

Mátrix

Dr. Szeghalmy Szilvia
Debreceni Egyetem, Informatikai Kar

Mátrix

- ▶ 2 vagy több dimenziós **tömb**
- ▶ Tulajdonságai ennek megfelelően:
 - Statikus
 - Homogén
 - Asszociatív
- ▶ Reprezentáció: **sorfolytonos** v. oszlopfolytonos
(a speciális eseteknél szétszórt reprezentáció is szóba jöhet)

indexek	0	1	...	M-1
0	5	0		3
1	1	7		4
...				
N-1	0	2		2

Művelet: létrehozás

▶ Létrehozásnál eldől

- típus
- dimenziók száma (2 vagy több)
- dimenziónként az indextartomány

indexek	0	1
0	5	0
1	1	7
2	3	4
3	0	2

```
import numpy as np
```

```
mat1 = np.empty( (4, 2), int )      # 4 sor, 2 oszlop, elemek típusa: egész, kezdőértékek: ismeretlen
```

```
mat2 = np.zeros( (10, 12), float )  # 10 sor, 12 oszlop, elemek típusa: valós, kezdőértékek: nulla
```

```
mat3 = np.array( [[5, 0],[1, 7],[3, 4],[0, 2]]) # 4 sor, 2 oszlop, elemek típusa: egész, kezdőérték: 5, 0, 1...
```

Művelet: elérés, csere

Közvetlen elérés dimenzióként egy index alapján.

```
import numpy as np  
mat = np.array( [[5, 0],[1, 7],[3, 4],[3, 2]] )
```

Feladat: Add meg a mat mátrix egyes elemeinek elérését:

5-ös: `mat[0, 0]`

2-es: `mat[3, 1]`

Feladat: Cseréld ki a 4-es elem értékét 9-esre.

```
mat[2, 1] = 9
```

indexek	0	1
0	5	0
1	1	7
2	3	4
3	0	2

Műveletei: bejárás

Írj (void) függvényt, mely **sorfolytonos** módon bejárja a paraméterben kapott NxM-es mátrix elemeit. Az érintett elemeket jelenítsd meg a képernyőn.

```
def sorfolytonos(mat, N, M): # 5, 0, ..., 3, 1, 7, ..., 4, ...
    for i in range(N):
        for j in range(M):
            print(mat[i, j], end=" ")
```

Írj (void) függvényt, mely **oszlopfolytonos** módon bejárja a paraméterben kapott NxM-es mátrix elemeit. Az érintett elemeket jelenítsd meg a képernyőn.

```
def oszlopfolytonos(mat, N, M): 5, 1, ..., 0, 0, 7,..., 2, ...
    for j in range(M):
        for i in range(N):
            print(mat[i, j], end=" ")
```

indexek	0	1	...	M-1
0	5	0		3
1	1	7		4
...				
N-1	0	2		2

Feldolgozás (egész)

Egy $N \times M$ -es mátrixban (*készlet*) tároljuk, hogy mely raktárban, mely termékből hány darab van éppen készleten.

Írj függvényt, mely meghatározza, hogy összesen hány termék van a raktárakban. Az értéket adja vissza a függvény.

```
def minden_cikk(keszlet, N, M): # osszeg
    s = 0
    for i in range(N):        # raktár indexek
        for j in range(M):    # cikk indexek
            s += keszlet[i, j]
    return s
```

	0. cikk	1. cikk	...	M-1. cikk
0. raktár	5	0		3
1. raktár	1	7		4
...				
N-1. raktár	0	2		2

Mátrix (1 sor)

Egy $N \times M$ -es mátrixban (*készlet*) tároljuk, hogy mely raktárban, mely termékből hány darab van éppen készleten.

Írj függvényt, mely visszaadja, hogy hány**féle** cikk van éppen raktáron a az r . raktárban (az r egész szám). Ha az r . raktár nem létezik, akkor -1-es értéket adja vissza.

```
def hanyfele_cikk(keszlet, N, M, r): # egy sor, darab
    if r < 0 or N <= r: # az r érvényes sorindex-e
        return -1

    db = 0
    for j in range(M): # mert egy sorban M elem áll
        if keszlet[r, j] > 0:
            db += 1
    return db
```

	0. cikk	1. cikk	...	M-1. cikk
0. raktár	5	0		3
1. raktár	1	7		4
...				
N-1. raktár	0	2		2

Feldolgozás (1 oszlop)

Egy $N \times M$ -es mátrixban (*készlet*) tároljuk, hogy mely raktárban, mely termékből hány darab van éppen készleten.

