# Diszkrét matematika
# 7. előadás

MÁRTON Gyöngyvér
mgyongyi@ms.sapientia.ro

Sapientia Egyetem,
Matematika-Informatika Tanszék
Marosvásárhely, Románia

2024, őszi félév

# Miről volt szó?

- valós számok, komplex számok,
- valós számok lánctört jegyei,
- polinom helyettesítési értéke, függvények ábrázolása,
- a `sin`, `ln` lánctörtképlete,
- a `numpy` és a `matplotlib` modulok,
- a `complex` típus,
- műveletek komplex számokkal: szorzat, hatványozás, stb
- fraktálok: Mandelbrot, Julia,
- számrendszerek
  - egész szám alapú számrendszerek,
  - a számrendszerek közötti kapcsolat,

# Miről lesz szó?

- egyéb számrendszerek:
  - a faktoriális számrendszer
  - a Fibonacci számrendszer
- az `n`-ik Fibonacci szám, exponenciális, lineáris, logaritmikus futásidejű algoritmusok
- bitműveletek

# Más számrendszerek

- Vegyes alapú számrendszerek
  - a számrendszer alapszáma változó
  - például, időmérés: 27. hét, 3. nap, 6 óra, 20 perc, 15 másodperc. $27_{52}3_{7}6_{24}20_{60}15_{60}$
  - az indexszám az alapszámot, a mértékegységet jelzi; a piros színű számok, a számjegyek azt jelzik, hogy az adott mértékegységből hányat használunk fel
- A faktoriális számrendszer
  - Cantor ábrázolásnak is mondják
  - alapját az képezi, hogy bármely szám egyértelműen felírható a faktoriális függvény értékeinek segítségével

$$nr = a_n \cdot n! + a_{n-1} \cdot (n-1)! + \cdots + a_2 \cdot 2! + a_1 \cdot 1!, \text{ ahol}$$
$$a_i \in [0, 1, \ldots, i], \text{ bármely } i \in [1, 2, \ldots, n]$$

  - a $nr$ szám faktoriális számrendszerben felírt számjegyeit a következő lista elemei alkotják: $[a_n, a_{n-1}, \ldots, a_2, a_1]$

# A faktoriális számrendszer

- Pl. Mennyi $(8172)_{10}$, faktoriális számrendszerbeli alakja?
  - $8172 = 1 \cdot 7! + 4 \cdot 6! + 2 \cdot 5! + 0 \cdot 4! + 2 \cdot 3! + 0 \cdot 2! + 0 \cdot 1!$
  - $8172 = 1 \cdot 5040 + 4 \cdot 720 + 2 \cdot 120 + 2 \cdot 6$
  - $8172 = [1, 4, 2, 0, 2, 0, 0]_{faktBase}$
- Pl. Mennyi $(45389)_{10}$, faktoriális számrendszerbeli alakja?
  - $45389 = 1 \cdot 8! + 1 \cdot 7! + 0 \cdot 6! + 0 \cdot 5! + 1 \cdot 4! + 0 \cdot 3! + 2 \cdot 2! + 1 \cdot 1!$
  - $45389 = 1 \cdot 40320 + 1 \cdot 5040 + 1 \cdot 24 + 2 \cdot 2 + 1 \cdot 1$
  - $45389 = [1, 1, 0, 0, 1, 0, 2, 1]_{faktBase}$
- Az algoritmus:
  - a $nr$ szám ábrázolása esetén létezik egy $n$ egész szám, amelyre: $n! \leq nr < (n+1)!$
  - az osztási algoritmus szerint $nr = a_n \cdot n! + r_n$, ahol $0 \leq a_n \leq n$ és $0 \leq r_n < n!$
  - hasonlóan, felírható: $r_n = a_{n-1} \cdot (n-1)! + r_{n-1}$, ahol $0 \leq a_{n-1} \leq (n-1)$ és $0 \leq r_{n-1} < (n-1)!$
  - tehát iterálva kapjuk az $a_n, a_{n-1}, \ldots a_1$ értékeket, amelyek a számrendszerbeli számjegyeket fogják jelenteni.

