Diszkrét matematika
6. előadás

MÁRTON Gyöngyvér
mgyongyi@ms.sapientia.ro

Sapientia Egyetem,
Matematika-Informatika Tanszék
Marosvásárhely, Románia

2022, őszi félév

Miről volt szó az elmúlt előadáson?

- Python: a `complex` típus,
- valós számok, komplex számok, számrendszerek,
- algoritmusok:
  - polinom helyettesítési értéke, függvények ábrázolása,
  - a `sin, log` függvények,
  - műveletek komplex számokkal: szorzat, hatványozás, stb
  - fraktálok: Mandelbrot, Julia
  - számrendszerek közötti átalakítások

Miről lesz szó?

- Python: az `ord`, `chr` függvények, bitműveletek,
- a számrendszerek közötti kapcsolat,
- más számrendszerek:
  - vegyes alapú számrendszerek,
  - a faktoriális számrendszer,
  - a Fibonacci számrendszer,
- algoritmusok: az `n`-ik Fibonacci szám: különböző futás idejű algoritmusok,

Kapcsolat a $2, 8, 16$ számrendszerek között

- bináris $\to$ oktális: hármas csoportokat formálunk:

  Pl. $[1,1,1,1,0,1,1] \to [1,\quad 1,1,1,\quad 0,1,1] \to [1,7,3]$

- bináris $\to$ hexadecimális: négyes csoportokat formálunk

  Pl. $[1,1,1,1,0,1,1] \to [1,1,1,\quad 1,0,1,1] \to [7,B]$

- bináris $\to 2^k$: k-as csoportokat formálunk

  Pl. $2 \to 256 = 2^8$, 8-as csoportokat formálunk:
  $[1,1,1,0,0,1,0,1,1,0,1,0,1,1] \to [1,1,1,0,0,1,\quad 0,1,1,0,1,0,1,1] \to [57,107]$

Átalakítás 2-es számrendszerből 2^k számrendszerbe

1. feladat

Írjunk programot, amely egy 2-es számrendszerben megadott számot átalakít 2^k számrendszerbe.

```
def conv2To2K(L, k):
    nr = 0
    m = len(L) % k
    for elem in L[:m]:
        nr = nr * 2 + elem
    nL = []
    if nr != 0: nL = [nr]
    nr, i = 0, 0
    for elem in L[m:]:
        nr = nr * 2 + elem
        i += 1
        if i == k:
            nL = nL + [nr]
            nr, i = 0, 0
    return nL
```

Átalakítás 2-es számrendszerből 2^k számrendszerbe

Függvénymeghívás, 2-es számrendszerből $8 = 2^3$-as számrendszerbe:

```
>>> conv2To2K([1, 1, 0, 1, 0, 1], 3)
    [6, 5]
```

Függvénymeghívás, 2-es számrendszerből $256 = 2^8$-as számrendszerbe:

```
>>> conv2To2K([1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1], 8)
    [57, 107]
```

Átalakítás 2^k számrendszerből 2 – *es* számrendszerbe

2. feladat

Írjunk programot, amely egy 2^k számrendszerben megadott számot átalakít 2-es számrendszerbe.

```
def conv2From2K(L, k):
    nL = []
    for elem in L:
        sL = []
        for i in range(k):
            sL = [elem % 2] + sL
            elem = elem // 2
        nL += sL
    return nL
```

Átalakítás 2^k számrendszerből 2 – *es* számrendszerbe

Függvénymeghívás, $8 = 2^3$ -as számrendszerből 2-es számrendszerbe

```
>>> conv2From2K([6, 5], 3)
    [1, 1, 0, 1, 0, 1]
```

Függvénymeghívás, $256 = 2^8$-os számrendszerből 2-es számrendszerbe

```
>>> conv2From2K([57, 107], 8)
    [0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1]
```

A két függvényt együtt alkalmazva:

```
>>> conv2To2K(conv2From2K([217, 107, 28, 142, 55], 8), 8)
    [217, 107, 28, 142, 55]
```

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 faktoriális számrendszer

3. feladat

Határozzuk meg egy szám faktoriális számrendszerbeli számjegyeit.

```
def digitToFaktB(nr):
    L = []
    n = 2
    while nr != 0:
        q = nr // n
        a = nr - q * n #a = nr % n
        #print (nr, q, i, a, end = "\n")
        nr = q
        L = [a] + L
        n = n + 1
    return L

>>> digitToFaktB(45389)
    [1, 1, 0, 0, 1, 0, 2, 1]
```

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:

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

- 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.

