Diszkrét matematika
3. előadás

MÁRTON Gyöngyvér
https://ms.sapientia.ro/∼mgyongyi/
mgyongyi@ms.sapientia.ro

Sapientia EMTE,
Matematika-Informatika Tanszék
Marosvásárhely, Románia

2023, őszi félév

Miről volt szó az elmúlt előadáson?

- Python: az `in` operátor, logikai kifejezések, függvények, a `tuple`, a `list`, az `str` adattípus, szeletelések, adatbevitel, `print` - a kimenet formázása,
- Algoritmusok:
  - tesztelés: egy szám négyzetszám-e,
  - osztók száma, osztók meghatározása,
  - nullások száma a faktoriális végén.

Miről lesz szó?

- Python: hibakezelés, függvények paraméterátadása,
- természetes számok, egész számok,
- algoritmusok futási ideje,
- maradékos osztás,
- Algoritmusok:
  - minimum keresés,
  - gyorshatványozás - iteratív, rekurzív változatok, szorzások száma
  - a faktoriális függvény - iteratív, rekurzív változatok,
  - legnagyobb közös osztó - iteratív, rekurzív változatok,
  - a kiterjesztett euklideszi algoritmus,
  - diofantoszi egyenletek.

Python adatbevitel

1. feladat

Írjunk egy Python függvényt, amely meghatározza a beolvasott elemek legkisebbikét.

```
def beolvasFloat(n):
    L = []
    for i in range(n):
        L += [float(input('elem = '))]
    return L

def sMinP():
    n = int(input('n = '))
    L = beolvasFloat(n)
    m = L[0]
    for elem in L:
        if elem < m: m = elem
    return m
```

```
def beolvasStr(n):
    L = []
    for i in range(n):
        L += [(input('elem = '))]
    return L
```

Python hibakezelés

Ha az adabevitel során nem megfelelő értéket olvasunk be egy adott változóba, akkor futási hiba adódik:

```
>>> beolvasFloat(4)
    elem = szoveg
    ...
    ValueError: could not convert string to float: 'szoveg'
```

Az ilyen típusú hibák elkerülése végett a megfelelő műveletsorokat egy `try...except` utasításblokk közé kell írni:

```
def beolvasFloat(n):
    L = []
    for i in range(n):
        try:
            L += [float(input("elem = "))]
        except:
            print("hibas bemenet")
            return
    return L
```

- az `try` részbe azokat a műveletsorokat kell írni, amelyek futási hibát okozhatnak,
- az `except` részbe pedig a hibaüzenetet, illetve a függvényből való kilépést kell megadni.

Python függvények paraméterátadása

```
def foF1():
    db = 0
    segedF1(db)
    print (db)

def segedF1(x):
    x += 10
```

```
>>> foF1()
    0
```

A `foF1` függvényben nem látjuk, hogy a `segedF1` függvényben megváltoztattuk az `X` paraméter értékét.

```
def foF2():
    db = 0
    db = segedF2(db)
    print (db)

def segedF2(x):
    x += 10
    return x
```

```
>>> foF2()
    10
```

A `foF2` függvényben látjuk, hogy a `segedF2` függvényben megváltoztattuk az `X` paraméter értékét.

Számok

- számtartományok: természetes számok, egész számok, racionális számok, irracionális számok, valós számok, komplex számok,
- alapműveletek számokkal: összegzés, különbség, szorzás, osztás, hatványozás, logaritmálás,
- természetes számok, egész számok: a diszkrét matematika gerincét alkotják,

Természetes számok

- halmazjelölés: $\mathbb{N} = \{0, 1, 2, 3, \dots\}$,
- tulajdonságok:
  - az összeadás, szorzás kommutatív: $a, b \in \mathbb{N}, a + b = b + a, a \cdot b = b \cdot a$
  - az összeadás, szorzás asszociatív: $a, b, c \in \mathbb{N}, (a + b) + c = a + (b + c), (a \cdot b) \cdot c = a \cdot (b \cdot c)$
  - az összeadás a szorzásra nézve disztributív: bármely $a, b, c \in \mathbb{N}, (a + b) \cdot c = a \cdot c + b \cdot c$
  - a természetes számok halmaza zárt az összeadásra, szorzásra nézve: bármely két természetes szám összeadható, szorozható az eredmény szintén természetes szám lesz, ez nem igaz a különbség és osztás műveletekre,
  - a 0 az összeadásra nézve semleges, az 1 a szorzásra nézve semleges elem,
  - jól rendezettség: a természetes számok minden nem üres részhalmazának van egy legkisebb eleme.

