

Diszkrét matematika

9. előadás

MÁRTON Gyöngyvér
mgyongyi@ms.sapientia.ro

Sapientia Egyetem,
Matematika-Informatika Tanszék
Marosvásárhely, Románia

2023, őszi félév



Miről volt szó az elmúlt előadáson?

- kódolási technikák: base64
- számelmélet:
 - prímszámok
 - prímtesztelés: osztási próba / brute force,
 - prímszámok listája: Eratoszthenész szitája,
 - a számelmélet alaptétele,
 - prímfaktorizáció: osztási próba / brute force
 - a prímszámtétel
 - prímszámokkal kapcsolatos sejtések

Miről lesz szó?

- kongruenciák
- moduláris hatványozás
- maradékosztályok, maradékrendszerek
- a kis Fermat tétel
- az Euler függvény, az Euler tétel
- az Euler függvényhez kapcsolódó összefüggések
- a Miller-Rabin prímteszt
- hatványok és generátor elemek

Kongruenciák

- a számelmélet alapjait képezik, amelyet Gauss dolgozott ki, a 19. században
- megjelenik a mindennapi életben: az órák $(\text{mod } 12)$ vagy $(\text{mod } 24)$ szerint, az év $(\text{mod } 7)$ 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 $a \not\equiv 0 \pmod{m}$, m -mel való osztási maradékot jelöl
- alkalmazási terület: prímtesztelő algoritmusok, adatbiztonság, kódelmélet

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:

- $a + c \equiv b + c \pmod{m}$,
- $a - c \equiv b - c \pmod{m}$,
- $a \cdot c \equiv b \cdot c \pmod{m}$,

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: $\dots - 15 \equiv -6 \equiv 3 \equiv 12 \equiv 21 \dots \pmod{9}$
- 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, mert **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, mert 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, mert 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, mert 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, mert 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, mert igaz, hogy $(5, 15) = 5$, így kapjuk: $10 \equiv 4 \pmod{3}$

Kongruenciák

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

- $a + c \equiv b + d \pmod{m}$,
- $a - c \equiv b - d \pmod{m}$,
- $a \cdot c \equiv b \cdot d \pmod{m}$,
- $a^n \equiv b^n \pmod{m}$, ahol n pozitív egész szám.

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 $4096 \equiv 15625 \pmod{9}$

Moduláris hatványozás

1. feladat

Írjunk egy Python függvényt, amely meghatározza $x^n \pmod{m}$ értékét.

```
def modPow(x, n, m):
    res = 1
    while n != 0:
        if n % 2 == 1: res = (res * x) % m
        x = (x * x) % m
        n = n // 2
    return res
```

>>> modPow(3, 43, 100)
27

Határozzuk meg $3^{43} \pmod{100}$ értékét: mindenegyes négyzetre emelés után meghatározzuk az osztási maradékot, és ezzel az értékkel dolgozunk tovább.

$$\begin{aligned} 3 &\equiv 3 \pmod{100} \\ 3^2 &\equiv 9 \pmod{100} \\ 3^4 &\equiv 9^2 \equiv 81 \pmod{100} \\ 3^8 &\equiv 81^2 \equiv 61 \pmod{100} \\ 3^{16} &\equiv 61^2 \equiv 21 \pmod{100} \\ 3^{32} &\equiv 21^2 \equiv 41 \pmod{100} \end{aligned}$$

$$3^{43} = 3^{32} \cdot 3^8 \cdot 3^2 \cdot 3^1 = 41 \cdot 61 \cdot 9 \cdot 3 = 27 \pmod{100}$$

Moduláris hatványozás

- a Python `pow` függvényével moduláris hatványozást is végezhetünk: három paraméterrel kell meghívni, ahol a harmadik paraméter a modulus értékét jelöli,
- soha nem kerül sor olyan számításokra, amelyek a **modulus négyzeténél nagyobb** számokkal való műveletvégzést jelentenének:

```
>>> pow(3, 43, 100)
27
```

- a Python `**` operátort is használhatjuk, de ebben az esetben először meghatározásra kerül a hatványérték és a legvégén kerül kiszámításra az osztási maradék,
- nagy számok esetében megnő a futási idő:

```
>>> (3 ** 43) % 100
27
```

- fontos, hogy moduláris hatványozás esetén a `pow` függvényt hívjuk három paraméterrel

A Wilson tétel

1. tétel (Wilson tétel)

Egy p szám akkor és csak akkor prímszám, ha $(p - 1)! = 1 \cdot 2 \dots (p - 2) \cdot (p - 1)$, p -vel való osztási maradéka $p - 1$.

