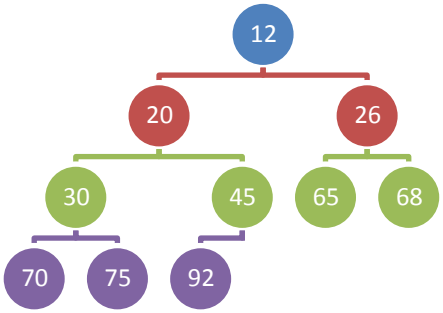
روش دوم محاسبه ی $S_{(n)}$ و $U_{(n)}$

می توان با اعداد داده شده یک درخت کامل ساخت و سپس مجموع تعداد گره ها در یک سطح و سطح آن گره ها را محاسبه کرد و نهایت حاصل را به تعداد تقسیم کرد.

مثال:

1	2	3	4	5	6	7	8	9	10
12	20	26	30	45	65	68	70	75	92



$$\frac{1\times 1+2\times 2+4\times 3+3\times 4}{10}=\frac{29}{10}$$

مثال: میانگین تعداد مقایسه ها برای پیدا کردن یک عنصر معلوم به روش جستوجوی دودویی در یک آرایه ی ۲۰۰ عنصری مرتب کدام است؟

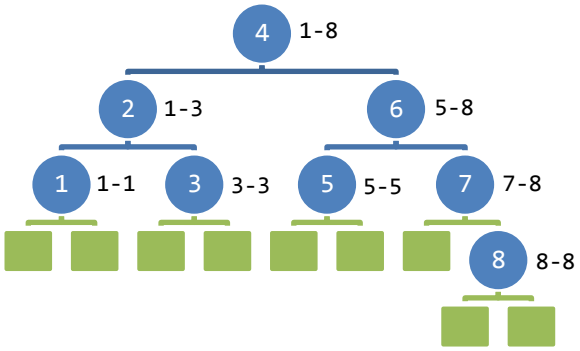
شماره سطر (k)	1	2	3	4	5	6	7	8
تعداد (N_k)	1	3	4	8	16	32	64	73

$$S_{(n)}=\frac{1}{n}\sum_{k=0}^{8}k.N_k=\frac{1\times 1+2\times 2+3\times 4+4\times 8+5\times 16+6\times 32+7\times 64+8\times 73}{200}$$

محاسبه ی کارایی جستجوی دودویی

$$k=\lfloor \log n\rfloor+1$$

$$k=\lceil \log n+1\rceil$$



n = 8

ارتفاع درخت k = 4

جستجوی ناموفق $U_{(n)}$ $\begin{cases} \text{حداقل} = 3 = k - 1 = \log n \\ \text{حداکثر} = 4 = k = \log n + 1 \end{cases}$

جستجوی موفق $S_{(n)}$ $\begin{cases} \text{حداقل} = 1 \\ \text{حداکثر} = 4 = k = \log n + 1 \end{cases}$

	بهترین	متوسط	بدترین
جستجوی موفق	$\Omega(1)$	$\theta(\log n)$	$O(\log n)$
جستجوی ناموفق	$\Omega(\log n)$	$\theta(\log n)$	$O(\log n)$

متوسط زمان:

$$k(1 + 2 + \cdots + k) = \frac{1}{k}\left(\frac{k(k+1)}{2}\right) = \frac{k+1}{2}$$

$$\frac{\lfloor \log n \rfloor + 1 + 1}{2} = \frac{1}{2}\lfloor \log n \rfloor + 1 = \log \sqrt{n} + 1$$