A faktoriális függvény - iteratív megoldások

2. feladat

Határozzuk meg n! értékét, azaz az összes n-nél kisebb pozitív szám szorzatát.

for ciklusutasítással:

```
def factorialFor(n):
    res = 1
    for i in range(1, n+1):
        res *= i
    return res
```

Könyvtárfüggvénnyel:

```
>>> from math import factorial
>>> factorial(10)
3628800
```

while ciklusutasítással:

```
def factorialWhile(n):
    res = 1
    while n >= 1:
        res = res * n
        n = n - 1
    return res
```

A faktoriális függvény - rekurzív megoldások

```
def factorialRek1(n):
    if n == 0: return 1
    return n * factorialRek1(n-1)

def factorialRek2(n):
    return factorialAux(n, 1)

def factorialAux(n, res):
    if n == 0: return res
    return factorialAux(n - 1, res * n)

def factorialRek3(n, res = 1):
    if n == 0: return res
    return factorialRek3(n - 1, res * n)
```

- a `factorialRek1` a következő sorrendbe szorozza össze a számokat: $1 \cdot 2, 1 \cdot 2 \cdot 3, 1 \cdot 2 \cdot 3 \cdot 4$, *stb*, azaz akkor számol mikor jön vissza a rekurzióból
- a `factorialRek2` feltételezve, hogy $n = 10$, a következő sorrendbe szorozza össze a számokat: $1 \cdot 10, 1 \cdot 10 \cdot 9, 1 \cdot 10 \cdot 9 \cdot 8$, *stb.*, azaz akkor számol mikor megy be a rekurzióba

A hatványozás algoritmusa

3. feladat

Írjunk egy Python függvényt, amely meghatározza x^n értékét.

- többféle megoldás létezik,
- a legkézenfekvőbb algoritmus az, ha `n`-szer összeszorozzuk az `x`-et, ezt a gondolatmenetet követi a `myPowLin` függvény:

```
def myPowLin(x, n):
    res = 1
    for i in range(n):
        res *= x
    return res
```

- az algoritmus futási ideje azonban nem jó, nagy számokra lassú,
- lemérhetjük a futási időt különböző bemenetekre: ha a hatványkitevő 500000 akkor már több másodpercig is eltarthat a számolási idő,
- megszámolhatjuk hány alapműveletet (szorzást) végez az algoritmus: ha a hatványkitevő 100, akkor 100 szorzást végez,
- az algoritmus futási ideje lineáris.

A gyorshatványozás algoritmusa

- x^n értékét meghatározhatjuk a gyorshatványozás algoritmusával: ha a hatványkitevő 100, akkor 9 darab szorzást végez az algoritmus,
- az algoritmus futási ideje logaritmikus

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

```
>>> myPow(2, 100)
    1267650600228229401496703205376L
```

```
def myPow1(x, n):
    res = 1
    while True:
        if n % 2 == 1:
            res = res * x
        n = n // 2
        if n == 0: break
        x = x * x
    return res
```

A gyorshatványozás algoritmusa

3^{100} meghatározása:

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

x	n	res
		1
3	100	1
$3^2 = 9$	50	1
$3^4 = 81$	25	81
$3^8 = 6561$	12	81
$3^{16} = 43046721$	6	81
$3^{32} = 1853020188851841$	3	150094635296999121
$3^{64} = 3433683820292512484657849089281$	1	515377520732011331036461129765621272702107522001
$3^{128} = 117\ldots961$	0	

Az eredmény: $3^{100} = 3^4 \cdot 3^{32} \cdot 3^{64} = 515377520732011331036461129765621272702107522001$.