Írj függvényt, mely **megjeleníti** azon raktárak sorszámát, melyekben a c . cikkből nincs raktáron. Feltételezheted, hogy a $0 \leq c < M$, *egész szám*.

```
def minden_cikk(keszlet, N, M, c): # oszlop
    for i in range(N): # mert egy oszlopban N elem áll
        if keszlet[i, c] == 0:
            print( i )
```

	0. cikk	1. cikk	...	M-1. cikk
0. raktár	5	0		3
1. raktár	1	7		4
...				
N-1. raktár	0	2		2

Mátrix (oszloponkénti)

Egy $N \times M$ -es mátrixban (*készlet*) tároljuk, hogy mely raktárban, mely termékből hány darab van éppen készleten.

Írj függvényt, mely meghatározza, hogy melyik cikkből mennyi van összesen raktáron. Az eredményt mentsd el az *ossz_készlet* tömb megfelelő helyére. (A tömb létezik, M elemű.)

```
def cikkenkenti_készlet(készlet, N, M, ossz_készlet):  
    for i in range(N):  
        s = 0 # minden cikknél 0-ról indul az összegzés  
        for j in range(M):  
            s += készlet[i, j]  
        ossz_készlet[i] = s
```

	0. cikk	1. cikk	...	M-1. cikk
0. raktár	5	0		3
1. raktár	1	7		4
...				
N-1. raktár	0	2		2

	0. cikk	1. cikk	...	M-1. cikk
ossz_készlet	?	?		?

Mátrix (cellák)

Egy $N \times M$ -es mátrixban (*készlet*) tároljuk, hogy mely raktárban, mely termékből hány darab van éppen készleten.

Az 1. raktárba (létezik) új szállítmány érkezik. Az M elemű *szallitmany* tömb i . értékén az áll, hogy az i . cikkből mennyit szállítottunk az adott raktárba.

Frissítsd ez alapján a készletet.

```
def uj_szallitmany(keszlet, szallitmany, N, M):  
    for j in range(M):  
        keszlet[1, j] += keszlet[j]
```

	0. cikk	1. cikk	...	M-1. cikk
0. raktár	5	0		3
1. raktár	1	7		4
...	3	4		5
N-1. raktár	0	2		2

Gyakorlás

Egy $N \times M$ -es mátrixban (*készlet*) tároljuk, hogy mely raktárban, mely termékből hány darab van éppen készleten.

Írj függvényt, mely meghatározza, hogy mely raktárban tároljuk a legtöbb cikket. (Tf. nincs holtverseny)

```
import math
```

```
def max_raktar(keszlet, N, M):
    maxidx = -1
    max = -math.inf
    #járd be a raktárakat
    for i in range(N):
        # számítsd ki az adott raktár összkészletét
        s = 0
        for j in range(M):
            s += keszlet[i, j]
        # ha szükséges, frissítsd a max és maxidx változókat
        if s > max:
            max = s
            maxidx = i

    return maxidx # az eredmény
```

	0. cikk	1. cikk	...	M-1. cikk
0. raktár	5	0		3
1. raktár	1	7		4
...				
N-1. raktár	0	2		2

Alsó háromszögmátrix

- ▶ Olyan négyzetes mátrix, melyben a főátló felett csupa 0 értékű elem áll.
- ▶ Cél: helytakarékos tárolás

Írj függvényt, mely bepakolja az NxN-es háromszögmátrix elemeit folytonosan egy tömbbe. A reprez tömb létezik és van elég hely benne.

```
def also_haromszog(mat, N, reprez):  
    db = 0  
    for i in range(N):  
        for j in range(i+1):  
            reprez[db] = mat[i, j]  
            db += 1
```

		j			
i		0	1	2	3
	0	5	0	0	0
	1	1	7	0	0
	2	3	4	2	0
	3	0	2	1	9

index	0	1	2	3	4	5	6	7	8	9
elem	5	1	7	3	4	2	0	2	1	9

Alsó háromszögmátrix: elérés

Írj függvényt, mely egy mátrix reprezentációt kap meg paraméterként és egy (érvényes) sor (y) és oszlopindexet (x). Add vissza a reprezentáció alapján a mátrix megfelelő elemét.

```
def also_haromszog_eleres(reprez, y, x):  
    # ha eredetileg a felső háromszögmátrixba tartozott  
    if x > y:  
        return 0  
    else:  
        idx = y*(y+1)//2 + x  
        return reprez[idx]
```

y	x			
	0	1	2	3
	0	5	0	0
	1	1	7	0
	2	3	4	2
	3	0	2	1
				9

index	0	1	2	3	4	5	6	7	8	9
elem	5	1	7	3	4	2	0	2	1	9

Melyik sor hol kezdődik a reprez-ben?
(= hány elem van előtte)

