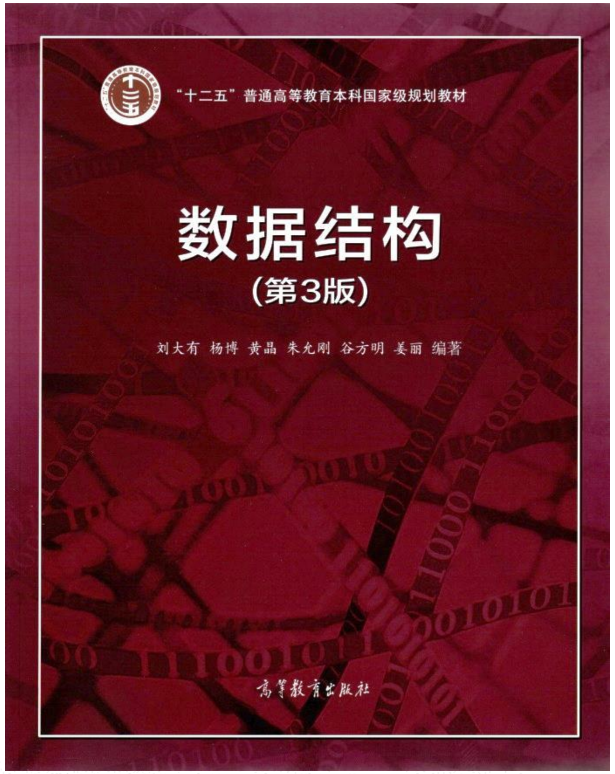


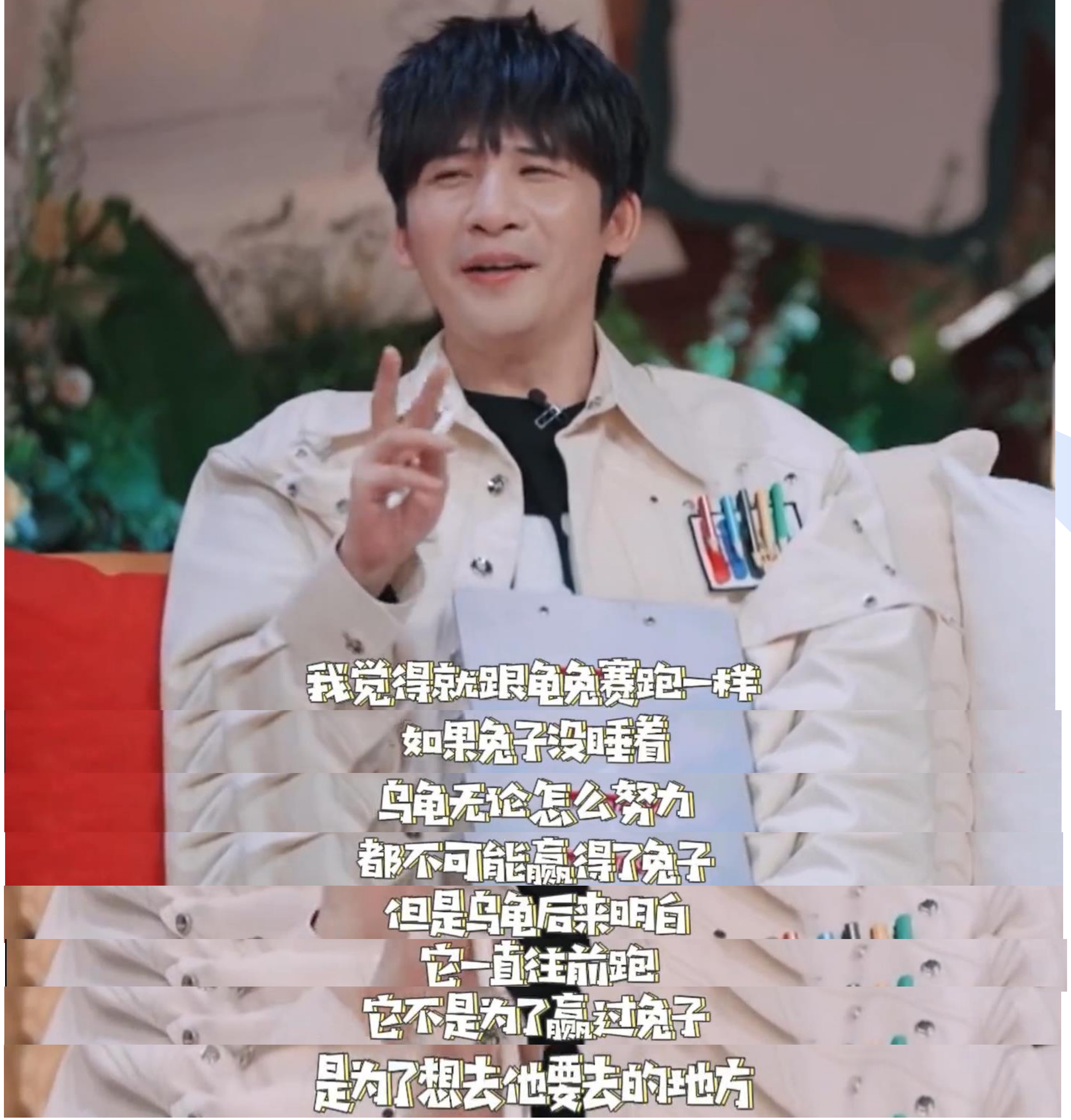
数组与矩阵

- 数组存储与寻址
- 特殊矩阵的压缩存储
- 三元组表
- 十字链表
- 动态规划初探
- 区间问题处理技巧
- 子集生成

数据之法
结构之美
算法之道

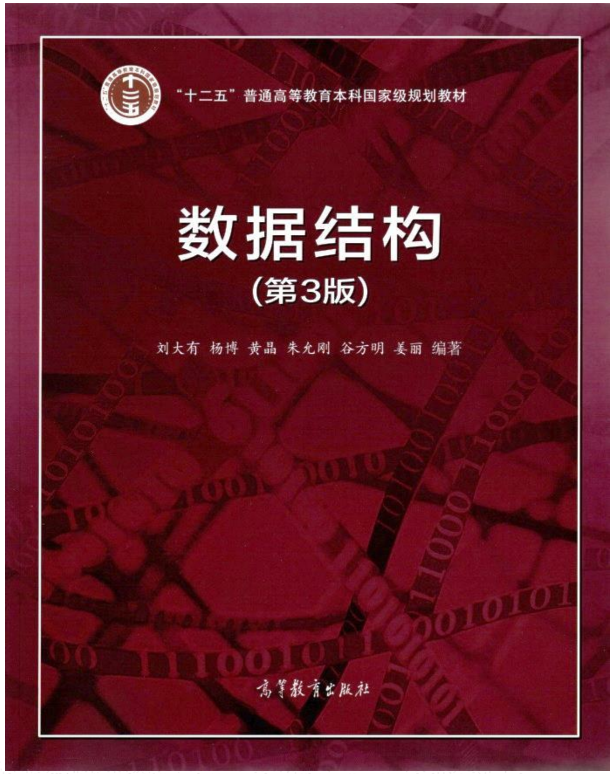
Last updated on 2023.9

zhuyungang@jlu.edu.cn


我觉得就跟龟兔赛跑一样
如果兔子没睡着
乌龟无论怎么努力
都不可能赢得了兔子
但是乌龟后来明白
它一直往前跑
它不是为了赢过兔子
是为了想去他要去的地方

慕课自学内容（必看，计入期末成绩）

自学内容	视频时长
数组的存储和寻址	1分40秒
一维数组类	20分58秒
矩阵类	22分35秒



数组与矩阵

- **数组存储与寻址**
- 特殊矩阵的压缩存储
- 三元组表
- 十字链表
- 动态规划初探
- 区间问题处理技巧
- 子集生成

数据之法
结构之美
算法之道

zhuyungang@jlu.edu.cn

n 维数组（按行优先存储）

- 各维元素个数 $m_1, m_2, m_3, \ldots, m_n$，每个元素占$C$个存储单元
- 下标为 $i_1, i_2, i_3, \ldots, i_n$ 的数组元素$a[i_1][i_2]\ldots[i_n]$ 的存储地址：

$$LOC(i_1, i_2, \ldots, i_n) = LOC(0,\ldots,0) + (i_1*m_2*m_3*\ldots*m_n + i_2*m_3*m_4*\ldots*m_n + i_3*m_4*\ldots*m_n + \ldots\ldots + i_{n-1}*m_n + i_n)*C$$

$$LOC(0,\cdots,0) + \sum_{k=1}^{n}\left(i_k * \prod_{p=k+1}^{n} m_p\right) * C$$

例子

- 已知数组A[3][5][11][3]

- 给出按行优先存储下的A[*i*][*j*][*k*][*l*]地址计算公式

$$Loc(A)+(i*5*11*3+j*11*3+k*3+l)*C$$

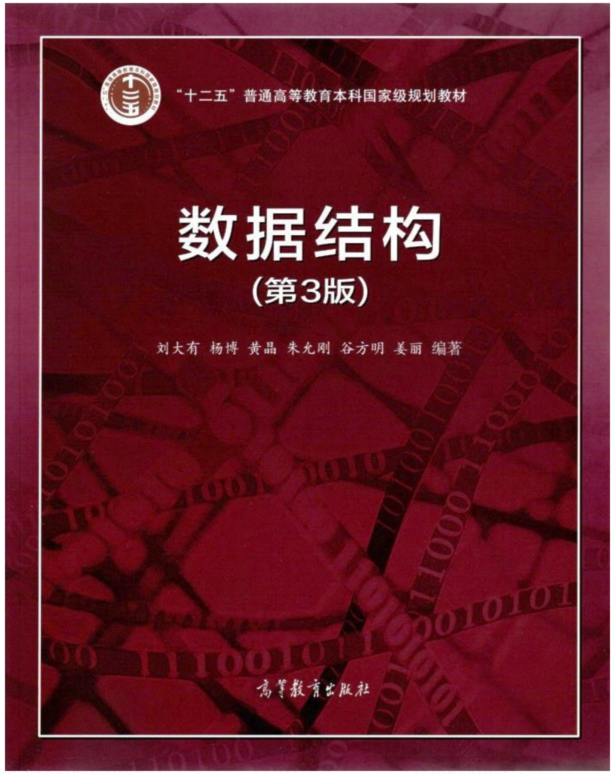
$$=Loc(A)+(165i+33j+3k+l)*C$$

课下练习

四维数组A[3][5][11][3]采用按行优先存储方式，每个元素占4个存储单元，若A[0][0][0][0]的存储地址是1000，则A[1][2][6][1]的存储地址是______.

$$\mathrm{Loc}(A[1][2][6][1])$$
$$=1000+4*(165i+33j+3k+l)$$
$$=1000+4*(165*1+33*2+3*6+1)$$
$$=1000+4*250$$
$$=2000$$

数据之法
结构之美
算法之道



数组与矩阵

- 数组存储与寻址
- **特殊矩阵的压缩存储**
- 三元组表
- 十字链表
- 动态规划初探
- 区间问题处理技巧
- 子集生成

Last updated on 2023.9

zhuyungang@jlu.edu.cn

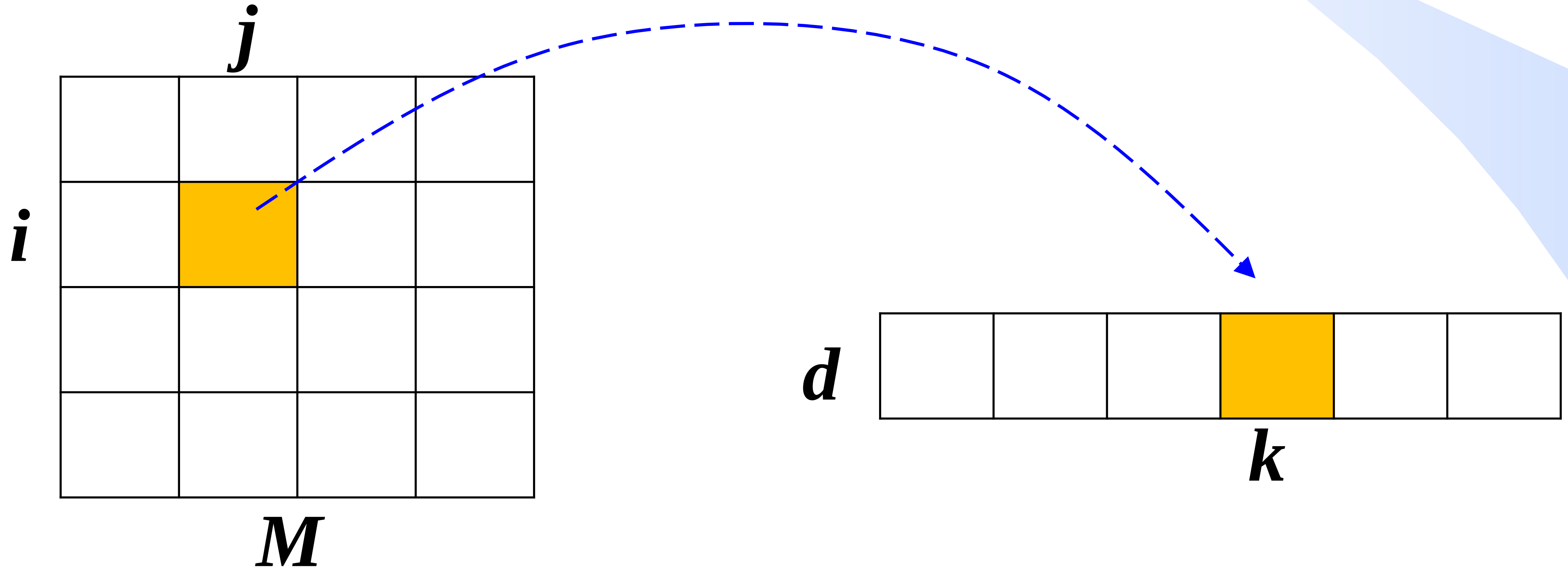
对角矩阵的压缩存储

- 若n×n的方阵M是对角矩阵，则对所有的$i \neq j$ $(1 \leq i, j \leq n)$ 都有M(i, j)=0，即非对角线上的元素均为0，非0元素只在对角线上。
- 对于一个n×n的对角矩阵，至多只有n个非0元素，因此只需存储n个对角元素。
- 可采用一维数组d[ ]来压缩存储对角矩阵。

$$\begin{pmatrix} A_1 & & & O \\ & A_2 & & \\ & & \ddots & \\ O & & & A_l \end{pmatrix}$$

特殊矩阵的压缩存储需考虑2个问题

- 需要多大存储空间：数组*d*[ ]需要多少元素
- 地址映射：矩阵的任意元素*M*(*i*, *j*)在*d*[ ]中的位置（下标），即把矩阵元素的下标映射到数组*d*的下标



对角矩阵的压缩存储

➢ 用一维数组$d[n]$存储对角矩阵，其中$d[i-1]$存储$M(i, i)$的值。

$$M = \begin{bmatrix} a_{11} & & & \\ & a_{22} & & \\ & & a_{33} & \\ & & & a_{44} \end{bmatrix}$$

$$d = [d_0 \quad d_1 \quad d_2 \quad d_3]$$

$$M(i, j) = \begin{cases} d[i-1], & i = j \\ 0, & i \neq j \end{cases}$$

三角矩阵的压缩存储

- 三角矩阵分为上三角矩阵和下三角矩阵。
- 方阵M是上三角矩阵，当且仅当$i>j$时有$M(i, j)=0$.
- 方阵M是下三角矩阵，当且仅当$i<j$时有$M(i, j)=0$.

