ساختمان داده ها
Data Structure

فصل دوم
آرایه ها

تهیه و تنظیم : محمد نعیمی

عضو هیات علمی دانشگاه آزاد اسلامی

آرایه ها

تعریف:
به خانه های هم نوع، از حافظه که دارای یک نام هستند و بر اساس اندیس های عددی از هم متفاوت میشوند آرایه گفته میشود

اندیس آرایه در زبان c++ از عدد 0 شروع می شود تا n-1 (۱–تعداد خانه). type a[n]
اما در زبانی مثل پاسکال از خانه دلخواه (L) تا خانه دلخواه (U) type a[L..U]

در رابطه با آرایه (هر خانه حافظه) به دو عمل اساسی نیاز است :

۱– ذخیره داده ها ۲– بازیابی داده ها

برای ذخیره سازی آرایه، آدرس اولین خانه آرایه در متغیر آرایه قرار داده می شود(متغیر آرایه عملا اشاره گر است). با انجام محاسباتی، آدرس هر خانه دلخواه محاسبه و دسترسی به آن خانه امکان پذیر می شود.

با توجه به تفاوت اندیس گذاری در زبان C++ و پاسکال و زبانهایی که مشابه این دو عمل میکنند ما روشها را بر اساس مدل C++ و پاسکال نمایش میدهیم.

برای ذخیره هر نوع داده (type) تعدادی خانه از حافظه نیاز است که در جدول روبرو نمایش داده شده است. این مقادیر در زبانهای برنامه نویسی ممکن است متفاوت باشد اما ملاک ما در این درس این اعداد است.

type	Size (type)
char	1
int	2
float	4
double	8

آدرس خانه ها در آرایه

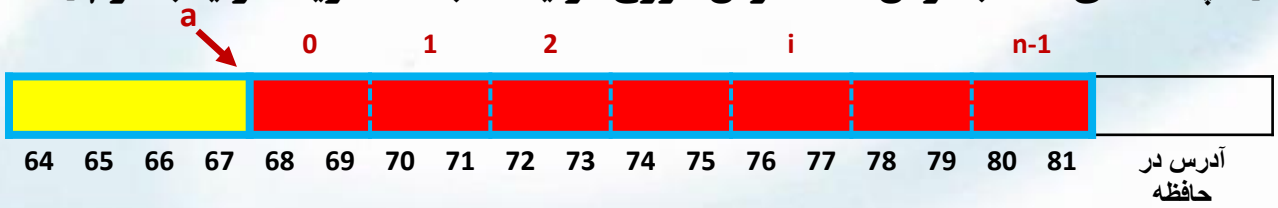
برای آرایه در زبان C++ ، با فرض تعریف آرایه به فرم type a[n] (یعنی آرایه از نوع type است) آدرس خانه i ام از فرمول زیر بدست می آید.

آدرس a[i] = a + i * size(type)

برای آرایه در زبان پاسکال ، با فرض تعریف آرایه به فرم type a[L..U] (یعنی آرایه از نوع type است) آدرس خانه i ام از فرمول زیر بدست می آید.

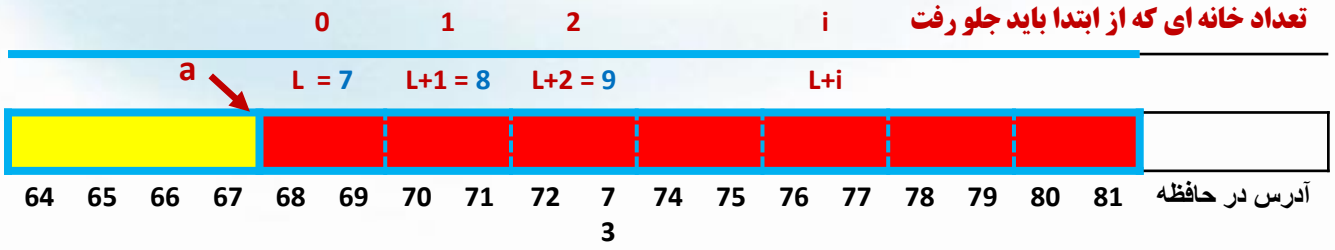
آدرس a[i] = a + (i - L) * size(type)