A gyorshatványozás algoritmusa - rekurzív változatok

```
def myPowRek1(x, n):
    if n == 0: return 1
    if n % 2 == 1: return x * myPowRek1(x * x, n // 2)
    return myPowRek1 (x * x, n // 2)

def myPowRek2(x, n):
    if n == 0: return 1
    temp = myPowRek2(x, n // 2)
    if n % 2 == 1: return x * temp * temp
    else: return temp * temp
```

Futási idők

4. feladat

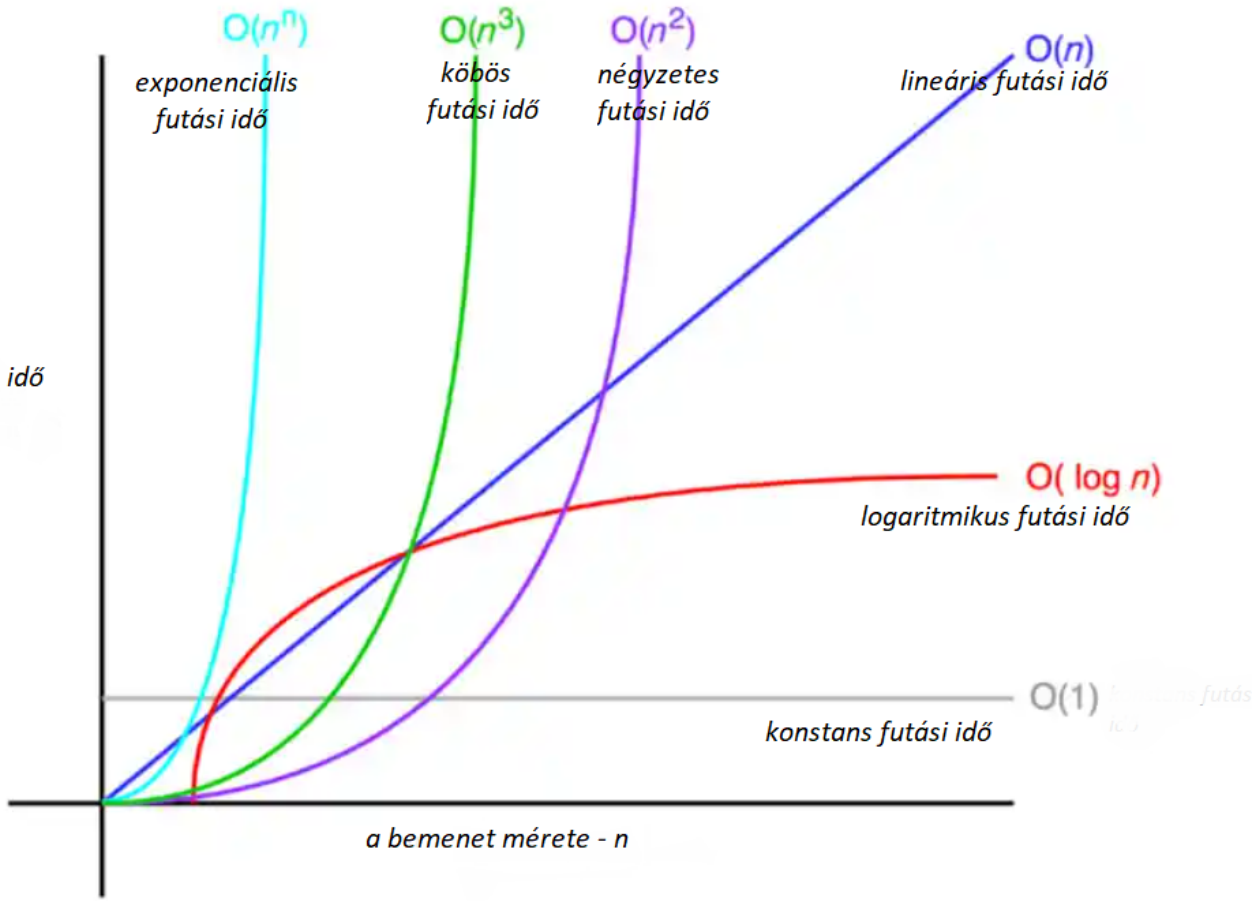
Írjunk Python függvényt, amely meghatározza különböző nagyságrendű bemenetek esetén a `myPowLin` és `myPow` függvények futási idejét.

```
from time import time

def mainTimeLog(x):
    for n in range(1, 10):
        st = time()
        myPow(x, 10**n)
        fs = time()
        print("kitevo: 10^",n, ", futasi ido:", fs - st)

def mainTimeLin(x):
    for n in range(1, 7):
        st = time()
        myPowLin(x, 10**n)
        fs = time()
        print("kitevo: 10^",n, ", futasi ido:", fs - st)
```

