ساختمان داده ها

دانشگاه صنعتی نوشیروانی بابل

دکتر حسام عمران پور

طراحان اسلاید:
زهرا ریحانیان و دانیال علیزاده

حل تمرین :
علی باقری

لینک کانال تلگرام اطلاع رسانی و حل تمرین:
t.me/ds_nit_4011

فصل سوم

ساختمان داده های مقدماتی

ساختمان داده

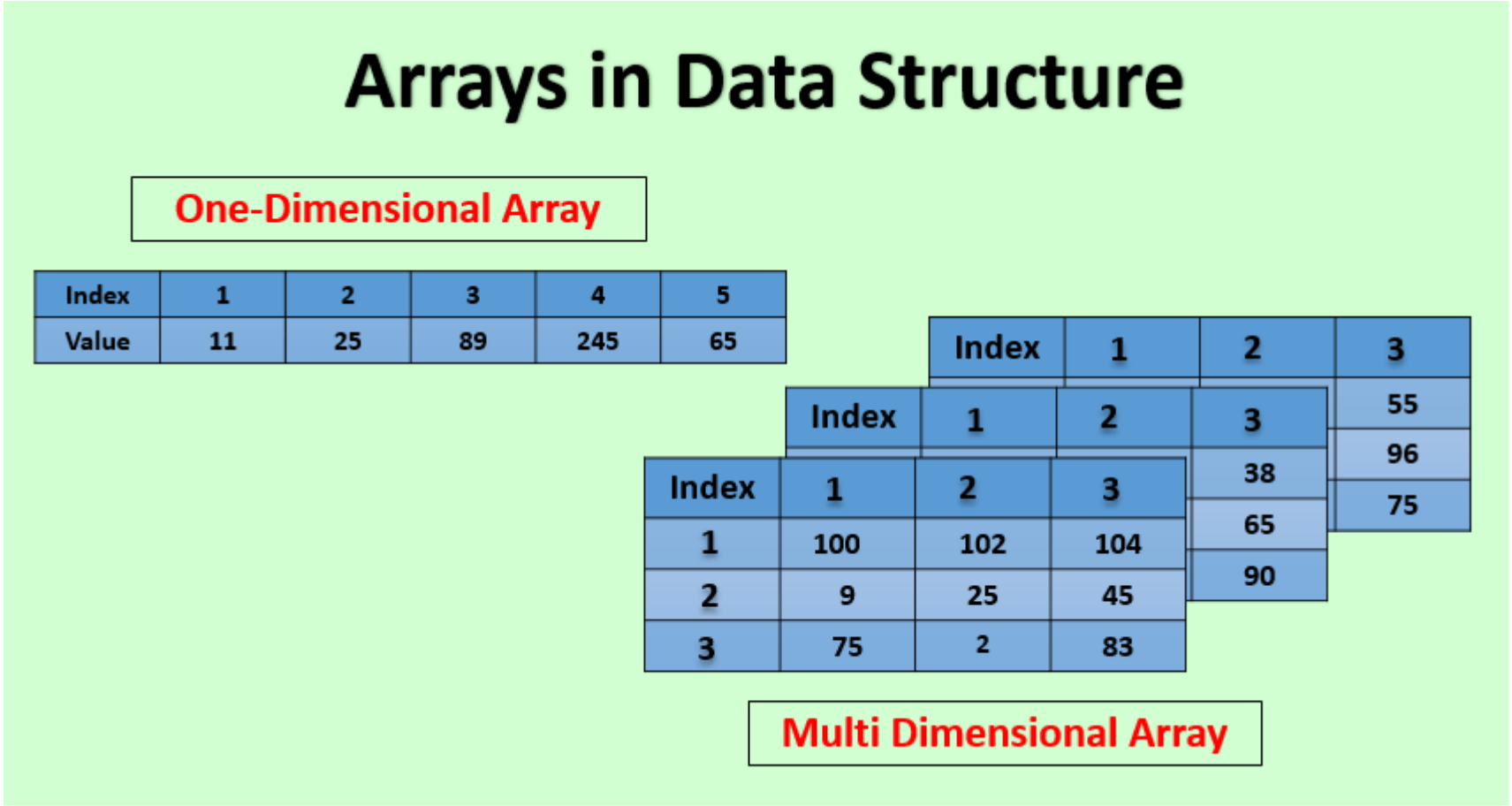
ساختمان داده چگونگی چینش (آرایش) داده ها را در حافظه کامپیوتر برای یک منظور خاص تعیین میکند.