Példa:

$1788! \pmod{1789} \equiv 1788$ tehát 1789 prímszám

```
>>> from math import factorial
>>> factorial(1788) % 1789
1788
```

A tételt nem lehet nagy számok esetében prímtesztelésre használni, mert nagyszámok esetében a faktoriálisok értékének a meghatározása időigényes.

A Wilson tétel

2. feladat

Írjunk egy Python függvényt, amely Wilson tételét alkalmazva megállapítja egy számról, hogy prímszám-e. Mérjük le a futási időket a következő prímszámokra: 626887, 3276599, 42877453.

```
from time import time
from math import factorial
def tesztWilson(nr):
    f = 1
    for i in range(1, nr):
        f = (f * i) % nr
    #f = factorial(nr - 1) % nr
    if f == nr - 1:
        return True
    return False
```

```
>>> idomeresWilson()
0.06800413131713867
0.3310239315032959
4.427326917648315
```

```
def idomeresWilson():
    st = time()
    tesztWilson(626887)
    fs = time()
    print(fs - st)

    st = time()
    tesztWilson(3276599)
    fs = time()
    print(fs - st)

    st = time()
    tesztWilson(42877453)
    fs = time()
    print(fs - st)
```

A kis Fermat tétel

- legjelentősebb számelméleti eredmény
- nem ugyanaz, mint a nagy Fermat tétel ($x^n + y^n = z^n$)
- Pl: legyen $m = 11, x = 3$, ekkor fennáll: $3^{10} = 59049 \equiv 1 \pmod{11}$

2. tétel (A kis Fermat tétel)

Ha m egy prímszám és x egy pozitív egész szám, úgy hogy $1 \leq x \leq m - 1$, akkor

$$x^{m-1} \equiv 1 \pmod{m}.$$

A kis Fermat tétel alapján kijelenthető, hogy egy m és egy x szám, esetében, ahol $(x, m) = 1$ és $1 \leq x \leq m - 1$

- a tétel fordítottja nem igaz,
- ha igaz, hogy $x^{m-1} \equiv 1 \pmod{m}$, akkor az m -ről nem állapítható meg egyértelműen, hogy prím, lehet összetett is
- ha nem igaz, hogy $x^{m-1} \equiv 1 \pmod{m}$, akkor a szám összetett.

A kis Fermat tétel

1. értelmezés

Azokat az m összetett számokat amelyekre *létezik* olyan x szám, hogy $1 \leq x \leq m - 1$ és $x^{m-1} \equiv 1 \pmod{m}$ x alapú *Fermat-féle áprímeknek* hívjuk.

Példa: 341, 2-es alapú Fermat-féle áprím, mert $2^{340} \equiv 1 \pmod{341}$, ugyanakkor $341 = 11 \cdot 31$.

2. értelmezés

Azokat az m összetett számokat, amelyek esetében *minden* x -re, ahol $(x, m) = 1$ és $1 \leq x \leq m - 1$ fennáll $x^{m-1} \equiv 1 \pmod{m}$ *Carmichael-számoknak* hívjuk.

- A legkisebb Carmichael-szám: $561 = 3 \cdot 11 \cdot 17$.
- A Carmichael-számok száma végtelen.
- A kis Fermat tétel a Carmichael számok miatt **nem alkalmazható** prímszám-tesztelő algoritmusban.