0: 0

1: 1

2: $1+2 = 3$

3: $1+2+3 = 6$

y : $1+2+3+\dots+y = (1+y)*y//2$

+ a sor x . eleme előtt van még pontosan x elem a reprezentációban.

Szimmetrikus mátrix

Egy szimmetrikus ($\text{mat}[i,j] = \text{mat}[j,i]$) mátrixot ugyanúgy reprezentálhatunk egy tömb segítségével, mint egy háromszögmátrixot. De hogyan nyerjük vissza az elemeit?

Írj függvényt, mely egy mátrix reprezentációt kap meg paraméterként és egy (érvényes) sor (y) és oszlopindexet (x). Add vissza a reprezentáció alapján a mátrix megfelelő elemét.

```
def szimm_matrix_eleres(reprez, y, x):  
    if x > y : # csak felcseréljük az indexeket  
        idx = (1+x)*x//2 + y  
        return reprez[idx]  
    else:  
        idx = (1+y)*y//2 + x  
        return reprez[idx]
```

		x			
y		0	1	2	3
	0	5	1	3	0
	1	1	7	4	2
	2	3	4	2	1
	3	0	2	1	9

index	0	1	2	3	4	5	6	7	8	9
elem	5	1	7	3	4	2	0	2	1	9

Melyik sor hol kezdődik a reprez-ben?
(= hány elem van előtte)

0: 0

1: 1

2: $1+2 = 3$

3: $1+2+3 = 6$

y : $1+3+\dots+y = (1+y)*y//2$

+ a sor x . eleme előtt van még pontosan x elem a reprezentációban.

Ritka mátrix

- ▶ Olyan mátrix, melyben egy elem nagyságrenddel többször szerepel, mint a többi elem összesen. A legtöbbször szereplő elemet **gyakori elem**nek nevezzük.
- ▶ Cél: helytakarékos tárolás
- ▶ Háromsoros reprezentáció: sorfolytonos módon bejárjuk a mátrixot és eltároljuk a nemgyakori elemeknél a sor, oszlop és érték egy tömbben (v. listában).

Írj függvényt, mely a eltárolja a *reprez* tömbben az NxM-es *mat* mátrix nem gyakori elemeit. A *reprez* tömb létezik, van elég hely az adatok tárolására.

(sor, oszlop, ertekek) rendezett hármastokat pakolj bele.

```
def letrehoz(mat, N, M, reprez, gyakori_elem):  
    db = 0  
    for i in range(N):  
        for j in range(M):  
            if mat[i, j] != gyakori_elem:  
                reprez[db] = (i, j, mat[i, j]) # sor,oszlop,ertekek  
                db += 1
```

	j					
	0	1	2	3	4	5
i	0	0	0	3	0	0
	1	0	0	1	0	0
	2	7	0	0	6	0
	3	0	0	4	0	2

sor	0	1	2	2	2	3	3
oszlop	2	2	0	3	5	2	4
ertekek	3	1	7	6	9	4	2

gyakori elem: 0

Ritka mátrix: elérés

Írj függvényt, mely egy háromsoros reprezentációt és egy érvényes *mátrix* indexet (y, x) kap paraméterként. A függvény adja vissza, hogy milyen érték volt eredetileg a mátrixban. A *reprez* egy n elemű, (*sor*, *oszlop*, *ertek*) rendezett hármasokból álló tömb

A példára: [(0, 2, 3), (1, 2, 1), (2, 0, 7), (2, 3, 6), (2, 5, 9), (3, 2, 4), (3, 4, 2)]

