



دانشگاه شهید بهشتی کرمان

# دو: آرایه ها

ساختمان داده ها و الگوریتم

مدرس: دکتر نجمه منصوری

نگارنده: سجاد هاشمیان

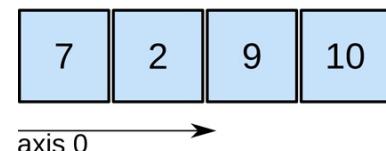


# آرایه

آرایه یا ماتریس یا جدول یا متغیر اندیس دار، مجموعه‌ای از فضاهای بهم پیوسته یا پی درپی از حافظه است.  
اعضای آرایه از یک نوعند و به یک اندازه حافظه نیاز دارند.

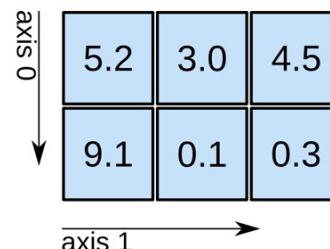
3D array

1D array

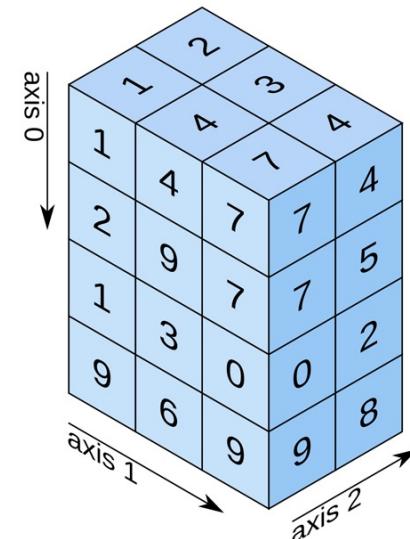


shape: (4,)

2D array



shape: (2, 3)



shape: (4, 3, 2)

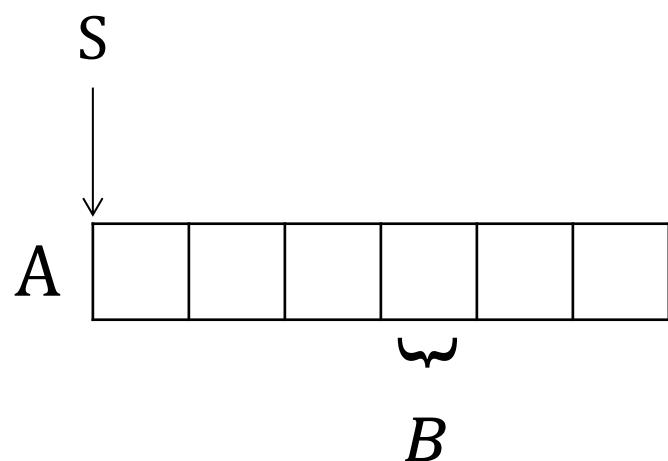
# آدرس خانه‌های آرایه

همه اعضای آرایه در حافظه پشت سرهم ذخیره می‌شوند. مثلاً آرایه A با آدرس شروع S که هر خانه آن B بایت از حافظه را نیاز دارد، در نظر بگیرید.

آدرس واقعی خانه اول:  $S$

آدرس واقعی خانه دوم:  $S + 1 \times B$

آدرس واقعی خانه nام:  $S + (n-1) \times B$



# آدرس خانه‌های آرایه

در آرایه دو بعدی  $A_{R \times C}$  آدرس واقعی درایه سطر  $r$  و ستون  $c$  برابر است با:

$$\text{Address}_{r,c} = S + [(r-1) \times C + (c-1)] \times B$$

برای مثال در آرایه زیر داریم:

			X	

$$\text{Address}_{3,4} = S + [(3-1) \times 5 + (4-1)] \times B$$

# آدرس خانه‌های آرایه

به طور کلی، آدرس خانه  $A[(L_1, \dots, U_1), (L_2, \dots, U_2), \dots, (L_n, \dots, U_n)]$  در آرایه  $A[x_1, x_2, \dots, x_n]$  با توجه به وضعیت سطری یا ستونی برابر است با:

$$A[x_1, x_2, \dots, x_n] = S + \left[ \sum_{i=1}^n (x_i - L_i) \left( \prod_{j=i+1}^n U_j - L_j + 1 \right) \right] \times B \quad \text{آدرس سطری:}$$

$$A[x_1, x_2, \dots, x_n] = S + \left[ \sum_{i=n}^1 (x_i - L_i) \left( \prod_{j=1}^{i-1} U_j - L_j + 1 \right) \right] \times B \quad \text{آدرس ستونی:}$$

که  $S$  در آن برابر با آدرس شروع آرایه و  $B$  برابر با فضای مورد نیاز برای ذخیره سازی نوع عناصر آرایه است؛ در صورت عدم بیان این دو مقدار به ترتیب آنها را برابر با 0 و 1 فرض می‌کنیم، در صورت عدم بیان نوع آدرس نیز، پیش‌فرض آدرس ستونی استفاده می‌کنیم.