Futási idők


O(n^n)
exponenciális futási idő
O(n^3)
köbös futási idő
O(n^2)
négyzetes futási idő
O(n)
lineáris futási idő
idő
O(log n)
logaritmikus futási idő
O(1)
konstans futási idő
a bemenet mérete - n

Egész számok

- halmazjelölés: $\mathbb{Z} = \{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\}$,
- tulajdonságok:
  - $\mathbb{N} \subset \mathbb{Z}$, és a két halmaz számossága ugyanaz,
  - két halmaz számossága, akkor egyezik meg, ha a két halmaz között létezik egy bijekció, pl: $f(x) = \begin{cases} 2x & \text{ha } x \geq 0, \\ -2x - 1 & \text{ha } x < 0, \end{cases}$
  - kommutatívitás, asszociativitás, disztributivitás,
  - az egész számok halmaza zárt az összeadásra, kivonásra, szorzásra nézve, de ez nem igaz az osztásra,
  - az összeadásra nézve minden elemnek van inverz eleme,
  - rendezettség: $a \leq b$, ha $b - a \in \mathbb{N}$.

Oszthatósági problémák

Legyen $a, b \in \mathbb{Z}$, ahol $b \neq 0$. a osztja b-t, ha létezik olyan c egész szám, amelyre fennáll: $b = a \cdot c$, jelölése: $a \mid b$. Azt az esetet, amikor a nem osztja b-t $a \nmid b$-vel jelöljük.

1. tétel (A maradékos osztás tétele)

Legyen a egy egész szám, c pedig egy pozitív egész szám. Ekkor egyértelműen létezik két olyan q és r egész szám, amelyekre igaz $a = c \cdot q + r$, $0 \leq r < c$.

- Az a és b egész számok legnagyobb közös osztója az a legnagyobb pozitív egész szám, amely osztja az a-t és b-t is. Jelölése: $lnko(a, b)$, (a, b), vagy $\gcd(a, b)$.
- Az a és b egész számok legkisebb közös többszöröse az a legkisebb pozitív egész szám amely osztható a-val és b-vel is. Jelölése: $lkkt(a, b)$, vagy $[a, b]$.
- Az a és b egész számok szorzata egyenlő az a és b legnagyobb közös osztójának és legkisebb közös többszörösének a szorzatával:

$$(a, b) \cdot [a, b] = a \cdot b.$$

Oszthatósági problémák

- A legnagyobb közös osztó meghatározására több algoritmus is ismert, az egyik az euklideszi algoritmus. Python-ban a `math` modulban található `gcd` függvény használható két szám legnagyobb közös osztójának a meghatározására:

```
>>> from math import gcd
>>> gcd(60, 45)
    15
>>> gcd(1789, 100)
    1
>>> gcd(-63, 45)
    9
```

- Azt mondjuk, hogy a és b relatív prímek, ha a legnagyobb közös osztójuk 1. A fenti példában 1789 és 100 relatív prímek, míg 60 és 45, illetve -63 és 45 nem.

2. tétel

Legyenek a, b, q, r egész számok, $a = b \cdot q + r$ és legyen $(a, b) = d$, azaz a és b legnagyobb közös osztója d. Ekkor igaz az, hogy: $d = (a, b) = (b, a - b \cdot q)$.

Megjegyzés:

```
a - b * q = a % b
```

Az euklideszi algoritmus

Legyenek a, b pozitív egész számok, ahol $a \geq b$ és legyen $r_0 = a$, $r_1 = b$:

$$
\begin{array}{rcll}
r_{\mathbf{0}} & = & r_{\mathbf{1}} \cdot q_{\mathbf{1}} + r_{\mathbf{2}} & 0 \leq r_{\mathbf{2}} < r_{\mathbf{1}}, \\
r_{\mathbf{1}} & = & r_{\mathbf{2}} \cdot q_{\mathbf{2}} + r_{\mathbf{3}} & 0 \leq r_{\mathbf{3}} < r_{\mathbf{2}}, \\
 & . & & \\
 & . & & \\
 & . & & \\
r_{n-\mathbf{2}} & = & r_n - 1 \cdot q_{n-\mathbf{1}} + r_n & 0 \leq r_n < r_{n-\mathbf{1}}, \\
r_{n-\mathbf{1}} & = & r_n \cdot q_n. &
\end{array}
$$

$$
\begin{array}{rcll}
(32, 76) & = & (76, 32 - 0 \cdot 76) & = (76, 32) \\
(76, 32) & = & (32, 76 - 2 \cdot 32) & = (32, 12) \\
(32, 12) & = & (12, 32 - 2 \cdot 12) & = (12, 8) \\
(12, 8) & = & (8, 12 - 1 \cdot 8) & = (8, 4) \\
(8, 4) & = & (4, 8 - 2 \cdot 4) & = (4, 0) = 4
\end{array}
$$