A Fibonacci számrendszer

4. feladat

Határozzuk meg egy szám Fibonacci számrendszerbeli alakját. Az eredményt egy `str` *típusú változóba generáljuk ki.*

A *fibL* függvény, meghatározza, az A 1, 2-vel kezdődő Fibonacci számsorozat elemeit, amelyek nem nagyobbak mint *nr*, egy lista típusú változóba. Ezt meg kell írni!

```
def digitToFib(nr):
    fL = fibL(nr)
    L = ''
    for elem in fL[::-1]:
        if nr >= elem:
            L += '1'
            nr = nr - elem
            #print (elem, end = ' ')
        else: L += '0'
    #print('')
    return L
```

```
>>> digitToFib(100)
    1000010100

    100 = 89 + 8 + 3
```

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

Az n-ik Fibonacci szám

5. 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

6. 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

7. feladat

Alkalmazva a következő képletet határozzuk meg az n-ik Fibonacci számot:

$$F_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(1 - \frac{1+\sqrt{5}}{2}\right)^n}{\sqrt{5}} = \texttt{round}\left(\frac{\left(\frac{1+\sqrt{5}}{2}\right)^n}{\sqrt{5}}\right)$$

```
from math import sqrt
def fibF1(n):
    temp = sqrt(5)
    phi = (1 + temp)/2
    return round((phi ** n)/temp)
```

```
from decimal import Decimal, getcontext
def fibF2(n):
    temp = my_sqrt(5)
    phi = (1 + temp)/2
    return round((phi ** n)/temp)
```

```
def my_sqrt(n):
    getcontext().prec = 50
    temp = Decimal(0)
    for x in range (0, 200):
        temp = (n - 1) / (2 + temp)
    return 1 + temp
```

```
>>> fibF1(100)
354224848179263111168
>>> fibF2(100)
354224848179261915075
```

Az n-ik Fibonacci szám

8. feladat

Alkalmazva a következő képletet határozzuk meg az n-ik Fibonacci számot:

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

ahol meg kell tehát határozni két 2×2 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

A következő algoritmus futási ideje logaritmikus:

```
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 fibAuxR (x, n):
   if n == 0: return (1,0,1)
   xS = szor(x, x)
   if n % 2 == 1:
      temp = szor(x, fibAuxR (xS, n // 2))
      return temp
   return fibAuxR (xS, n // 2)

def fibR(n):
   x = fibAuxR((1,1,0), n)
   return x[1]

>>> fibR(100)
    354224848179261915075
```

Az n-ik Fibonacci szám

A következő, az előző algoritmus iteratív változata, ahol a szor függvény is az előző oldalon van megadva:

```
def fib(n):
    res = (1, 0, 1)
    alap = (1, 1, 0)
    n -= 1
    while n != 0:
        if n % 2 == 1: res = szor(res, alap)
        n = n // 2
        alap = szor(alap, alap)
    return res[0]
```

```
>>> fib(100)
    354224848179261915075
```

Az n-ik Fibonacci szám

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

```
def fib(n):
        res = (1, 0, 1)
        alap = (1, 1, 0)
        n -= 1
        while n != 0:
            if n % 2 == 1: res = szor(res, alap)
            n = n // 2
            alap = szor(alap, alap)
        return res[0]
```

alap	*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:

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

Python, az ord, chr függvények

A számítástechnikában használt karakterekhez, egy kódolási eljárás segítségével, számokat rendelnek. A Pythonban az `ord` megadja egy adott karakter kódját, a `chr`, pedig meghatározza a kódértékhez tartozó karaktert.

```
def fugv1(L):
    nL = []
    for elem in L:
        nL += [ord(elem)]
    return nL

def fugv2(L):
    nL = ""
    for elem in L:
        nL += chr(elem)
    return nL

>>> fugv1("matematika informatika tanszek")
    [109, 97, 116, 101, 109, 97, 116, 105, 107, 97, 32, 105, ...]
>>> fugv2([109, 97, 116, 101, 109, 97, 116, 105, 107, 97])
    'matematika'
```

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: $\sim$

```
>>> x = 4
>>> y = ~x
>>> y
    -5
>>> format(x,  'b')
    '100'
>>> format(y, 'b')
    '-101'
```

C++:

```
#include <iostream>
using namespace std;
int main() {
    unsigned char x = 4;
    unsigned char y = ~x;
    cout << hex << (0xFF & x) << " ";
    cout << hex << (0xFF & y) << " ";
    cout << dec << (int)y << endl;
}
```

Megjegyzések:

- a C++ futtatás eredménye: `4 fb 251`,
- a negatív számok tárolása a memóriában 1-es, illetve 2-es komplemens alakban történik,
- a Python és a C++ negatív számok esetében 2-es komplemens szerint határozza meg a $\sim$ művelet eredményét,
- 1-es komplemens: minden 0-ás bit 1-re, és minden 1-es bit 0-ra cserélődik,
- 2-es komplemens: az 1-es komplemenshez hozzáadunk 1-et,
- Pythonban $-5_{10} = -101_2$ a kiírt érték, mert $4_{10} = 100_2 = -(100+1)_2 = -101_2 = -5_{10}$, ez tulajdonképpen egyenlő $\ldots 111111011_2$-vel, de ez nem használható, nem is portábilis (az `Int` mérete géptől függ),
- C++-ban, `char` típus esetében $251_{10} = 11111011_2$ az eredmény,
- ritkán használják az 1-es komplemenst

1-es komplemens		2-es komplemens	
		1111 1011	-5
1111 1011	-4	1111 1100	-4
1111 1100	-3	1111 1101	-3
1111 1101	-2	1111 1110	-2
1111 1110	-1	1111 1111	-1
1111 1111	-0		
0000 0000	+0	0000 0000	+0
0000 0001	+1	0000 0001	+1
0000 0010	+2	0000 0010	+2
0000 0011	+3	0000 0011	+3
0000 0100	+4	0000 0100	+4

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 pozicióval, 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 pozicióval 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")
```