# مثال

آرایه ۲-بعدی  $X[-5..5, 3..33]$  در آدرس ۴۰۰ به بعد حافظه قرار دارد و هر خانه آن به ۴ بایت نیاز دارد.  
آدرس خانه  $X[4, 10]$  به روش ستونی را محاسبه کنید.

$$X[-5..5, 3..33] \quad S = 400 \quad B = 4$$

$$\begin{aligned} X[4, 10] &= S + \left[ (10 - 3) \left( 5 - (-5) + 1 \right) + \left( 4 - (-5) \right) \right] \times B \\ &= 400 + \left[ (10 - 3) \left( 5 - (-5) + 1 \right) + \left( 4 - (-5) \right) \right] \times 4 = 400 + 344 = 744 \end{aligned}$$

# مثال

آرایه ۳-بعدی  $A[1:15, -5:5, 10:25]$  در آدرس ۲۰۰۰ به بعد حافظه قرار دارد و هر خانه آن به ۲ بایت نیاز دارد. آدرس خانه  $A[5,2,15]$  به روش سط्रی را محاسبه کنید.

$$A[1 : 15, -5 : 5, 10 : 25] \quad S = 2000 \quad B = 2$$

$$A[5, 2, 15] = S + \left[ (5-1)(5-(-5)+1)(25-10+1) + (2-(-5))(25-10+1) + (15-10) \right] \times B$$
$$= 2000 + 821 \times 2 = 2000 + 1642 = 3642$$

# مثال

آرایه ۳-بعدی  $A[1:m, 1:n, 1:p]$  در یک آرایه یک بعدی  $B[1\dots m\times n\times p]$  به روش سطر به سطر ذخیره شده است. آدرس عنصر  $A[i,j,k]$  در  $B$  کدام است؟

$$A[1..m, 1..n, 1..p] \quad S = 0 \quad B = 1$$

$$\begin{aligned} A[i, j, m] &= S + \left[ (i - 1)(n - 1 + 1)(p - 1 + 1) + (j - 1)(p - 1 + 1) + (k - 1) \right] \times B \\ &= (i - 1)np + (j - 1)p + (k - 1) \end{aligned}$$

# ماتریس ها

به هر آرایه دو بعدی  $n \times m$  یک ماتریس یا جدول با  $m$  سطر و  $n$  ستون گفته می‌شود؛ که تعداد  $mn$  خانه در آن وجود دارد.

**ماتریس مربعی:** به هر ماتریس با تعداد برابر سطر و ستون گفته می‌شود، در هر ماتریس مربعی  $n \times n$  مانند  $A$  روابط زیر برای  $[i,j] A$  وجود دارد:

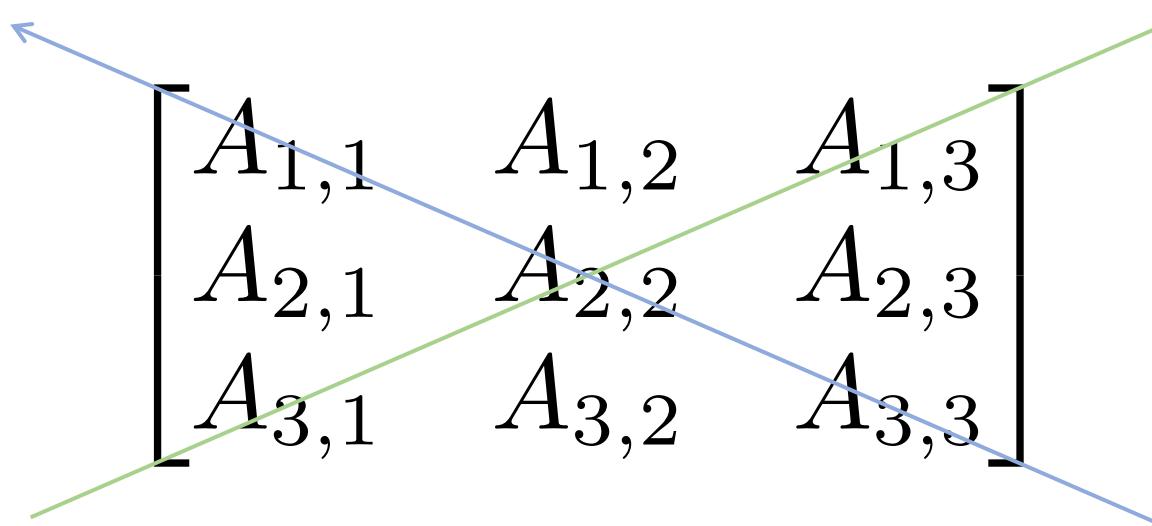
$$\begin{cases} i < j & A_{ij} \text{ بالا قطر اصلی است} \\ i = j & A_{ij} \text{ روی قطر اصلی است} \\ i > j & A_{ij} \text{ پایین قطر اصلی است} \end{cases}$$

$$\begin{cases} i + j < n + 1 & A_{ij} \text{ بالا قطر فرعی است} \\ i + j = n + 1 & A_{ij} \text{ روی قطر فرعی است} \\ i + j > n + 1 & A_{ij} \text{ پایین قطر فرعی است} \end{cases}$$

# قطر ها

قطر اصلی  
 $i=j$

قطر فرعی  
 $i+j=n+1$



# ماتریس های مثلثی

ماتریس پایین مثلثی

$$\begin{bmatrix} X & 0 & 0 & \cdots & 0 \\ X & X & 0 & \cdots & 0 \\ X & X & X & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ X & X & X & \cdots & X \end{bmatrix}$$

عناصر بالای قطر اصلی، همگی صفر هستند.

ماتریس بالا مثلثی

$$\begin{bmatrix} X & X & X & \cdots & X \\ 0 & X & X & \cdots & X \\ 0 & 0 & X & \cdots & X \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & X \end{bmatrix}$$

عناصر زیر قطر اصلی، همگی صفر هستند.