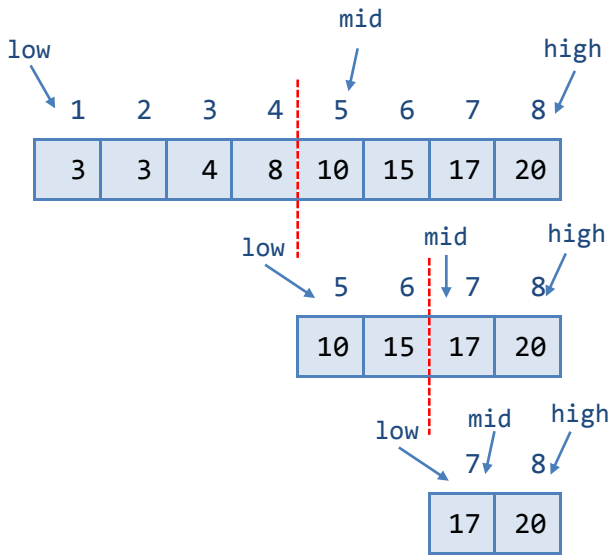
برای متوسط زمان هم $\log n$ و هم $\log \sqrt{n}$ قبول است. اما در تست ها و کنکور اگر دادند $\log \sqrt{n}$ دقیق تر است.

اشکال جستوجوی دودویی

اشکال این است که دو if دارد: ۱- مساوی بودن ۲- چپ یا راست بودن

الگوریتم جستجوی دودویی بهینه

همیشه مساوی بودن را چک می کردیم و `low=mid+1` ولی در اینجا چون چک نکردیم `low=mid` می باشد.



کد جستجوی دودویی بهینه در تعداد مقایسه ها

```
Boolean binery_search(int low,int high)
        {
            if (low < high - 1)
            {
                mid = (low + high) / 2;
                if (num < x[mid])
                    return binery_search(low, mid);
                else
                    return binery_search(mid, high);
            }
            else
            {
                if (x[mid] == num)
                    return true;
                else
                    return false;
            }
        }
```

تمرین ۱: زمان اجرای جستجوی دودویی بهینه را بدست آورید.

$a_1 < a_2 < a_3 < \cdots < a_i + a_{i+1} < \cdots < a_n$

تمرین ۲: الگوریتم غیر بازگشتی آن را بنویسید.

چند مسئله از جستوجوی دودویی

۱. یک فهرست از اعداد مرتب شده ی چرخشی به شما داده شده است، موفقیت کوچک ترین عدد فهرست را بیابید. (برای سادگی فرض می کنیم عنصر کوچک یکتاست)

۲. دنباله ای مرتب از اعداد صحیح متمایز به صورت a_1 و a_2 و ... و a_n داده شده است. مشخص کنید آیا اندیسی مانند i وجود دارد که $a_i = i$ است یا خیر.

۳. جستوجوی دودویی در دنباله هایی با اندازه های نامشخص پیاده سازی کنید و نشان دهید اگر پیاده سازی شود زمان جستوجوی دودویی بهینه می شود.

۴. جستوجوی دودویی را با درون یابی پیاده سازی نمایید.

$$mid = \left\lfloor low + \frac{(num - x[low])(high - low)}{x[high] - x[low]} \right\rfloor$$

زمان اجرا = $O(\log\log n)$

۵. حداقل و حداکثر ارتفاع درخت را بدست آورید.

۶. برنامه ای بنویسید که در یک آرایه مرتب تعداد عناصر برابر num را چاپ کند. (بازمان $\log n$)

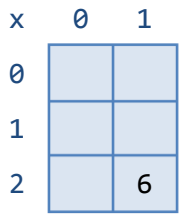
آرایه های دو بعدی

در زبان C

نوع آرایه نام آرایه [تعداد سطر ها][تعداد ستون ها]

مثال:

```
int x[3][2]
```



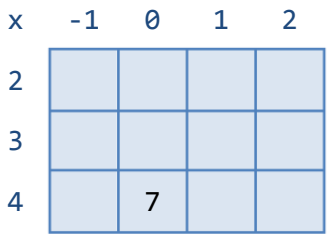
تعداد عناصر = تعداد سطرها × تعداد ستون ها

```
x[3][2] = 6
```

در زبان پاسکال (Pascal)

نام آرایه: array $[L_1..U_1,L_2..U_2]$of نوع آرایه

```
x: array [2..4,-1..2]of integer
```




```
x[4,0]:=7
```

