

Diszkrét matematika

7. előadás

MÁRTON Gyöngyvér
mgyongyi@ms.sapientia.ro

Sapientia Egyetem,
Matematika-Informatika Tanszék
Marosvásárhely, Románia

2022, őszi félév



Miről volt 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,

Miről lesz szó?

- Python: , az `index` metódus, az `str`, `bytes` típusok, átalakítások, a `random` modul, bináris állományok,
- kódolási technikák: ASCII, UTF-8, Unicode,
- algoritmusok: a Hamming súly, titkosítás (az XOR egy alkalmazása),

Bitműveletek

1. feladat

Hány 1-es bitet tartalmaz egy természetes szám kettes számrendszerbeli alakja?

Más megfogalmazásban határozzuk meg a Hamming-súlyát egy adott számnak. Nem kell összetéveszteni a Hamming távolsággal!

```
def HammingW1(nr):  
    db = 0  
    while nr:  
        db += (nr & 1)  
        nr >>= 1  
    return db
```

```
>>> HammingW1(1023)  
10  
>>> HammingW1(1024)  
1
```

```
def HammingW2(nr):  
    db = 0  
    while nr:  
        nr &= nr - 1  
        db += 1  
    return db
```

```
>>> HammingW2(1023)  
10  
>>> HammingW2(1024)  
1
```

Bitműveletek

Az előző oldalon megadott algoritmusok közül melyik a hatékonyabb? Számoljuk meg, hogy hányszor kerülnek végrehajtásra a while-ban levő utasítások.

```
def HammingW1(nr):  
    db = 0  
    itNr = 0  
    while nr:  
        db += (nr & 1)  
        nr >>= 1  
        itNr += 1  
    return db, itNr
```

```
>>> HammingW1(1023)  
(10, 10)  
>>> HammingW1(1024)  
(1, 11)
```

```
def HammingW2(nr):  
    db = 0  
    itNr = 0  
    while nr:  
        nr &= nr - 1  
        db += 1  
        itNr += 1  
    return db, itNr
```

```
>>> HammingW2(1023)  
(10, 10)  
>>> HammingW2(1024)  
(1, 1)
```

Bitműveletek

kiíratjuk a részadatok értékét:

```
def HammingW1(nr):  
    db = 0  
    while nr:  
        db += (nr & 1)  
        print(format(nr, 'b'))  
        print(format(nr & 1, 'b'))  
        print()  
        nr >>= 1  
    return db
```

```
>>> HammingW1(59)  
111011  
1  
  
11101  
1  
  
1110  
0  
  
111  
1  
  
11  
1  
  
1  
1  
  
5
```

Bitműveletek

kiíratjuk a részadatok értékét:

<pre>def HammingW2(nr): db = 0 while nr: print(format(nr, 'b')) print(format(nr-1, 'b')) print(format(nr & (nr-1), 'b')) print() nr &= nr - 1 db += 1 return db</pre>	<pre>>>> HammingW2(63) 111111 111110 111110 111110 111101 111100 111100 111011 111000 111000 101111 100000 100000 011111 000000 6</pre>
<pre>>>> int('111110', 2) 62 >>> int('111101', 2) 61</pre>	
<pre>>>> int('111100', 2) 60 >>> int('111011', 2) 59</pre>	
<pre>>>> int('111000', 2) 56 >>> int('110111', 2) 55</pre>	

Python, a random könyvtárcsomag

2. feladat

Generáljunk véletlenszerűen n darab egész számot, ahol $k_1 \leq nr \leq k_2$.

```
import random
def myRand1(n, k1, k2):
    L = []
    for i in range(0, n):
        nr = random.randint(k1, k2)
        L += [nr]
    return L

>>> myRand1(10, 0, 100)
[47, 13, 33, 88, 98, 75, 67, 70, 25, 76]

>>> myRand1(10, 65, 75)
[75, 66, 65, 67, 69, 72, 67, 66, 69, 74]
```


Python, a random könyvtárcsomag

3. feladat

Generáljunk véletlenszerűen n darab **különböző** egész számot, ahol $k1 \leq nr \leq k2$.

```
def myRand2(n, k1, k2):  
    i = 0  
    L = []  
    while True:  
        nr = random.randint(k1, k2)  
        if not nr in L:  
            L += [nr]  
            i += 1  
            if i == n: return L  
  
>>> myRand2(10, 1, 100)  
[1, 95, 2, 15, 72, 84, 47, 9, 10, 80]
```

