19	23	29	31	37	41	43	47	~~49~~
53	59	61	67	71	73	~~77~~	79	83	89	~~91~~	97	

a maradék 11-el kezdődő számokat tartalmazó listában a megmaradt páratlan sorszámú elemek prímszámok lesznek, mert 11 első többszöröse, ami a listában szerepel az a $11 \cdot 11 = 121$ nagyobb mint 100:

11	13	17	19	23	29	31	37	41	43	47	53	59
61	67	71	73	79	83	89	97					

100-ig a prímszámok, tehát:

2	3	5	7	11	13	17	19	23	29	31	37	41
43	47	53	59	61	67	71	73	79	83	89	97	

Prímszámok listája - Eratosztenész szitája

4. feladat

Írjunk egy Python függvényt, amely Eratosztenész szitájával kigenerálja n-ig a prímszámokat.

```
def eratL_(n):
    L = [True] * (n + 1)
    i = 3
    while i * i <= n + 1:
        if L[i] == True:
            for j in range(i * i, n + 1, i): L[j] = False
        i += 2
    Prim = [2]
    for i in range(3, len(L), 2):
        if L[i]: Prim += [i]
    return Prim

>>> eratL_(50)
    [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
>>> len(eratL_)(1000000))
    78498
```

Prímszámok listája - Eratosztenész szitája

Az `eratL_` függvényt optimalizálhatjuk, ha a tesztelést csak $6 \cdot i + 5$, vagy $6 \cdot i + 1$ alakú számokkal végezzük:

```
def eratL(n):
    L = [True] * (n + 1)
    i = 5
    while i * i <= n + 1:
        if L[i] == True:
            for j in range(i * i, n + 1, i): L[j] = False
        i += 2
        if L[i] == True:
            for j in range(i * i, n + 1, i): L[j] = False
        i += 4
    Prim = [2, 3]
    for i in range(5, len(L) - 2, 6):
        if L[i]: Prim += [i]
        if L[i + 2]: Prim += [i + 2]
    return Prim

>>> len(eratL(10000000))
    664579
```

A számelmélet alaptétele

4. tétel

Bármely 1-nél nagyobb, pozitív egész szám a szorzótényezők sorrendjétől eltekintve, egyértelműen felírható prímszámok szorzataként. A kapott szorzatot ***prímtényezős*** *vagy törzstényezős felbontásnak hívjuk.*

- egy nr szám prímtényezős felbontásában egy prímszám többször is előfordulhat,
- azt az alakot, amelyben minden prímszámot csak egyszer, a megfelelő hatványértéken írunk kanonikus alaknak hívjuk, jelölése:

$$nr = p_1^{a_1} \cdot p_2^{a_2} \cdot \ldots \cdot p_n^{a_n} = \prod_{i=1}^{n} p_i^{a_i}$$

- Példák:
  $2200 = 2 \cdot 2 \cdot 2 \cdot 5 \cdot 5 \cdot 11 = 2^3 \cdot 5^2 \cdot 11$
  $361 = 19 \cdot 19 = 19^2$
  $10127 = 13 \cdot 19 \cdot 41$
- a prímtényezős felbontás folyamatát prímfelbontásnak, prímfaktorizációnak, vagy csak egyszerűen faktorizációnak nevezzük,

Prímfaktorizációs módszerek

- fontosak azok az algoritmusok, amelyek képesek hatékonyan megadni egy nagy szám prímtényezős felbontását,
- nagy számok esetében a prímtényezős felbontás meghatározására, ha a számításokat nem kvantumszámítógépen végezzük nem ismert hatékony eljárás,
- a kriptográfiában 2048 bites prímszámokkal/összetett számokkal dolgoznak, hivatalosan a jelenlegi kvantumszámítógépek 22 bites számokat képesek faktorizálni:

```
>>> random.getrandbits(22)
    ...
>>> random.getrandbits(2048)
    ...
```