Az Euler függvény

3. értelmezés

Legyen n pozitív egész szám, ekkor az **Euler függvény** meghatározza azoknak a pozitív egész számoknak a számát, amelyek nem nagyobbak mint n és relatív prímek n -nel. $\phi(n)$ -nel jelöljük.

Példa: az Euler függvény értékeire, ha $1 \leq n \leq 12$:

n	1	2	3	4	5	6	7	8	9	10	11	12
$\phi(n)$	1	1	2	2	4	2	6	4	6	4	10	4

- angolul: Euler's totient function,
- nagy összetett n szám esetében **nem ismert hatékony algoritmus** a $\phi(n)$ meghatározására,
- az Euler függvény értelmezése alapján megadható az Euler tétel,
- az Euler tétel **tetszőleges modulus** szerinti adott hatványérték viselkedéséről ad információt,
- a kis Fermat tétel **prím modulus** szerinti adott hatványérték viselkedéséről ad információt,
- az Euler tétel a kis Fermat tétel általánosítása, alapját képezi a véletlenszerűen működő prímtesztelő algoritmusoknak.

Az Euler tétel

3. tétel (Az Euler tétel)

Ha x, m pozitív egész számok, amelyekre $(x, m) = 1$, akkor:

$$x^{\phi(m)} \equiv 1 \pmod{m}.$$

pl: legyen $m = 12, x = 5$, ekkor fennáll $\phi(12) = 4$ és:

$$5^{\phi(12)} = 5^4 = 625 \equiv 1 \pmod{12}$$

4. tétel

Ha p egy prímszám, akkor $\phi(p) = p - 1$. A tétel fordítottja is igaz, azaz, ha p egy pozitív egész szám, amelyre fennáll, hogy $\phi(p) = p - 1$, akkor p prímszám.

Például: $\phi(61) = 60$.

5. tétel

Ha p egy prímszám és a egy pozitív egész szám, akkor $\phi(p^a) = p^a - p^{a-1}$.

Például: $\phi(256) = \phi(2^8) = 2^8 - 2^7 = 128$.

Az Euler függvény

6. tétel

Ha a és b relatív prímek, akkor $\phi(a \cdot b) = \phi(a) \cdot \phi(b)$.

Például: $\phi(75) = \phi(3 \cdot 25) = \phi(3) \cdot \phi(25) = 2 \cdot (5^2 - 5) = 40$.

7. tétel

Ha x prímtényezőss felbontása: $x = p_1^{a_1} \cdot p_2^{a_2} \cdot \dots \cdot p_n^{a_n}$, akkor

$$\phi(x) = x \cdot \left(1 - \frac{1}{p_1}\right) \cdot \left(1 - \frac{1}{p_2}\right) \dots \left(1 - \frac{1}{p_n}\right)$$

$\phi(x) - t$ felírjuk a következőképpen is:

$$\phi(x) = x \cdot \left(\frac{p_1 - 1}{p_1}\right) \cdot \left(\frac{p_2 - 1}{p_2}\right) \dots \left(\frac{p_n - 1}{p_n}\right)$$

Példák:

$$\begin{aligned}\phi(60) &= (2^2 - 2^1) \cdot (3 - 1) \cdot (5 - 1) = 16, \text{ mert } 60 = 2^2 \cdot 3 \cdot 5 \\ \phi(100) &= 100 \cdot \left(1 - \frac{1}{2}\right) \cdot \left(1 - \frac{1}{5}\right) = 40, \text{ mert } 100 = 2^2 \cdot 5^2\end{aligned}$$

A Miller-Rabin valószínűségi prímteszt

- 1980-ban publikálta O. Rabin, Gary L. Miller egy korábban megadott algoritmusát módosította,
- alkalmas annak a megállapítására, hogy egy több száz számjegyből álló páratlan szám prím-e vagy sem,
- azok után szokták alkalmazni, hogy előzetesen megállapítják, hogy a tesztelendő számnak nincsenek kis prímosztói,
- az algoritmus gyakorlatban való használhatósága a következő tételen alapszik:

8. tétel

Euler kritérium Ha az m prímszám, akkor minden olyan x szám esetében, ahol $(x, m) = 1$, és $m - 1 = 2^s r$, ahol r páratlan szám, fennáll a következő két összefüggés közül valamelyik:

$$\begin{aligned} x^r &\equiv 1 \pmod{m}, \\ \exists j, 0 \leq j \leq s-1 : x^{2^j r} &\equiv m-1 \pmod{m}. \end{aligned}$$

Példa:

- $m = 61$ esetében $m - 1 = 60 = 2^2 \cdot 15$ és legyen $x = 49$. A következő hatványértékeket kell megvizsgálni $(\text{mod } 61)$ szerint: $49^{15}, 49^{2 \cdot 15}$.
- $m = 273$ esetében $m - 1 = 272 = 2^4 \cdot 17$ és legyen $x = 5$. A következő hatványértékeket kell megvizsgálni $(\text{mod } 273)$ szerint: $5^{17}, 5^{2 \cdot 17}, 5^{2^2 \cdot 17}, 5^{2^3 \cdot 17}$.

A Miller-Rabin valószínűségi prímteszt

Az Euler kritérium alapján kijelenthető:

- a tétel fordítottja nem igaz, azaz ha fennáll a két összefüggés közül valamelyik abból nem következik az hogy a szám prím, mert vannak olyan m összetett számok, melyekre $\exists x: 1 \leq x \leq m - 1$, melyre fennáll a fenti két összefüggés közül valamelyik,
- a tétel mégis alkalmazható primtesztelő algoritmusként, mert az m összetett szám esetében maximum $\phi(m)/4$ az ilyen x számok száma,
- tehát annak a valószínűsége, hogy egy összetett m szám t darab különböző x esetében megfeleljen a tételben megfogalmazott két kritériumnak kisebb mint $(1/4)^t$,
- ha $t = 10$ véletlenszerűen generált x számra fennáll a kritérium, akkor a tévedés valószínűsége $1/4^{10} \sim 10^{-6}$, tehát m nagy valószínűséggel prímszám,
- a Miller-Rabin valószínűségi prímteszt a 8. tétel alapján adható meg. Bemenete az $m \geq 3$ páratlan szám és a $t \geq 1$ biztonsági paraméter. A kimenet True, ha az m szám prímszám figyelembe véve a t biztonsági paramétert, és False, ha összetett,
- az algoritmus kimenete tehát csak nagy valószínűséggel állítja a bemenetről hogy prím.

A Miller-Rabin valószínűségi prímteszt

3. feladat

Írjunk egy Python függvényt, amely a 8. tétel alapján végzi a prímtesztelést, azaz írjuk meg a Miller-Rabin prímtesztet.

```
from random import randint
def miller_rabinT(m, t):
    s, r = 0, m - 1
    while r % 2 == 0:
        s, r = s + 1, r // 2
    for i in range(0, t):
        x = randint(2, m - 1)
        y = pow(x, r, m)
        if y == 1 or y == (m - 1): continue
        for j in range(1, s):
            y = (y * y) % m
            if y == 1: return False #összetett szám esetében a hatványertek lehet 1
            if y == m - 1: break
        if y != (m - 1): return False
    return True
```

A 8. tétel:

Ha az m prímszám, akkor minden olyan x szám esetében, ahol $(x, m) = 1$, és $m - 1 = 2^s r$, ahol r páratlan szám, fennáll a következő két összefüggés közül valamelyik:

$$x^r \equiv 1 \pmod{m},$$
$$\exists j, 0 \leq j \leq s - 1 : x^{2^j r} \equiv m - 1 \pmod{m}.$$

A Miller-Rabin valószínűségi prímteszt

```
>>> miller_rabinT(12189489607157289733, 10)
True
>>> miller_rabinT(61, 10)
True
```