$$\mathbf{U}=\begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\ & u_{2,2} & u_{2,3} & \cdots & u_{2,n} \\ \vdots & & \ddots & \ddots & \vdots \\ & (0) & & \ddots & u_{n-1,n} \\ 0 & & \cdots & & u_{n,n} \end{bmatrix} \qquad \mathbf{L}=\begin{bmatrix} l_{1,1} & & \cdots & & 0 \\ l_{2,1} & l_{2,2} & & (0) & \\ l_{3,1} & l_{3,2} & \ddots & & \vdots \\ \vdots & \vdots & \ddots & \ddots & \\ l_{n,1} & l_{n,2} & \cdots & l_{n,n-1} & l_{n,n} \end{bmatrix}$$

三角矩阵的压缩存储

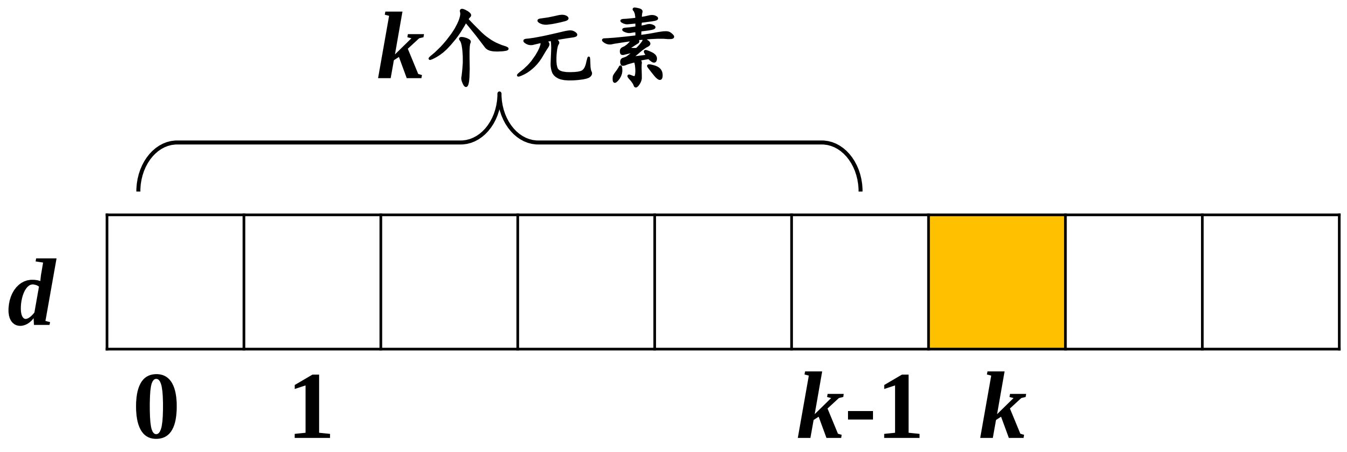
以下三角矩阵M为例，讨论其压缩存储方法：

- ➢将下三角矩阵压缩存放在一维数组*d*
- ✓*d*需要多少个元素？ ***n*(*n*+1)/2**
- ✓M(*i*, *j*)在数组d的什么位置？

$$\mathbf{L} = \begin{bmatrix} l_{1,1} & & \cdots & & 0 \\ l_{2,1} & l_{2,2} & & (0) & \\ l_{3,1} & l_{3,2} & \ddots & & \vdots \\ \vdots & \vdots & \ddots & \ddots & \\ l_{n,1} & l_{n,2} & \cdots & l_{n,n-1} & l_{n,n} \end{bmatrix}$$

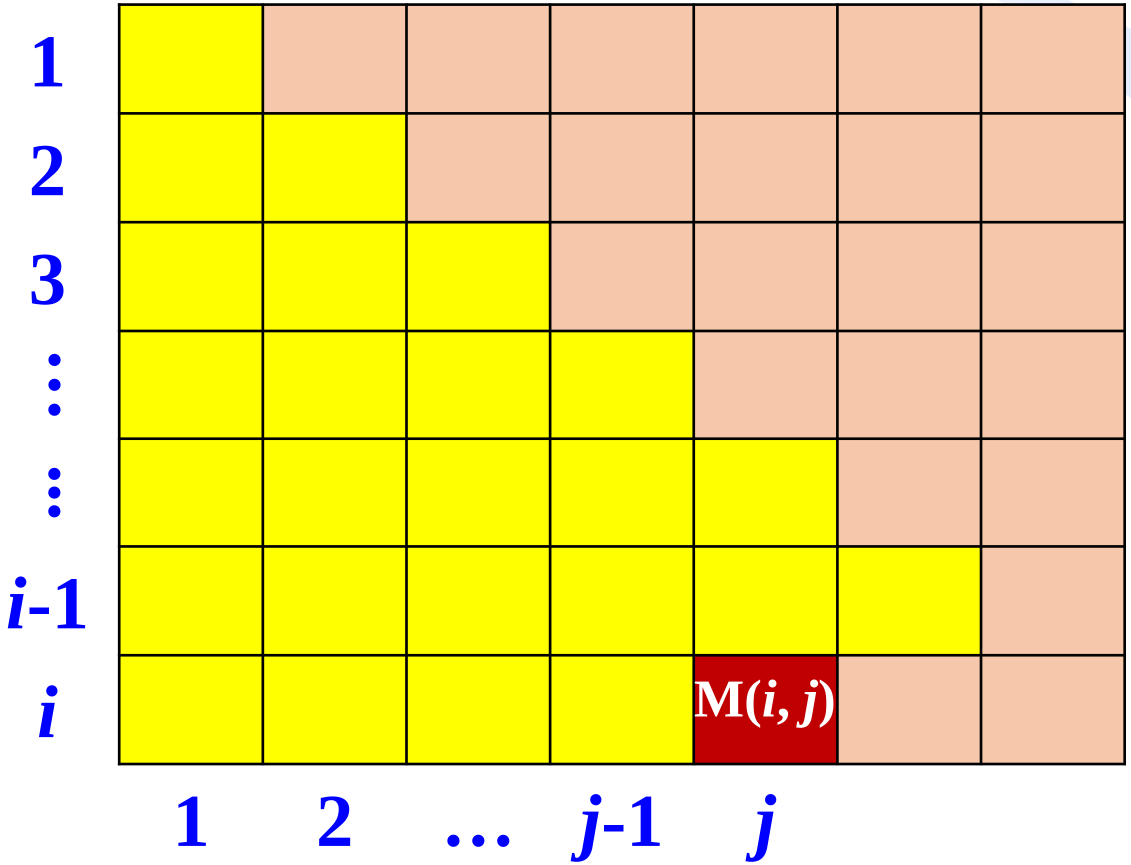
M(*i*, *j*)在数组d的什么位置

➢ 若在矩阵中元素M(*i*, *j*)前面有*k*个元素，则M(*i*, *j*)存储在d[*k*]位置



下三角矩阵的压缩存储

- 设元素M(i, j)前面有k个元素，可以计算出
- $k = 1+2+\ldots+(i-1)+(j-1) = i(i-1)/2+(j-1)$



下三角矩阵的压缩存储

- 设元素M(i, j)前面有k个元素，可以计算出
- k =1+2+…+ (i –1) + (j–1)= i(i–1)/2 + (j–1)
- M(i, j)= d[k]=d[i(i–1)/2 + (j–1)]

$$M(i,j)=\begin{cases} d[i(i-1)/2+(j-1)], & i\geq j \\ 0, & i<j \end{cases}$$

对称矩阵M的压缩存储

- 方阵$M_{n\times n}$是对称矩阵，当且仅当对于任何$1\le i, j \le n$，均有$M(i, j) = M(j, i)$.
- 因为对称矩阵中$M(i, j)$与$M(j, i)$的信息相同，所以只需存储M的下三角部分的元素信息。

- 将对称矩阵存储到一维数组d
- d需要多少个元素？ $n(n+1)/2$
- $M(i, j)$的寻址方式是什么？

- ➢ $i \geq j$，M(i, j)=d[k]，$k = i(i-1)/2 + (j-1)$

对于对称矩阵中的下三角元素M(i, j) ($i \geq j$)，和下三角矩阵压缩存储的映射公式一样

对于上三角元素M(i, j) ($i<j$)，元素值与下三角矩阵中的元素M(j, i)相同

- ➢ $i < j$, M(i, j)=M(j, i)=d[q]，$q= j(j-1)/2 + (i-1)$

$$M(i,j) = \begin{cases} d[i(i-1)/2 + (j-1)], & i \geq j \\ d[j(j-1)/2 + (i-1)], & i < j \end{cases}$$

课下思考

设有一个12*12的对称矩阵M，将其上三角元素M(*i*, *j*) (1≤ *i*, *j* ≤ 12)按行优先存入C语言的一维数组N中，则元素M(6, 6) 在N中的下标是______．【2018年考研题全国卷】

M第一行12个元素，
第二行11个元素，
第三行10个元素，
第四行9个元素，
第五行8个元素，
M(6, 6)是第6行第1个元素，
即前面有50个元素，故M(6, 6) =N[50]

$$\mathrm{U} = \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\ & u_{2,2} & u_{2,3} & \cdots & u_{2,n} \\ \vdots & & \ddots & \ddots & \vdots \\ & & & \ddots & u_{n-1,n} \\ & & \cdots & & u_{n,n} \end{bmatrix}$$

课下思考

设有一个10*10的对称矩阵M，将其上三角元素M(*i*, *j*) (1≤ *i*, *j* ≤ 10)按列优先存入C语言的一维数组N中，则元素$M_{7,2}$在N中的下标是______．【2020年考研题全国卷】

M(7, 2) =N[22]

三对角矩阵M的压缩存储

方阵$M_{n\times n}$中任意元素M(i, j)，当 | i - j | >1时，有M(i, j) =0， 则M称为三对角矩阵。

$$M=\begin{bmatrix} a_{11} & a_{12} & & \\ a_{21} & a_{22} & a_{23} & \\ & a_{32} & a_{33} & a_{34} \\ & & a_{43} & a_{44} \end{bmatrix}$$

三对角矩阵M的压缩存储

$a_{1,1}$	$a_{1,2}$						
$a_{2,1}$	$a_{2,2}$	$a_{2,3}$					
	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$				
		$a_{4,3}$	$a_{4,4}$	$a_{4,5}$			
			…	…			
				$a_{i,i-1}$	$a_{i,i}$	$a_{i,i+1}$	
					…	…	
					$a_{n-1,n-2}$	$a_{n-1,n-1}$	$a_{n-1,n}$
						$a_{n,n-1}$	$a_{n,n}$

$$M(i,j)=\begin{cases} d[2i+j-3], & |i-j|\le 1 \\ 0, & |i-j|>1 \end{cases}$$

M(i, j)前面有k个元素

$k = 2+(i-2)*3+(j-i)+1$

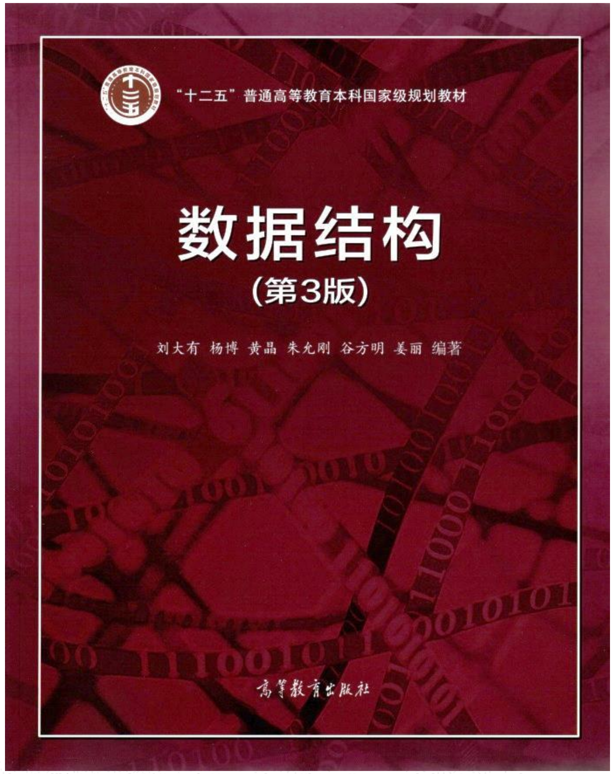
$=2i+j-3$

$a_{1,1}$	$a_{1,2}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	…	$a_{n-1,n}$	$a_{n,n-1}$	$a_{n,n}$

课下思考

有一个100阶的三对角矩阵M，其元素M(i, j) (1≤ i, j ≤ 100)按行优先依次压缩存入下标从0开始的一维数组N中，则元素M(30, 30) 在N中的下标是______.【2016年考研题全国卷】

$$2i + j - 3 = 2*30 + 30 - 3 = 87$$



数组与矩阵

- 数组存储与寻址
- 特殊矩阵的压缩存储
- **三元组表**
- 十字链表
- 动态规划初探
- 区间问题处理技巧
- 子集生成

数据之法
结构之美
算法之道

Last updated on 2023.9

zhuyungang@jlu.edu.cn

稀疏矩阵的压缩存储

定义：设矩阵 $A_{m \times n}$ 中非零元素的个数远远小于零元素的个数，则称 **A** 为稀疏矩阵。

✓ 稀疏矩阵特点：零元素多，且其分布一般没有规律。

✓ 压缩存储：仅存储非零元素，节省空间。

➢对于矩阵 $A_{m\times n}$ 的每个元素a_{ij}，知道其行号i和列号j，就可以确定该元素在矩阵中的位置。因此，如果用**一个结点**来存储**一个非零元素**的话，那么该结点可以设计如下：

i	j	a_{ij}

三元组结点

➢矩阵的每个非零元素可由一个**三元组**结点唯一确定。

- 如何在三元组结点的基础上实现对整个稀疏矩阵的存储？
- **顺序存储**方式实现：三元组表
- **链接存储**方式实现：十字链表

i	j	a_{ij}

三元组表

将表示稀疏矩阵的**非零元素**的三元组结点**按行优先**的顺序排列，得到一个线性表，将此线性表用顺序存储结构存储起来，称之为三元组表。

稀疏矩阵

$$A=\begin{pmatrix} 50 & 0 & 0 & 0 \\ 10 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 \\ -30 & 0 & -60 & 5 \end{pmatrix}$$

三元组表

B[0]	1	1	50
B[1]	2	1	10
B[2]	2	3	20
B[3]	4	1	-30
B[4]	4	3	-60
B[5]	4	4	5

```
struct Triple{
   int row;
   int col;
   int value;
};
Triple B[100];
```

三元组表

若采用三元组表存储稀疏矩阵M，除三元组及M包含的非零元素个数外，下列数据中还需要保存的是________.【2023年考研题全国卷】

I. M 的行数　　II. M 中包含非零元素的行数

III. M 的列数　IV. M 中包含非零元素的列数

A.仅 I、III　　B.仅I、IV　　C.仅 II、IV　　D. I、II、III、IV

求稀疏矩阵的转置

稀疏矩阵 $A=\begin{pmatrix} 50 & 0 & 0 & 0 \\ 10 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 \\ -30 & 0 & -60 & 5 \end{pmatrix}$

转置矩阵 $B=\begin{pmatrix} 50 & 10 & 0 & -30 \\ 0 & 0 & 0 & 0 \\ 0 & 20 & 0 & -60 \\ 0 & 0 & 0 & 5 \end{pmatrix}$

转置操作

转置前

$$A=\begin{pmatrix} 50 & 0 & 0 & 0 \\ 10 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 \\ -30 & 0 & -60 & 5 \end{pmatrix}$$