# ماتریس های مثلثی

$$\begin{array}{ccccc} X & X & X & \cdots & X \\ 0 & X & X & \cdots & X \\ 0 & 0 & X & \cdots & X \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & X \end{array} \quad \begin{array}{c} n \\ +n-1 \\ +n-2 \\ \vdots \\ +1 \end{array}$$
$$\frac{n(n+1)}{2}$$

تعداد کل خانه ها =  $n^2$

تعداد خانه های مخالف صفر =  $\frac{n(n+1)}{2}$

تعداد خانه های برابر با صفر =  $\frac{n(n-1)}{2}$

# ماتریس Sparse (تنک، خلوت، پراکنده)

هر ماتریسی که در آن تعداد خانه های صفر و بی ارزش (non value) از تعداد خانه های مخالف صفر بیشتر باشد، را ماتریس پراکنده تعریف می کنیم.

دقت کنید که لزومی ندارد که حاصل جمع یا ضرب ماتریس های اسپارس، ماتریس اسپارس شود:

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

# ذخیرهسازی ماتریس اسپارس

طبق تعریف ماتریس های اسپارس می دانیم بیش از نصف خانه ها برابر با صفر است؛ که برای نگهداری از یک ماتریس مجبور به ذخیرهسازی بی جهت این داده های تکراری هستیم، از این رو دو روش زیر برای نگهداری ماتریس های اسپارس ارائه می شود:

- روش عمومی: آرایه دو بعدی و ذخیره مختصات خانه های مخالف صفر
- به کمک یک رابطه جانبی: استفاده از یک آرایه یک بعدی و نگهداری از مقادیر مخالف صفر به کمک یک فرمول یا رابطه، مثلا برای ماتریس های مثلثی.

روش دوم از لحاظ صرفه جویی در حافظه بهتر است، البته اگر بتوان رابطه ای بین اندیس های آرایه و اندیس های ماتریس اسپارس پیدا کرد.

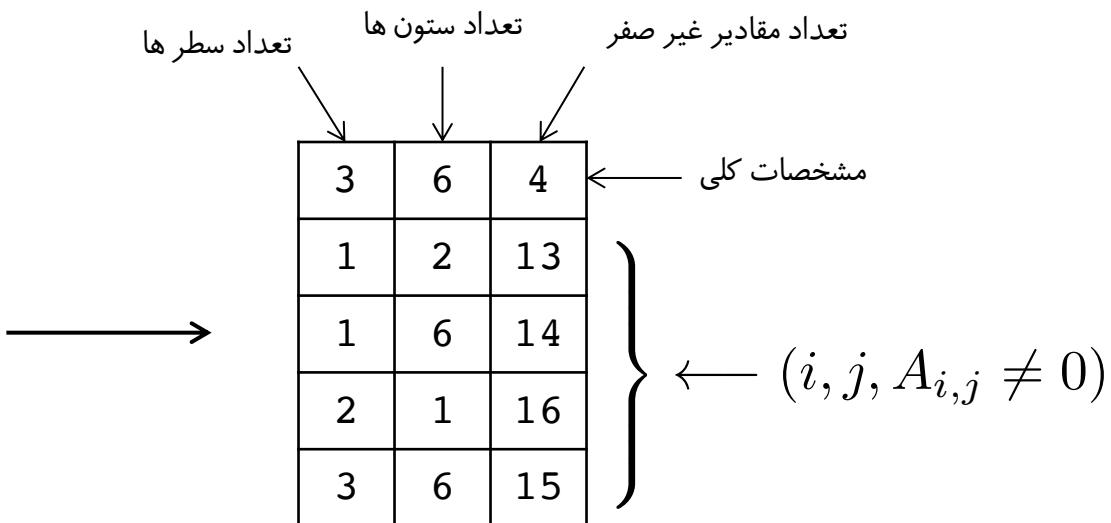
# روش عمومی

اگر ماتریس اسپارس  $m \times n$  با  $r$  خانه مخالف صفر موجود باشد، آنگاه برای ذخیره سازی آن از یک آرایه دو بعدی با  $r+1$  سطر و  $3$  ستون استفاده می‌کنیم:

- در سطر اول به ترتیب: تعداد سطرها ( $m$ )، تعداد ستون‌ها ( $n$ ) و تعداد خانه‌های مخالف صفر ( $r$ ) قرار داده می‌شود.
- از سطر دوم به بعد، هر سطر شامل سه مولفه است که همان مختصات و مقدار خانه‌های مخالف صفر است.

0	13	0	0	0	14
16	0	0	0	0	0
0	0	0	0	0	15

ماتریس اسپارس



# کارایی روش عمومی

اما روش مطرح شده تنها در صورتی به صرفه است که تعداد خانه های ماتریس نگهدارنده از تعداد خانه های ماتریس اولیه به مراتب کمتر باشد:

$$3(r + 1) < mn \xrightarrow{3(r+1) \approx 3r} r < \frac{mn}{3}$$

بنابراین تنها در صورتی استفاده از این روش توصیه می شود که تعداد خانه های مخالف صفر کمتر از  $1/3$  خانه های ماتریس اسپارس اولیه باشد.