مثال آدرس خانه a[5] چه عددی است با فرض انکه ادرس شروع آرایه 68 باشد. تعریف آرایه به فرم int a[100] است



آدرس a[i] = a + i * size(int) = 68 +5*2 = 78

مثال آدرس خانه a[9] چه عددی است با فرض انکه ادرس شروع آرایه 68 باشد. تعریف آرایه به فرم int a[7..15] است



آدرس a[i] = a + (i – L) * size(int) = 68 + (9 – 7) * 2= 68+4=72

ماتریس ها

تعریف:
به آرایه هایی که بیش از یک بعد دارند ماتریس گفته می شود (دو بعد یا بیشتر)

تعریف ماتریس در زبان C++ به فرم مقابل است.

type a[n1][n2];	ماتریس دو بعدی دارای n1 سطر و n2 ستون
type a[n1][n2]...[nk];	ماتریس k بعدی

تعریف ماتریس در زبان پاسکال به فرم مقابل است.

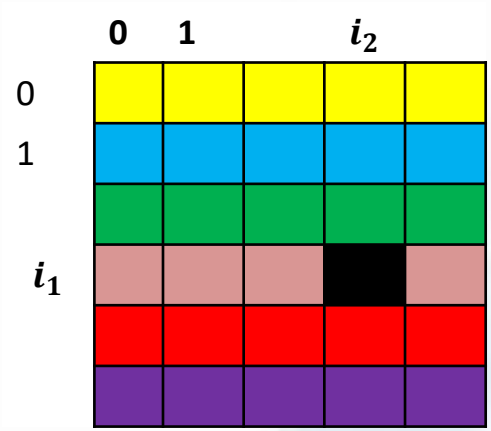
type a[L1..U1][L2.. U2];	ماتریس دو بعدی دارای U1-L1+1 سطر و U2-L2+1 ستون
type a[L1..U1][L2.. U2] ...[Lk.. Uk]; ;	ماتریس k بعدی

از آنجایی که حافظه سیستم تک بعدی است(خانه ها شماره ترتیبی دارند) لازم است آرایه با دو بعد یا بیشتر در ساختاری مشخص با یک بعد ذخیر شوند.
برای این کار میتوان ماتریس را سطر به سطر یا ستون به ستون ذخیره نمود. (فعلا مفهوم سطر و ستون را در فضای دو بعدی در نظر بگیرید)

تهیه و تنظیم : محمد نعیمی

آدرس خانه ها در ماتریس (روش سطری)

برای ذخیره کردن ماتریس دو بعدی میتوان از روش سطری استفاده نمود.
در مثال زیر سطر ها به ترتیب پشت سر هم ذخیره می شوند.



برای ماتریس مقابل آدرس خانه a[i_1][i_2] را با فرض آنکه آدرس شروع آرایه a باشد را در دو حالت C++ و پاسکال نمایش می دهیم

C++ روش سطری — **تعریف ماتریس** type a[n_1][n_2]

آدرس a[i_1] [i_2] = a + [i_1 * n_2 + i_2] * size(type)

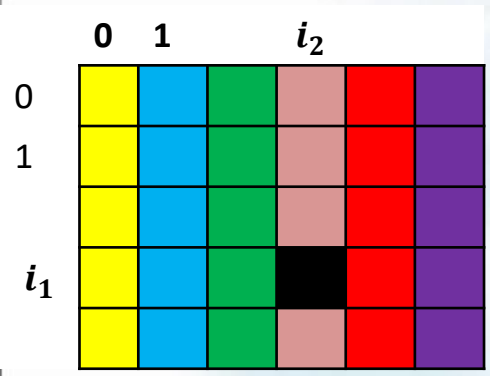
پاسکال روش سطری — **تعریف ماتریس** type a[L_1..U_1][L_2..U_2]

آدرس a[i_1] [i_2] = a + [(i_1 - L_1) * (U_2 - L_2+1) + i_2 - L_2] * size(type)

آدرس خانه ها در ماتریس (روش ستونی)

برای ذخیره کردن ماتریس دو بعدی میتوان از روش سطری استفاده نمود.
در مثال زیر سطر ها به ترتیب پشت سر هم ذخیره می شوند.