A legnagyobb közös osztó egyenlő lesz a fenti számítási sorozat során meghatározott utolsó nem nullás maradékkal:

$$(a, b) = (r_{\mathbf{0}}, r_{\mathbf{1}}) = (r_{\mathbf{1}}, r_{\mathbf{2}}) = \ldots = (r_{n-\mathbf{2}}, r_{n-\mathbf{1}}) = (r_{n-\mathbf{1}}, r_n) = (r_n, r_{\mathbf{0}}) = r_n.$$

Az euklideszi algoritmus

5. feladat

Írjunk egy Python függvényt, amely az euklideszi algoritmus meghatározza két egész szám legnagyobb közös osztóját.

```
def lnkoF_(a, b):
    while b != 0:
        r = a % b
        a = b
        b = r
    return a

>>> lnkoF(-32, -76)
    4
>>> lnkoF(32.7, 76)
    'hibás bemenet'
```

```
def lnkoF(a, b):
    if not isinstance(a, int) or not isinstance(b, int):
        return 'hibás bemenet'
    a = abs(a)
    b = abs(b)
    while True:
        r = a % b
        if r == 0: break
        a = b
        b = r
    return b
```

Az euklideszi algoritmus

6. feladat (Rekurzív változat)

Írjunk egy Python függvényt, amely az euklideszi algoritmussal, rekurzívan határozza meg két egész szám legnagyobb közös osztóját.

```
def lnkoR(a, b):
    temp = a % b
    if temp == 0: return b
    return lnkoR(b, temp)
```

Az euklideszi algoritmus futási idejét az osztások száma határozza meg, ezzel kapcsolatban megadható a következő tétel:

3. tétel (Lamé tétele)

Az euklideszi algoritmus során végzett osztások száma nem lesz nagyobb, mint ötször a kisebbik szám számjegyeinek száma

Az euklideszi algoritmus

7. feladat

Írjunk egy Python függvényt, amely meghatározza egy adott racionális szám irreducibilis alakját.

Az $\frac{a}{b}$ racionális számot az `(a, b)` értékpárral fogjuk jelölni, azaz `tuple` típusú adatként kezeljük. Egy tört irreducibilis alakját úgy határozzuk meg, hogy kiszámoljuk a számláló és nevező legnagyobb közös osztóját, majd végig osztunk ezzel az értékkel.

```
def iAlak(a, b):
    temp = lnkoF(a, b)
    return (a // temp, b // temp)
```

A kiterjesztett euklideszi algoritmus

- a d legnagyobb közös osztó valamely x, y egész számokkal mindig felírható az a és b lineáris kombinációjaként:
$$d = x \cdot a + y \cdot b,$$
- az euklideszi algoritmus meghatározza a d legnagyobb közös osztót, a kiterjesztett euklideszi algoritmus pedig az x, y egész számokat is,
- a kiterjesztett euklideszi algoritmus futási ideje logaritmikus éppen ezért fontos szerepet tölt be a számítógépes adatbiztonság területén

Legyenek a, b pozitív egész számok, ekkor felírható a következő összefüggés:

$$(a, b) = x_n \cdot a + y_n \cdot b,$$

ahol x_n és y_n a következő számítási sorozat n-ik tagjai, ahol $j \in \{2, 3, \ldots, n\}$,

$$\begin{aligned} x_0 &= 1, \ y_0 = 0 \\ x_1 &= 0, \ y_1 = 1 \\ x_j &= x_{j-2} - q_{j-1} \cdot x_{j-1} \\ y_j &= y_{j-2} - q_{j-1} \cdot y_{j-1}, \end{aligned}$$

Megállapítható, hogy minden $j = \{2, 3, \ldots, n\}$-re igaz a következő is:

$$r_j = x_j \cdot a + y_j \cdot b.$$

A r_j, illetve q_j értékek az iteráció során kiszámolt osztási maradékok, illetve osztási egész részek.