转置后

$$B=\begin{pmatrix} 50 & 10 & 0 & -30 \\ 0 & 0 & 0 & 0 \\ 0 & 20 & 0 & -60 \\ 0 & 0 & 0 & 5 \end{pmatrix}$$

a[0]	1	1	50
a[1]	2	1	10
a[2]	2	3	20
a[3]	4	1	-30
a[4]	4	3	-60
a[5]	4	4	5

b[0]	1	1	50
b[1]	1	2	10
b[2]	1	4	-30
b[3]	3	2	20
b[4]	3	4	-60
b[5]	4	4	5

a 转置前

1	1	50
2	1	10
2	3	20
4	1	-30
4	3	-60
4	4	5

b 转置后

1	1	50
1	2	10
1	4	-30
3	2	20
3	4	-60
4	4	5

```
void Transpose(Triple a[], int m, int n, int t, Triple b[])
{   //将三元组表a表示的m行n列矩阵转置，保存在三元组表b中，a中非0元素个数为t
    int j = 0;                              //j标识当前填三元组b的第几位
    if (t == 0) return;                     // a空
    for (int k = 1; k <= n; k++)            //填转置后的矩阵的第k行的元素
        for (int i = 0; i < t; i++)         //扫描矩阵a，看哪些元素列号是k
            if (a[i].col == k) {            //看a的哪些元素列号是k
                b[j].row = k;               //转置后行号应为k
                b[j].col = a[i].row;        //列号应为其在a中的行号
                b[j].value = a[i].value;
                j++;                        //赋值三元组表b中的下一个结点
            }
}
```

```
struct Triple {
    int row, col;
    int value;
};
```

时间复杂度

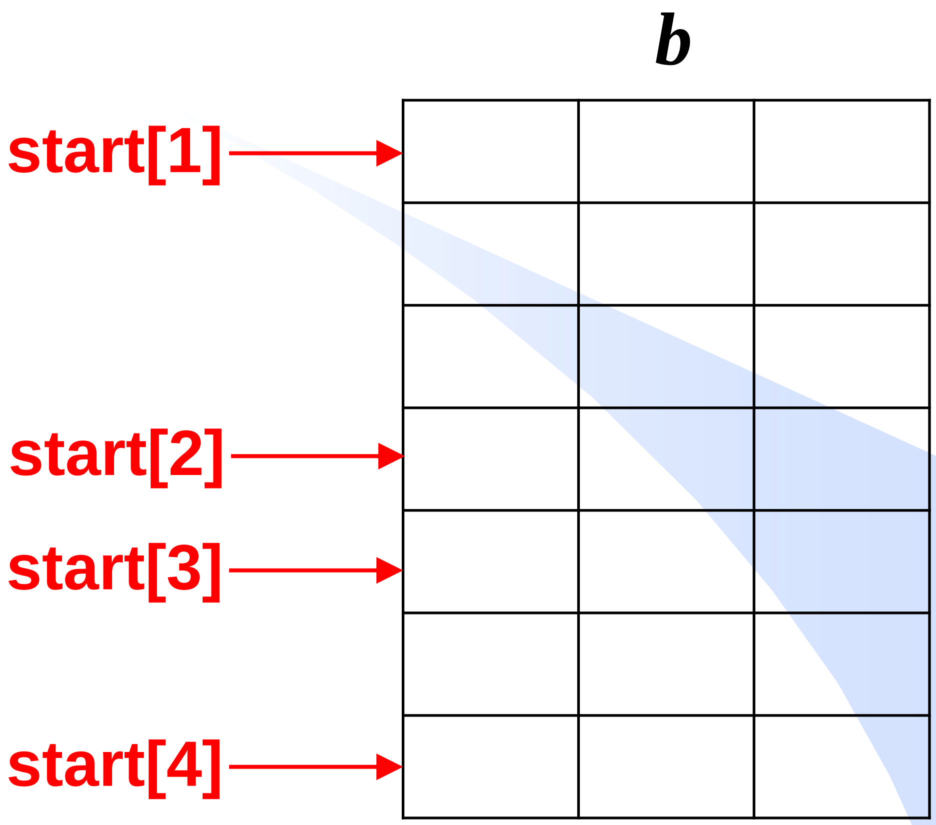
O(*nt*)

快速转置算法

是否存在$O(n+t)$的算法？

a

1	1	50
2	1	10
2	3	20
3	2	5
4	1	-30
4	3	6
4	4	5




```
start[1]=0
start[2]=start[1]+转置后第1行的元素个数
```

rowsize[]:长度为n，存放转置后各行非0元素的个数，即转置前各列非0元素的个数。

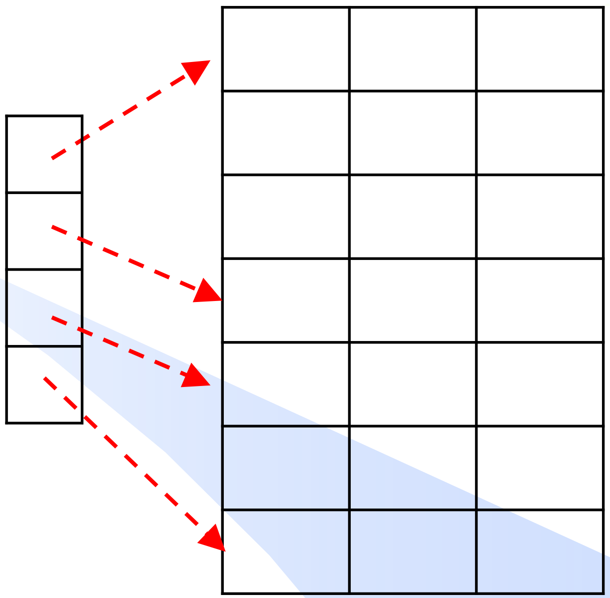
```
for (i = 1; i <= n; i++)
    rowsize[i] = 0;

for (i = 0; i < t; i++) {
    k = a[i].col;
    rowsize[k]++;
}
```

1	1	50
2	1	10
2	3	20
3	2	5
4	1	-30
4	3	6
4	4	5

1	**2**	**3**	**4**

start[]:长度为n，存放转置后的矩阵每行在三元组表b中的起始位置。

```
start[1] = 0;
for (i = 2; i <= n; i++)
    start[i] = start[i-1]+rowsize[i-1];
for (i = 0; i < t; i++) {
    k = a[i].col;
    j = start[k];
    b[j].row = a[i].col;
    b[j].col = a[i].row;
    b[j].value = a[i].value;
    start[k]++;
}
```

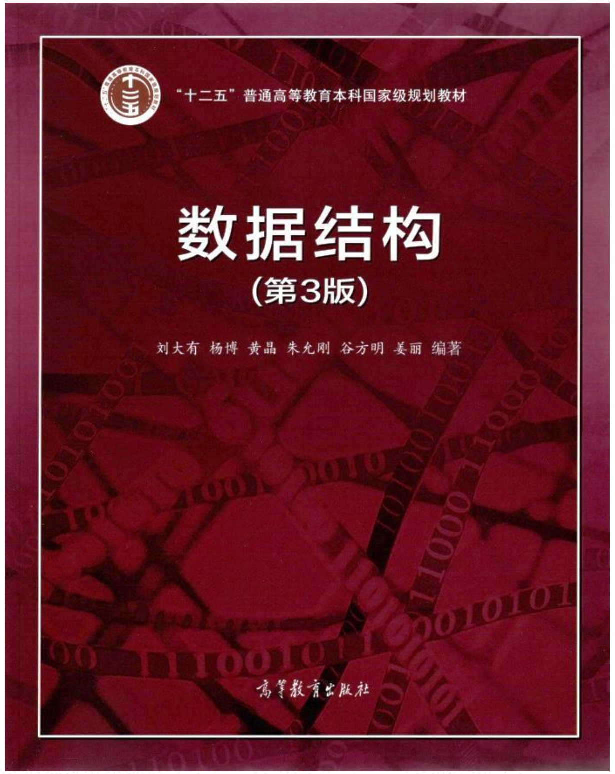


1	1	50
2	1	10
2	3	20
3	2	5
4	1	-30
4	3	6
4	4	5

时间复杂度O($n+t$)

稀疏矩阵的三元组表存储方式分析

- 节省空间，但对于非零元的位置或个数经常发生变化的矩阵运算就显得不太适合。
- 如：矩阵某些位置频繁的加上或减去一个数，使有的元素由0变成非0，由非0变成0。导致三元组表频繁进行插入删除操作，需要频繁元素移动。



数组与矩阵

- 数组存储与寻址
- 特殊矩阵的压缩存储
- 三元组表
- **十字链表**
- 动态规划初探
- 区间问题处理技巧
- 子集生成

数据之法
结构之美
算法之道

zhuyungang@jlu.edu.cn

十字链表

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & & & & \\ 2 & & \multicolumn{2}{c}{\begin{pmatrix} 0 & 0 & 6 & 0 \\ 4 & 0 & 0 & 0 \\ 0 & 9 & 0 & 7 \\ 0 & 0 & 0 & 8 \end{pmatrix}} & \\ 3 & & & & \\ 4 & & & & \end{array}$$

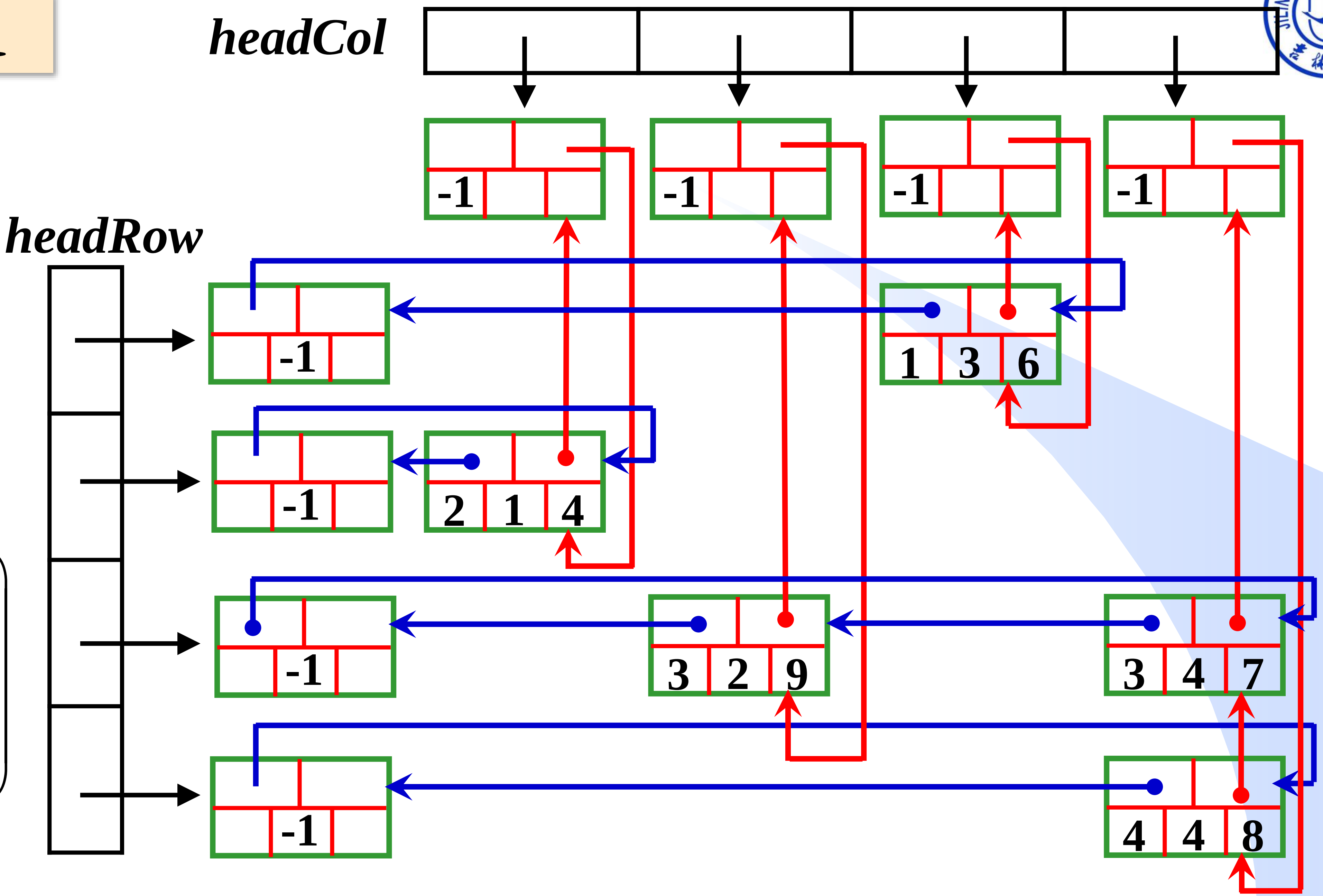
left		up
row	col	data

```
struct ListNode{
    int data;          //数据
    int row;           //该结点所在行
    int column;        //该结点所在列
    ListNode *left;    //指向左侧相邻非零元素的指针
    ListNode *up;      //指向上方相邻非零元素的指针
};
```

十字链表

left		up
row	col	data

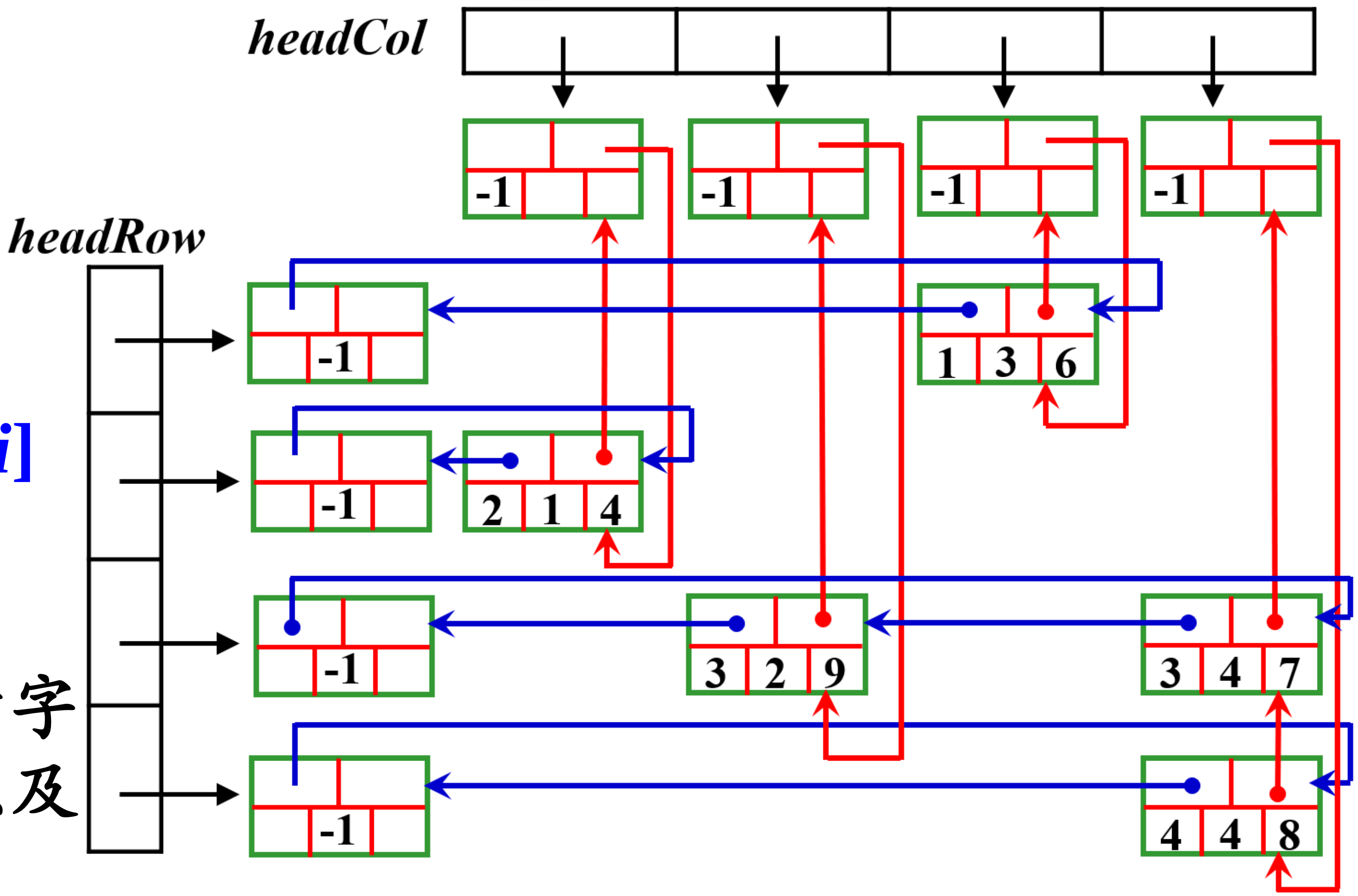
$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 0 & 6 & 0 \\ 4 & 0 & 0 & 0 \\ 0 & 9 & 0 & 7 \\ 0 & 0 & 0 & 8 \end{pmatrix} \end{array}$$