Abstract Data Type (ADT) یا نوع داده انتزاعی :

یک نوع داده به همراه تمام عملیاتی است کـه بـر روی آن قابـل اجرا باشد.

❖ *ADT* به صورت *انتزاعی* تعریف می شود، یعنی چگونگی پیاده سـازی عملیـات در آن مشخص نمی شود. به عبارت دیگر، **در نوع داده ی انتزاعی تنهـا مشـخص مـی شـود کـه چـه عملیاتی بر روی داده ها انجام می گیرد.**

آرایه


Arrays in Data Structure
One-Dimensional Array
Index 1 2 3 4 5
Value 11 25 89 245 65
Index 1 2 3
1 100 102 104
2 9 25 45
3 75 2 83
Index 1 2 3
55
96
75
Index 1 2 3
38
65
90
Multi Dimensional Array

آرایه مجموعه ای از داده هایی است که خصوصیات زیر را داشته باشند :

۱) همگی از یک نوع داده باشند.

۲) در خانه های پیوسته ی حافظه قرار گیرند.

۳) دسترسی به عناصر به صورت مستقیم است.

این دسترسی با استفاده از آدرس اولین عنصر و **Index** عنصر مورد نظر امکان پذیر می باشد.

$$\text{Index} \xrightarrow{map} Address$$

آرایه ساختاری ایستا است و اندازه ی آن (حداکثر تعداد داده هایی که می تواند ذخیـره کنـد) باید در هنگام تعریف آرایه مشخص شود.

در شبه کدهای کتاب ساختمان داده ، آرایه به صورت شبه کد زیر تعریف می شود:

var ArrayName : ElementType[ArraySize] ;

var ArrayName : ElementType[LowIndex…HighIndex] ;

آرایه یک بعدی

هنگام استفاده از آرایه، در زمان اجرای برنامه آدرس داده ی مورد نیاز محاسبه می شـود. بـا فرض این که زبان برنامه سازی مورد بحث، شاخص را از یک آغاز کند، محاسبات انجام گرفته به منظور دستیابی به آدرس عنصر i ام آرایه ی A به شکل زیر می باشد :

$$A[i] \rightarrow \alpha + (i + 1) \times (size\ of\ element\ type)$$

1	2	3	4	….	n

در آرایه ی فرضی Array[55] که آدرس اولین خانه ی آن ۵۰۰۰ است و شماره ی شاخص های آن از یک آغاز می شود و با فرض این که اندازه ی داده های آرایه همگی سه بایت می باشد؛ آدرس عناصر زیر را محاسبه نمایید :

Array[2] = ?

Array[15] = ?

Array[55] = ?

Array[2] = 5000 + (2-1) × 3 = 5003

Array[15] = 5000 + (15-1) × 3 = 5042

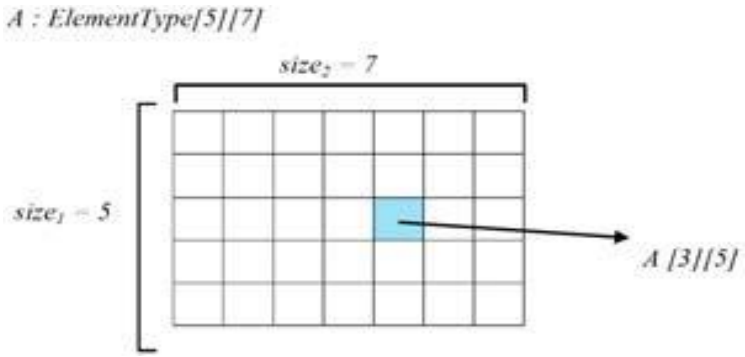
Array[55] = 5000 + (55-1) × 3 = 5162

آرایه چند بعدی

آرایه ای را که تا این جا مورد بحث قرار گرفت، آرایه ی تک بعدی مـی نامنـد . امـا نـوع دیگـرآرایه، نوع چند بعدی آن است. از آرایه های چند بعدی برای پیاده سازی ماتریس ها و دیگر جـداول استفاده می شود. در اغلب زبان های برنامه سازی رایج، تعداد بعد های یک آرایه، هیچ محـدودیتی ندارد. یک آرایه ی دو بعدی بدین صورت در شبه کدها تعریف می گردد:

var ArrayName : ElementType[size1][size2] ;

var ArrayName : ElementType[low1..high1][low2..high2] ;

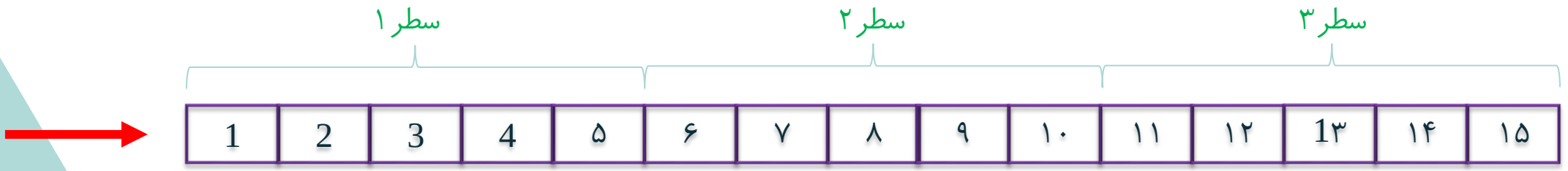


جای گیری آرایه دوبعدی در RAM

Size2 = 5

Size1 = 3

۱	۲	۳	۴	۵
۶	۷	۸	۹	۱۰
۱۱	۱۲	۱۳	۱۴	۱۵



در آرایه ی دو بعدی A که شاخص خانـه ی اول آن در هـر دو بعـد برابـر یـک بـوده و آدرس اولین عنصر آن α است، آدرس عنصر $[i][j]$ ام آن بدین صورت محاسبه می گردد:

var ArrayName : ElementType[size1][size2] ;

تعداد ستون

سطری :

$$A[i][j] = \alpha + \big((i-1) \times n + (j-1)\big) \times (size\ of\ element\ type)$$

ستونی :

$$A[i][j] = \alpha + \big((j-1) \times m + (i-1)\big) \times (size\ of\ element\ type)$$

تعداد سطر

مثال: آدرس A[3][4] را با فرض اندیس شروع از 1 به هر دو صورت سطری و ستونی بدست آورید.

$$A[10][20]\ ,\ \alpha = 2000\ ,\ \text{ElementType : int}\ ,\ \text{Size(int)} = 2$$

سطری :

$$A[3][4] = 2000 + (2 \times 20 + 3) \times 2 = 2086$$

ستونی :

$$A[3][4] = 2000 + (3 \times 10 + 2) \times 2 = 2064$$

var ArrayName : ElementType[low1..high1][low2..high2] ;

سطری :

$$A[i][j] = \alpha + \big((i - low1) \times (high2 - low2 + 1) + (j - low2)\big) \times (size\ of\ element\ type)$$

ستونی :

$$A[i][j] = \alpha + \big((j - low2) \times (high1 - low1 + 1) + (i - low1)\big) \times (size\ of\ element\ type)$$

تمرین :

برای حالت ۳ بعدی و چهار بعدی و همچنین (ستونی و سطری برای هر یک)

و (شروع از ۱ و low i و high i) روابط را بیایید .