# A faktoriális számrendszer

Hatékonyabb algoritmus a $nr$ szám faktoriális számrendszerbeli alakjának a meghatározására:

- meghatározzuk $nr$ 2-vel való osztási egészrészét ($q_1$), illetve osztási maradékát ($a_1$), felírható: $nr = 2 \cdot q_1 + a_1$
- meghatározzuk $q_1$ 3-mal való osztási egészrészét, illetve osztási maradékát, felírható: $q_1 = 3 \cdot q_2 + a_2$
- folytatjuk az osztási folyamatot a 4, 5, ... értékekkel, amíg az osztási egészrész nem lesz 0
- az osztási folyamat során kiszámolt maradékok lesznek $nr$ faktoriális számrendszerbeli számjegyei
- fennáll:
  $nr = 2 \cdot (3 \cdot (4 \cdot (5 \cdot (\ldots) + a_4) + a_3) + a_2) + a_1 =_{\ldots} \cdot 4! \cdot a_4 + 3! \cdot a_3 + 2! \cdot a_2 + a_1$
- $nr = [a_n, a_{n-1}, \ldots a_2, a_1]_{faktBase}$

# A faktoriális számrendszer

Határozzuk meg az előző oldalon megadott algoritmus szerint, 8172 faktoriális számrendszerbeli számjegyeit:

| *nr* | *q* | *n* | *a* |
|---|---|---|---|
| 8172 | 4086 | 2 | 0 |
| 4086 | 1362 | 3 | 0 |
| 1362 | 340 | 4 | 2 |
| 340 | 68 | 5 | 0 |
| 68 | 11 | 6 | 2 |
| 11 | 1 | 7 | 4 |
| 1 | 0 | 8 | 1 |

```
q = nr // n
a = nr - q * n #a = nr % n
nr = q
```

$$8172 = 2 \cdot (3 \cdot (4 \cdot (5 \cdot (6 \cdot (7 \cdot (8 \cdot 0 + 1) + 4) + 2) + 0) + 2) + 0) + 0$$
$$8172 = 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \cdot 7 \cdot 1 + 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \cdot 4 + 2 \cdot 3 \cdot 4 \cdot 5 \cdot 2 + 2 \cdot 3 \cdot 4 \cdot 0 + 2 \cdot 3 \cdot 2 + 2 \cdot 0 + 0$$
$$8172 = [1, 4, 2, 0, 2, 0, 0]_{faktBase}$$

# A Fibonacci számrendszer

- a Fibonacci számrendszert, Zeckendorf ábrázolásnak is hívjuk
- alapját az képezi, hogy bármely szám egyértelműen felírható Fibonacci számok összegeként:

$$nr = a_n \cdot F_n + a_{n-1} \cdot F_{n-1} + \cdots + a_2 \cdot F_2 + a_1 \cdot F_1, \text{ ahol}$$

$a_i \in [0, 1]$, bármely $i \in [1, 2, \ldots, n]$, és $F_n, F_{n-1}, \ldots, F_2, F_1$ a 0,1 kezdőértékeket elhagyva kapott Fibonacci számsorozat

- a $nr$ szám Fibonacci számrendszerben felírt számjegyeit a következő lista elemei alkotják: $[a_n, a_{n-1}, \ldots, a_2, a_1]$
- legyen nr = 237, ekkor felírható:

$$237 = 233 + 3 + 1$$
$$237 = 1 \cdot 233 + 0 \cdot 144 + 0 \cdot 89 + 0 \cdot 55 + 0 \cdot 34 + 0 \cdot 21 + 0 \cdot 13 + 0 \cdot 8 + 0 \cdot 5 + 1 \cdot 3 + 0 \cdot 2 + 1 \cdot 1$$
$$237 = [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1]_{fibBase}$$

- legyen nr = 100, ekkor a Fibonacci számrendszerbeli alak: $[1, 0, 0, 0, 0, 1, 0, 1, 0, 0]_{fibBase}$