A reprezentáció i . tagja:

sor: `reprez[i][0]`

oszlop: `reprez[i][1]`

ertek: `reprez[i][2]`

```
def eler(reprez, n, y, x, gyakori_elem):  
    # ha a reprez-ben megtalálod a megfelelő sort és oszlopot  
    # add vissza a hozzá tartozó értéket  
    for i in range(n):  
        if reprez[i][0] == y and reprez[i][1] == x:  
            return reprez[i][2]  
    # ha nincs a tömbben, akkor térj vissza a gyakori elemmel  
    return gyakori_elem
```

		x					
y		0	1	2	3	4	5
	0	0	0	3	0	0	0
	1	0	0	1	0	0	0
	2	7	0	0	6	0	9
	3	0	0	4	0	2	0

sor	0	1	2	2	2	3	3
oszlop	2	2	0	3	5	2	4
ertek	3	1	7	6	9	4	2

gyakori elem: 0

Gyakorlás

Írj függvényt, mely egy háromsoros reprezentációt kap paraméterként. A *reprez* egy $n > 0$ elemű, $(sor, oszlop, ertek)$ rendezett hármassokból álló tömb.

A példára: $[(0, 2, 3), (1, 2, 1), (2, 0, 7), (2, 3, 6), (2, 5, 9), (3, 2, 4), (3, 4, 2)]$

A reprezentáció i . tagja:

sor: `reprez[i][0]`

oszlop: `reprez[i][1]`

ertek: `reprez[i][2]`

A függvény adja vissza a nem gyakori elemek átlagát.

Megj.: ne feledd, a *reprez* csak a nem gyakori elemeket tartalmazza.

```
def atlag (reprez, n):
```

```
    ossz = 0
```

```
    for i in range(n):
```

```
        ossz += reprez[i][2]
```

```
    return ossz/n
```

		x					
y		0	1	2	3	4	5
	0	0	0	3	0	0	0
	1	0	0	1	0	0	0
	2	7	0	0	6	0	9
	3	0	0	4	0	2	0

sor	0	1	2	2	2	3	3
oszlop	2	2	0	3	5	2	4
ertek	3	1	7	6	9	4	2

gyakori elem: 0

Gyakorlás

A netet pásztázó mesterséges intelligenciánk feladata, hogy összehasonlítsa a különböző bankok által nyújtott szolgáltatásokat. Az eredményt egy $N \times M$ -es mátrixban (*adat*) mátrixban tároljuk. Ha a mátrix i . sorában és j . oszlopában álló érték 1, akkor az i . bank nyújtja a j . szolgáltatást. Ha 0 áll a cellában, akkor nem.

	0. szolg	1. szolg	...	M-1. szolg
0. bank	0	0		1
1. bank	1	0		0
...				1
N-1. bank	0	1		1

Jelenítsd meg azon bankok sorszámát, melyek minden szolgáltatást nyújtanak!

```
def bank(adat, N, M):  
    for i in range(N):  
        mindent_nyujt = True  
        for j in range(M):  
            if adat[i, j] == 0:  
                mindent_nyujt = False  
                break  
        if mindent_nyujt:  
            print(i)
```

Alapötlet: Tegyük fel, hogy az i . bank minden szolgáltatást nyújt. Ennek megfelelően állítsuk be a *mindent_nyujt* változót True értékre. Ha találunk egy 0-át az i . sorban, akkor billentsük át a *mindent_nyujt* változó értékét. A belső for ciklus után ellenőrizzük a *mindent_nyujt* értékét. Ha True értékű, akkor a bank minden szolgáltatást nyújt.

Gyakorlás

Alternatív megoldás az előző feladathoz (kevésbé hatékony, de technikailag könnyebben kivitelezhető).

Számoljuk meg minden sorban, az 1-esek számát. Ha pont M darabot találunk egy sorban, akkor az adott bank minden szolgáltatást nyújt.

```
def bank(adat, N, M):  
    for i in range(N):  
        db = 0  
        for j in range(M):  
            if adat[i, j] == 1:  
                db += 1  
        if db == M: # M - a szolgáltatások száma  
            print(i)
```

	0. szolg	1. szolg	...	M-1. szolg
0. bank	0	0		1
1. bank	1	0		0
...				1
N-1. bank	0	1		1

Gyakorlás

Alternatív megoldás az előző feladathoz (kevésbé hatékony, de technikailag könnyebben kivitelezhető).