الگوریتم های مهم در آرایه

```
findMax (A : integer[n], n : integer) : integer
{
    var maxIndex : integer ;
    maxIndex = 1;
    for i = 2 to n do
        if A[i] > A[maxIndex] then
            maxIndex = i;
    return A[maxIndex];
}
```

```
findMinMax (A : integer[n]; ref maxIndex, ref minIndex
    : integer)
{
    minIndex = 1;
    maxIndex = 1;
    for i = 2 to n do
    {
        if A[i] > A[maxIndex] then
            maxIndex = i;
        else if A[i] < A[maxIndex] then
            minIndex = i;
    }
}
```

```
BinarySearch (L : Array; n : integer; k : KeyType) : integer
{
    // L is a sorted list (increasing) of elements
    // returns index of the machesd element, or -1
    // l =lower bound, u = upper bound, m = (u+1)/2
    var l, u : integer;
    l = 1;
    u = n;
    while l <= u do
    {
        m = (l + u) / 2;
        if L[m] == k then
            return m;
        else if L[m] < k then
            l = m + 1;
        else
            u = m - 1;
    }
    return -1;
}
```

حالت بازگشتی جستجوی دودویی

```
BinarySearch (A : integer[n]; lowIndex, highIndex, k : integer) : integer
{
   if (lowIndex > highIndex)
      return -1;
   else
   {
      m = [(lowIndex + highIndex) / 2]
      if A[m] == k
         return A[m];
      if A[m] < k
         return BinarySearch(A, m + 1, highIndex, k);
      else
         return BinarySearch(A, lowIndex, m - 1, k);
   }
}
```

ماتریس ها

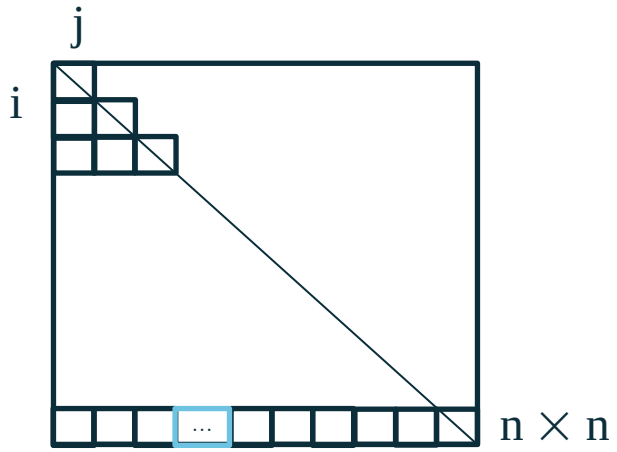
- **ماتریس** یک ابزار ریاضی است و در انجام بسیاری از محاسبات از آن استفاده می شود. درکامپیوتر می توان ماتریس را با استفاده از یک **آرایه ی دو بعدی** نمایش داد.
- ماتریس A را که دارای m سطر و n ستون است، در ریاضی ات به صورت $A_{m\cdot n}$ تعریف می کنند.
- تعداد اعضای این ماتریس، برابر $m\cdot n$ است. اگر تعداد سطرها و ستون های یک ماتریس برابر باشد، آن را ماتریس **مربعی** نامیده و به صورت $m\cdot m$ نمایش می دهند.
- به هر یک از عناصر ماتریس، **درایه** می گویند. در علوم ریاضی، درایه های از ماتریس A که در سطر iام و ستون jام قرار دارد، با $A_{i\,j,}$ نمایش داده میشود در این جا با توجه به این که برای نمایش ماتریس از آرایه ی دو بعدی استفاده می شود، این عنصر به صورت $A[i][j]$ نشان داده خواهدشد.

انواع ماتریس

ماتریس پایین مثلثی

ماتریس پایین مثلثی، یک ماتریس مربعی است که تمامی مقادیر بالای قطر اصلی آن صفر اســت .

$$\begin{bmatrix} 2 & 0 & 0 \\ 4 & 8 & 0 \\ 1 & 3 & 1 \end{bmatrix}$$



i	j	index
1	1	1
2	1	2
2	2	3

1	2	3	4	….	n(n+1)/2

تعداد سطرهای رد شده (ما قبل i) = $i - 1$

I تعداد سلول های موجود در سطرهای ما قبل = $1 + 2 + 3 + \cdots + (i-2) + (i-1) = \frac{(i-1)i}{2}$

تعداد سلول های موجود در سطر i (ما قبل این عنصر) = j - 1

$$f(i,j) = \frac{i(i-1)}{2} + j - 1 \Rightarrow index = \frac{i(i-1)}{2} + j$$

$$\text{آدرس} = \alpha + \big(f(i,j)\big) \times SizeOfElementType$$

ماتریس قطری

ماتریس قطری، ماتریس مربعی است که **تمام درایه های غیر واقع بر قطر اصلی آن صفر** هستند.