# A Fibonacci számrendszer

- egy szám Fibonacci számrendszerbeli alakjában hasonlóan a kettes számrendszerhez csak két fajta szimbólum szerepel, a 0 és 1-es szimbólumok. A két alakot nem szabad összetéveszteni!
- Fibonacci számrendszerben egy szám felírásához több számjegyre (nullásra és egyesre) van szükség mint a kettes számrendszerbeli alak esetében,
- Fibonacci kód:
  - egy szám Fibonacci számrendszerben, plusz a végére még írnak egy 1-est,
  - az ábrázolásban minden két 1-es érték között lesz legalább egy 0-ás,
  - legyen nr = 237, ekkor a Fibonacci kód: [1,0,0,0,0,0,0,0,0,1,0,1,1],
  - legyen nr = 100, ekkor a Fibonacci kód: [1,0,0,0,0,1,0,1,0,0,1].

# A Fibonacci számsorozat

- A számsorozat: $0, 1, 1, 2, 3, 5, 8, 13, \ldots$,
- A rekurziós képlet: $F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$,
- léteznek hatékonyabb rekurziós összefüggések,
- alkalmazásuk:
  - kombinatorika,
  - algoritmusok futási idejének elemzése,
  - minden pozitív egész szám felírható Fibonacci számok összegeként,
  - aranymetszés, zene, művészetek, természet.

A Lucas számok, ugyanaz az összefüggés a számok között, más a két kezdeti érték:

- A számsorozat: $2, 1, 3, 4, 7, 11, 18, 29, 47, \ldots$,
- Az egyik rekurziós képlet: $L_0 = 2, L_1 = 1, L_n = L_{n-1} + L_{n-2}$,

# Az n-ik Fibonacci szám

**1. feladat**

*Határozzuk meg az n-ik Fibonacci számot.*

A következő algoritmus futási ideje exponenciális, az előző oldalon megadott rekurzív képletet alkalmazza:

```
def fibR1 (n):
    if n == 0: return 0
    if n == 1: return 1
    return fibR1 (n-1) + fibR1 (n-2)

>>> fibR1(10)
    55
```

# Az n-ik Fibonacci szám

**2. feladat**

*Határozzuk meg az n-ik Fibonacci számot.*

A következő két algoritmus a `fibR2` és `fibR3` futási ideje lineáris:

```
def fAux(a, b, n):
    if n == 1: return a
    return fAux(b, a + b, n-1)
def fibR2(n):
    return fAux(1, 1, n)

>>> fibR2(100)
    354224848179261915075

def fibR3(n, a = 1, b = 1):
   if n == 1: return a
   return fibR3(n-1, b, a + b)

>>> fibR3(200)
    280571172992510140037611932413038677189525
```

# Az n-ik Fibonacci szám

**3. feladat**

*Határozzuk meg az n-ik Fibonacci számot:*

A következő képletet alkalmazó algoritmus futási ideje logaritmikus:

$$F_n = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1}, \qquad F_5 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{4} = \begin{pmatrix} 5 & 3 \\ 3 & 2 \end{pmatrix}$$

ahol meg kell tehát határozni két $2 \times 2$ mátrix szorzatát:

$$\begin{pmatrix} a & b \\ b & d \end{pmatrix} \cdot \begin{pmatrix} e & f \\ f & g \end{pmatrix} = \begin{pmatrix} a \cdot e + b \cdot f & a \cdot f + b \cdot g \\ a \cdot f + b \cdot g & b \cdot f + d \cdot g \end{pmatrix}$$

- fennáll: $a \cdot f + b \cdot g == b \cdot e + d \cdot f$
- a hatékonyságból, a mátrixot egy számhármassal lehet ábrázolni, mert a mellékátlón az elemek megegyeznek

# Az n-ik Fibonacci szám

```
def szor(t1, t2):
    a, b, d = t1
    e, f, g = t2
    return (a * e + b * f, a * f + b * g, b * f + d * g)

def fibR(n, x = (1,1,0), res = (1,0,1)):
    if n == 0: return res[1]
    if n % 2 == 1:
        res = szor(res, x)
        x = szor(x, x)
        return fibR(n // 2, x, res)
    x = szor(x, x)
    return fibR(n // 2, x, res)

>>> fibR(100)
    354224848179261915075
```