برای ماتریس مقابل آدرس خانه a[i_1][i_2] را با فرض آنکه ادرس شروع آرایه a باشد را در دو حالت C++ و پاسکال نمایش می دهیم



C++ روش ستونی **تعریف ماتریس** type a[n_1][n_2]

آدرس a[i_1] [i_2] = a + [i_2 * n_1 + i_1] * size(type)

پاسکال روش ستونی **تعریف ماتریس** type a[$L_1..U_1$][$L_2..U_2$]

آدرس a[i_1] [i_2] = a + [(i_2 _ L_2) * (U_1 - L_1 +1) + i_1 - L_1] * size(type)

آدرس خانه ها در ماتریس سه بعدی

سطری C++

type a[n_1][n_2][n_3]

آدرس a[i_1] [i_2] [i_3] = a + (i_1 * n_2 * n_3 + i_2 * n_3 + i_3) * size(type)

سطری پاسکال

type a[L_1..U_1][L_2..U_2][L_3..U_3]

آدرس a[i_1][i_2][i_3]= a + ((i_1 - L_1) * (U_2 - L_2+1)* (U_3 - L_3+1) + (i_2 - L_2) *(U_3 - L_3+1) + (i_3 - L_3)) * size(type)

ستون C++

type a[n_1][n_2][n_3]

آدرس a[i_1] [i_2] [i_3] = a + (i_3 * n_2 * n_1 + i_2 * n_1 + i_1) * size(type)

ستون پاسکال

type a[L_1..U_1][L_2..U_2][L_3..U_3]

آدرس a[i_1][i_2][i_3]= a + ((i_3 - L_3) * (U_2 - L_2+1)* (U_1 - L_1+1) + (i_2 - L_2) *(U_1 - L_1+1) + (i_1 - L_1)) * size(type)

برای فهم راحت تر فرمولهای بالا، در روش سطری هر اندیس در طول تمام بُعد های بعدش ضرب میشود و در روش ستونی در طول بُعد های قبلش

برای تبدیل فرمول از C++ به حالت پاسکال کافی است تغییرات مقابل را رو فرمولهای زبان C++ انجام دهیم

$i_t \rightarrow i_t - L_t$

$n_t \rightarrow U_t - L_t + 1$

تهیه و تنظیم: محمد نعیمی

آدرس خانه ها در ماتریس چند بعدی

فرمول را برای حالت C++ می نویسیم.

برای آرایه type a$[n_1][n_2]...[n_k]$ آدرس خانه a$[i_1][i_2]...[i_k]$چه عددی است با فرض انکه ادرس شروع آرایه a باشد

روش سطری C++

آدرس a$[i_1]$ $[i_2]$... $[i_k]$ = a + ($i_1 * n_2 * n_3 * \cdots * n_k + i_2 * n_3 * \cdots * n_k$ +...+ $i_{k-1} * n_k + i_k$) * size(type)

روش ستونی C++

آدرس a$[i_1]$ $[i_2]$... $[i_k]$ = a + ($i_k * n_{k-1} * n_{k-2} * \cdots * n_1 + i_{k-1} * n_{k-2} * \cdots * n_1$+... + $i_2 * n_1 + i_1$) * size(type)

برای فهم راحت تر فرمولهای بالا، در روش سطری هر اندیس در طول تمام بُعد های بعدش ضرب میشود و در روش ستونی در طول بُعد های قبلش

برای یافتن فرمولهای فوق برای حالت پاسکال کافی است تغییرات مقابل را رو فرمولهای زبان C++ انجام دهیم

$i_t \rightarrow i_t - L_t$

$n_t \rightarrow U_t - L_t + 1$

مثال:

سوال: برای آرایه int a[5][7][4]آدرس خانه a[3][6][2] در حالت سطری و ستونی چه عددی است. با فرض انکه آدرس شروع آرایه 2000 باشد؟

پاسخ:

فرم تعریف آرایه مدل c++ است و فضای int دارای طول 2 است لذا

آدرس سطری = 2000 + [3*7*4 + 6*4 + 2] * 2 = 2000 + (84+24+2)*2 = 2000+220=2220

آدرس ستونی = 2000 + [2*7*5 + 6*5 + 3] * 2 = 2000 + (70+30+3)*2 = 2000+206=2206

سوال: برای آرایه int a[2..6][3..9][4..7]آدرس خانه a[5][9][6] در حالت سطری و ستونی چه عددی است. با فرض انکه آدرس شروع آرایه 2000 باشد؟

پاسخ:

فرم تعریف آرایه مدل پاسکال است و فضای int دارای طول 2 است. میتوان مسئله را طبق فرمول گفته شده به فرم c++ تبدیل نمود :

آرایه int a [2..6] [3..9] [4..7] آدرس خانه a [5] [9] [6]

آرایه int a[6-2+1][9-3+1][7-4+1] آدرس خانه a[5-2][9-3][6-4]

آرایه int a[5][7][4] آدرس خانه a[3][6][2]

آدرس سطری = 2000 + [3*7*4 + 6*4 + 2] * 2 = 2000 + (84+24+2)*2 = 2000+220=2220

آدرس ستونی = 2000 + [2*7*5 + 6*5 + 3] * 2 = 2000 + (70+30+3)*2 = 2000+206=2206

ماتریس پایین مثلثی در C++:

ماتریس پایین مثلثی ماتریسی است که عناصر بالای قطر اصلی آن 0 می باشند.
در این روش دنبال آدرس نیستیم بلکه میخواهیم عناصر ماتریس را در آرایه ذخیره کنیم به نحوی که با یک فرمول بدانیم هر عنصر در چه خانه ای ذخیره می شود و خانه های 0 را اصلا ذخیره نکنیم.

سطر	تعداد عناصر قبل از آن سطر
0	0
1	1
2	1+2
3	1+2+3
i	$1+2+3+\ldots+i = \frac{i(i+1)}{2}$

در سطر i ام قبل از عنصر ستون 0 ام 0 عنصر، قبل از 1 ام 1 عنصر و ... قبل از عنصر ستون j ام j عنصر ذخیر شده که باید به همان اندازه جلو برویم

$$A[i][j] = \begin{cases} B\left[\frac{i*(i+1)}{2}+j\right] & i \geq j \\ 0 & i < j \end{cases}$$

آرایه B جهت ذخیره سازی ماتریس A

A[0][0]	A[1][0]	A[1][1]	A[2][0]	A[2][1]	A[2][2]	A[3][0]	A[3][1]	A[3][2]	A[3][3]	

برای پاسکال با شروع اندیس از عدد 1 کافیست بجای (i+1) عبارت (i-1) قرار دهیم

برای بالا مثلثی کافیست در سمت راست فرمول به طور کامل i و j ها را با هم جابجا کنیم

تهیه و تنظیم: محمد نعیمی

ماتریس سه قطری و مثال از ماتریس مثلثی

سوال: برای ذخیره سازی ماتریس پایین مثلثی، عنصر a[6][3] در چه خانه ای از b ذخیره می شود؟

پاسخ:

چون پایین مثلثی است و i>=j است جواب می شود.

$$\frac{6 * 7}{2} + 3 = 24$$

یعنی در خانه b[24] ذخیره می شود

سوال: برای ذخیره سازی ماتریس پایین مثلثی، عنصر a[3][6] در چه خانه ای از b ذخیره می شود؟

پاسخ:

چون پایین مثلثی است و i<j است لذا این عنصر بالای قطر بوده و صفر می باشد و در آرایه نگهداری نمی شود

سوال: برای ذخیره سازی ماتریس بالا مثلثی، عنصر a[5][7] در چه خانه ای از b ذخیره می شود؟

پاسخ:

چون بالا مثلثی است و j>=i است جواب می شود.

$$\frac{7 * 8}{2} + 5 = 33$$

یعنی در خانه b[33] ذخیره می شود

برای ذخیره ماتریس سه قطری در آرایه فرمول زیر وجود دارد

پاسکال به شرطی که اندیس شروع 1 باشد

$$A[i][j] = \begin{cases} B[2i + j - 2] & |i - j| \leq 1 \\ 0 & other \end{cases}$$