Példa: legyen $m = 61$, $t = 4$, $m - 1 = 2^2 \cdot 15$

x	j	Magyarázat
3	0	$3^{15} \equiv 60 \pmod{61} \rightarrow$ valószínűleg prím
7	0	$7^{15} \equiv 11 \pmod{61}$
	1	$7^{2 \cdot 15} \equiv 60 \pmod{61} \rightarrow$ valószínűleg prím
42	0	$42^{15} \equiv 1 \pmod{61} \rightarrow$ valószínűleg prím
24	0	$24^{15} \equiv 11 \pmod{61}$
	1	$24^{2 \cdot 15} \equiv 60 \pmod{61} \rightarrow$ nagyon nagy valószínűséggel prím

A Miller-Rabin valószínűségi prímteszt

Legyen $m = 91$, $t = 4$, $m - 1 = 90 = 2 \cdot 45$

x	j	Magyarázat
16	0	$16^{45} \equiv 1 \pmod{91} \rightarrow$ valószínűleg prím
75	0	$75^{45} \equiv 90 \pmod{91} \rightarrow$ valószínűleg prím
13	0	$13^{45} \equiv 13 \pmod{91} \rightarrow$ biztosan összetett

Megjegyzés: az algoritmus ebben az esetben nem kell $t = 4$ tesztet elvégezzen, mert a harmadik x érték után megállapítható, hogy a szám összetett.

A Miller-Rabin valószínűségi prímteszt

4. feladat

A Miller-Rabin valószínűségi prímteszt segítségével generáljunk egy k bites prímszámot, ahol $k \geq 128$ bit.

```
from random import getrandbits
def primeGen(k, t):
    nr = getrandbits(k)
    if not (nr & 1): nr += 1
    while True:
        if miller_rabinT(nr, t): break
        nr += 2
    return nr

>>> primeGen(256, 10)
88725395404043359043589...111217400738357967924821159093157
```


Hatványok és generátor elemek

- az $x^n \pmod{m}$ értékek meghatározásakor számos tulajdonság figyelhető meg,
- Példa: meghatározzuk $x^n \pmod{11}$, értékeit $x = 2, 3, \dots, 10$ -re, illetve $n = 1, 2, \dots, 10$ -re:

$2^1 = 2$	$3^1 = 3$	$4^1 = 4$	$5^1 = 5$	$6^1 = 6$	$7^1 = 7$	$8^1 = 8$	$9^1 = 9$	$10^1 = 10$
$2^2 = 4$	$3^2 = 9$	$4^2 = 5$	$5^2 = 3$	$6^2 = 3$	$7^2 = 5$	$8^2 = 9$	$9^2 = 4$	$10^2 = 1$
$2^3 = 8$	$3^3 = 5$	$4^3 = 9$	$5^3 = 4$	$6^3 = 7$	$7^3 = 2$	$8^3 = 6$	$9^3 = 3$	$10^3 = 10$
$2^4 = 5$	$3^4 = 4$	$4^4 = 3$	$5^4 = 9$	$6^4 = 9$	$7^4 = 3$	$8^4 = 4$	$9^4 = 5$	$10^4 = 1$
$2^5 = 10$	$3^5 = 1$	$4^5 = 1$	$5^5 = 1$	$6^5 = 10$	$7^5 = 10$	$8^5 = 10$	$9^5 = 1$	$10^5 = 10$
$2^6 = 9$	$3^6 = 3$	$4^6 = 4$	$5^6 = 5$	$6^6 = 5$	$7^6 = 4$	$8^6 = 3$	$9^6 = 9$	$10^6 = 1$
$2^7 = 7$	$3^7 = 9$	$4^7 = 5$	$5^7 = 3$	$6^7 = 8$	$7^7 = 6$	$8^7 = 2$	$9^7 = 4$	$10^7 = 10$
$2^8 = 3$	$3^8 = 5$	$4^8 = 9$	$5^8 = 4$	$6^8 = 4$	$7^8 = 9$	$8^8 = 5$	$9^8 = 3$	$10^8 = 1$
$2^9 = 6$	$3^9 = 4$	$4^9 = 3$	$5^9 = 9$	$6^9 = 2$	$7^9 = 8$	$8^9 = 7$	$9^9 = 5$	$10^9 = 10$
$2^{10} = 1$	$3^{10} = 1$	$4^{10} = 1$	$5^{10} = 1$	$6^{10} = 1$	$7^{10} = 1$	$8^{10} = 1$	$9^{10} = 1$	$10^{10} = 1$

4. értelmezés

x *rendjén*, \pmod{p} szerint, azt a legkisebb k kitevőt értjük, amelyre fennáll: $x^k \equiv 1 \pmod{p}$.