به عبارت دیگر، ماتریس همزمان بالامثلثی و پایین مثلثی را ماتریس قطری می نامند. در شکل پایین ماتریس قطری نشان داده شده است.

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

ماتریس اسپارس

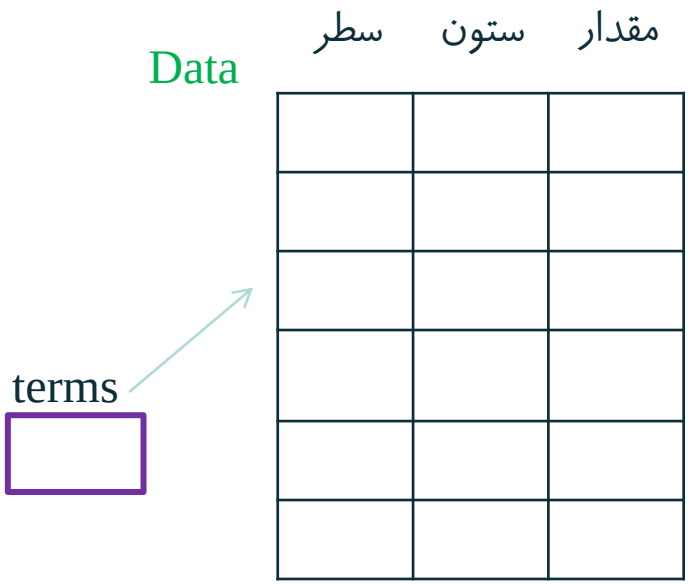
ماتریس اسپارس ، ماتریسی است که تعداد قابل توجهی از درایه های آن صـفر باشـد ، ولی درایه های غیر صفر آن بر خلاف ماتریس قطری و ماتریس مثلثی، نظم خاصـی ندارنـد.

نمونه ای از یک ماتریس اسپارس در شکل پایین مشاهده می شود :

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 8 & 0 \\ 0 & 0 & 7 & 2 & 0 & 0 & 4 \\ 0 & 9 & 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

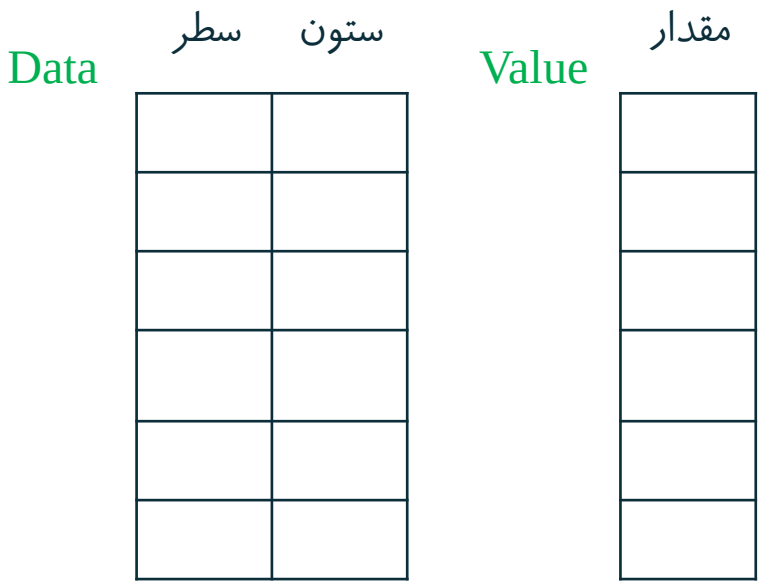
ساختمان داده بهینه برای ماتریس اسپارس ۱

```
Type Sparse1 = Record
{
        terms , originRow , originColumn : integer
        Data : integer [terms][3]
}
```