# ترانهاده ماتریس اسپارس

برای ترانهاده کردن یک ماتریس اسپارس که به روش عمومی ذخیره شده، کافیست روش زیر را اجرا کنیم:

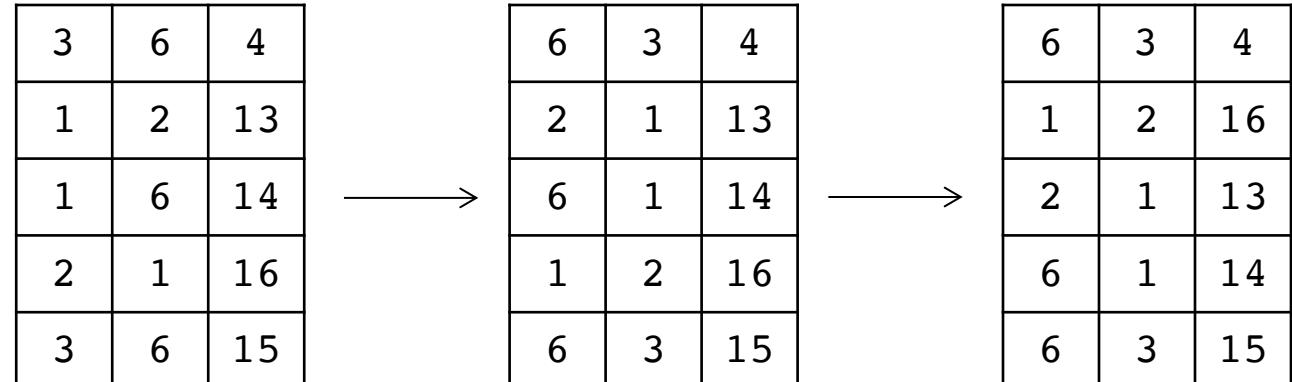
- ستون اول و دوم را جابه جا کن.
- سطرهای دوم تا  $r+1$  ماتریس حاصله را به صورت نزولی مرتب کن.

```
def transpose(A:matrix):  
    B.shape=A.shape  
    for i in (1,columns):  
        for j in (1,rows):  
            B[j][i]=A[i][j]  
    return B
```

```
def sparse_transpose(A:sparse_matrix):  
    B.shape=A.shape  
    B[1]=A[2]  
    B[2]=A[1]  
    B[3]=A[3]  
    sort(B[:,2:])  
    return B
```

فرض کنید  $A$  ماتریس اسپارس  $m \times n$  با  $r$  خانه مخالف صفر باشد. آنگاه الگوریتم اول از مرتبه  $O(mn)$  اجرا می‌شود، اما اگر روش مرتب‌سازی مناسبی داشته باشیم الگوریتم دوم با مرتبه اجرایی  $O(r)+O(\text{sort})$  به مراتب سریع‌تر است.

(به ترتیب اولویت ابتدا با مقدار سطر در ستون اول و سپس ستون دوم)



# نگهداری با کمک روابط

بعضی ماتریس ها مانند ماتریس های بالا مثلثی، پایین مثلثی و L-قطری اسپارس نیستند، اما زمانی که تعداد عناصر در آنها زیاد می شود، تعداد صفر های ماتریس به صورت قابل توجه ای زیاد می شود، در این صورت بهتر است مقادیر مخالف صفر ماتریس به صورت زیر ذخیره شود:

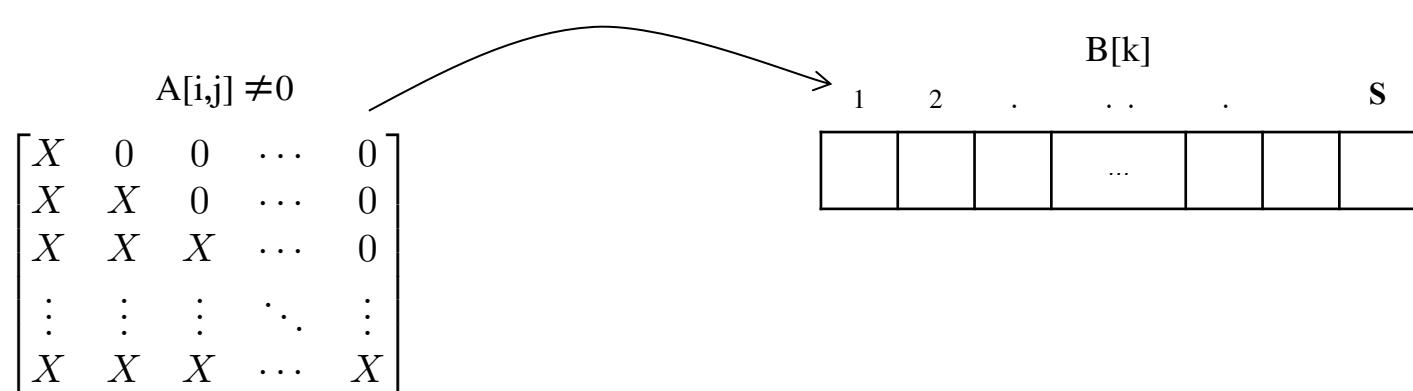
- یک آرایه یک بعدی برای نگهداری مقادیر مخالف صفر ماتریس به صورت سطری یا ستونی، که تعداد عناصر آرایه یک بعدی به تعداد مقادیر مخالف صفر ماتریس می باشد.
- یک رابطه یا فرمول که محل مقادیر مخالف صفر ماتریس را در یک آرایه یک بعدی مشخص می کند.

# نگهداری ماتریس پایین مثلثی

ماتریس پایین مثلثی  $A[1:n, 1:n]$  را در نظر بگیرید که عناصر مخالف صفر آن را به صورت سطری در یک آرایه یک بعدی نگهداری کردہ ایم، در این صورت:

- در ماتریس  $S = \frac{n(n+1)}{2}$  خانه مخالف صفر قرار دارد، برای همین به یک آرایه یک بعدی  $B[1:S]$  نیاز داریم.
- در هر خانه  $A[i,j] \neq 0$  در ماتریس پایین مثلث با رابطه (فرمول سطرب) زیر محل خود را در آرایه  $B[k]$  مشخص می کند.