Python, a random könyvtárcsomag

4. feladat

*Generáljunk véletlenszerűen n darab karaktert, amelyek nr kódjára fennáll:
 $k1 \leq nr \leq k2$.*

```
def myRand3(n, k1 = 65, k2 = 65 + 25):  
    L = ''  
    for i in range(0, n):  
        temp = chr(random.randint(k1, k2))  
        L += temp  
    return L
```

```
>>> myRand3(10)  
'AUQXEVJLRQ'
```

```
>>> myRand3(10, 97, 97+25)  
'khobnyeqin'
```

```
>>> myRand3(10, 50000, 53000)
```

Bináris állományok

5. feladat

Írjunk programot, amely megjeleníti egy állomány bájtjait hexa számrendszerben.

```
def fileHexa():
    fnev = input('kerek egy allomanynevet:')
    try:
        inf = open(fnev, 'rb')
        bajtL = inf.read()
        inf.close()
    except:
        print('megnyitasi/olvasasi hiba!!!')
        return
    out = open('hexImage.txt', 'wt')
    bajtR = ''
    for bajt in bajtL:
        bajtR += format(bajt, 'x') + ' '
    out.write(bajtR)
    out.close()
```

Kódolási technikák

Többféle kódolási technika létezik, ahol a kódolás a feldolgozandó adat egy átalakítási módját jelenti. Kódolási módszerek, kódok:

- gépi kód: bináris formában tárolja az adatokat és utasításokat,
- Ascii kód, Unicode, UTF-8,
- a leíró nyelvek kódrendszere: HTML, LaTeX,
- tömörítést megvalósító kódok: Huffman kód,
- hibajelző/hibajavító kódok: Hamming kód,
- base64 kódolás:
 - az SMTP (Simple Mail Transfer Protocol) átviteli protokollban használták először,
 - adatbázisok esetében ID-k tárolása,
 - weboldalak tárolása,
 - kriptográfia protokollok: kulcsok tárolása.

Az ASCII kódolás

- 1968-ben írták le a standardot, különböző szimbólumokhoz rendel numerikus értéket
- kezdetben 128 szimbólumot kódolt, később 256 (extended ASCII)

6. feladat

Írjunk Python függvényt, amely kiírja a képernyőre az ASCII szimbólumokat és a hozzájuk tartozó kódértékeket.

```
def myASCIITable0():  
    for i in range(128):  
        print(chr(i), i)  
  
def myASCIITable1(n = 0, m = 256):  
    for i in range(n, m):  
        print(chr(i), i)  
  
>>> myASCIITable1(65, 91)  
  
>>> myASCIITable1(97, 123)
```

Az ASCII kódolás

Ha a kimenetet szeretnénk formázni:

```
def myASCIITable2(n = 0, m = 256):  
    for i in range(n, m):  
        print('%4s %5i %s' % (chr(i), i, ''), end = '\t')  
        if i % 3 == 2: print(end = '\n')  
  
>>> myASCIITable2()  
...
```

Az ASCII kódolás

7. feladat

Írjunk Python függvényt, amely kiírja a [képernyőre](#) az ASCII szimbólumokat, a hozzájuk tartozó kódértékeket, és ezek bináris, oktális, illetve hexa alakját.

```
def myASCIITable3(n = 32, m = 128):  
    for i in range(n, m):  
        print('%4s%5i' % (chr(i), i), end = ''),  
        print('%10s' % format(i, 'b'), end = ''),  
        print('%4s' % format(i, 'o'), end = ''),  
        print('%3s' % format(i, 'x'), end = ''),  
        print('%5s' % ' ', end = '')  
        if i % 2 == 1: print(end = '\n')  
  
>>> myASCIITable3()
```