ساختمان داده بهینه برای ماتریس اسپارس ۲

```
Type Sparse2 = Record
{
        terms , originRow , originColumn : integer
        Data : integer [terms][2]
        Value : ElementType[terms]
}
```



ماتریس اصلی

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 9 & 0 \\ 0 & 8 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 2 & 0 & 0 & 0 \end{bmatrix}$$

ماتریس اسپارس

Rows	Columns	Values
0	4	9
1	1	8
2	0	4
2	3	2
3	5	5
4	2	2

ماتریس 1000 * 500 داریم که دارای m مقدار غیر صفر است. m حداکثر چقدر باشد تا استفاده از ساختمان داده اسپارس فضای بهینه تر از فضای اولیه را بدهد ؟ (نوع عناصر int است)

$m * 3 * 2 < 500 * 1000 * 2$ ⟹ $m < 166666.66$

⟹ $\boxed{m = 166666}$

اگر ElementType باشد :

$m * 2 * 2 + m * SoET < 500 * 1000 * SoET$

⟹ $\boxed{m = \frac{500000 * SoET}{4 + SoET}}$

* SoET = size of ElementType

شبه کد ترانهاده ماتریس معمولی

```
Transpose ( A : MATRIX ) : MATRIX
{
        var B : MATRIX ;
        CREATE(B, A.col, A.row);
        for i = 1 to A.row do
                for j = 1 to A.col do
                        B[j][i] = A[i][j];
        return B;
}
```

ضرب در ماتریس معمولی

```
Multiply ( A, B : MATRIX ) : MATRIX
{
        var C: MATRIX ;
        CREATE(C, A.row, B.col);
        for i = 1 to A.row do
                for j = 1 to B.col do
                {
                        var sum : integer;
                        sum = 0;
                        for k = 1 to A.col do
                                sum += A[i][k] * B[k][j];
                        C[i][j] = sum;
                }
        return C;
}
```

الگوریتم اول ترانهاده ماتریس اسپارس

```
Transpose( A : SPARSE) : SPARSE
{
    var B : SPARSE;
    CREATE(B, A.originColumn, A.originRow);
    B.terms = A.terms;
    for i = 1 to B.terms do
    {
        B.Data[i][1] = A.Data[i][2];
        B.Data[i][2] = A.Data[i][1];
        B.Data[i][3] = A.Data[i][3];
    }
    stably sort B.Data on the first column;
    return B;
}
```

بدنه ی حلقه ی for ، از مرتبه ی $\theta(terms)$ است. ولی عملیات مرتب سازی، از مرتبه ی $\theta(terms \times \log(terms))$ است. پس می توان نتیجه گرفت که مرتبه ی کل الگوریتم ، $\theta(terms \times \log(terms))$ می باشد و اگر terms از مرتبه ی $\theta(originRow \times originColumn)$ باشد، پیچیدگی زمانی الگوریتم در بدترین حالت برابر خواهد بود با :

$$\theta(originRow \times originColumn \times \log(originRow \times originColumn))$$

الگوریتم دوم ترانهاده ماتریس اسپارس

```
Transpose( A : SPARSE) : SPARSE
{
    var B : SPARSE;
    CREATE(B, A.originColumn, A.originRow);
    B.terms = A.terms;
    if B.terms > 0 then
    {
        var indexB : integer;
        indexB = 1;
        for cols = 1 to A.originColumn do
            for i = 1 to A.terms do
                if A.Data[i][2] == cols then
                {
                    B.Data[indexB][1] = cols;
                    B.Data[indexB][2] = A.Data[i][1];
                    B.Data[indexB][3] = A.Data[i][3];
                    indexB++;
                }
    }
    return B;
}
```

بدنه ی حلقه از مرتبه ی $\theta(originColumn \times terms)$ می باشد. در بدترین حالت که terms از مرتبه ی $\theta(originRow \times originColumn)$ باشد ، پیچیدگی زمانی الگوریتم فوق برابر

$$\boldsymbol{\theta(originRow \times originColumn^2)}$$

خواهد شد. این مرتبه ی پیچیدگی زمانی بسیار بدتر از الگوریتم ترانهاده در ماتریس معمولی است که از مرتبه ی $\theta(originRow \times originColumn)$ می باشد.