A kiterjesztett euklideszi algoritmus, példa

Határozzuk meg 76 és 32 legnagyobb közös osztóját, majd azokat az x és y egész számokat, melyekre fennáll a következő összefüggés: $76 \cdot x + 32 \cdot y = d$, ahol $d = (76,\ 32)$.

a	b	q	r	x_0	x_1	y_0	y_1	
				1	0	0	1	
76	32	2	12	0	1	1	-2	$1 \cdot 76 + (-2) \cdot 32 = 12$
32	12	2	8	1	-2	-2	5	$(-2) \cdot 76 + 5 \cdot 32 = 8$
12	8	1	4	-2	3	5	-7	$3 \cdot 76 + (-7) \cdot 32 = 4$
8	4	2	0					

Tehát a megoldás: $d = 4,\ x = 3,\ y = -7$, és fennáll a következő összefüggés: $3 \cdot 76 + (-7) \cdot 32 = 4$.

A kiterjesztett euklideszi algoritmus, példa

Határozzuk meg 15 és 56 legnagyobb közös osztóját, majd azokat az x és y egész számokat, melyekre fennáll a következő összefüggés: $15 \cdot x + 56 \cdot y = d$, ahol $d = (15,\ 56)$.

a	b	q	r	x_0	x_1	y_0	y_1
				1	0	0	1
15	56	0	15	0	1	1	0
56	15	3	11	1	-3	0	1
15	11	1	4	-3	4	1	-1
11	4	2	3	4	-11	-1	3
4	3	1	1	-11	15	3	-4
3	1	3	0				

Tehát a megoldás: $d = 1,\ x = 15,\ y = -4$, és fennáll a következő összefüggés: $15 \cdot 15 + 56 \cdot (-4) = 1$.

A kiterjesztett euklideszi algoritmus

8. feladat

Írjunk egy Python függvényt, amely a kiterjesztett euklideszi algoritmusával határozza meg az a és b legnagyobb közös osztóját, a d-t és azokat az x, y egész számokat amelyekre fennáll: $d = x \cdot a + y \cdot b$.

```
def extEuclid(a, b):
    x0, x1, y0, y1 = 1, 0, 0, 1
    while True:
        q = a // b
        r = a - b * q
        if r == 0:
            return b, x1, y1
        x = x0 - q * x1
        y = y0 - q * y1
        x0, x1, y0, y1 = x1, x, y1, y
        a, b = b, r

>>> extEuclid(15, 56)
    (1, 15, -4)
```

A kiterjesztett euklideszi algoritmus

Kiegészítjük az előző függvényt, hogy kiírassuk a részértékeket:

```
def extEuclid_(a, b):
    x0, x1, y0, y1 = 1, 0, 0, 1
    print("%4s%4s%4s%4s%4s%4s%4s%4s" % ("a","b","q","r","x0","x1","y0","y1"))
    print("%4s%4s%4s%4s%4i%4i%4i%4i" % ("","","","",x0,x1,y0,y1))
    while True:
        q = a // b
        r = a - b * q
        if r == 0:
            print("%4i%4i%4i%4i" % (a,b,q,r))
            return b, x1, y1
        x = x0 - q * x1
        y = y0 - q * y1
        x0, x1, y0, y1 = x1, x, y1, y
        print("%4i%4i%4i%4i%4i%4i%4i%4i" % (a,b,q,r,x0,x1,y0,y1))
        a, b = b, r

>>> extEuclid_(15, 56)
    a   b   q   r  x0  x1  y0  y1
                   1   0   0   1
    ...
```

Diofantoszi egyenletek

- egész együtthatós többismeretlenes algebrai egyenletek, amelyeknek megoldásai egész számok (ritkán természetes vagy racionális számok),
- elnevezése Diophantosz (3. század), görög matematikus után

4. tétel

Legyenek a, m egész számok, úgy hogy $lnko(a, m) = d$. Ha $d \not| b$, azaz ha d nem osztja b-t, akkor az $a \cdot x + m \cdot y = b$ egyenletnek nincs megoldása az egész számok körében, Ha $d \mid b$, akkor az egyenletnek végtelen sok megoldása van.