Példa:

- $\pmod{11}$ szerint a 2-es szám rendje 10, a 3 szám rendje 5, míg a 10-es szám rendje 2.

Hatványok és generátor elemek

9. tétel

Legyen p egy prímszám. Ekkor mindig létezik egy g egész szám, $g \in \{1, 2, \dots, p-1\}$, amelynek hatványértékei $(\text{mod } p)$ szerint előállítják az $\{1, 2, \dots, p-1\}$ halmaz elemeit egy tetszőleges sorrendben. Ezeket az elemeket **primitív gyököknek**, vagy **generátor** elemeknek hívjuk.

Példa:

- $(\text{mod } 11)$ szerint 2, 6, 7, 8 generátor elemek.
- fontos feladat, hogy egy adott prímszám esetében meghatározzunk egy generátor elemet, erre általános esetben nincs hatékony algoritmus,

5. értelmezés

Biztonságos prímekek (safe prime) hívjuk azokat a p prímszámokat, amelyek egyenlők $2 \cdot q + 1$ -el, ahol q is prímszám.

Példa:

- $11 = 2 \cdot 5 + 1$, $47 = 2 \cdot 23 + 1$ biztonságos prímekek,
- 3, 13, 17, 41, stb. nem biztonságos prímekek,

Biztonságos prímelek és generátor elemek

10. tétel

Egy p biztonságos prím esetében a g generátor elem lesz $(\text{mod } p)$ szerint, ha fennáll:

$$g \not\equiv \pm 1 \pmod{p} \text{ és } g^q \not\equiv 1 \pmod{p}.$$

- egy biztonságos prím meghatározása jóval időigényesebb, mint egy "közönséges" prím kiválasztása,
- ha a p biztonságos prím, akkor a generátor elem kiválasztása nem számít nehéz feladatnak, egy hatványértéket kell megvizsgálni

Példa:

- legyen $p = 47 = 2 \cdot 23 + 1$, $p - 1 = 46$,
 - $g = 13$ generátor elem?
 - **Igen**, mert $13^{23} = 46 \pmod{47}$.
 - $g = 3$ generátor elem?
 - **Nem**, mert $3^{23} = 1 \pmod{47}$.

Ha p biztonságos prím, akkor a generátor elemek száma
 $\phi(p - 1) = \phi(2) \cdot \phi(q) = q - 1$.

Biztonságos prímek és generátor elemek

5. feladat

Írjunk Python függvényt, amely generál egy k bites biztonságos prímet és meghatározza a prím egy generátor elemét.

```
from random import getrandbits, randint
from eload9 import miller_rabinT
def safePrime(k):
    q = getrandbits(k)
    if not (q & 1): q += 1
    while True:
        p = 2 * q + 1
        if miller_rabinT(q, 10) and miller_rabinT(p, 10): break
        q = getrandbits(k)
        if not (q & 1): q += 1
    while True:
        g = randint(2, p-2)
        temp = pow(g, q, p)
        if temp != 1: break
    return p, g
```

>>> safePrime(32)
(518189279, 466351168)

A diszkrét logaritmus (DL) probléma

- a diszkrét logaritmus problémának (DL probléma) több verziója is megadható,
- a \mathbb{Z}_p^* -ben, illetve az elliptikus görbéken megadott értelmezések mindegyikét széles körben alkalmazzák

6. értelmezés (A diszkrét logaritmus probléma \mathbb{Z}_p^* -ben)

Az egész számok $\mathbb{Z}_p^* = \{1, 2, \dots, p - 1\}$ halmaza esetében, ahol p prímszám, g generátor elem, és $A \in \mathbb{Z}_p^*$ a diszkrét logaritmusprobléma azt jelenti, hogy megkeressük azt az a pozitív egész számot, melyre fennáll:

$$g^a \equiv A \pmod{p}.$$

- az a számot A g alapú **diszkrét logaritmusának** hívjuk,
- nagy számok esetében a DL problémára **nem ismert hatékony** algoritmus, jelenleg **minimum 1024 bites** prímszámot használnak,
- számos kriptorendszer biztonsága alapszik a DL problémán.