Az ASCII kódolás

8. feladat

Írjunk Python függvényt, amely kiírja egy [állományba](#) az ASCII szimbólumokat, a hozzájuk tartozó kódértékeket, és ezek bináris, oktális, illetve hexa alakját.

```
def myASCIItableF(n = 32, m = 128):  
    outf = open('myAscii.txt', 'wt', encoding = 'utf-8')  
    for i in range(n, m):  
        outf.write('%5s%5i' % (chr(i), i))  
        outf.write('%11s' % format(i, 'b'))  
        outf.write('%5s' % format(i, 'o'))  
        outf.write('%4s' % format(i, 'x'))  
        outf.write('%6s' % ' ' )  
        if i % 3 == 1: outf.write('\n')  
    outf.close()  
  
>>> myASCIItableF()
```


Az ASCII kódolás

Az állományba a `print` függvény segítségével is írhatunk:

```
def myASCIItableF1(n = 32, m = 128):  
    outf = open('myAscii.txt', 'wt', encoding='utf-8')  
    for i in range(n, m):  
        print('%4s%7i' % (chr(i), i), end = '', file = outf),  
        print('%12s' % bin(i), end = '', file = outf)  
        print('%6s' % oct(i), end = '', file = outf)  
        print('%5s' % hex(i), end = '', file = outf)  
        print('%5s' % ' ', end = '', file = outf)  
        if i % 2 == 1: print(end = '\n', file = outf)  
    outf.close()  
  
>>> myASCIItableF1(0, 256)
```

Az Unicode szabvány

- egységes karakterkódolás, karakterosztályozás, szabvány, fejlesztője az Unicode Consortium
- tartalmazza a világ különböző nyelveinek szimbólumait, műszaki szimbólumokat, stb.
- az Unicode karakterek implementálása többféleképpen is lehetséges, az egyik legismertebb az UTF (Unicode Transformation Formats)

```
def myUnicodeTable(n = 51700, m = 51710):  
    for i in range(n, m):  
        print('%3c%7i' % (chr(i), i), end = ''),  
        print('%18s' % format(i, 'b'), end = '')  
        print('%8s' % format(i, 'o'), end = '')  
        print('%6s' % format(i, 'x'), end = '')  
        print()  
  
def myUnicodeTableF(n = 0, m = 55296):  
    outf = open('myUnicode.txt', 'wt', encoding = 'utf-8')  
    for i in range(n, m):  
        outf.write('%5s%7i' % (chr(i), i))  
        outf.write('%18s' % format(i, 'b'))  
        outf.write('%8s' % format(i, 'o'))  
        outf.write('%6s' % format(i, 'x'))  
        outf.write('\n')  
    outf.close()
```

UTF-8 kódolás

- az Unicode szabvány segítségével a világon használt összes szimbólumnak megfeleltethető egy egész szám, pontosabban ezt kódpontnak hívják, ez azonban nem mindig tárolható el egy bájtton
- kérdés: milyen technikával tároljuk ezeket a kódpontokat? azaz mi legyen a bájtrend, hány bájtot használjunk?
- a leggyakrabban alkalmazott technika az UTF-8 kódolás

bájt szám	bit szám	1.kódpont	2.kódpont	1.bájt	2. bájt	3. bájt	4. bájt
1	7	U+0000	U+007F	0bbbbbbb			
2	11	U+0080	U+07FF	110bbbbbb	10bbbbbbb		
3	16	U+0800	U+FFFF	1110bbbb	10bbbbbbb	10bbbbbbb	
4	21	U+10000	U+10FFFF	11110bbb	10bbbbbbb	10bbbbbbb	10bbbbbbb

karakter	kód	bin	1.bájt			2.bájt		
			bin	hex	int	bin	hex	int
ű	369	101 110001	110 00101	0xc5	197	10 110001	0xb1	177

Bájtsorrend

- big-endian: a legnagyobb helyiértékű bájtt a legkisebb címen van tárolva, azaz az MSB (most significant byte) van elől, a hálózatok a csomagok címeiben így használják
- little-endian: a legnagyobb helyiértékű bájtt a legnagyobb címen van tárolva, azaz az MSB van hátul, az Intel alapú számítógépek bájtt tárolási módja