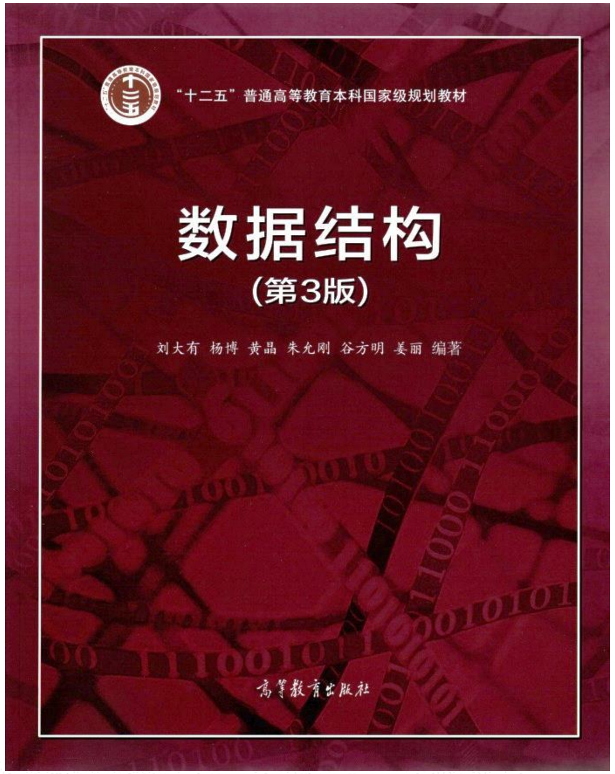


十字链表

矩阵的每一行、列都设置为由一个哨位结点引导的循环链表

- ***headRow[i]*是第*i*行链表的头指针，指向第*i*行链表的哨位结点**
- ***headCol[j]*是第*j*列链表的头指针，指向第*j*列链表的哨位结点**
- **每行哨位结点的col值为-1：**
  ***headRow[i]*->col = -1**
- **每列哨位结点的row值为-1：**
  ***headCol[j]*->row = -1**
- **若第*i*行无非零元素，则**
  headRow[i]*->left== *headRow[i]
- **若第*j*列无非零元素，则**
  headCol[j]*->up== *headCol[j]
- **对矩阵的运算实质上就是在十字链表中插入结点、删除结点以及改变某个结点的数据域的值。**





数组与矩阵

- 数组存储与寻址
- 特殊矩阵的压缩存储
- 三元组表
- 十字链表
- **动态规划初探**
- 区间问题处理技巧
- 子集生成

数据之法
结构之美
算法之道

Last updated on 2023.9

zhuyungang@jlu.edu.cn

递归算法求解斐波那契数列

$$F(n)=\begin{cases}0 & n=0\\ 1 & n=1\\ F(n-1)+F(n-2) & n\geq 2\end{cases}$$

```
int F(int n){
    if(n<=1) return n;
    return F(n-1)+F(n-2);
}
```

Fibonacci
（1170年—1250年）
意大利数学家

时间复杂度

```c
int F(int n){
    if(n<=1) return n;
    return F(n-1)+F(n-2);
}
```

$$T(n)=\begin{cases}0 & n\le 1\\ 1 & n=2\\ T(n-1)+T(n-2)+1 & n\ge 2\end{cases}$$

$$T(n-1)=T(n-2)+T(n-3)+1$$

$$\begin{aligned}T(n) &\le 2T(n-1)\\ &\le 2^2T(n-2)\\ &\le 2^3T(n-3)\\ &\le 2^4T(n-4)\\ &\cdots\\ &\le 2^{n-2}T(2)\\ &= 2^{n-2}\end{aligned}$$

$$\boxed{T(n)=O(2^n)}$$

时间复杂度

```
int F(int n){
    if(n<=1) return n;
    return F(n-1)+F(n-2);
}
```

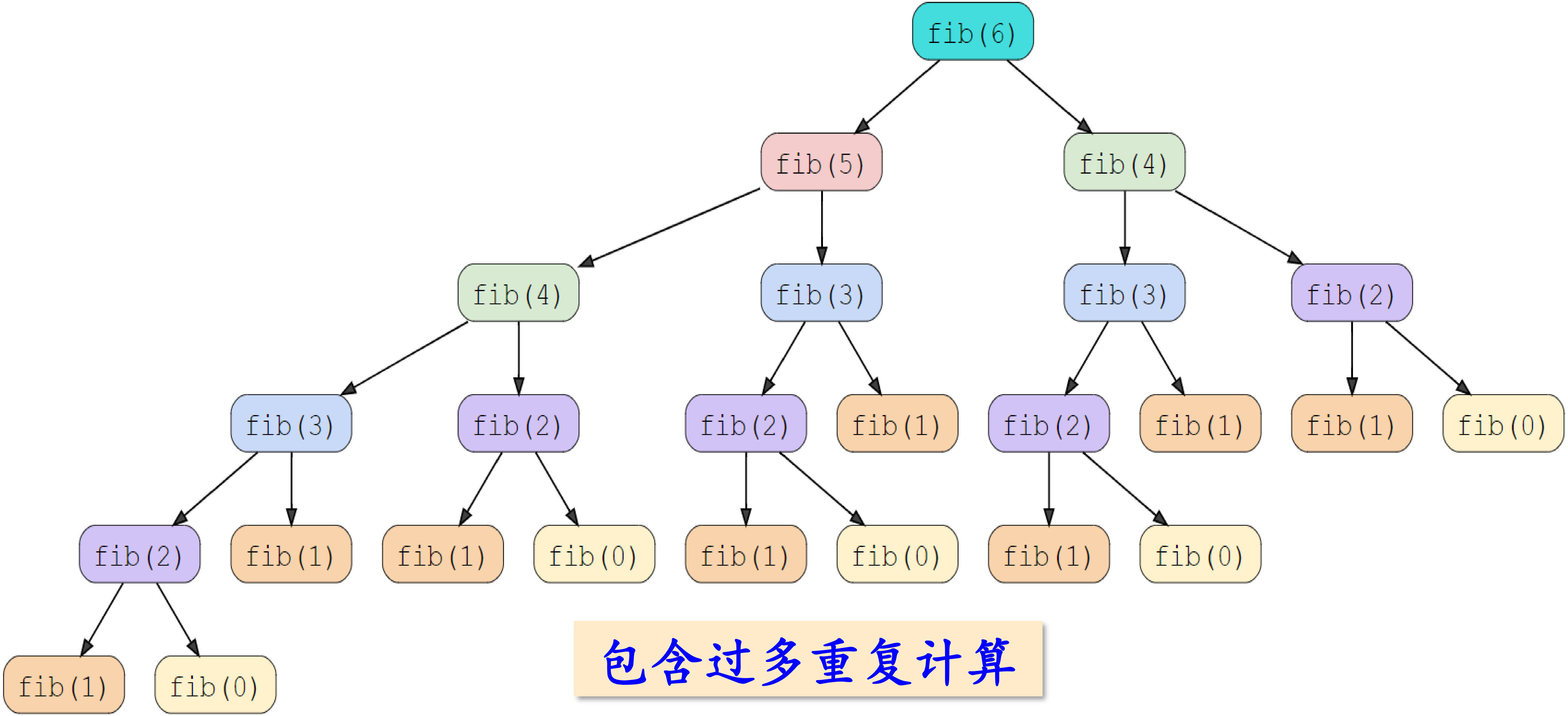
$$T(n) = \begin{cases} 0 & n \le 1 \\ 1 & n = 2 \\ T(n-1) + T(n-2) + 1 & n \ge 2 \end{cases}$$

$T(n-1)=T(n-2)+T(n-3)+1$

$$\begin{aligned} T(n) &\ge 2T(n-2) \\ &\ge 2^2T(n-4) \\ &\ge 2^3T(n-6) \\ &\ge 2^4T(n-8) \\ &\cdots \\ &\ge 2^{(n-2)/2}T(2) \\ &= 2^{(n-2)/2} \end{aligned}$$

$$T(n)= \Omega(2^{n/2})$$

计算效率低的原因



包含过多重复计算

动态规划 (*Dynamic Programming, DP*)

- 将一个复杂的问题分解成若干个子问题，通过综合子问题的解来得到原问题的解。
- 自底向上先求解最小的子问题，并把结果存储在表格中，在求解大的子问题时直接从表格中查询小的子问题的解，以避免重复计算，从而提高效率。
- 往往可以通过“递推”来实现。

动态规划（*Dynamic Programming, DP*）的提出者

Richard Bellman
（1920-1984）
南加州大学教授
美国科学院院士
美国工程院院士

递推

```
int F[N];
int Fib(int n){
    F[0]=0; F[1]=1;
    for(int i=2; i<=n; i++)
        F[i]=F[i-1]+F[i-2];
    return F[n];
}
```

$$F(n)=\begin{cases}0 & n=0\\ 1 & n=1\\ F(n-1)+F(n-2) & n\geq 2\end{cases}$$

从前往后

当算到 F[*i*] 时，F[*i*-1] 和 F[*i*-2] 已经算完了

时间复杂度 O(*n*)

空间复杂度 O(*n*)

	0	1	2	3	4	5	6	7	8	9	10
F	0	1	1	2	3	5	8	13	21	34	55

空间优化

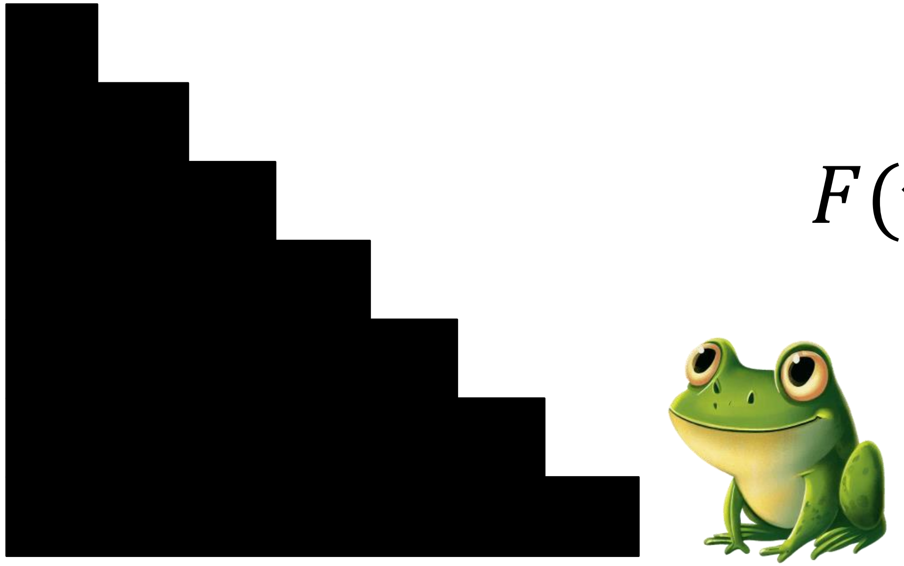
```
int Fib(int n){
    int prev1=0,prev2=1;
    for(int i=2; i<=n; i++){
        cur = prev1+prev2;
        prev1 = prev2;
        prev2 = cur;
    }
    return cur;
}
```

空间复杂度
O(1)

	0	1	2	3	4	5	6	7	8	9	10
F	0	1	1	2	3	5	8	13	21	34	55

跳台阶问题

一个台阶总共有n级。如果一只青蛙一次可以跳1级，也可以跳2级。编写算法对于给定的n，计算出青蛙跳到最顶层总共有多少种跳法。【大厂面试题LeetCode70】

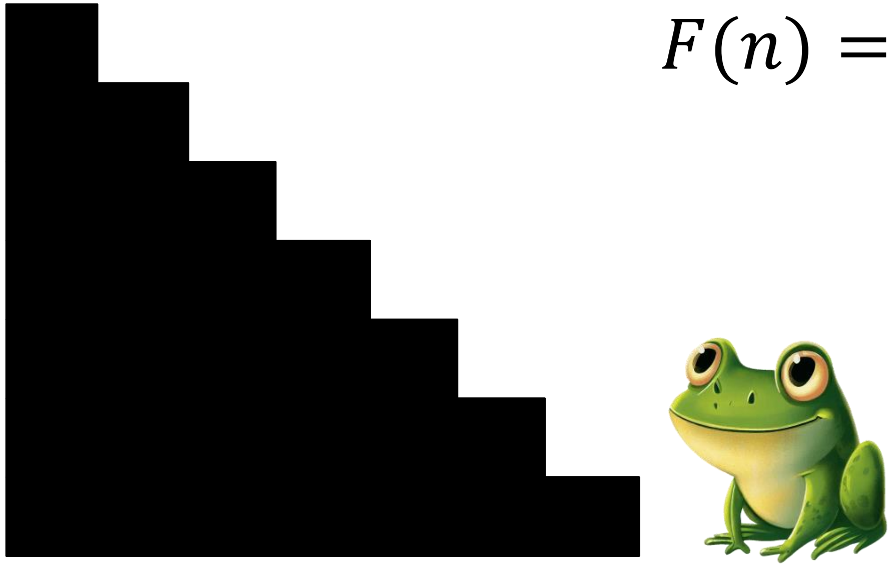


$$F(n)=\begin{cases}1 & n=1\\ 2 & n=2\\ F(n-1)+F(n-2) & n>2\end{cases}$$

课下思考

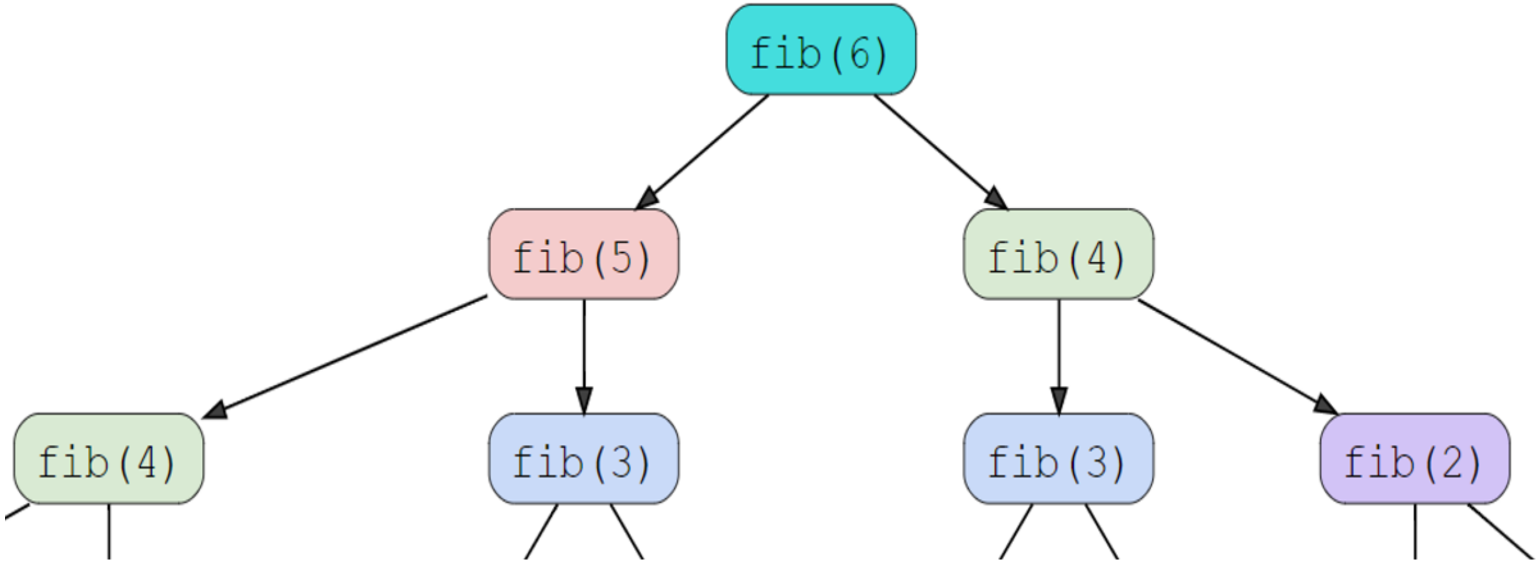
- 一个台阶总共有**n**级。如果一只青蛙一次可以跳1级，也可以跳2级，也可以跳3级。编写算法对于给定的**n**，计算出青蛙跳到最顶层总共有多少种跳法。

$$F(n)=\begin{cases}1 & n=1\\ 2 & n=2\\ 4 & n=3\\ F(n-1)+F(n-2)+F(n-3) & n>3\end{cases}$$