A Diffie-Hellman kulcscsere

- 1976-ban publikálták a szerzők, biztonsága a DL problémán alapszik,
- mérőföldkőnek számít, amely alapjaiban meghatározta/meghatározza a számítógépes adatcsere biztonságát,
- két távoli egység (számítógép, mobileszköz, stb.) **kulcscsere** mechanizmusára adott elsőként megoldást: hogyan tud megosztani két távoli egység egy nyilvános csatornán egy egész számot (bájt szekvenciát), amelyet rajtuk kívül más egység nem tud meghatározni,
- a **TLS 1.3** protokolban az elliptikus görbéken értelmezett Diffie Hellman típusú kulcscserét használják
- feltételezve, hogy a kommunikációban résztvevő két egység **A** és **B**, akkor a protokoll első lépéseként **A** és **B** egy hiteles szervertől lekér egy p prímszámot, és p -nek egy g generátor elemét
- a p és g értékek publikusak,
- az **A** és **B** között megosztott egész számot a szakirodalom **mester kulcsnak**, **szimmetrikus kulcsnak** (master key) nevez, amely szigorúan titkos információ.

A Diffie-Hellman kulcscsere

- **A** a p, g ismeretében meghatározza az a, A értékeket, ahol:
 - $a \in \{2, \dots, p-2\}$ véletlenszerű, szigorún titkos,
 - $A = g^a \pmod{p}$,
 - az A -t pedig elküldi **B**-nak,
 - a gyakorlatban fontos, hogy egy megbízható hatóság garantálja, hogy az A értéke az **A** segítségétől származik, azaz szükséges A -t hitelesíteni
- **B** a p, g ismeretében meghatározza a b, B értékeket, ahol:
 - $b \in \{2, \dots, p-2\}$ véletlenszerű, szigorún titkos,
 - $B = g^b \pmod{p}$,
 - B -t pedig elküldi **A**-nak,
 - a gyakorlatban fontos, hogy egy megbízható hatóság garantálja, hogy a B értéke az **B** segítségétől származik, azaz szükséges B -t hitelesíteni
- **A** a közös K kulcsot a következőképpen határozza meg:
$$K = B^a \pmod{p}.$$
- **B** a közös K kulcsot a következőképpen határozza meg:
$$K = A^b \pmod{p}.$$
- helyesség:
$$K = A^b = B^a = g^{ab} \pmod{p}.$$
- az A illetve B értékek hitelesítése nem része az alapprotokollnak

A Diffie-Hellman kulcscsere, példa

- Legyen $p = 47, g = 13$, ahol $p = 2 \cdot 23 + 1$. Fennáll, hogy $13^{23} \equiv 46 \not\equiv 1 \pmod{47}$, azaz 13 generátor elem.
 - **A számításai:**
 - választ egy random a számot, legyen ez 12 és meghatározza $A = 13^{12} \equiv 9 \pmod{47}$,
 - elküldi **B**-nek az $A = 9$ -es számot,
 - **B számításai:**
 - választ egy random b számot, legyen ez 34 és meghatározza $B = 13^{34} \equiv 21 \pmod{47}$,
 - elküldi **A**-nak a $B = 21$ -es számot,
 - **A számítása:** $K = 21^{12} = 16 \pmod{47}$,
 - **B számítása:** $K = 9^{34} = 16 \pmod{47}$.
- a közös kulcs: $K = 16$.

A következő linken egy Diffie-Hellman kulcserén alapuló *kliens-szerver* Python program található.