الگوریتم سوم ترانهاده ماتریس اسپارس

این الگوریتم ابتدا تعداد عناصر موجود در هر ستون ماتریس اصلی A (ماتریس اولیه) را تعیین می کند که این مقدار برابر تعداد عناصر هر سطر ماتریس اصلی B (ماتریس ترانهاده) است. سپس، با استفاده از مقادیر به دست آمده، می تواند مشخص کند که عنصر با شماره سطر جدید k ، باید در چه مکانی از ماتریس B قرار بگیرد؛ زیرا تعداد عناصر هر سطر مشخص شده است و تنها کافی است جمع تعداد عناصر سطر های قبلی محاسبه شود تا مشخص گردد عنصر با شماره سط k ، کدام یک از ردیف های ماتریس B را اشغال می کند. حال که محل قرارگیری هر عنصر مشخص شد، عناصر یکی یکی در ماتریس اسپارس B درج می شوند.

```
Transpose( A : SPARSE) : SPARSE
{
    var B : SPARSE;
    CREATE(B, A.originColumn, A.originRow);
    B.terms = A.terms;
    if B.terms > 0 then
    {
        var rowSize : integer[B.originRow];
 9      for i = 1 to B.originRow do
            rowSize[i] = 0;
 11     for i = 1 to A.terms do
            rowSize[A.Data[i][2]]++;
        var startOfRow : integer[B.originRow];
        startOfRow[1] = 1;
 15     for i = 2 to B.originRow do
            startOfRow[i] = startOfRow[i-1] + rowSize[i-1];
 17     for i = 1 to A.terms do
        {
            int x = startOfRow[A.Data[i][2]];
            B.Data[x][1] = A.Data[i][2];
            B.Data[x][2] = A.Data[i][1];
            B.Data[x][3] = A.Data[i][3];
            startOfRow[A.Data[i][2]] ++ ;
        }
    }
    return B;
}
```

حلقه ی خط ۹ و ۱۵ از مرتبه ی $\theta(originRow)$ و حلقه ی خط ۱۱ و ۱۷ از مرتبه ی $\theta(terms)$ می باشند.
پس در مجموع ، الگوریتم دارای پیچیدگی زمانی $\theta(terms + originRow)$ می باشند.
چون در بدترین حالت ، terms از مرتبه ی $\theta(originRow \times originColumn)$ است ، پس این اگوریتم در بدترین حالت از مرتبه ی

$$\boldsymbol{\theta(originRow \times originColumn)}$$

می باشد که برابر مرتبه ی زمانی الگوریتم ترانهاده در یک ماتریس معمولی است.

این الگوریتم الهام گرفته از الگوریتم مرتب سازی شمارشی است.

در «الگوریتم اول برای ترانهاده ی ماتریس اسپارس» از یک الگوریتم مرتب سازی استفاده شد که از مرتبه $\theta(n \times \log n)$ بود ، و این موضوع سبب شد که **پیچیدگی زمانی الگوریتم اول ترانهاده ، در بدترین حالت** برابر مقدار زیر شود :

$$\theta(originRow \times originColumn \times \log(originRow \times originColumn))$$

پس از استفاده از الگوریتم مرتب سازی شمارشی (فصل ۱۱) ، مرتبه ی الگوریتم اول ترانهاده به $\theta(terms + originRow)$ و در بدترین حالت به $\theta(originRow \times originColumn)$ بهبود یافت.

تمرین برنامه نویسی

برنامه ای بنویسید که عملیات جمع ، تفریق و ضرب ماتریس های اسپارس را انجام دهد.

ورودی های برنامه ماتریس هایی با تعداد زیادی صفر هستند و شما باید در ابتدا آنها را اسپارس کنید و صفرهای آن را به صورت درست حذف کنید و سپس ماتریس های تبدیل شده را در توابع خود استفاده کنید.

سعی کنید برنامه را بصورت بهینه بنویسید.

* گرافیک نمره اضافه دارد.

پایان