- első lépésben, kiterjesztett euklideszi algoritmussal meghatározzuk a $d, \hat{x}, \hat{y}$-t, úgy hogy teljesüljön: $a \cdot \hat{x} + m \cdot \hat{y} = d$,
- az egyenlet első megoldása: $x_0 = \hat{x} \cdot (b/d), y_0 = \hat{y} \cdot (b/d)$,
- az egyenlet többi megoldásait az $n = \ldots, -2, -1, 0, 1, 2, \ldots$ egész számok alapján a következőképpen számoljuk ki:

$$x = x_0 + n \cdot (m/d), y = y_0 - n \cdot (a/d).$$

Diofantoszi egyenletek

- nagyon gyakran egy elsőfokú, kétismeretlenes diofantoszi egyenlet pozitív megoldásait kell meghatározni
- Például: Egy elárusító 1676 ron értékben rendelt almát és körtét. Minden láda alma 36 ronba, és minden láda körte 50 ronba kerül. Hány láda almát és hány láda körtét rendelhetett?
- tulajdonképpen a $36 \cdot x + 50 \cdot y = 1676$ diofantoszi egyenlet pozitív megoldásait kell megkeresnünk.
- egy $a \cdot x + m \cdot y = b$ egyenlet pozitív megoldásai esetében fenn kell álljon:

$$x_0 + n \cdot \frac{m}{d} \geq 0 \Rightarrow n \geq \frac{-x_0}{\frac{m}{d}}$$

$$y_0 - n \cdot \frac{a}{d} \geq 0 \Rightarrow n \leq \frac{y_0}{\frac{a}{d}}$$

- keressük tehát azokat az n természetes számokat, amelyek kielégítik a fenti két egyenlőtlenséget

Diofantoszi egyenletek, példa

A $36 \cdot x + 50 \cdot y = 1676$ diofantoszi egyenlet pozitív megoldásait keressük, ahol a megoldás menete a következő:

- a kiterjesztett euklideszi algoritmussal meghatározzuk: $d = 2, \hat{x} = 7, \hat{y} = -5$ értékeket, azaz fennáll $7 \cdot 36 + (-5) \cdot 50 = 2$
- megvizsgáljuk, hogy $d = 2$ osztja-e 1676-ot
- meghatározzuk:

$$x_0 = \hat{x} \cdot \frac{b}{d} = 7 \cdot \frac{1676}{2} = 7 \cdot 838 = 5866$$

$$y_0 = \hat{y} \cdot \frac{b}{d} = -5 \cdot \frac{1676}{2} = -5 \cdot 838 = -4190$$

- fennáll:

$$5866 \cdot 36 - 4190 \cdot 50 = 1676$$

Diofantoszi egyenletek

- Keressük a pozitív megoldásokat, fenn kell álljon:

$$5866 + n \cdot \frac{50}{d} = 5866 + n \cdot 25 \quad \geq 0 \quad \Rightarrow n \quad \geq -234.64$$

$$-4190 - n \cdot \frac{36}{d} = -4190 - n \cdot 18 \quad \geq 0 \quad \Rightarrow n \quad \leq -232.7$$

$$\begin{aligned} \Rightarrow \quad n &= -234 \\ x_1 &= 5866 + 25 \cdot (-234) = 16 \\ y_1 &= -4190 - 18 \cdot (-234) = 22 \end{aligned}$$

$$\begin{aligned} \Rightarrow \quad n &= -233 \\ x_2 &= 5866 + 25 \cdot (-233) = 41 \\ y_2 &= -4190 - 18 \cdot (-233) = 4 \end{aligned}$$

Tehát:

- 1. megoldás: 16 láda almát és 22 láda körtét rendelt, $16 \cdot 36 + 22 \cdot 50 = 1676$.
- 2. megoldás: 41 láda almát és 4 láda körtét rendelt, $41 \cdot 36 + 4 \cdot 50 = 1676$.

Diofantoszi egyenletek

9. feladat

Írjunk egy Python függvényt, amely meghatározza egy kétismeretlenes diofantikus egyenlet pozitív megoldásait.

```
def diofant (a, m, b):
    (d, xk, yk) = extEuclid (a, m)
    if b % d != 0:
        print ('nincs megoldas')
        return
    x0 = xk * b//d
    y0 = yk * b//d
    bd = m//d
    ad = a//d
    n1 = int (ceil(-x0 / bd))
    n2 = int (floor(y0 / ad))
    if n1 <= n2:
        print('megoldasok:')
        for i in range(n1, n2 + 1):
            print (x0 + bd * i, y0 - ad * i)
    else: print('nincs pozitiv megoldas')
```

```
>>> diofant(36, 50, 1676)
...
```