什么问题可以用动态规划方法高效解决

- **最优子结构**：问题的最优解所包含的子问题的解也是最优的。大问题的解包含子问题的解，可以通过子问题的解推导出大问题的解。
- **无后效性**：将每个子问题的求解过程作为一个阶段，当前阶段的求解只与之前阶段有关，而与之后的阶段无关。
- **重叠子问题**：求解过程中每次产生的子问题并不总是新问题，有大量子问题重复。动态规划在处理重叠子问题时，每个子问题只计算一次，从而避免重复计算，这也是动态规划效率高的原因。



使用动态规划方法求解问题的一般过程

- 寻找子问题：把原问题分解成若干子问题，子问题和原问题形式相似，只不过规模小一些，例如从原来的n变成n-1。
- 定义状态：和子问题相关的各个变量的的一组取值称为一个状态。
- 导出递推公式：从一个或多个已知状态的值，推导出未知状态的值，递推公式也称“状态转移方程”。
- 确定边界条件：确定递推的终止条件和边界条件。
- 如何找子问题、如何定义状态、如何推导出递推公式，没有固定的规则，需要具体问题具体分析。多做题，熟能生巧。

二维情况

m和n为正整数

$$F(m,n)=\begin{cases}1 & m=0\text{或}n=0\\ F(m,n-1)+F(m-1,n) & \text{其他}\end{cases}$$

递归算法

```
int F(int m, int n){
    if(m==0 || n==0) return 1;
    return F(m,n-1)+F(m-1,n);
}
```

动态规划算法

$$F(m,n)=\begin{cases}1 & m=0\text{或}n=0\\ F(m,n-1)+F(m-1,n) & \text{其他}\end{cases}$$

				n
				F[*m*-1][*n*]
m			**F[*m*][*n*-1]**	**F[*m*][*n*]**

动态规划算法

$$F(i,j)=\begin{cases}1 & i=0或j=0\\ F(i,j-1)+F(i-1,j) & 其他\end{cases}$$

			j	
			F[*i*-1][*j*]	
i		**F[*i*][*j*-1]**	**F[*i*][*j*]**	

动态规划算法

$$F(i,j)=\begin{cases}1 & i=0\text{或}j=0\\ F(i,j-1)+F(i-1,j) & \text{其他}\end{cases}$$

1	**1**	**1**	**1**
1			
1			
1			
1			

动态规划算法

```
const int N=110;
int CalF(int m, int n) {
    int F[N][N]={{0}};
    for(int i=0; i<=m; i++)
        F[i][0] = 1; //填第0列
    for(int j=0; j<=n; j++)
        F[0][j] = 1; //填第0行
    for(int i=1; i<=m; i++)
        for(int j=1; j<=n; j++)
            F[i][j] = F[i-1][j] + F[i][j-1];
    return F[m][n];
}
```

1	1	1	1
1			
1		F[*i*-1][*j*]	
1	F[*i*][*j*-1]	F[*i*][*j*]	
1			

时间复杂度 $O(m*n)$
空间复杂度 $O(m*n)$

空间优化

		F[i-1][j]	
	F[i][j-1]	F[i][j]	

空间优化

		F[i-1][j]	F[i-1][j+1]
	F[i][j-1]	F[i][j]	F[i][j+1]

空间优化

	b	*d*	*f*
a	*c*	*e*	*g*

$$c = a + b$$
$$e = c + d$$
$$g = e + f$$

空间优化——滚动数组

$i = 1$

1	1	1	1

空间优化——滚动数组

$i = 1$

1	2	1	1

空间优化——滚动数组

$i = 1$

1	2	3	1

空间优化——滚动数组

$i = 1$

1	2	3	4

空间优化——滚动数组

$i = 2$

1	2	3	4

空间优化——滚动数组

$i = 2$

1	3	3	4

空间优化——滚动数组

$i = 2$

1	3	6	4

空间优化——滚动数组

$i = 2$

1	3	6	10

空间优化——滚动数组

$i = 2$

1	3	6	10

```
const int maxn=110;
int CalF(int m, int n) {
    int F[maxn]={0};
    for (int j=0; j<=n; j++)
        F[j] = 1;
    for (int i=1; i<=m; i++)
        for (int j=1; j<=n; j++)
            F[j] = F[j-1] + F[j];
    return F[n];
}
```

时间复杂度 $O(mn)$

空间复杂度 $O(n)$

源码之前
了无秘密

应用举例

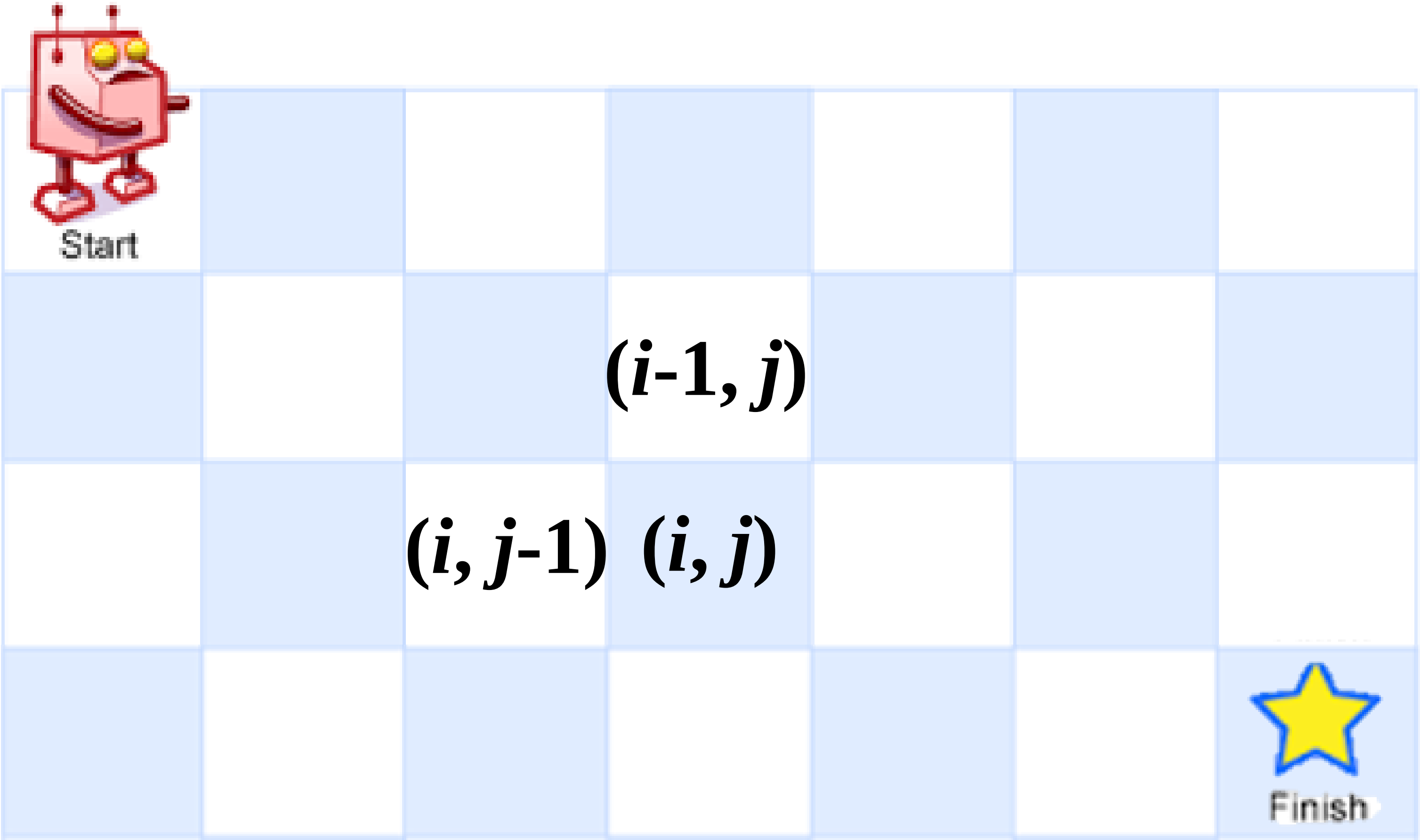
一个机器人位于一个$m \times n$网格的左上角，每次只能向下或者向右移动一步。机器人试图达到网格的右下角，编写程序计算总共有多少条不同的路径。【字节跳动、腾讯、华为、京东、苹果、微软、谷歌面试题LeetCode62】

令$F(i, j)$表示起点到点(i, j)的路径条数

从起点到点(i, j)只有两种途径：

①先到点$(i\text{-}1, j)$，再向下走一步

②先到点$(i, j\text{-}1)$，再向右走一步



$$F(i,j)=\begin{cases}1 & i=0\text{或}j=0\\ F(i,j-1)+F(i-1,j) & \text{其他}\end{cases}$$

课下阅读

时间复杂度O(*m*n*)

```
const int maxn=110;
int uniquePaths(int m, int n){
   int dp[maxn][maxn]={{0}};
   for(int i=0; i<m; i++)
     dp[i][0] = 1;
   for(int j=0; j<n; j++)
     dp[0][j] = 1;
   for(int i=1; i<m; i++)
     for(int j=1; j<n; j++)
       dp[i][j]=dp[i-1][j]+dp[i][j-1];
   return dp[m-1][n-1];
}
```

空间复杂度O(*m*n*)

```
const int maxn=110;
int uniquePaths(int m, int n) {
   int dp[maxn]={0};
   for(int j=0; j<n; j++)
        dp[j] = 1;
   for(int i=1; i<m; i++)
     for(int j=1; j<n; j++)
        dp[j]=dp[j-1]+dp[j];
    return dp[n-1];
}
```

滚动数组优化
空间复杂度O(*n*)

因为无障碍物，所以还有另一解法

因为只能向右或向下走，想要从左上角到右下角，则向右走的步数一定是n-1步，向下走的步数一定是m-1步，合计走n+m-2步。

另一解法

路径的总数 = 从$n+m$-2步中选择m-1步向下移动的方案数

$$C_{n+m-2}^{m-1}$$

1	**2**	**3**	**…**	**…**	**…**			***n*+*m*-2**

如何高效计算C_n^m

➢给定两个整数n和m，满足$n \geq 1$且$0 \leq m \leq n$，编写程序计算C_n^m，输入数据保证最后结果在2^{31}-1以内。【POJ2249】

$$C_n^m = \frac{n!}{m!(n-m)!} = \frac{n\cdot(n-1)\ldots(n-m+3)\cdot(n-m+2)\cdot(n-m+1)}{m\cdot(m-1)\ldots3\cdot2\cdot1}$$

我是天才

```
int C(int n, int m) {
      long long a = 1, b = 1;
      for(int i=n; i>=n-m+1; i--)
            a = a*i;
      for(int i=m; i>=1; i--)
            b = b*i;
      return a / b;
}
```

中间结果溢出

如何高效计算C_n^m

➢给定两个整数n和m，满足$n \geq 1$且$0 \leq m \leq n$，编写程序计算C_n^m，输入数据保证最后结果在2^{31}-1以内。【POJ2249】

$$C_n^m = \frac{n!}{m!(n-m)!} = \frac{n\cdot(n-1)\ldots(n-m+3)\cdot(n-m+2)\cdot(n-m+1)}{m\cdot(m-1)\ldots3\cdot2\cdot1}$$

$$= \frac{n}{m}\cdot\frac{n-1}{m-1}\cdot\frac{n-2}{m-2}\cdots\frac{n-m+1}{1}$$

中间结果可能是小数

如何高效计算C_n^m

$$C_n^m = \frac{n\cdot(n-1)\ldots(n-m+3)\cdot(n-m+2)\cdot(n-m+1)}{m\cdot(m-1)\ldots3\cdot2\cdot1}$$

$$= \boxed{\frac{n-m+1}{1}}\cdot\frac{n-m+2}{2}\cdot\frac{n-m+3}{3}\cdots\frac{n-m+k}{k}\cdots\frac{n-1}{m-1}\cdot\frac{n}{m}$$

$$C_{n-m+1}^{1}$$

中间结果一定是整数

如何高效计算C_n^m

$$C_n^m = \frac{n\cdot(n-1)\ldots(n-m+3)\cdot(n-m+2)\cdot(n-m+1)}{m\cdot(m-1)\ldots3\cdot2\cdot1}$$

$$= \boxed{\frac{n-m+1}{1}\cdot\frac{n-m+2}{2}}\cdot\frac{n-m+3}{3}\cdots\frac{n-m+k}{k}\cdots\frac{n-1}{m-1}\cdot\frac{n}{m}$$

$$C_{n-m+2}^2$$

中间结果一定是整数

如何高效计算C_n^m

$$C_n^m = \frac{n\cdot(n-1)\ldots(n-m+3)\cdot(n-m+2)\cdot(n-m+1)}{m\cdot(m-1)\ldots3\cdot2\cdot1}$$

$$= \boxed{\frac{n-m+1}{1}\cdot\frac{n-m+2}{2}\cdot\frac{n-m+3}{3}}\cdots\frac{n-m+k}{k}\cdots\frac{n-1}{m-1}\cdot\frac{n}{m}$$

$$C_{n-m+3}^{3}$$

中间结果一定是整数

如何高效计算C_n^m

$$C_n^m = \frac{n\cdot(n-1)\ldots(n-m+3)\cdot(n-m+2)\cdot(n-m+1)}{m\cdot(m-1)\ldots3\cdot2\cdot1}$$

$$= \boxed{\frac{n-m+1}{1}\cdot\frac{n-m+2}{2}\cdot\frac{n-m+3}{3}\cdots\frac{n-m+k}{k}}\cdots\frac{n-1}{m-1}\cdot\frac{n}{m}$$

$$C_{n-m+k}^{k}$$

中间结果一定是整数

如何高效计算C_n^m

$$C_n^m=\frac{n-m+1}{1}\cdot\frac{n-m+2}{2}\cdot\frac{n-m+3}{3}\cdots\frac{n-m+k}{k}\cdots\frac{n-1}{m-1}\cdot\frac{n}{m}$$

```
int C(int n, int m) {
    if (m == 0 || n == m) return 1;
    if (m > n/2) m = n - m;
    long long ans = 1; //通过i扫描分子，通过j扫描分母
    for (int i=n-m+1, j=1; i<=n; i++,j++)
        ans = ans*i/j;
    return ans;
}
```