تعداد سطرها $= U_1 - L_1 + 1$

تعداد ستون ها $= U_2 - L_2 + 1$

تعداد عناصر آرایه $= (U_1 - L_1 + 1)(U_2 - L_2 + 1)$

نکته: در ماتریس A، عنصر A[i,j] روی قطر فرعی، i+j=n+1

نحوه ی ذخیره سازی آرایه های دو بعدی

۱- سطری

۲- ستونی

روش سطری در زبان C

```
int x[3][4] = {{7,4,1,-2},{0,3,6,19},{1,4,8,9}}
```

x	0	1	2	3
0	7	4	1	-2
1	0	3	6	19
2	1	4	8	9

حافظه مصرفی = 4 (بایت)

آدرس شروع : α = 1000

1000	1004	1008	1012	1016	1020	1024	1028	1032	1036	1040	1044
7	4	1	2-	0	3	6	19	1	4	8	9

آدرس x[2][1] = 1000 + (2 × 4 + 1) × 4 = 1036

آدرس شروع — سطر — تعداد ستون درون سطر — ستون — حافظه مصرفی

x[n][m] m: تعداد سطر ها n: تعداد ستون ها

x[i][j] = α + (i × m + j) × حافظه مصرفی

مثال: x[70][100] و آدرس شروع 500 و آدرس x[10][15] را به روش سطری بدست آورید. (حافظه مصرفی = 4)

x[70][100]

آدرس x[10][15] = 500 + (10 × 100 + 15) × 4 = 4560

روش ستونی در زبان C

1000	1004	1008	1012	1016	1020	1024	1028	1032	1036	1040	1044
7	0	1	4	3	4	1	6	8	2-	19	9

آدرس x[1][3] = 1000 + (3 × 3 + 1) × 4 = 1040

آدرس شروع — ستون — تعداد سطر درون ستون — سطر — حافظه مصرفی

x[n][m] m: تعداد سطر ها n: تعداد ستون ها

x[i][j] = α + (i × n + j) × حافظه مصرفی

حل مثال قبل به روش ستونی:

آدرس x[10][15] = 500 + (15 × 70 + 10) × 4 = 4740

توجه: اگر نگفتند سطری یا ستونی است، پیش فرض سطری است.

توجه: اگر حافظه مصرفی را مشخص نکردند از مقدار متغیر های زبان C استفاده می کنیم.

مثال:

x[30][70]

حافظه مصرفی = 3 سطری : 500 + (12 × 70 + 14) × 3

α = 500 ستونی : 500 + (14 × 30 + 12) × 3

آدرس x[12][14]

ذخیره سازی آرایه های دو بعدی در زبان پاسکال (Pascal)

روش سطری: آدرس A[i,j] = α + [(i − L_1) ×(U_2 − L_2 + 1)+(j − L_2)]

روش ستونی: آدرس A[i,j] = α + [(j − L_2) ×(U_1 - L_1 + 1)+(i − L_1)]

در آرایه های سه بعدی

x[n][m][l]

روش سطری: آدرس x[i][j][k] = α + (i × m × l + j × l + k) × حافظه مصرفی

روش ستونی: آدرس x[i][j][k] = α + (k × n × m + j × n + i) × حافظه مصرفی

نکته: اگر 1 = حافظه مصرفی = α آنگاه آدرس x[i][j][k] همان عنصر چندم آرایه است.

اعمال آرایه های دو بعدی

جمع: $O(n^2)$ ⟵ دو تا حلقه for است

جمع: $O(n^3)$ ⟵ سه تا حلقه for است

در ضرب دو ماتریس $A_{m\times n} \times B_{n\times k} = C_{m\times k}$ به $n \times m \times k$ ضرب نیاز دارد.