C++

$$A[i][j] = \begin{cases} B[2i + j] & |i - j| \leq 1 \\ 0 & other \end{cases}$$

تهیه و تنظیم: محمد نعیمی

ماتریس خلوت Sparse

ماتریسی که حداقل $\frac{2}{3}$ از عناصر آن 0 یا یک عدد ثابت باشند.
جهت ذخیره ماتریس خلوت از آرایه از رکورد استفاده میکنیم. هر خانه آرایه سه فیلد دارد(row , col , val)
کافیست به صورت سطر به سطر عناصر غیر صفر را به صورت شماره سطر row، ستون col و مقدار val ذخیر نماییم.
در سطر 0 در خانه row تعداد کل سطر ها، در خانه col تعداد کل ستونها و در خانه val تعداد کل مقادیر غیر صفر را قرار دهیم.

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

ذخیره ماتریس اسپارس

	row	col	val
0	6	8	7
1	0	0	5
2	0	2	4
3	0	5	1
4	1	3	2
5	1	6	4
6	4	1	2
7	5	5	7
8			

کد تولید ماتریس خلوت Sparse و چاپ آن

تعریف آرایه mat
از نوع رکورد

```
struct sparse_item{
                    int row,col,val;
                   }mat[100];
```

ساخت آرایه mat به
عنوان ماتریس خلوت از
روی ماتریس خلوت a که
m سطر و n ستون دارد

```
int i,j,count=0;
for (i=0;i<m;i++)
    for (j=0;j<n;j++)
        if (a[i][j]!=0)
            {
              count++;
              mat[count].row=i;
              mat[count].col=j;
              mat[count].val=a[i][j];
            }
mat[0].row=m;
mat[0].col=n;
mat[0].val=count;
```

چاپ ماتریس خلوت mat

```
for(i=0;i<=mat[0].val;i++)
      cout<<mat[i].row<<" "<<mat[i].col<<" "<<mat[i].val<<"\n";
```

الگوریتم ترانهاده (تعویض جای سطر و ستون)ماتریس خلوت

در ماتریس اسپارس ترتیب عناصر باید طبق سطر مرتب باشدو در سطر های یکسان طبق ستون. هنگام ترانهاده کردن جای سطر و ستون عوض میشود، لذا کافی است برای هر ستون (از 0 تا ستون n-1) کل ماتریس را از ابتدا تا انتها پیمایش نماییم و هر عنصری که ستونش برابر این مقدار شد به ماتریس اسپارس ترانهاده منتقل شود. در انتها هم سطر صفر فقط کافی است تعداد سطر و ستون را جا بجا کنیم. پیچیدگی این عملیات برابر با **تعداد ستون * تعداد عناصر** می باشد.

```
count=0;
   for(i=0;i<mat[0].col;i++)
    for(j=1;j<=mat[0].val;j++)
      if (mat[j].col==i)
        {
            count++;
            matT[count].row=mat[j].col;
            matT[count].col=mat[j].row;
            matT[count].val=mat[j].val;
        }
   matT[0].row=mat[0].col;
   matT[0].col=mat[0].row;
   matT[0].val=mat[0].val;
```