Most használjuk ki azt, hogy a mátrixban csak 0 és 1-es érték áll, ezért ha összeadjuk a sorban lévő értékeket, akkor pont az 1-esek számát kapjuk meg.

```
def bank(adat, N, M):  
    for i in range(N):  
        db = 0  
        for j in range(M):  
            db += adat[i, j]  
        if db == M: # M - a szolgáltatások száma  
            print(i)
```

	0. szolg	1. szolg	...	M-1. szolg
0. bank	0	0		1
1. bank	1	0		0
...				1
N-1. bank	0	1		1

Gyakorlás

A netet pásztázó mesterséges intelligenciánk feladata, hogy összehasonlítsa a különböző bankok által nyújtott szolgáltatásokat. Az eredményt egy $N \times M$ -es mátrixban (*adat*) mátrixban tároljuk. Ha a mátrix i . sorában és j . oszlopában álló érték 1, akkor az i . bank nyújtja a j . szolgáltatást. Ha 0 áll a cellában, akkor nem.

	0. szolg	1. szolg	...	M-1. szolg
0. bank	0	0		1
1. bank	1	0		0
...				1
N-1. bank	0	1		1

Írj függvényt, mely visszaadja, hogy hány banknál lehet kötvényt leköti a neten keresztül. A szolgáltatás kódja (mátrixbeli oszlop indexe) a 10-es. (Tf. $10 < M-1$)

```
def bank2(adat, N, M):  
    db = 0  
    for i in range(N):  
        if adat[i, 10] > 0:  
            db += 1  
    return db
```

Gyakorlás

A netet pásztázó mesterséges intelligenciánk feladata, hogy összehasonlítsa a különböző bankok által nyújtott szolgáltatásokat. Az eredményt egy $N \times M$ -es mátrixban (*adat*) mátrixban tároljuk. Ha a mátrix i . sorában és j . oszlopában álló érték 1, akkor az i . bank nyújtja a j . szolgáltatást. Ha 0 áll a cellában, akkor nem.

Írj függvényt, mely meghatározza, hogy mely bankban, hány szolgáltatás érhető el. Az eredményeket az N elemű t tömbbe másold.

```
def bank3(adat, N, M, t):  
    for i in range(N):  
        db = 0  
        for j in range(M):  
            db += adat[i, j]  
        t[i] = db
```

	0. szolg	1. szolg	...	M-1. szolg	t
0. bank	0	0		1	?
1. bank	1	0		0	?
...					...
N-1. bank	0	1		1	?

Gyakorlás

A netet pásztázó mesterséges intelligenciánk feladata, hogy összehasonlítsa a különböző bankok által nyújtott szolgáltatásokat. Az eredményt egy $N \times M$ -es mátrixban (*adat*) mátrixban tároljuk. Ha a mátrix i . sorában és j . oszlopában álló érték 1, akkor az i . bank nyújtja a j . szolgáltatást. Ha 0 áll a cellában, akkor nem.

Írj függvényt, mely True értéket ad vissza, ha a b . bankban elérhető az sz . szolgáltatás, különben False értéket. Feltételezhető, hogy a b és az sz egész szám. Ha a b vagy az sz érvénytelen index, akkor a függvény None értéket adjon vissza.

```
def bank4(adat, N, M, b, sz):  
    if 0 <= b < N and 0 <= sz < M:  
        return adat[b, sz] == 1  
    return None
```

	0. szolg	1. szolg	...	M-1. szolg
0. bank	0	0		1
1. bank	1	0		0
...				
N-1. bank	0	1		1

Gyakorlás

A netet pásztázó mesterséges intelligenciánk feladata, hogy összehasonlítsa a különböző bankok által nyújtott szolgáltatásokat. Az eredményt egy $N \times M$ -es mátrixban (*adat*) mátrixban tároljuk. Ha a mátrix i . sorában és j . oszlopában álló érték 1, akkor az i . bank nyújtja a j . szolgáltatást. Ha 0 áll a cellában, akkor nem.

Hasonlítsd össze a KSH (0. sorszámú bank) és az OTP bankot (3. sorszámú bank), majd add vissza azon bank nevét, amelyik több szolgáltatást nyújt. Holtverseny esetén üressztringet adj vissza. ($N > 4$)

```
def bank5(adat, N, M):  
    db_otp = 0  
    db_ksh = 0  
    for j in range(M):  
        db_otp += adat[0, j]  
        db_ksh += adat[3, j]  
  
    if db_otp > db_ksh:  
        return "OTP"  
    if db_otp < db_ksh:  
        return "KSH"  
    return ""
```

	0. szolg	1. szolg	...	M-1. szolg
0. bank	0	0		1
1. bank	1	0		0
...				
N-1. bank	0	1		1