时间复杂度O(m)

空间复杂度O(1)

另一解法

一个机器人位于一个$m×n$网格的左上角，每次只能向下或者向右移动一步。机器人试图达到网格的右下角，编写程序计算总共有多少条不同的路径。【字节跳动、腾讯、华为、京东、苹果、微软、谷歌面试题LeetCode62】

路径的总数 = C_{n+m-2}^{m-1}

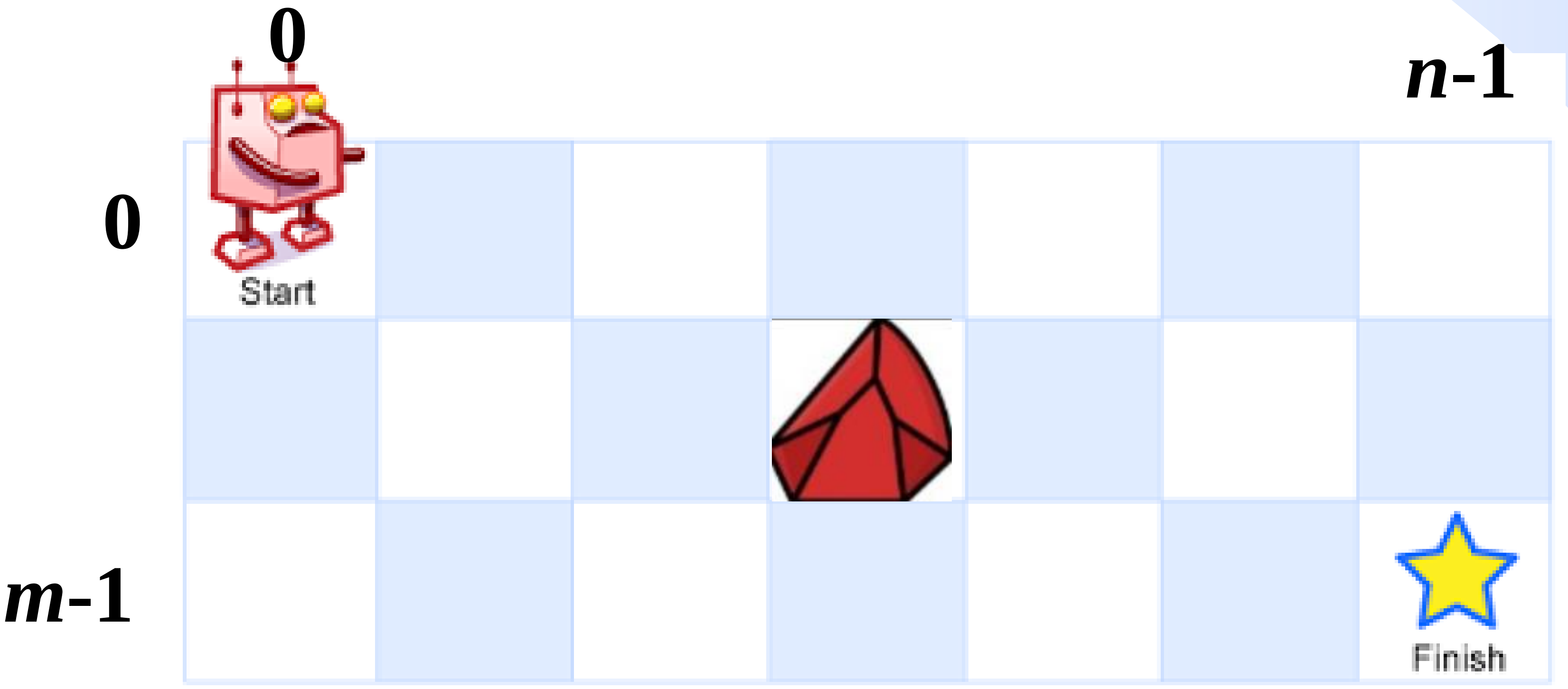
```
int uniquePaths(int m, int n) {
    return C(n+m-2,m-1);
}
```

时间复杂度 $O(m)$

空间复杂度 $O(1)$

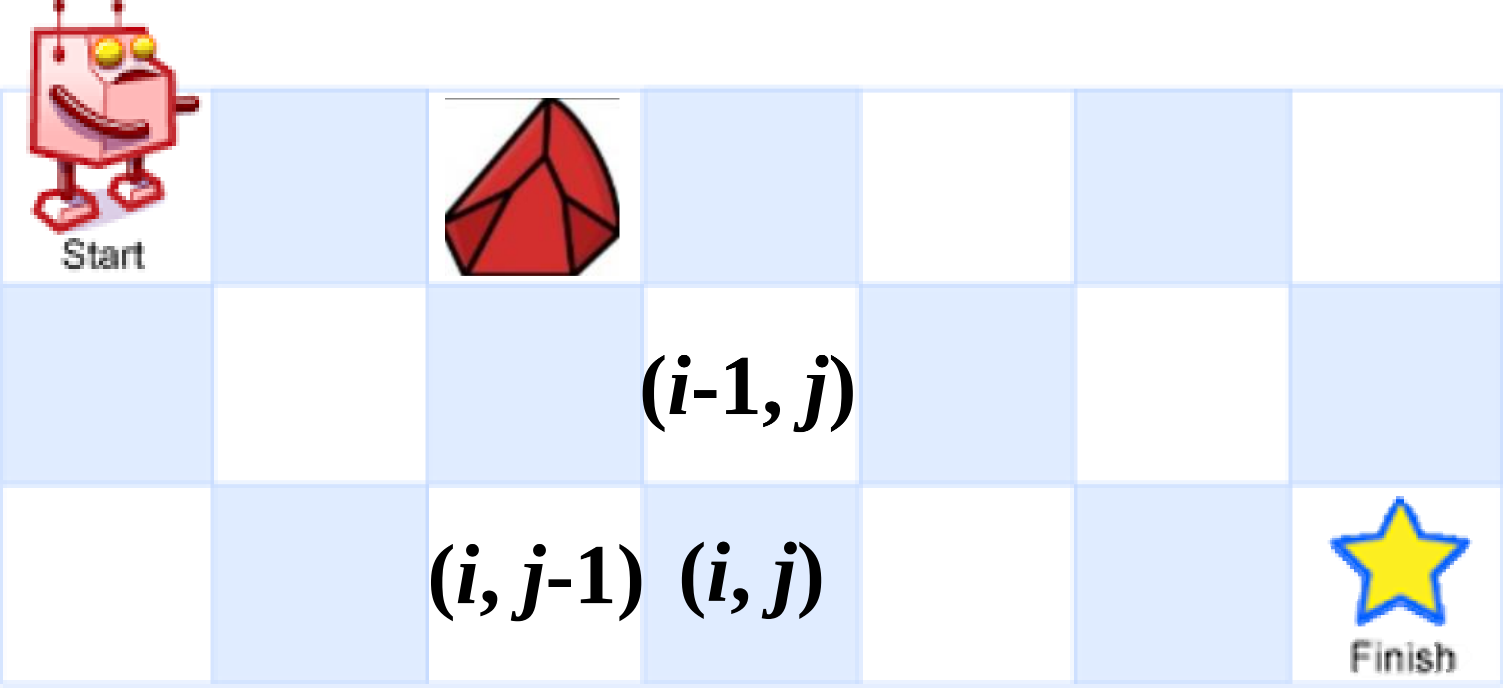
课下思考

一个机器人位于一个$m \times n$网格的左上角，每次只能向下或者向右移动一步。网格中有障碍物，机器人试图达到网格的右下角，编写程序计算总共有多少条不同的路径。【字节跳动、微软、谷歌、小红书面试题LeetCode63】



令$F(i, j)$表示起点到点(i, j)的路径条数

从起点到点(i, j)只有两种途径:
①先到点$(i\text{-}1, j)$,再向下走一步
②先到点$(i, j\text{-}1)$,再向右走一步



$$F(i,j)=\begin{cases} 0 & ,map[i][j]=1 \\ F(i,j-1)+F(i-1,j) & ,i>0\text{且}j>0\text{且}map[i][j]=0 \\ 1 & ,i=0\text{且}j=0\text{且}map[i][j]=0 \\ F(i,j-1) & ,i=0\text{且}j>0\text{且}map[i][j]=0 \\ F(i-1,j) & ,i>0\text{且}j=0\text{且}map[i][j]=0 \end{cases}$$

课下阅读（注意函数参数与LeetCode不同）

```
const int N=110;
int PathsII(int map[N][N],int m,int n){
   int dp[N][N]={{0}};
   for(int i=0;i<m && map[i][0]==0;i++)
      dp[i][0]=1;
   for(int j=0;j<n && map[0][j]==0;j++)
      dp[0][j]=1;
   for(int i=1;i<m;i++)
      for(int j=1;j<n;j++)
        if(map[i][j]==1) dp[i][j]=0;
        else
          dp[i][j]=dp[i][j-1]+dp[i-1][j];
   return dp[m-1][n-1];
}
```

空间复杂度O(*m***n*)

```
const int N=110;
int PathsII(int map[N][N],int m,int n){
   int dp[N]={0};
   for(int j=0;j<n && map[0][j]==0;j++)
      dp[j]=1;
   for(int i=1;i<m;i++)
      for(int j=0;j<n;j++)
         if(map[i][j]==1) dp[j]=0;
         else if(j>0)
            dp[j]=dp[j]+dp[j-1];
   return dp[n-1];
}
```

滚动数组优化
空间复杂度O(*n*)

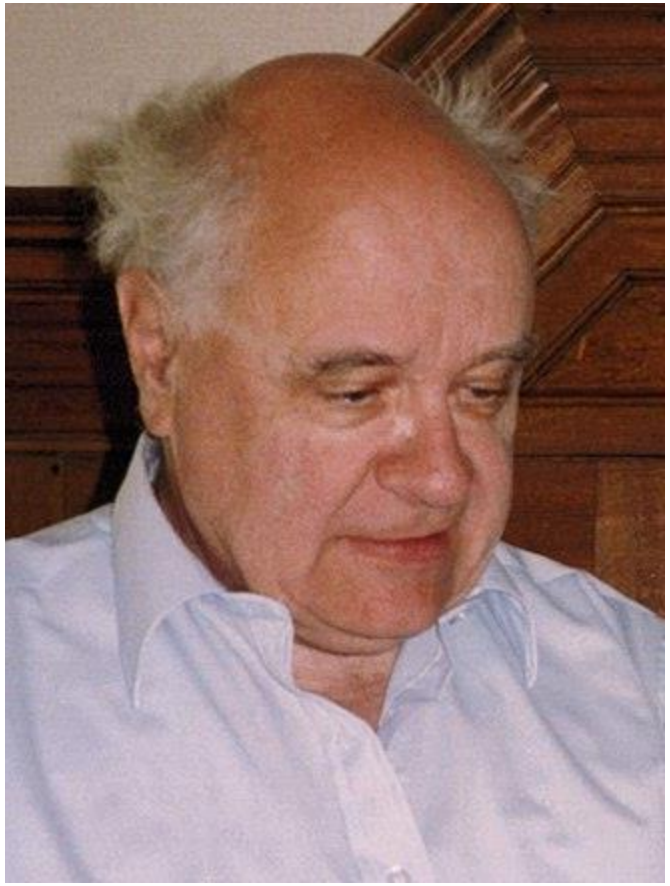
最大子数组和

给定一个含n个整数的数组，数组里可能有正数、负数和零。数组中连续的一个或多个元素组成一个子数组，每个子数组都有一个和。请设计算法求所有子数组之和的最大值。【字节跳动、阿里、微软、谷歌、腾讯、华为、百度、快手、携程、滴滴面试题，浙江大学研究生复试机试LeetCode53、HDU 1003】

-2	5	3	-6	4	-8	6

1	-2	3	10	-4	7	-2	-5

最大子数组和



Ulf Grenander
(1923 - 2016)
斯德哥尔摩大学教授
布朗大学教授
瑞典科学院院士
美国科学院外籍院士

C/C++ 表示正负无穷大的常用方法

方案1：limits.h头文件下的INT_MAX和INT_MIN常量

✓ 正无穷大：INT_MAX = 2^{31}-1 = 0x7fffffff

✓ 负无穷大：INT_MIN = -2^{31} = 0x80000000

✓ 问题：$\infty + d \neq \infty$，$\infty + \infty \neq \infty$

方案2：自定义常量

✓ 正无穷大：0x3f3f3f3f = 1061109567

✓ 负无穷大：0xc0c0c0c0 = -1061109568

✓ 好处：① 加上一个数或翻倍都不会溢出

② 将数组a中每个元素都初始化为无穷大

```
memset(a,0x3f,sizeof(a));
memset(a,0xc0,sizeof(a));
```

最大子数组和

算法1：遍历所有子数组。

```
#include <limits.h>
int maxSubArray(int a[], int n){
   int maxsum = INT_MIN;
   for(int i=0; i<n; i++)
      for(int j=i; j<n; j++){      //考察子数组a[i]…a[j]
         int sum = 0;
         for(int k=i; k<=j; k++)  //sum=a[i]+…+a[j]
            sum += a[k];
         if(sum > maxsum) maxsum = sum;
      }
   return maxsum;
}
```

时间复杂度$O(n^3)$

1	-2	3	10	-4	7	-2	-5

最大子数组和

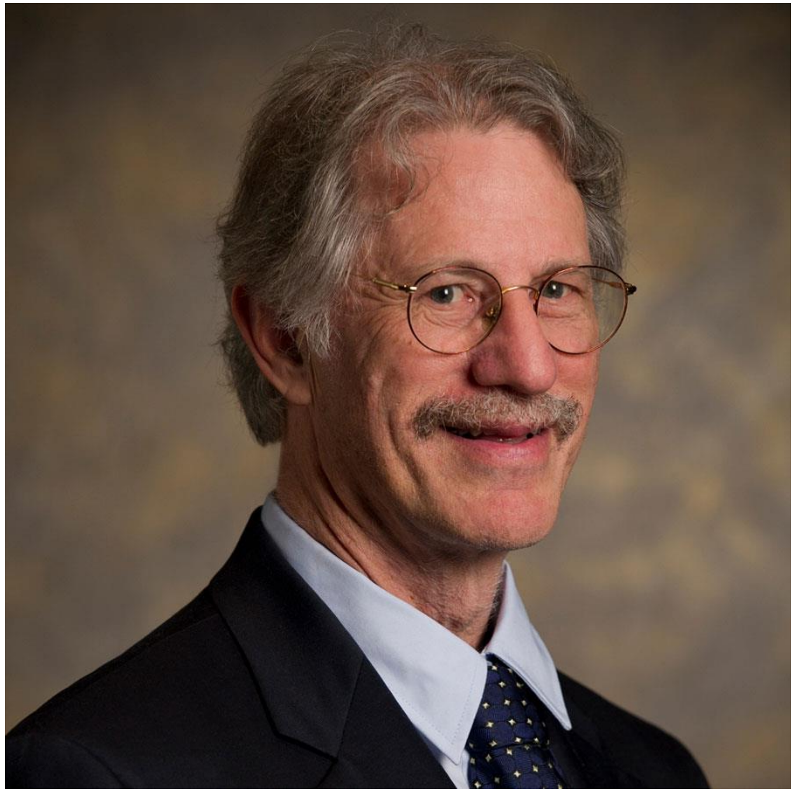
算法2：利用sum[i…j]=sum[i…j-1]+a[j]优化。

```
int maxSubArray(int a[], int n){
    int maxsum = INT_MIN;
    for(int i=0; i<n; i++){
        int sum = 0;
        for (int j=i; j<n; j++) {
            sum += a[j];
            if (sum > maxsum) maxsum = sum;
        }
    }
    return maxsum;
}
```