# Az n-ik Fibonacci szám

$(1,1,0)^{28}$, azaz a 29-ik Fibonacci szám meghatározása:

```
def fibR(n, x = (1,1,0), res = (1,0,1)):
    if n == 0: return res[1]
    if n % 2 == 1:
         res = szor(res, x)
         x = szor(x, x)
         return fibR(n // 2, x, res)
    x = szor(x, x)
    return fibR(n // 2, x, res)
```

| $x$ | $n$ | $res$ |
|---|---|---|
| | | $(1,0,1$ |
| $(1,1,0)$ | 28 | $(1,1,0)$ |
| $(1,1,0)^2 = (2,1,1)$ | 14 | $(1,1,0)$ |
| $(1,1,0)^4 = (5,3,2)$ | 7 | $(5,3,2)$ |
| $(1,1,0)^8 = (34,21,13)$ | 3 | $(233,144,89)$ |
| $(1,1,0)^{16} = (1597,987,610)$ | 1 | $(514229,317811,196418)$ |

Az eredmény:

$$F_{29} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{28} = \begin{pmatrix} 514229 & 317811 \\ 317811 & 196418 \end{pmatrix}$$

# Bitműveletek: &, |, ^

```
>>> x, y = 34, 48
>>> format(x, 'b')
    '100010'
>>> format(y, 'b')
    '110000'

>>> x & y #bitenkenti és (AND) művelet
    32
>>> format(32, 'b')
    '100000'

>>> x | y #bitenkenti vagy (OR) művelet
    50
>>> format(50, 'b')
    '110010'

>>> x ^ y #bitenkenti kizáró vagy (XOR) művelet
    18
>>> format(18, 'b')
    '10010'
```

| *x* | *y* | *AND(x,y)* | *OR(x,y)* | *XOR(x,y)* |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |

# Bitműveletek: <<

Szorzás 2-vel: a bináris alak, jobbról kiegészül egy 0-val:

```
>>> 17 * 2
    34
>>> 17 << 1
    34
>>> format(17, 'b'), format(34, 'b')
    ('10001', '100010')
```

Szorzás $2^k$-val: a bináris alak, jobbról kiegészül $k$ darab 0-val:

```
>>> 17 * 2**4
    272
>>> 17 << 4
    272
>>> format(17, 'b'), format(272, 'b')
    ('10001', '100010000')
```

# Bitműveletek: >>

Osztás 2-vel: a bináris alak bitértékei egy pozícióval, jobbra tolódnak:

```
>>> 59 // 2
    29
>>> 59 >> 1
    29
>>> format(59, 'b'), format(29, 'b')
    ('111011', '11101')
```

Osztás $2^k$-val: a bináris alak bitértékei k pozícióval jobbra tolódnak:

```
>>> 59 // 2**4
    3
>>> 59 >> 4
    3
>>> format(59, 'b'), format(3, 'b')
    ('111011', '11')
```

# Bitműveletek: maradékosztás 2-vel

Maradékos osztás, bitműveletekkel: ha 2-vel szeretnénk meghatározni az osztási maradékot, akkor a maradékos osztás helyett az & (és) bitműveletet használjuk, és a második operandus 1 lesz.

```
def bitm1(szam):
    if szam % 2:
        print ("paratlan szam")
    else: print ("paros szam")

def bitm2(szam):
    if szam & 1:
        print ("paratlan szam")
    else: print ("paros szam")
```

# Bitműveletek: maradékosztás $2^k$-val

Maradékos osztás, bitműveletekkel: ha $2^k$-val szeretnénk meghatározni az osztási maradékot, akkor a maradékos osztás helyett az & (és) bitműveletet használjuk, és a második operandus $2^k - 1$ lesz:

```
def bitm3(szam):
    temp = szam % 256
    if temp >= 32:
        print (chr(temp))
    else: print ("nem nyomtathato karakter")

def bitm4(szam):
    temp = szam & 255
    if temp >= 32:
        print (chr(temp))
    else: print ("nem nyomtathato karakter")
```