3DF149D1 tárolása a 32-es címtől kezdve:

		big-endian, 1 bájtos tárolás			
memória cím		32	33	34	35
érték		3D	F1	49	D1

		big-endian, 2 bájtos tárolás			
memória cím		32		33	
érték		F1	3D	D1	49

		little-endian, 1 bájtos tárolás			
memória cím		32	33	34	35
érték		D1	49	F1	3D

		little-endian, 2 bájtos tárolás			
memória cím		32		33	
érték		D1	49	F1	3D

Az Unicode szabvány

9. feladat

Generáljunk véletlenszerűen n darab karaktert, amelyek nr kódjára fennáll:
 $k1 \leq nr \leq k2$ és írassuk ki a karaktert, a karakter kódját, a kategóriáját, nevét.

```
import random, unicodedata
def unicode(n, k1, k2):
    for i in range(0, n):
        temp = random.randint(k1, k2)
        try:
            c = chr(temp)
            print(i, c, ord(c), end = ' ')
            print(unicodedata.category(c), end = ' ')
            print(unicodedata.name(c), end = '\n')
        except:
            print('hiba', end = '\n')
```

```
>>> unicode(5, 65, 91)
0 O 79 Lu LATIN CAPITAL LETTER O
1 R 82 Lu LATIN CAPITAL LETTER R
2 E 69 Lu LATIN CAPITAL LETTER E
3 L 76 Lu LATIN CAPITAL LETTER L
4 C 67 Lu LATIN CAPITAL LETTER C
```

```
>>> unicode(5, 65, 65535)
0 뽕 47434 Lo HANGUL SYLLABLE RWILP
1 켤 51708 Lo HANGUL SYLLABLE JJAEN
2 淸 20938 Lo CJK UNIFIED IDEOGRAPH-51CA
3 * 10038 So SIX POINTED BLACK STAR
4 肴 32948 Lo CJK UNIFIED IDEOGRAPH-80B4
```

Python, az index metódus

Az `index` megadja azt a sorszámot (a legelsőt), amelyen egy adott karakter előfordul a karakterláncban, ha nincs benne, futási hibával leáll:

```
>>> 'http://www.ms.sapientia.ro/'.index('/')
5
>>> 'matematika informatika'.index('t')
2
```

```
def betu_kod():
    abece = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    c = input('kerek egy nagybetut: ')
    try:
        betu = abece.index(c)
        return betu, ord(c)
    except:
        print('nem nagy betu', end = ' ')
```

```
>>> betu_kod()
      kerek egy nagybetut:  B
      (1, 66)
>>> betu_kod()
      kerek egy nagybetut:  &
      nem nagy betu
```

Az Unicode szabvány

```
>>> chr(int('d6ff', 16))
...
>>> chr(int('1920', 10))
...
>>> chr(int('50000', 10))
...

def Unicode_kod():
    abece = [chr(i) for i in range(65535)]
    c = input('kerek egy karaktert: ')
    try:
        kar = abece.index(c)
        return kar, ord(c)
    except:
        print('nem nyomtathato karakter')

>>> Unicode_kod()
    kerek egy karaktert: ...
```

Python3 str, bytes

- a Python3 különböző módon kezeli a bináris adatokat (bytes) és a karakterláncokat (str)
- Python3-ban minden str típusú karakter Unicode-ként van eltárolva, a bináris adatok pedig bájtok (bytes) szekvenciáját jelentik
- egy str típusú adat átalakítható bytes típusúvá alkalmazva az **encode**, **bytes** függvényeket; a csak ASCII kódokat tartalmazó stringek a **b** prefix segítségével is átalakíthatóak bytes típusú adattá
- egy bytes típusú adat a decode függvényt alkalmazva alakítható át str típusú adattá

```
>>> mStr = 'muszaki'
>>> ord(mStr[0]), ord(mStr[1]), ord(mStr[2])
(109, 117, 115)
```

```
>>> mStr = 'műszaki'
>>> ord(mStr[0]), ord(mStr[1]), ord(mStr[2])
(109, 369, 115)
```


Python3, str, bytes

Példák, str típusról bytes típusra:

```
>>> mStr = 'muszaki'
>>> type(mStr)
<class 'str'>
>>> mStr[0], mStr[1], mStr[2]
('m', 'u', 's')
```

```
>>> mStr = b'muszaki'
>>> type(mStr)
<class 'bytes'>
>>> mStr[0], mStr[1], mStr[2]
(109, 117, 115)
```

```
>>> mStr = b'műszaki'
SyntaxError: bytes can only contain ASCII literal characters.
```

```
>>> mStr = 'műszaki'.encode()
>>> mStr[0], mStr[1], mStr[2], mStr[3]
(109, 197, 177, 115)
```