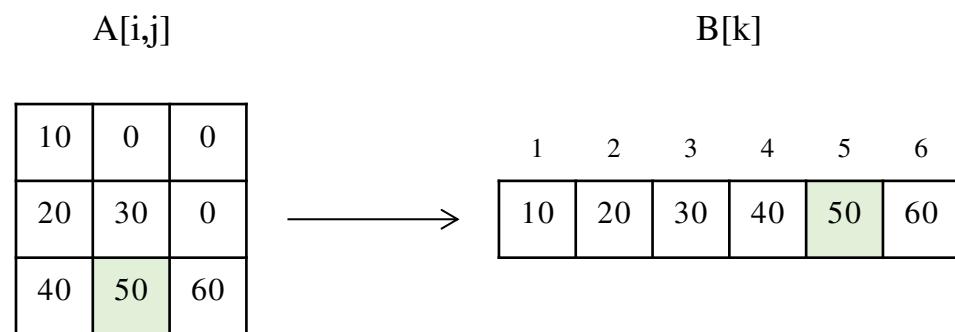
$$k_{(i,j)} = \frac{i(i-1)}{2} + j$$



# مثال

به طور مثال موقعیت خانه  $A[3,2]=50$  در آرایه یک بعدی  $B[5]$  است، رابطه بین  $(i=3, j=2)$  در ماتریس و  $(k=5)$  در آرایه یک بعدی توسط رابطه زیر مشخص می‌شود:

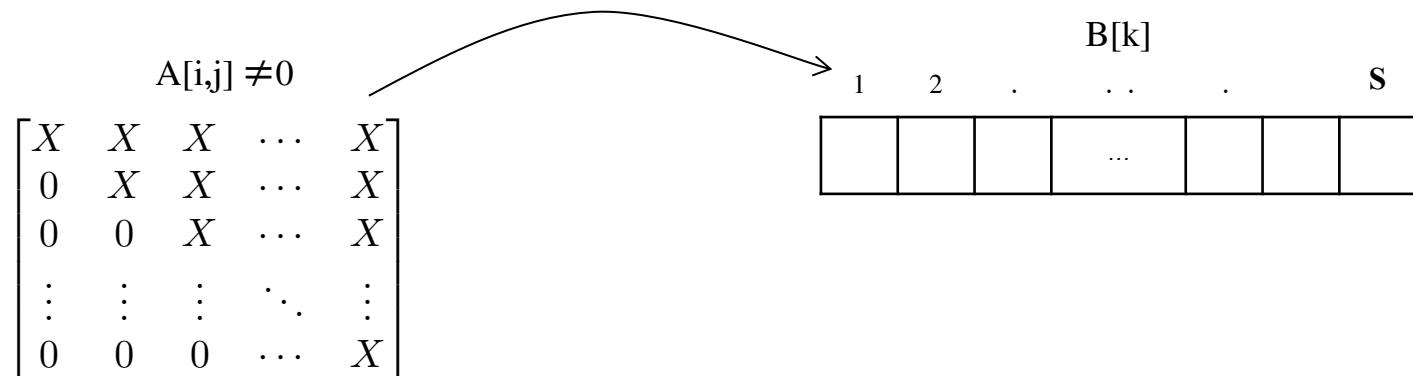
$$\left. \begin{array}{l} A[i, j] : A[3, 2] \\ k_{(i,j)} = \frac{i(i-1)}{2} + j \end{array} \right\} \longrightarrow k_{(3,2)} = \frac{3(3-1)}{2} + 2 = 5 \longrightarrow B[k] = B[5]$$



# نگهداری ماتریس بالا مثلثی

اگر  $A[1:n, 1:n]$  ماتریسی بالا مثلثی باشد، آنگاه موقعیت خانه  $A[i,j]$  در آرایه یک بعدی سطري  $B[k]$  را با توجه به رابطه زیر تعیین می شود:

$$k_{(i,j)} = (i-1) \left( n - \frac{i}{2} \right) + j$$



# مثال

به طور مثال موقعیت خانه  $A[1,3] = 87$  در آرایه یک بعدی  $B[3]$  است، رابطه بین  $(i=1, j=3)$  در ماتریس و  $(k=3)$  در آرایه یک بعدی توسط رابطه زیر مشخص می‌شود:

$$A[i, j] : A[1, 3] \\ k_{(i,j)} = (i - 1) \left( n - \frac{i}{2} \right) + j \quad \left. \right\} \rightarrow k_{(1,3)} = (1-1) \left( 4 - \frac{1}{2} \right) + 3 = 3 \rightarrow B[k] = B[3]$$