- az egyik legegyszerűbb prímfaktorizációs módszer a prímtesztelésnél bemutatott osztási próba módszeréhez hasonlít,
- ez is brute-force típusú algoritmus, és a szakirodalomban erre is trial division néven hivatkoznak,
- léteznek hatékonyabb prímfaktorizációs algoritmusok

Prímfaktorizáció - sympy.ntheory.factorint

5. feladat

Írjunk egy Python függvényt, amely különböző páratlan számok esetében meghatározza a szám prímtényezős felbontását és a futási időt is leméri:

```
from time import time
def idomeres():
    nr1 = 280843460918712339738662400896655331143466806317667350915045303
    st = time()
    print(sympy.ntheory.factorint(nr1))
    fs = time()
    print('idoigeny: ', round(fs - st, 4))

    nr2 = 837410004219653860054476988443006445403
    st = time()
    print(sympy.ntheory.factorint(nr2))
    fs = time()
    print('idoigeny: ', round(fs - st, 4))
```

Prímfaktorizáció - sympy.ntheory.factorint

```
>>> idomeres()
    {487: 1, 576680617902900081598896100403809714873648472931555135349169: 1}
    idoigeny:  0.0
    {25155009348433269769: 1, 33289989783756939587: 1}
    idoigeny:  3.2576
```

- rosszabb futási időt kaptunk akkor, amikor a kisebb számot próbáltuk faktorizálni: a bemenet két nagyobb prímszám szorzatából áll,
- az első meghívás esetében a prímtényezők egyike egy háromjegyű szám,
- az igazi kihívást a prímfaktorizációs algoritmusok terén az jelenti, hogy nagy prímszámok szorzatából álló számnak határozuk meg a prímtényezőit
- a `sympy.ntheory.factorint` függvény kimenetének típusa dictionary

```
>>> sympy.ntheory.factorint(33195825)
    {3: 4, 5: 2, 13: 2, 97: 1}
```

  - jele a kapcsos zárójel, és értékpárokat tárol
  - a kimenetet `kulcs: érték` formájában tárolja
  - minden `kulcs`-hoz tartozik egy `érték`
  - a kulcs értéke egyedi, azaz nem szerepelhet többször az adatszerkezetben, az `érték` azonban többször is szerepelhet és nem változtatható meg
  - különbség van a kis és nagybetűk között

A prímszámtétel

1. értelmezés

$\pi(x)$, az x-nél nem nagyobb prímszámok számát jelöli, ahol x egy pozitív valós szám.

Például: $\pi(10) = 4$, $\pi(100) = 25$.

A prímszámtétel a prímszámok eloszlását írja le, amelyet Gauss adott meg, 1793-ban, mint sejtést, bizonyítani Hadamard bizonyította be 1896-ban:

5. tétel (Prímszámtétel)

$$\lim_{x\to\infty} \frac{\frac{\pi(x)}{x}}{\frac{}{ln(x)}} = 1,$$

ahol $ln(x)$ a természetes logaritmusfüggvény.

Aszimptotikus jelöléssel: $\pi(x) \sim \frac{x}{\ln(x)}$

A prímszámtétel

Nagyobb x értékekre a következő megközelítések pontosabbak:

$$\pi(x) \sim \frac{x}{\ln(x)-1}, \qquad \pi(x) \sim \frac{x}{\ln(x)-1-\frac{1}{\ln(x)}}.$$

Példák:

$$\pi(10) = 4 \qquad \frac{10}{ln(10)} = 4.34 \qquad \frac{10}{ln(10)-1} = 7.67,$$

$$\pi(1000000) = 78498 \qquad \frac{1000000}{ln(1000000)} = 72382.41 \qquad \frac{1000000}{ln(1000000)-1} = 78030.44.$$

A prímszámtétel alapján állítható, hogy az x-ik prímszám közelített értéke a következő képletek valamelyikével is meghatározható:

$$x \cdot \ln(x), \qquad x \cdot (\ln(x) + \ln(\ln(x) - 1)).$$

Példák:
A 10-ik prímszám 29, és fennáll:

$$10 \cdot ln(10) = 23.02,$$
$$10 \cdot (ln(10) + ln(ln(10) - 1)) = 25.66.$$

A 10000-ik prímszám 104729, és fennáll:

$$10000 \cdot ln(10000) = 92103.40,$$
$$10000 \cdot (ln(1000) + ln(ln(10000) - 1)) = 113157.34.$$