Python3, str, bytes

Példák, str típusról bytes típusra:

```
>>> mStr = bytes('műszaki', 'ascii')  
...UnicodeEncodeError: 'ascii' codec can't encode character  
...
```

```
>>> mStr = bytes('műszaki', 'utf-8')  
>>> mStr  
b'm\xcc5\xbb1szaki'
```

A bytes alkalmas arra is, hogy egy megfelelő szerkezetű listát bytes típusú adattá alakítsunk:

```
>>> bytes([109, 197, 177, 115])  
b'm\xcc5\xbb1s'
```

```
>>> bytes([109, 300, 177, 115])  
...ValueError: bytes must be in range(0, 256)
```

Python3, str, bytes

Példák, bytes típusról str típusra:

```
>>> mStr = 'műszaki'.encode()
```

```
>>> mStr
```

```
b'm\xc5\xb1szaki'
```

```
>>> mStr.decode()
```

```
'műszaki'
```

```
>>> mStr = bytes([109,197,177,115,122,97,107,105])
```

```
>>> mStr.decode()
```

```
'műszaki'
```

```
>>> mStr = bytes([109,197,190,115,122,97,107,105])
```

```
>>> mStr.decode()
```

```
'mžszaki'
```

```
>>> mStr = bytes([109,197,200,115,122,97,107,105])
```

```
>>> mStr.decode()
```

```
...UnicodeDecodeError: 'utf-8' codec can't decode ...
```

```
>>> mStr = bytes([109,117,115,122,97,107,105])
```

```
>>> mStr.decode()
```

```
'muszaki'
```

Az XOR egy alkalmazása: titkosítás

10. feladat

Titkosítsunk egy karakterláncot: végezzünk XOR műveletet a karakterlánc karakterjei és egy kulcs-karakterlánc karakterjei között.

Megj: Valós alkalmazások esetében is használják a XOR-t titkosításra, mert gyors műveletvégzést tesz lehetővé, de mivel a lenti kódsorban a kulcsot körkörösén alkalmazzuk, a titkosítás nem lesz biztonságos!!

```
def crypt(myStr, key):
    crStr = ''
    i = 0
    n = len(key)
    for elem in myStr:
        crStr += cryptE(elem, key[i])
        if i == n - 1: i = -1
        i += 1
    return crStr

def cryptE(kar, crKar):
    return chr(ord(kar) ^ ord(crKar))
```

Az XOR egy alkalmazása: titkosítás

Ahhoz, hogy visszkapjuk az eredeti karakterláncot, a titkosításhoz és visszafejtéshez is, ugyanazt a kulcsot kell használni, mert:

```
>>> elem = 97
>>> kulcs = 120
>>> elem ^ kulcs
25
>>> 25 ^ kulcs
97

def mainCR():
    szoveg = input('kerek egy karakterláncot: ')
    kulcs = input('kerek egy kulcsot: ')
    tSzoveg = crypt(szoveg, kulcs)
    print('a titkosított szoveg: ', tSzoveg)

    vSzoveg = crypt(tSzoveg, kulcs)
    print('a visszafejtett szoveg: ', vSzoveg)

>>> mainCR()
    kerek egy karakterláncot: matematika informatika tanszek
    kerek egy kulcsot: kulcs2020
    a titkosított szoveg: ...
```

Az XOR egy alkalmazása: titkosítás

A mainCR függvény elvégzi a titkosítást és a visszafejtést is, ahol a titkosított szöveget hexában írjuk ki, mert amúgy is értelmetlen:

```
def mainCR():
    szoveg = input('kerek egy karakterlancot: ')
    kulcs = input('kerek egy kulcsot: ')
    tSzoveg = crypt(szoveg, kulcs)
    print('a titkosított szoveg: ')
    for c in tSzoveg:
        print(hex(ord(c)), end = ' ')
    print()

    vSzoveg = crypt(tSzoveg, kulcs)
    print('a visszafejtett szoveg: ', vSzoveg)

>>> mainCR()
kerek egy karakterlancot: matematika informatika tanszek
kerek egy kulcsot: kulcs2020
a titkosított szoveg: 0x6 0x14 0x18 0x6 0x1e 0x53 0x44 ...
a visszafejtett szoveg: matematika informatika tanszek
```