$A[i,j]$

25	98	87	91
0	16	85	19
0	0	70	30
0	0	0	11

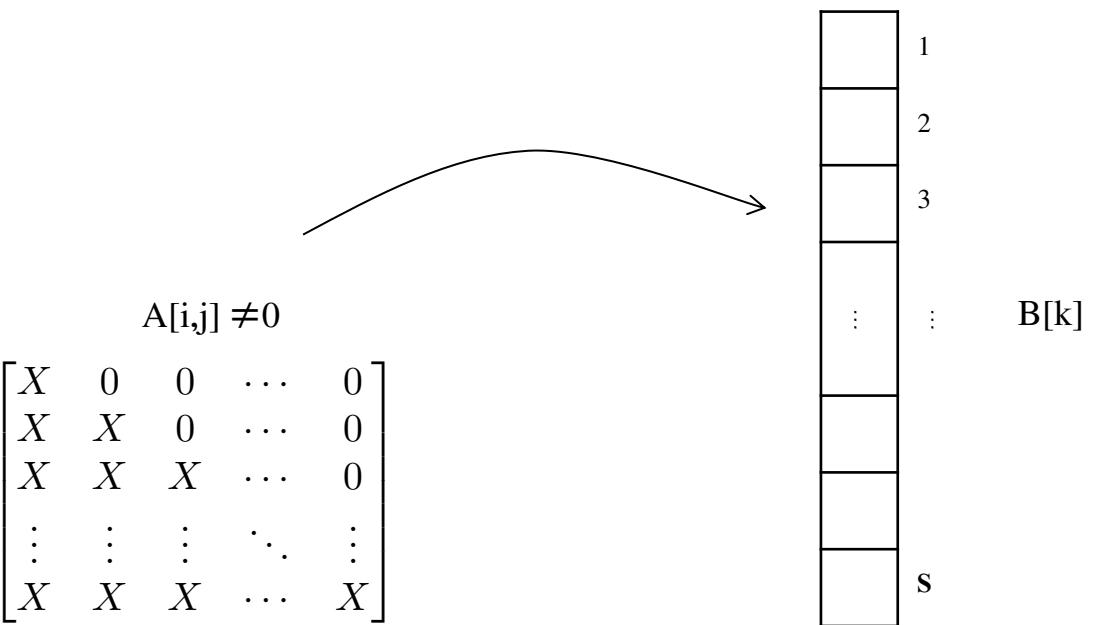
$B[k]$

1	2	3	4	5	6	7	8	9	10
25	98	87	91	16	85	19	70	30	11

# نگهداری ستونی ماتریس پایین مثلثی

اگر  $A[1:n, 1:n]$  ماتریسی پایین مثلثی باشد، آنگاه موقعیت خانه  $A[i,j]$  در آرایه یک بعدی ستونی  $B[k]$  را با توجه به رابطه زیر تعیین می‌شود:

$$k_{(i,j)} = (i-1) \left( n - \frac{i}{2} \right) + j$$



# مثال

به طور مثال موقعیت خانه  $A[4,3]=17$  در آرایه یک بعدی  $B[9]$  است، رابطه بین  $(i=4, j=3)$  در ماتریس و  $(k=9)$  در آرایه یک بعدی توسط رابطه زیر مشخص می‌شود:

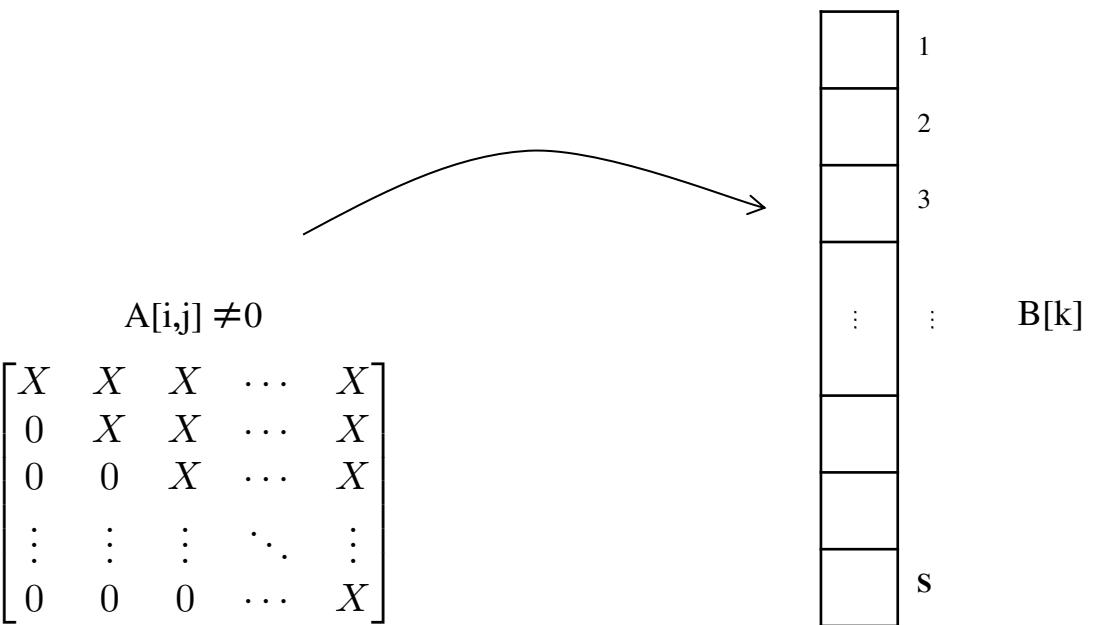
$$k_{(i,j)} = (j - 1) \left( n - \frac{j}{2} \right) + j \quad \left. \begin{array}{l} A[i,j] : A[4,3] \\ \longrightarrow k_{(4,3)} = (3-1) \left( 4 - \frac{3}{2} \right) + 4 = 9 \longrightarrow B[k] = B[9] \end{array} \right\}$$

A[i,j]					B[k]
25	0	0	0		1
52	21	0	0		2
29	24	35	0		3
99	61	17	58	→	4
					5
					6
					7
					8
					9
					10

# نگهداری ستونی ماتریس بالا مثلثی

اگر  $A[1:n, 1:n]$  ماتریسی بالا مثلثی باشد، آنگاه موقعیت خانه  $A[i,j]$  در آرایه یک بعدی ستونی  $B[k]$  را با توجه به رابطه زیر تعیین می‌شود:

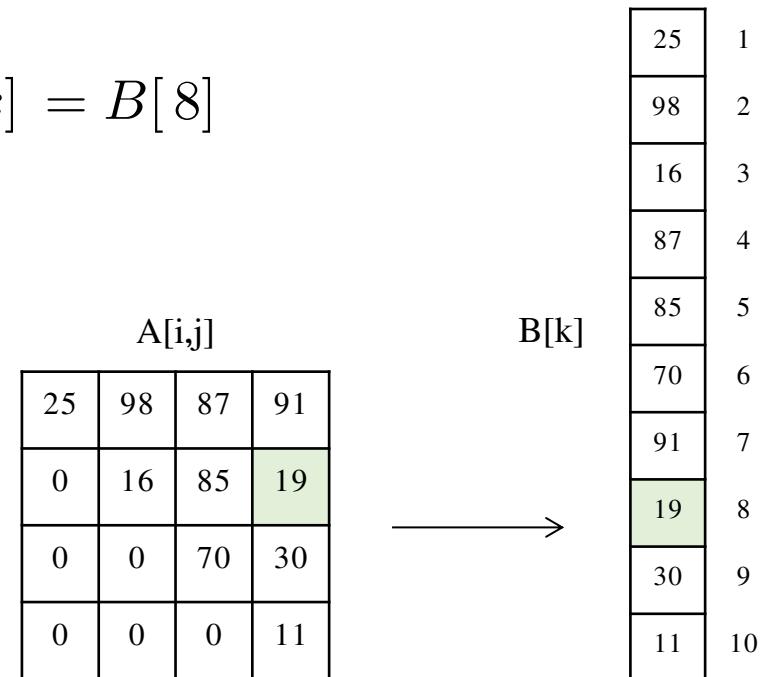
$$k_{(i,j)} = \frac{j(j - 1)}{2} + i$$



# مثال

به طور مثال موقعیت خانه  $A[2,4]=19$  در آرایه یک بعدی  $B[8]$  است، رابطه بین  $(i=2, j=4)$  در ماتریس و  $(k=8)$  در آرایه یک بعدی توسط رابطه زیر مشخص می‌شود:

$$\left. \begin{array}{l} A[i, j] : A[2, 4] \\ k_{(i, j)} = \frac{j(j-1)}{2} + i \end{array} \right\} \rightarrow k_{(2, 4)} = \frac{4(4-1)}{2} + 2 = 8 \rightarrow B[k] = B[8]$$



# ضرب ماتریس ها

برای آنکه ضرب دو ماتریس  $A$  و  $B$  امکان پذیر باشد، باید بعد میانی آنها برابر باشد، به عبارت بهتر برای محاسبه پذیر بودن ضرب دو ماتریس  $A$  و  $B$  باید تعداد ستون های  $A$  با تعداد سطر های  $B$  برابر باشد:

$$A_{m \times n} \times B_{n \times k} \rightarrow C_{m \times k}$$

- ضرب ماتریس ها خاصیت جابه جایی ( $AB \neq BA$ ) ندارد.
- ضرب ماتریس ها خاصیت شرکت پذیری ( $(AB)C = A(BC) = (AB)C$ ) دارد.

```
def mult(A: matrix(m,n),B: matrix(n,k)):  
    C=matrix(m,k)  
    for i in (1:m):  
        for j in (1:k):  
            C[i][j]=0  
            for l in (1:n):  
                C[i][j]=C[i][j]+A[i][l]*B[l][j]  
    return C
```

در این الگوریتم تمام جمع و ضرب ها در خط ۷ اتفاق می افتد بنابراین تعداد آنها مساوی بوده و برابر  $m \times n \times k$  خواهد بود، این یعنی اگر سه متغیر  $m$  و  $n$  و  $k$  هم مرتبه هم باشند، آنگاه ضرب ماتریسی از مرتبه  $O(n^3)$  خواهد بود!

# ضرب ماتریس‌ها

- هرگاه یک ماتریس بالا مثلث را در یک ماتریس بالا مثلث ضرب کنیم، حاصل یک ماتریس بالا مثلث است.
- هرگاه یک ماتریس پایین مثلث را در یک ماتریس پایین مثلث ضرب کنیم، حاصل یک ماتریس پایین مثلث است.
- هرگاه یک ماتریس قطری را در یک ماتریس قطری ضرب کنیم، حاصل یک ماتریس قطری است.

به طور کلی ضرب یک ماتریس در یک ماتریس با همان مشخصه، خاصیت ماتریس را تغییر نمی‌دهد! (اما آیا برای هر مشخصه ای؟)

# شرکت پذیری ضرب ماتریس ها

تعداد حالات شرکت پذیری در ضرب  $n+1$  ماتریس ها از رابطه زیر بدست می آید:

$$T(n+1) = C_n = \frac{1}{n+1} \binom{2n}{n}$$

e.g.  $T(4) = C_3 = \frac{1}{4} \binom{6}{3} = 5$

e.g.  $T(3) = C_2 = \frac{1}{3} \binom{4}{2} = 2$

شرکت پذیری ۳ ماتریس  
A,B,C

A(BC)  
(AB)C

شرکت پذیری ۴ ماتریس  
A,B,C,D

((AB)C)D  
(A(BC))D  
(AB)(CD)  
A((BC)D)  
A(B(CD))

# تعداد ضرب در ضرب چند ماتریس

تعداد ضرب های حاصل ضرب ۳ ماتریس ۳ در ۱۰، ۱۰ در ۵ و ۵ در ۴ را با توجه به حالات مختلف شرکت پذیری محاسبه کنید:

$$A \in \mathcal{M}_{3 \times 10} \quad B \in \mathcal{M}_{10 \times 5} \quad C \in \mathcal{M}_{5 \times 4}$$

$$A_{3 \times 10}(BC)_{10 \times 4} = (3 \times 10 \times 4) + (10 \times 5 \times 4) = 120 + 200 = 320$$

$$(AB)_{3 \times 5}C_{5 \times 4} = (3 \times 10 \times 5) + (3 \times 5 \times 4) = 150 + 60 = 210$$

# تعداد ضرب در ضرب چند ماتریس

همانطور که دیدیم ضرب ماتریسی عملیاتی پر هزینه است، همچنین در حالات مختلف طبق شرکت پذیری تعداد کل ضرب ها برای یک عبارت متفاوت است، بنابراین ما به دنبال حالتی هستیم که تعداد ضرب ها در آن کمینه باشد تا عمل ضرب ماتریسی سریعتر انجام شود.