row	col	val
6	8	7
0	0	5
0	2	4
0	5	1
1	3	2
1	6	4
4	1	2
5	5	7

ترانهاده ماتریس اسپارس

row	col	val
8	6	7
0	0	5
1	4	2
2	0	4
3	1	2
5	0	1
5	5	7
6	1	4

الگوریتم ضرب دو ماتریس

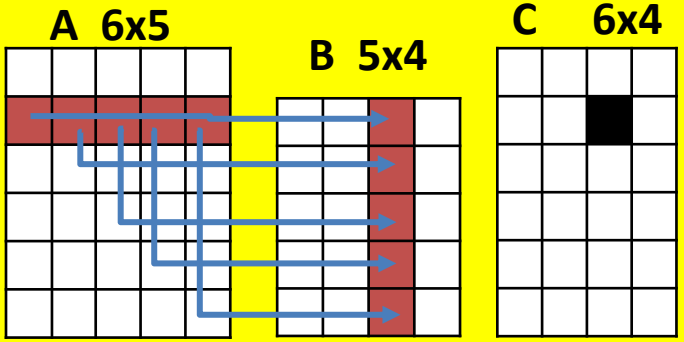
```
void mul_mat(int a[][30], int b[][30], int c[][30],int m,int n,int p);
{
   for (int i=0;i<m;i++)
       for (int j=0;j<p;j++)
           {
               c[i][j]=0;
               for (int k=0;k<n;k++)
                   c[i][j]+=a[i][k]*b[k][j];

           }
}
```

$$A_{m\times n} \times B_{n\times p} = C_{m\times p}$$

$$C_{ij} = \sum_{k=0}^{n-1} A_{ik} \times B_{kj}$$

m=3 n=5 p=4



مثال عددی

2	5
3	6
1	4

3	8	4	6
7	1	2	9

2*3+5*7	2*8+5*1	2*4+5*2	2*6+5*9
3*3+6*7	3*8+6*1	3*4+6*2	3*6+6*9
1*3+4*7	1*8+4*1	1*4+4*2	1*6+4*9

c[1][2]=a[1][0]*b[0][2]+a[1][1]*b[1][2]+a[1][2]*b[2][2]+a[1][3]*b[3][2]+a[1][4]*b[4][2]

برخی الگوریتم های ماتریس

جمع دو ماتریس: باید دو ماتریس از نظر تعداد سطر با هم برابر و از نظر تعداد ستون نیز برابر باشند

```
void sum_mat(int a[][30], int b[][30], int c[][30],int m,int n);
{
   for (int i=0;i<m;i++)
      for (int j=0;j<n;j++)
               c[i][j]=a[i][j]*b[i][j];
}
```

$$A_{m\times n} + B_{m\times n} = C_{m\times n}$$

$$C_{ij} = A_{ij} + B_{ij}$$

ترانهاده: در ترانهاده باید جای سطر و ستون را عوض کرد یعنی عنصر a[i][j] را با عنصر a[j][i] جابجا کرد. در حالتی که ماتریس مربعی نیست باید فضای در نظر گرفته شده برای سطر و ستون به یک اندازه باشد چون تعداد سطر و ستون ها عوض می شود.

```
void tran_mat(int a[][30],int m,int n);
{
   for (int i=0;i<m;i++)
      if (m>=n)
         for (int j=0;j<i && j<n;j++)
            swap(a[i][j],a[j][i]);
      else
         for (int j=i+1;j<n;j++)
            swap(a[i][j],a[j][i]);
}
```

برای جابجایی یا باید به عناصر بالای قطر اصلی گفت که با عنصر متناظر در پایین قطر اصلی جابجا شوند یا بر عکس.
در ماتریس های غیر مربعی اگر تعداد سطرها بیشتر باشد باید به عناصر زیر قطر اعلام کرد تا با متناظر آنها(برای سطرهای پایینی گرچه به صورت منطقی وجود ندارد اما در ماتریس آن خانه ها مقدارشان صفر است) جا بجا شوند و برای حالتی که تعداد ستون ها بیشتر است به عناصر بالای قطر.

جستجو خطی آرایه

دو نوع جستجو در آرایه ها وجود دارد

- **خطی:** این جستجو برای هر نوع آرایه ای قابل انجام است. پیچیدگی این نوع جستجو o(n) می باشد.
- **دودویی:** این جستجو را فقط روی آرایه های مرتب می توان انجام داد. پیچیدگی این نوع جستجو o(log n) است

```
int find_line(int a[], int n , int x);
{
   for (int i=0;i<n;i++)
      if (a[i]==x)
         return i;
   return -1 ;
}
```

در این نوع جستجو از **ابتدای آرایه** به ترتیب تا **آخرین خانه** آرایه پیمایش نموده و هنگامی که عنصری برابر با x پیدا کردیم کار را تمام کرده اندیس مورد نظر را بر می گردانیم.