时间复杂度O(n^2)

1	-2	3	10	-4	7	-2	-5

最大子数组和——线性时间算法



Joseph B. Kadane

(1941 -)

卡内基梅隆大学教授

美国艺术与科学院院士

最大子数组和

算法3：$f(i)$表示所有以位置i结尾的子数组的最大和。

		…		$a[i]$			
			i-1	i			

✓ $f(i\text{-}1)$表示所有以位置i-1结尾的子数组的最大和。

✓ 以位置i结尾的最大子数组一定包含$a[i]$；但包不包含$a[i]$前面的元素？不一定，可能包含也可能不包含。

最大子数组和

情况1：当 $f(i\text{-}1) > 0$ 时

✓以位置*i*结尾的最大子数组=以位置*i*-1结尾的最大子数组+*a*[*i*]

$$f(i) = f(i-1) + a[i], \qquad f(i-1) > 0$$

$f(i\text{-}1)=6$

a								
	-2	3	1	2	3			

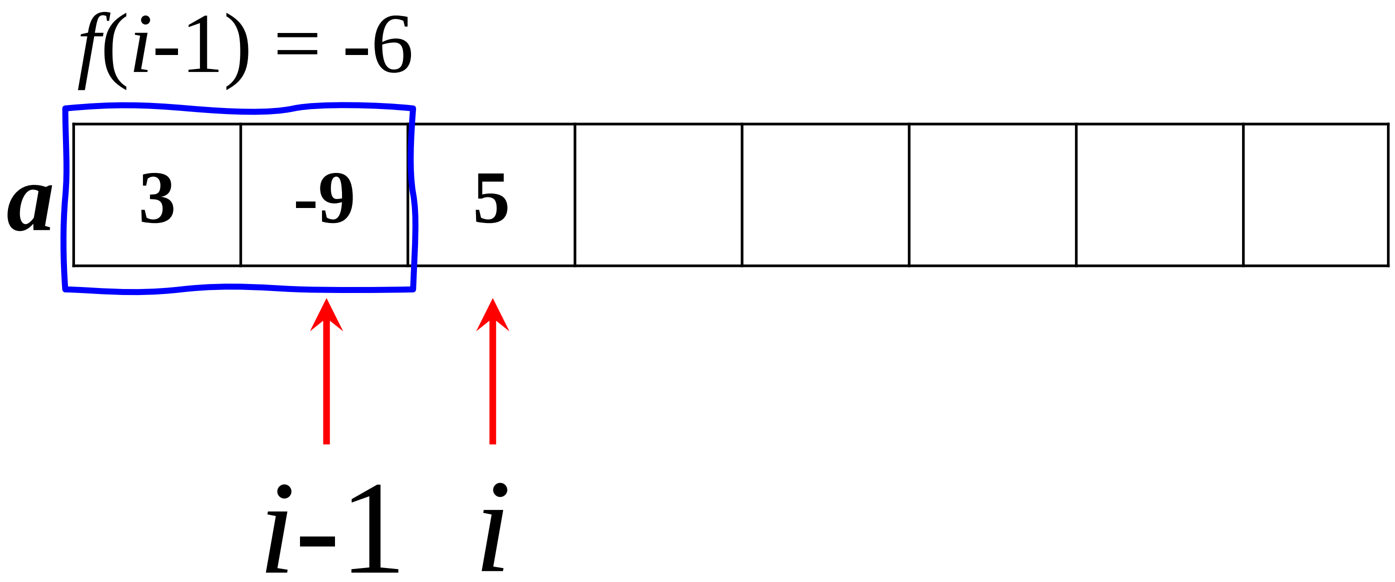
i-1 *i*

最大子数组和

情况2：当 $f(i\text{-}1) \le 0$ 时，则 $a[i] + f(i\text{-}1) \le a[i]$

✓以位置 i 结尾的最大子数组=$a[i]$

$$f(i) = a[i], \qquad f(i-1) \le 0$$



最大子数组和

算法3：*f*(*i*)表示以位置*i*结尾的最大子数组的和。

		…		***a*[*i*]**			

i*-1** ***i

$$f(i)=\begin{cases} f(i-1)+a[i] & ,\ f(i-1)>0 \\ a[i] & ,\ f(i-1)\le 0 \mid i=0 \end{cases}$$

$$\text{所有子数组的最大和}=\max_{i} f(i)$$

最大子数组和

```
const int maxn = 1e5+10;
int maxSubArray(int a[], int n){
    int f[maxn];  f[0] = a[0];
    int maxsum = a[0];
    for (int i=1; i<n; i++) {
        if (f[i-1] > 0) f[i] = f[i-1] + a[i];
        else f[i] = a[i];
        if (f[i] > maxsum) maxsum = f[i];
    }
    return maxsum;
}
```

时间复杂度O(n)
空间复杂度O(n)

$$f(i) = \begin{cases} f(i-1)+a[i] & , f(i-1)>0 \\ a[i] & , f(i-1)\le 0 \mid i=0 \end{cases}$$

最大子数组和——空间优化

```
const int maxn = 1e5+10;
int maxSubArray(int a[], int n){
    int f = a[0];
    int maxsum = a[0];
    for (int i=1; i<n; i++) {
        if (f > 0) f = f + a[i];
        else f = a[i];
        if (f > maxsum) maxsum = f;
    }
    return maxsum;
}
```

时间复杂度O(n)
空间复杂度O(1)

$$f(i)=\begin{cases} f(i-1)+a[i] & , f(i-1)>0 \\ a[i] & , f(i-1)\le 0 \mid i=0 \end{cases}$$

课下思考

找出最大子数组，若存在多个最大子数组，则返回长度最长的，输入数据保证最长的最大子数组只有一个。【大厂面试题，牛客JZ85】

```
int* maxSubArray(int a[], int n, int &subArraySize){ //参数类型与牛客不同
     int f = a[0], maxsum = a[0];
     int start = 0, end = 0, maxstart = 0, maxend = 0;
     for (int i = 1; i < n; i++) {
          if (f >= 0) f = f + a[i];           //起点是f(i-1)的起点
          else { f = a[i]; start = i; }       //起点就是i
          end = i;
          if (f >= maxsum) {     //若有多个子数组满足条件，返回最长的
               maxsum = f; maxstart = start; maxend = end;
          }
     }
     subArraySize = maxend - maxstart + 1;
     return &a[maxstart]; //返回以maxstart为起点，长度为subArraySize的子数组
}
```

时间复杂度 $O(n)$
空间复杂度 $O(1)$

拓展：最大子数组乘积

给定一个整数数组，请找出数组中乘积最大的非空连续子数组（该子数组中至少包含一个数字），并返回该子数组所对应的乘积。【华为、字节跳动、拼多多、微软、谷歌面试题LeetCode152】

-2	4	0	3	2	8	-1

			…	a[i]			
				i			

***f*(*i*)表示以位置*i*结尾的最大子数组的乘积。**

***g*(*i*)表示以位置*i*结尾的最小子数组的乘积。**

$$f(i)=\max\{a[i], f(i-1)\times a[i], g(i-1)\times a[i]\}$$

$$g(i)=\min\{a[i], f(i-1)\times a[i], g(i-1)\times a[i]\}$$

最大子数组乘积——课下阅读

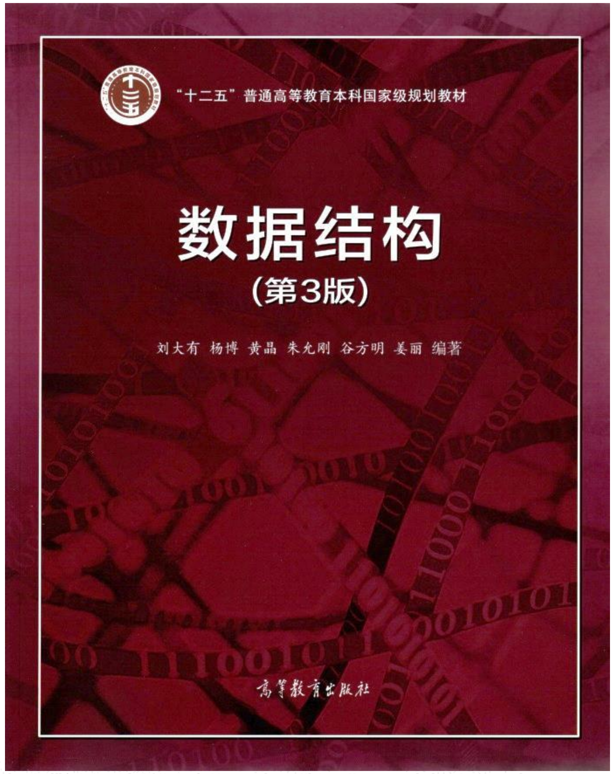
```
int max(int a, int b, int c){
    int maxval=a;
    if(b>maxval) maxval=b;
    if(c>maxval) maxval=c;
    return maxval;
}
```

```
int min(int a, int b, int c){
    int minval=a;
    if(b<minval) minval=b;
    if(c<minval) minval=c;
    return minval;
}
```

```
int maxProduct(int a[], int n){
    int f=a[0], g=a[0], maxproduct=a[0];
    for(int i=1;i<n;i++){
        int pref=f*a[i], preg=g*a[i];
        f=max(a[i], pref, preg);
        g=min(a[i], pref, preg);
        if(f>maxproduct) maxproduct=f;
    }
    return maxproduct;
}
```

时间复杂度O(n)
空间复杂度O(1)

数组与矩阵



- 数组存储与寻址
- 特殊矩阵的压缩存储
- 三元组表
- 十字链表
- 动态规划初探
- **区间处理技巧**
  - **前缀和**
  - 差分数组
  - ST表
  - 尺取法
- 子集生成

数据结构之美
算法之道
数据之法

zhuyungang@jlu.edu.cn

例：老师想统计学生考试的平均分，假定一共有n名学生，学号为1至n。现按学号递增顺序给定每个学生的分数，请编写程序，帮助老师统计学号*i*到学号*j*之间的学生的平均分【吉林大学2019级上机考试题】。

输入格式：输入第一行为2个整数n和m($0<n,m\le10^5$)，n为学生人数，m为查询次数。第二行为n个正整数，表示学号1至n的学生的成绩。接下来m行，每行两个正整数i和j（$1\le i, j\le n$），表示一个查询，即查询学号i至学号j间的学生。

输出格式：输出m行，每行为所求的平均分，截尾取整

输入	输出
9 2 10 20 30 40 50 60 70 80 90 1 9 3 6	50 45

区间求和问题

本质：给定一个长度为n的数组a，做m次查询，每次查询区间i至j的元素之和。【大厂面试题，洛谷B3612】

暴力方法：


```
while(m--){
    scanf("%d %d",&i,&j);
    sum=0;
    for(int k=i;k<=j;k++)
        sum+=a[k];
}
```

时间复杂度
$O(nm)$

前缀和：一种重要的数据预处理技巧

- 令 $sum_i = a_1+a_2+\ldots+a_i$,
- 则 $a_i+\ldots+a_j = sum_j - sum_{i-1}$

每次区间求和O(1)

$$\underbrace{\overbrace{a_1 \;\; \ldots \;\; a_{i-1}}^{sum_{i-1}} \;\; a_i \;\; \ldots \;\; a_j}_{sum_j}$$

前缀和：一种重要的数据预处理技巧

- 令$sum_i = a_1+a_2+\ldots+a_i$，
- $sum_i = a_1+a_2+\ldots+a_{i-1}+a_i$
  $= sum_{i-1}+a_i$

```
sum[0]=0;
for(int i=1;i<=n;i++)
    sum[i]=sum[i-1]+a[i];
```

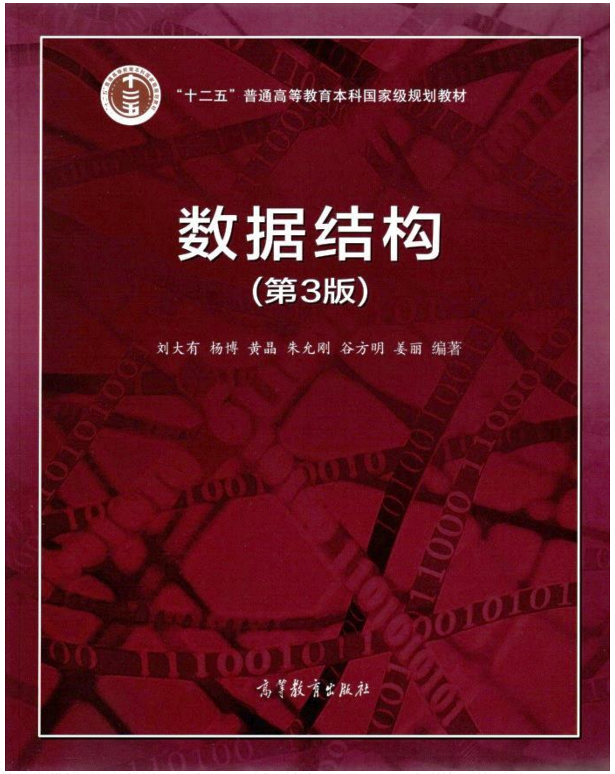
【洛谷B3612】

```
#include<stdio.h>
using namespace std;
const int N = 1e5 + 10;
int main() {
	int sum[N], a[N], n, m, i, j;
	sum[0] = 0;
	scanf("%d", &n);
	for (i = 1; i <= n; i++) {
		scanf("%d", &a[i]);
		sum[i] = sum[i-1] + a[i];
	}
	scanf("%d", &m);
	while (m--) {
		scanf("%d %d", &i, &j);
		printf("%d\n", sum[j]-sum[i-1]);
	}
	return 0;
}
```

$O(n+m)$

前缀和技巧适用场合：数组在不被修改的情况下，频繁查询某个区间的累加和。

预处理$O(n)$
每次区间查询$O(1)$



数组与矩阵

- 数组存储与寻址
- 特殊矩阵的压缩存储
- 三元组表
- 十字链表
- 动态规划初探
- **区间处理技巧**
  - 前缀和
  - **差分数组**
  - ST表
  - 尺取法
- 子集生成

数据之法
结构之美
算法之道

Last updated on 2023.9

zhuyungang@jlu.edu.cn

差分数组：一种重要的数据预处理技巧

给定一个长度为n的数组a（下标从1开始），做m次操作，每次操作为3个整数i、j、d，表示对区间i至j的所有元素加上d，返回最后的数组。【大厂面试题，洛谷P2367、LeetCode1109】

示例：

	1	2	3	4	5	6	7	8	9
a	6	10	20	29	35	50	60	70	80

操作

3 6 8	6	10	**28**	**37**	**43**	**58**	60	70	80
4 7 10	6	10	28	**47**	**53**	**68**	**70**	70	80
2 8 2	6	**12**	**30**	**49**	**55**	**70**	**72**	**72**	80