قضیه: در هنگام ضرب ۳ ماتریس  $A, B, C$  داریم:  $(AB)C \iff \frac{1}{m} + \frac{1}{k} > \frac{1}{n} + \frac{1}{L}$  تعداد ضرب

در واقع طبق یک حساب سرانگشتی می توان گفت که هنگام ضرب چند ماتریس، اگر ماتریس هایی را که زود تر ضرب کنیم که بعد میانی آنها بزرگتر و بعد کناری آنها به نسبت کوچکتر است، بدین ترتیب تعداد ضرب ها کوچک تر می شود.

برای مثال قبل، در هنگام ضرب ۳ ماتریس، از آنجا  $\frac{1}{5} + \frac{1}{3} > \frac{1}{4} + \frac{1}{6}$  که پس تعداد ضرب های  $C(AB)$  از تعداد ضرب های  $A(BC)$  کمتر است. در اینجا بعد میانی  $AB$  بزرگترین بود.

# مثال

اگر  $A$  یک ماتریس  $13 \times 5$ ،  $B$  یک ماتریس  $5 \times 89$ ،  $C$  یک ماتریس  $89 \times 3$  و  $D$  ماتریسی  $3 \times 34$  باشند، کمینه تعداد ضرب برای محاسبه حاصل ضرب آنها چقدر است؟

ترتیب ضرب ها	تعداد ضرب ها
$B_{5 \times 89} \times C_{89 \times 3} = (B \times C)_{5 \times 3}$	$5 \times 89 \times 3 = 1335$
$A_{13 \times 5} \times (BC)_{5 \times 3} = (A \times (BC))_{13 \times 3}$	$13 \times 5 \times 3 = 195$
$(A(BC))_{13 \times 3} \times D_{3 \times 34} = ((A(BC))D)_{13 \times 34}$	$13 \times 3 \times 34 = 1326$
	<hr/> $total = 2856$

# مثال

اگر  $A$  یک ماتریس  $10 \times 20$ ،  $B$  یک ماتریس  $20 \times 50$ ،  $C$  یک ماتریس  $50 \times 1$  و  $D$  ماتریسی  $1 \times 100$  باشند، پرانتر بندی که سبب کمینه تعداد ضرب برای محاسبه حاصل ضرب  $ABCD$  شود را بیابید.

$$M = A_{10 \times 20} \times B_{20 \times 50} \times C_{50 \times 1} \times D_{1 \times 100}$$

$$M = A_{10 \times 20} \times (B \times C)_{20 \times 1} \times D_{1 \times 100}$$

$$M = (A \times (B \times C))_{10 \times 1} \times D_{1 \times 100}$$

$$M = ((A \times (B \times C)) \times D)_{10 \times 100}$$

# کاربردهای دیگر: چند جمله‌ای‌ها

چند جمله‌ای‌ها، نوع خاصی از عبارت‌های جبری محسوب می‌شوند که در آن‌ها فقط یک متغیر نقش دارد. همانطور که گفته شد، «چند جمله‌ای» یک رابطه بین توان‌های مختلف یک متغیر است. به یاد دارید که یک چند جمله‌ای بر حسب  $x$  از درجه  $n$  را به صورت زیر می‌نویسیم:

$$P_n(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \dots + a_nx^n$$

البته عبارت بالا را برای راحتی با استفاده از نماد جمع به شکل زیر نیز نشان می‌دهند:

$$P_n(x) = \sum_{i=0}^n a_i x^i$$

# نگهداری چندجمله ای ها

برای ذخیره سازی یک چندجمله ای از درجه  $n$  می توانیم از یک آرایه عددی به طول  $n+1$  استفاده کنیم، به این صورت که در خانه  $i$  ام آن ضریب جمله  $x^i$  را قرار می دهیم.

$a_0$	$a_1$	$a_2$	$a_3$	$\dots$	$a_n$
-------	-------	-------	-------	---------	-------

$$P_n(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \dots + a_nx^n$$

برای مثال ذخیره سازی چندجمله ای  $p(x) = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!}$  به این صورت خواهد بود:

1	$\frac{1}{1!}$	$\frac{1}{2!}$	$\frac{1}{3!}$
0	1	2	3

# جمع چندجمله‌ای‌ها

در این حالت برای جمع دو چندجمله‌ای کافیست تا مقدار خانه‌های متناظر را با هم جمع کنیم.

0	1	0	0	2	3	0
---	---	---	---	---	---	---

$$P(x) = x + 2x^5 + 3x^6$$

1	7	2	0	0	0	1
---	---	---	---	---	---	---

$$Q(x) = 1 + 7x + 2x^2 + x^7 \quad +$$

---

1	8	2	0	2	3	1
---	---	---	---	---	---	---

$$T(x) = 1 + 8x + 2x^2 + 2x^5 + 3x^6 + x^7$$

# توسعه پیاده‌سازی چندجمله‌ای‌ها

این روش را می‌توان به سادگی به دیگر عملیات‌های چندجمله‌ای (مثلاً ضرب، توان،...) گسترش داد، اما همانگونه که حدس زدید، اگر تعداد ضریب‌های صفر یک چندجمله‌ای زیاد باشد ( $P(x) = x^{100}$ ) دیگر این روش دخیره‌سازی مقرن به صرفه نیست.

آیا ایده‌ای دارید؟