Prímszámok, tulajdonságok, sejtések

- Ikerprímek: Azok a $(p, p+2)$ számpárok, ahol p prímszám és $p+2$ is prímszám.
  Példa: $(3,5)$, $(5,7)$, $(11,13)$.
- Ikerprím sejtés: Végtelen sok ikerprím létezik.
  - Az ikerprím sejtés tulajdonképpen úgy is megfogalmazható, hogy a szomszédos prímek közötti különbség végtelen sokszor nagyon kicsi.
  - A prímek között az ikerprímek ritkán fordulnak elő.
    Példa: 10^{18}-nál kevesebb mint $8 \cdot 10^{15}$ pár található. A 2016-ban felfedezett aktuálisan legnagyobb ikerprím párok 388342 számjeggyel rendelkeznek.
- Goldbach sejtés:
  - Minden 2-nél nagyobb páros szám felírható két prímszám összegeként.
  - Minden 5-nél nagyobb páratlan szám felírható három prímszám összegeként.
    Példa:

$$\begin{array}{ll} 4 = 2 + 2 & 10 = 7 + 3 \\ 6 = 3 + 3 & 12 = 7 + 5 \\ 8 = 5 + 3 & 14 = 7 + 7 \end{array}$$

Kongruenciák

- a számelmélet alapjait képezik, amelyet Gauss dolgozott ki, a 19. században
- megjelenik a mindennapi életben: az órák $\pmod{12}$ vagy $\pmod{24}$ szerint, az év $\pmod{7}$ szerint működik, stb.
- legyen m egy pozitív egész szám:
  - azt mondjuk, hogy a kongruens b-vel modulo m szerint, ha $m \mid (a - b)$
  - ezt $a \equiv b \pmod{m}$-el jelöljük
  - ellenkező esetben azt mondjuk, hogy a inkongruens b-vel modulo m szerint
- átalakítva egyenletté: $a \equiv b \pmod{m}$ akkor és csakis akkor, ha létezik egy k egész szám, úgy hogy: $a = b + km$
- $a \cdot x \equiv b \pmod{m}$ típusú egyenletek megoldásából indulunk ki, ahol a $\pmod{m}$, m-mel való osztási maradékot jelöl
- alkalmazási terület: prímtesztelő algoritmusok, adatbiztonság, kódelmélet

Kongruenciák

Alaptulajdonságok

- reflexív: ha a egy egész szám, akkor $a \equiv a \pmod{m}$,
- szimmetrikus: ha a, b egész számok és $a \equiv b \pmod{m}$, akkor $b \equiv a \pmod{m}$,
- tranzitív: ha a, b, c egész számok és $a \equiv b \pmod{m}$, $b \equiv c \pmod{m}$, akkor $a \equiv c \pmod{m}$,

Aritmetikai műveletekkel kapcsolatos tulajdonságok: legyenek a, b, c, m egész számok, ahol $m > 0$ és $a \equiv b \pmod{m}$. Ekkor fennállnak a következők:

$$\begin{aligned} a + c &\equiv b + c \pmod{m}, \\ a - c &\equiv b - c \pmod{m}, \\ a \cdot c &\equiv b \cdot c \pmod{m}. \end{aligned}$$

Kongruenciák

Példák:

- $21 \equiv 3 \pmod 9$, mert $9 \mid (21-3)$, és felírható $21 = 3 + 2 \cdot 9$
- $48 \equiv 3 \pmod 9$, mert $9 \mid (48-3)$, és felírható $48 = 3 + 5 \cdot 9$
- tehát $21 \equiv 48 \equiv 3 \pmod 9$,
- fennáll: $\ldots -15 \equiv -6 \equiv 3 \equiv 12 \equiv 21 \ldots \pmod 9$
- 3, a legkisebb pozitív maradék
- -6, a legnagyobb negatív maradék