دترمینان آرایه: $n!$

حل دستگاه n معادله و n مجهول (از روش کرامر): $n!$

$$A_{20\times 2} \times B_{2\times 30} \times C_{30\times 12} \times D_{12\times 8}$$

$$A\,(\,B\,(\,C\,D\,)\,) = 30 \times 12 \times 8 + 2 \times 30 \times 8 + 2\,0 \times 2 \times 8 = 3680$$

$$(\,A\,B\,)\,(\,C\,D\,) = 20 \times 2 \times 30 + 30 \times 12 \times 8 + 20 \times 30 \times 8 = 8880$$

$$A\,(\,(\,B\,C\,)\,D\,) = 2 \times 30 \times 12 + 2 \times 12 \times 8 + 20 \times 2 \times 8 = 1230$$

$$(\,(\,A\,B\,)\,C\,)\,D = 20 \times 2 \times 30 + 20 \times 30 \times 12 + 20 \times 12 \times 8 = 10320$$

$$(\,A\,(\,B\,C\,)\,)\,D = 2 \times 30 \times 12 + 20 \times 2 \times 12 + 20 \times 12 \times 8 = 3120$$

نکته: تعداد حالات شرکت پذیری ضرب n+1 ماتریس برابر است با: $\frac{\binom{2n}{n}}{n+1}$ (عدد کاتلان)

نکته: چه شرطی وجود داشته باشد تا ضرب سه ماتریس A_{wx} و B_{xy} و C_{yz} به صورت $(AB)C$ نسبت به $A(BC)$ بهتر باشند یعنی عملیات ضرب کمتری نیاز داشته باشند؟

$$\text{ضرب } (AB)C = wxy + wyz$$

$$\text{ضرب } A(BC) = xyz + wxz$$

$$\text{ضرب } (AB)C \;<\; \text{ضرب } A(BC)$$

$$wxy + wyz \;<\; xyz + wxz$$

$$\xrightarrow{\times \frac{1}{wxyz}} \boxed{\frac{1}{z} + \frac{1}{x} < \frac{1}{w} + \frac{1}{y}}$$

انواع ماتریس های معروف

۱- ماتریس اسپارس (خلوت) (Sparse)

۲- ماتریس مثلثی

۳- ماتریس سه قطری

۴- ماتریس متقارن

۵- ماتریس پله ای

ماتریس اسپارس: ماتریسی که دارای تعداد نسبتا زیادی عنصر صفر دارد را ماتریس می نامند. در این ماتریس برای کاهش حافظه مصرفی و زمان اجرا فقط عناصر غیر صفر ماتریس ذخیره می شود، ماتریس را می توان به دو طریق ذخیره کرد:

	1	2	3	4	5	6	7
1	0	1	0	0	0	0	1
2	0	0	0	2	0	0	0
3	0	0	0	0	0	0	0
4	3	0	0	0	4	0	0
5	7	0	0	0	0	8	0
6	0	0	0	0	0	1	0

$_{6\times7}$

سطر `row = 6`

ستون `col = 7`

تعداد عناصر غیر صفر `row = 6`

حافظه مصرفی `= 2`

خانه ی مصرفی

$2 \times 6 \times 7 = 84$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	2	1	1	7	1	2	4	2	4	1	3	4	5	4	5	1	7	5	6	8	6	6	1