بهترین حالت وقتی است که اول عنصر آرایه برابر با x باشد که یک مقایسه خواهیم داشت

بدترین حالت وقتی است که آخرین عنصر ارایه برابر با x باشد که n مقایسه خواهیم داشت.

به طور متوسط $\frac{n}{2}$ پیمایش خواهیم داشت و پیچیدگی در حالت متوسط و بدترین حالت o(n) خواهد بود.

جستجو دودویی (باینری) آرایه

جستجوی دودویی فقط بر روی آرایه های مرتب قابل انجام است.
در این جستجو هر مرحله بازه جستجو را نصف میکنیم تا یا عدد پیدا شود یا بازه تهی شود (left>right) که با تهی شدن بازه میتوان اعلام کرد عدد پیدا نشده است. چون بازه هر بار تقسیم به 2 می شود پیچیدگی این روش o(log n) می باشد.
بازه توسط دو متغیر left و right مشخص می شود و ابتدا کل اندیس ها را شامل می شود [0 n-1]
هر بار وسط بازه را پیدا میکنیم mid=(left+right)/2. اگر عنصر آنجا بود، اندیس را بر میگردانیم وگرنه دو حالت داریم:

اگر x از محتوای خانه mid کوچکتر بود پس باید در نیمه اول بازه دنبال x بگردیم لذا right را به خانه ماقبل mid منتقل می کنیم و گرنه باید در نیمه دوم به دنبال x باشیم یعنی left رابه خانه بعد از mid منتقل میکنیم.

تابع به صورت غیر بازگشتی

```
int find_bin(int a[], int n , int x);
{
   int left=0,right=n-1,mid;
   while(left<=right)
      {
         mid=(left+right)/2;
         if (a[mid]==x)
             return(mid);
         if (x<a[mid])
             right=mid-1;
         else
             left=mid+1;
      }
   return -1;
}
```

تابع به صورت بازگشتی

```
int find_bin_rec(int a[], int left,int right , int x);
{
   int mid;
   if (left>right)
       return -1;
   mid=(left+right)/2;
   if (a[mid]==x)
             return(mid);
   if (x<a[mid])
        return(find_bin_rec(a,left,mid-1,x));
   return(find_bin_rec(a,mid+1,right,x));
}
```

نمایش چند جمله ای با آرایه

برای نمایش چند جمله ای توسط آرایه ساده ترین راه این است که محتوای خانه i ام به عنوان ضریب x^i در نظر گرفته می شود.

$$3x^7 + x^6 - 2x^3 + 9$$

9	0	0	-2	0	0	1	3
0	1	2	3	4	5	6	7

روش قبل برای حالاتی مثل $5x^{100} - 9x^{45} + 4$ حافظه خالی زیادی مصرف میکند. راه حل بهینه استفاده از آرایه ای با دو فیلد توان و ضریب می باشد. در این حالت یک متغیر هم برای مشخص کردن تعداد خانه های قابل قبول آرایه داریم

ساختار رکورد چند جمله ای

```
struct poly{
        int pow,coef;
    };
```

ضریب coef	5	-9	4
توان pow	100	45	0
	0	1	2

تابع مجموع دو چند جمله ای

```
int som_poly(poly a[], int n , poly b[], int m, poly c[] , int &k);
{
   int i,j=0;
   k=0;
   while(i<n && j<m)
     {
        if (a[i].pow>b[j].pow)
          {
            c[k].pow=a[i].pow;
            c[k++].coef=a[i++].coef;
          }
       else if (a[i].pow<b[j].pow)
          {
            c[k].pow=b[j].pow;
            c[k++].coef=b[j++].coef;
          }
       else if (a[i].coef+ b[j].coef!=0)
          {
            c[k].pow=a[i].pow;
            c[k++].coef=a[i++].coef+ b[j++].coef;
          }
       else
           {
              i++;
              j++;
          }
     }
   while(i<n)
     {
        c[k].pow=a[i].pow;
        c[k++].coef=a[i++].coef;
     }
   while(j<m)
     {
        c[k].pow=b[j].pow;
        c[k++].coef=b[j++].coef;
     }
}
```