Pythonban a % a legkisebb pozitív maradékot határozza meg:

```
>>> (-15) % 9
3
```

- $79 \equiv 7 \pmod 9$, ekkor
  - $79+3 \equiv 7+3 \pmod 9$, azaz $82 \equiv 10 \pmod 9$
  - $79-4 \equiv 7-4 \pmod 9$, azaz $75 \equiv 3 \pmod 9$
  - $79 \cdot 2 \equiv 7 \cdot 2 \pmod 9$, azaz $158 \equiv 14 \pmod 9$
- $14 \not\equiv 5 \pmod{12}$, mert $12 \nmid (14-5) = 9$

Kongruenciák

Az osztással kapcsolatos tulajdonság:

- Legyenek a, b, c, m egész számok, ahol $m > 0$, $d = (c, m)$ és $a \cdot c \equiv b \cdot c \pmod{m}$. Ekkor fennáll: $a \equiv b \pmod{m/d}$
- a kongruenciát végig tudjuk osztani c-vel, ha a modulust is osztjuk a c és a modulus legnagyobb közös osztójával

Példák:

- Legyen $14 \equiv 8 \pmod{6}$
  - A kongruenciát nem tudjuk végigosztani 2-vel a szokásos módon, nem igaz az, hogy $7 \equiv 4 \pmod{6}$.
  - A kongruenciát viszont végig tudjuk osztani 2-vel, úgy hogy osztjuk a modulust is 2-vel igaz, hogy $(2, 6) = 2$, így kapjuk: $7 \equiv 4 \pmod{3}$
- Legyen $42 \equiv 77 \pmod{5}$
  - A kongruenciát végig tudjuk osztani 7-tel, úgy hogy osztjuk a modulust 1-el igaz, hogy $(7, 5) = 1$, így kapjuk: $6 \equiv 11 \pmod{5}$

Kongruenciák

Példák:

- Legyen $98 \equiv 182 \pmod{21}$
  - A kongruenciát végig tudjuk osztani úgy 14-gyel, hogy osztjuk a modulust is 7-tel igaz, hogy $(14, 21) = 7$, így kapjuk: $7 \equiv 13 \pmod 3$
- Legyen $50 \equiv 20 \pmod{15}$
  - A kongruenciát végig tudjuk osztani úgy 10-zel, hogy osztjuk a modulust is 5-tel igaz, hogy $(10, 15) = 5$, így kapjuk: $5 \equiv 2 \pmod 3$
  - A kongruenciát végig tudjuk osztani úgy 5-tel is, úgy hogy osztjuk a modulust is 5-tel igaz, hogy $(5, 15) = 5$, így kapjuk: $10 \equiv 4 \pmod 3$

Kongruenciák

További tulajdonságok: legyenek a, b, c, d, m, n egész számok, ahol $m > 0$ és

$$a \equiv b \pmod{m},$$
$$c \equiv d \pmod{m}.$$

Ekkor fennállnak a következők:

$$a + c \equiv b + d \pmod{m}$$
$$a - c \equiv b - d \pmod{m}$$
$$a \cdot c \equiv b \cdot d \pmod{m}$$
$$a^n \equiv b^n \pmod{m}$$

Példák:

- ha $4 \equiv -5 \pmod{9}$ és $79 \equiv 7 \pmod{9}$ akkor
  - $4 + 79 \equiv -5 + 7 \pmod{9}$, azaz $83 \equiv 2 \pmod{9}$
  - $4 - 79 \equiv -5 - 7 \pmod{9}$, azaz $-75 \equiv -12 \equiv 6 \pmod{9}$
  - $4 \cdot 79 \equiv (-5) \cdot 7 \pmod{9}$, azaz $316 \equiv -35 \equiv 1 \pmod{9}$
  - $4^6 \equiv (-5)^6 \pmod{9}$, azaz $4096 \equiv 15625 \pmod{9}$