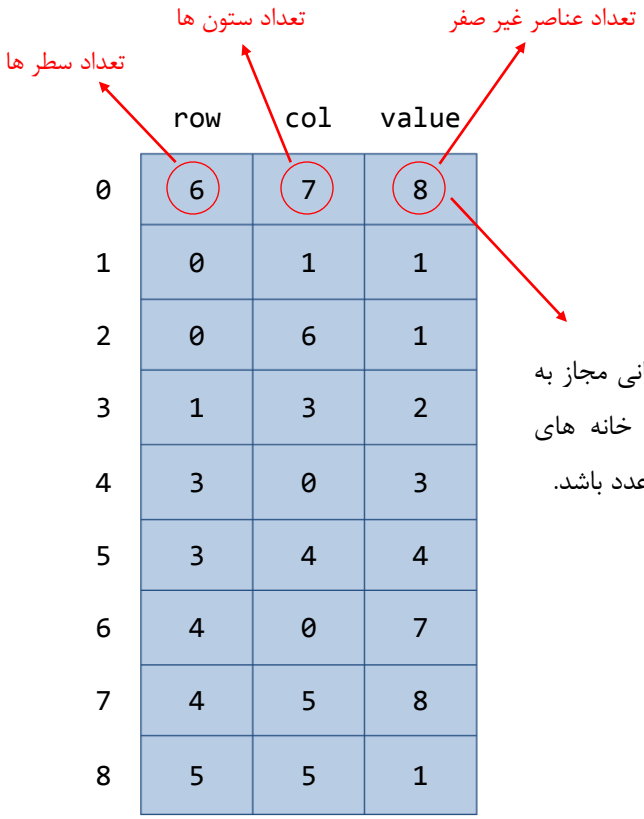
row col value

خانه ی مصرفی

$24 \times 2 = 48$

پیاده سازی ماتریس اسپارس به کمک آرایه ای از رکورد ها

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array}\begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 4 & 0 & 0 \\ 7 & 0 & 0 & 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}_{6\times7} \end{array}$$




```
Struct
{
      int row,col;
      var value;
};

sparse mat[n + 1]
```

تعداد عناصر غیر صفر

توجه: این خانه فقط زمانی مجاز به پر شدن است که نوع خانه های ماتریس اسپارس از نوع عدد باشد.

آیا جمع دو ماتریس اسپارس یک ماتریس اسپارس است؟

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 7 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 6 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{5\times7} \end{array} + \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -2 & 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 \end{bmatrix}_{5\times7} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 5 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 6 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 2 & 0 & 0 \end{bmatrix}_{5\times7}$$

$$O(row \times col)$$

	row	col	value
0	5	7	6
1	0	1	1
2	1	2	7
3	2	0	1
4	2	4	6
5	3	1	1
6	4	0	2

	row	col	value
0	5	7	5
1	0	5	1
2	1	2	-2
3	1	6	1
4	2	0	-1
5	4	4	2

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

$$O(\,(n_A)+(n_B)\,)$$

پیاده سازی ترانهاده در ماتریس اسپارس

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 7 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 6 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{5\times 7} \xrightarrow{\text{ترانهاده}} \begin{bmatrix} 0 & 0 & 1 & 0 & 2 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{7\times 5}$$

$$\theta(row \times col\,)$$

	row	col	value
0	5	7	6
1	0	1	1
2	1	2	7
3	2	0	1
4	2	4	6
5	3	1	1
6	4	0	2

ترانهاده

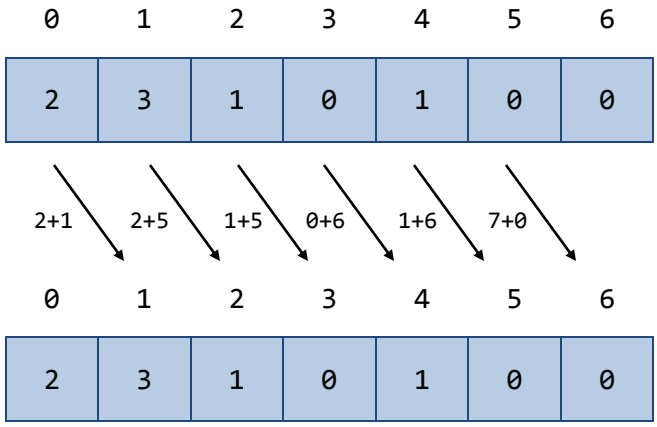
	row	col	value
0	5	7	6
1	0	2	1
2	0	4	2
3	1	0	1
4	1	3	1
5	2	1	7
6	4	2	6

$$\theta(n \times col\,)$$

	row	col	value
0	5	7	6
1	0	1	1
2	1	2	7
3	2	0	1
4	2	4	6
5	3	1	1
6	4	0	2

ترانهاده

	row	col	value
0	5	7	6
1	0	2	1
2	0	4	2
3	1	0	1
4	1	3	1
5	2	1	7
6	4	2	6



$$\theta(n + col\ )$$