给定一个长度为*n*的数组*a*，做*m*次操作，每次操作为3个整数*i*、*j*、*d*，表示对区间*i*至*j*的所有元素加上*d*。

暴力方法：


```
while(m--){
   scanf("%d %d %d",&i,&j,&d);
   for(int k=i;k<=j;k++)
      a[k]+=d;
}
```

时间复杂度
O(*nm*)

差分数组 $diff[i]=a[i]-a[i-1]$

	1	2	3	4	5	6	7	8
a	1	2	5	6	9	8	10	7
$diff$								

差分数组 $diff[i]=a[i]-a[i-1]$

	1	2	3	4	5	6	7	8
a	1	2	5	6	9	8	10	7
$diff$	1	1	3	1	3	-1	2	-3

$$diff[i]=a[i]-a[i-1]$$

	1	2	3	4	5	6	7	8
a	1	2	5+3	6+3	9+3	8+3	10	7
$diff$	1	1	3+3	1	3	-1	2-3	-3

i j

```
diff[i]+=d;
if(j<n) diff[j+1]-=d;
```

1次区间操作
O(1)

给定一个长度为n的数组a（下标从1开始），做m次操作，每次操作是对区间i至j的所有元素加上d，返回最后的数组。

```
const int N=5e6+10;
void RangeIncrement(int a[],int n,int m){
      int diff[N],i,j,d;
      diff[1]=a[1];    //计算差分数组
      for(int i=2;i<=n;i++)
            diff[i]=a[i]-a[i-1];
      while(m--){       //a[i]…a[j]加d
            scanf("%d %d %d",&i,&j,&d);
            diff[i]+=d;
            if(j<n) diff[j+1]-=d;
      }
      a[1]=diff[1];    //利用diff反推/恢复数组a
      for(int i=2; i<=n; i++)
            a[i]=a[i-1]+diff[i];
}
```

$O(n+m)$

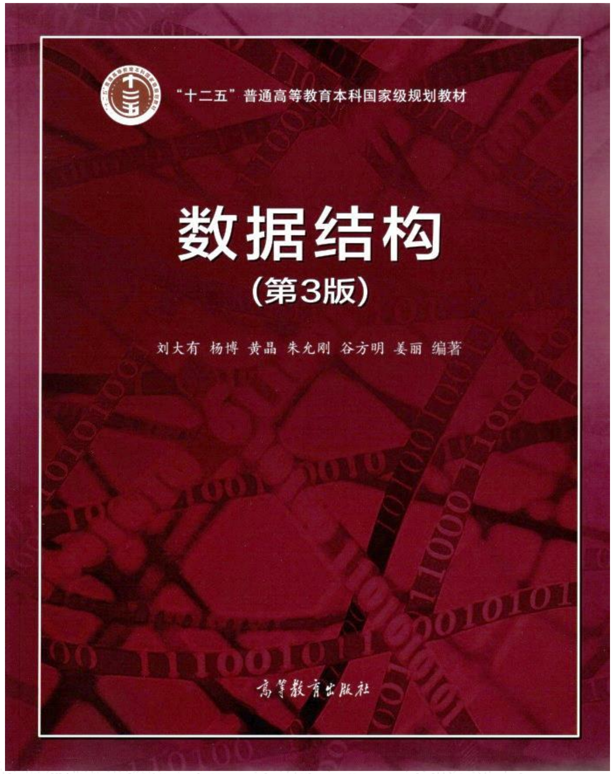
$diff[i]=a[i]-a[i-1]$

预处理$O(n)$
每次区间操作$O(1)$

差分数组适用场合：频繁对数组的某个区间的元素进行增减。

课下思考
若数组a下标从0开始，如何修改代码

数据之法
结构之美
算法之道



数组与矩阵

- 数组存储与寻址
- 特殊矩阵的压缩存储
- 三元组表
- 十字链表
- 动态规划初探
- **区间处理技巧**
  - 前缀和
  - 差分数组
  - **ST表**
  - 尺取法
- 子集生成

zhuyungang@jlu.edu.cn

区间最值问题（*Range Maximum/Minimum Query, RMQ*）

给定一个长度为n的数组a（下标从1开始），做m次查询，每次查询为2个整数i、j，表示区间i至j的最大值。【洛谷P3865】

示例：

	1	2	3	4	5	6	7	8	9
a	3	9	1	2	5	6	0	7	8

查询	输出
3 6	6
5 8	7
2 7	9

区间最值问题（*Range Maximum/Minimum Query, RMQ*）

给定一个长度为*n*的数组*a*（下标从1开始），做*m*次查询，每次查询为2个整数*i*、*j*，表示区间*i*至*j*的最大值。

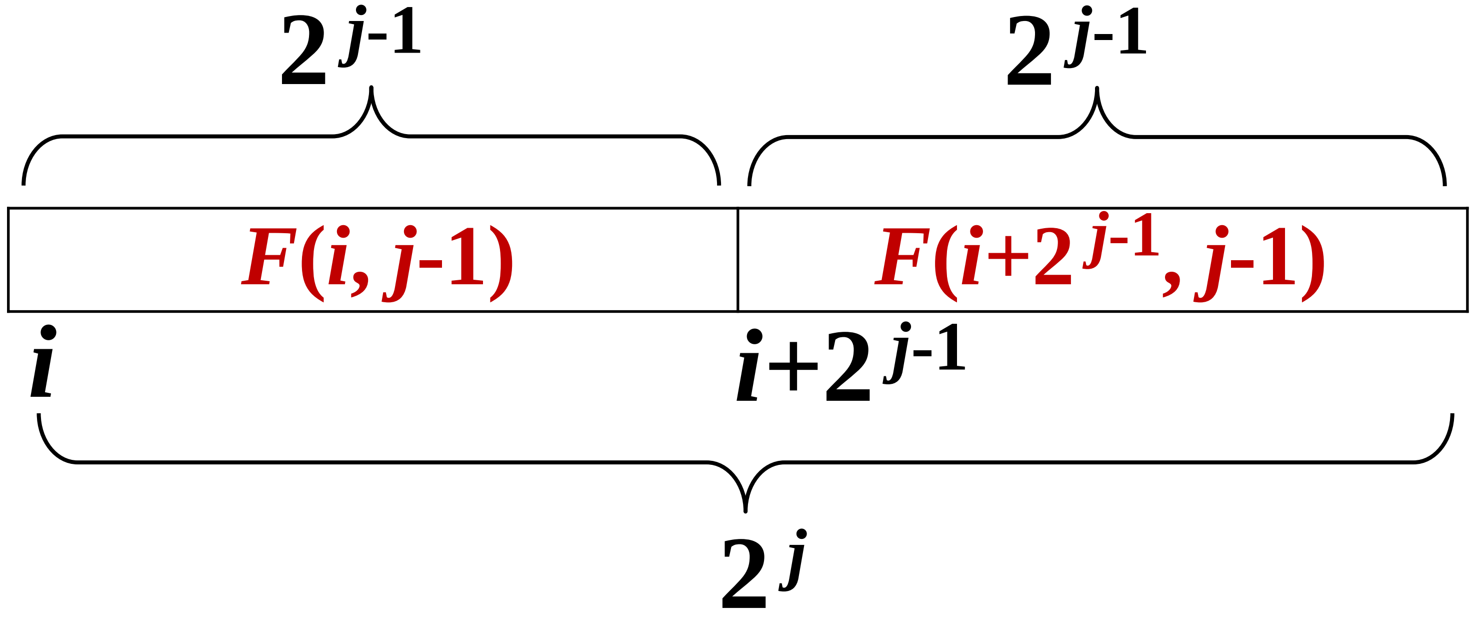
暴力方法


```
while(m--){
    scanf("%d %d",&i,&j);
    int max=INT_MIN;
    for(int k=i;k<=j;k++)
        if(a[k]>max) max=a[k];
}
```

时间复杂度 O(*nm*)

ST表（Sparse Table）

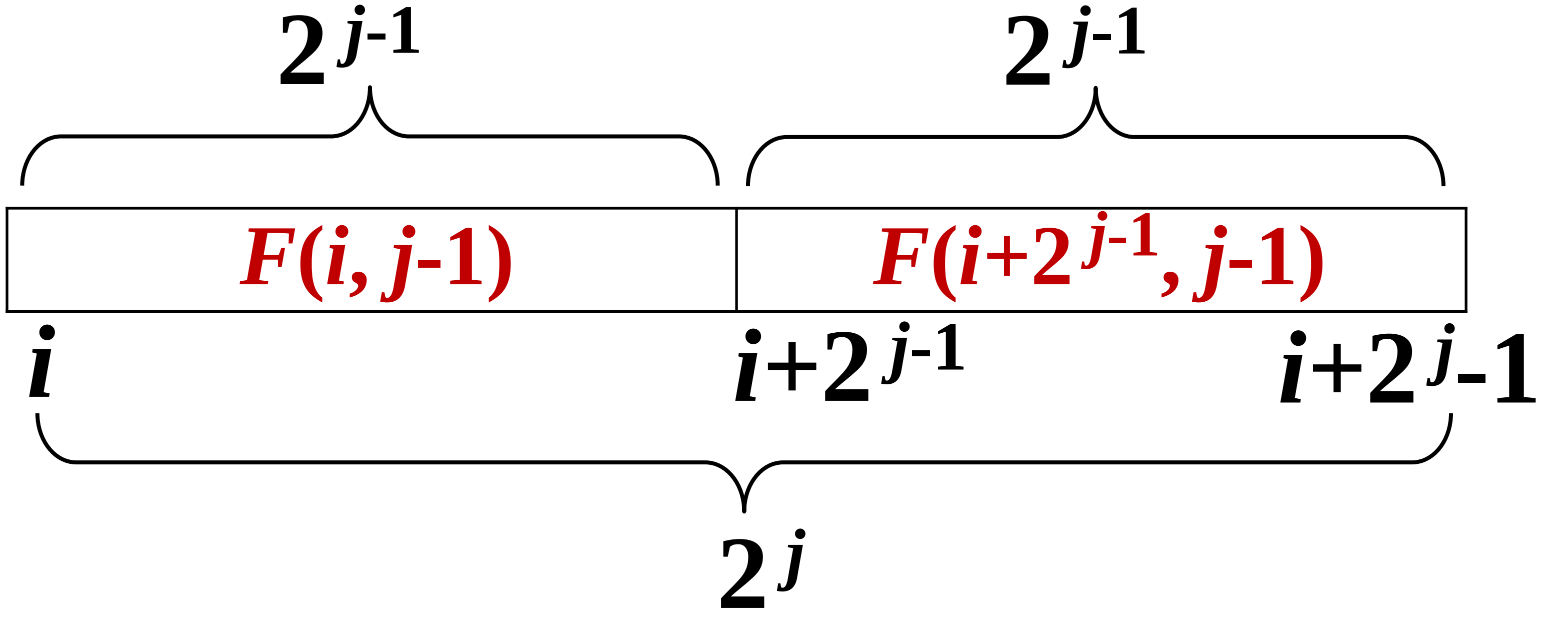
- 令$F(i, j)$表示数组A中从下标i开始的2^j个数的最大值，即子区间$[i, i+2^j-1]$的最大值。
- 区间$[i, i+2^j-1]$由两个区间组成：①下标i开始的长度为2^{j-1}的区间，其最值为$F(i, j-1)$；②下标$i+2^{j-1}$开始的长度为2^{j-1}的区间，其最值为$F(i+2^{j-1}, j-1)$；



ST表（Sparse Table）

- 长度为2^j的区间的最大值是左右两半长度为2^{j-1}的区间的最大值中的较大者。

$$F(i,j)=\begin{cases}\max\{F(i,j-1),F(i+2^{j-1},j-1)\}, & j>0\\ a[i], & j=0\end{cases}$$



$0\le j\le \log_2 n$
$1\le i\le n-2^j+1$

ST表（Sparse Table）

$$F(i,j)=\begin{cases}\max\{F(i,j-1),F(i+2^{j-1},j-1)\}, & j>0\\ a[i], & j=0\end{cases}$$

		j-1	j	
i		$F[i][j\text{-}1]$	$F[i][j]$	
		⋮		
$i+2^{j-1}$		$F[i+2^{j-1}][j\text{-}1]$		

如何计算2^j

十进制表示	二进制表示	相当于
2^0	0 0 0 1	

如何计算2^j

十进制表示	二进制表示	相当于
2^0	0 0 0 1	
2^1	0 0 1 0	1左移1位

如何计算2^j

十进制表示	二进制表示	相当于
2^0	0 0 0 1	
2^1	0 0 1 0	1左移1位
2^2	0 1 0 0	1左移2位

如何计算2^j

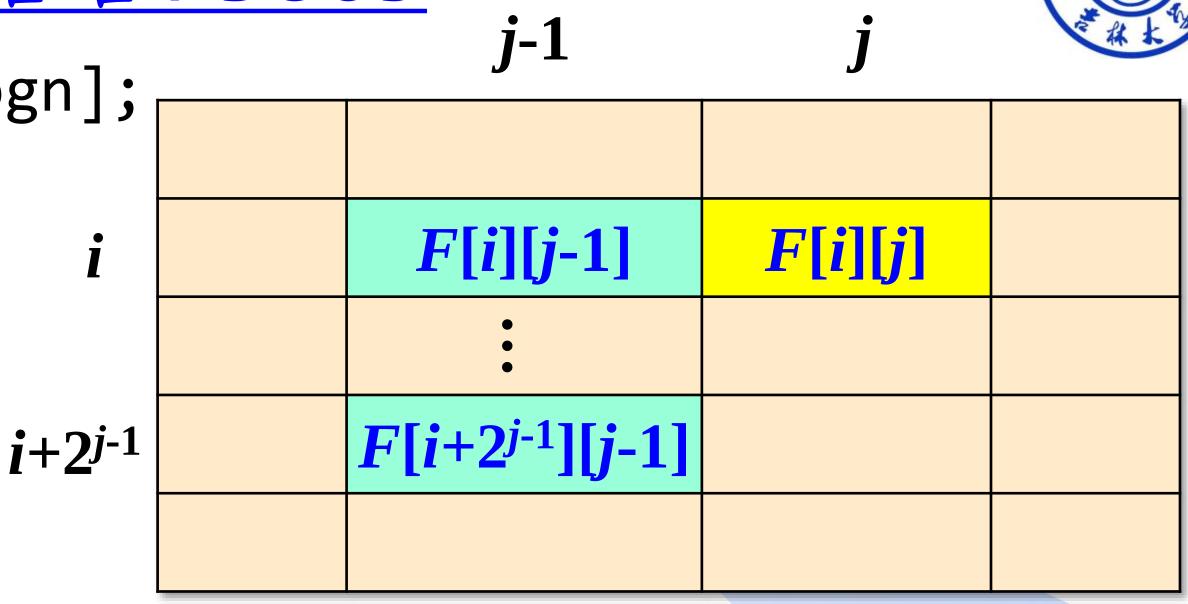
十进制表示	二进制表示	相当于
2^0	0 0 0 1	
2^1	0 0 1 0	1左移1位
2^2	0 1 0 0	1左移2位
2^3	1 0 0 0	1左移3位

2^j：将1对应的二进制数左移j位

C/C++位运算：1<<j

构建ST表 洛谷P3865

```
int main() {
    int n,m,L,R,a[maxn],F[maxn][maxlogn];
    scanf("%d %d", &n, &m);
    for (int i=1; i<=n; i++) {
        scanf("%d", &a[i]);
        F[i][0] = a[i];
    }
    int logn = log2(n);
    for (int j=1; j<=logn; j++)
        for (int i=1; i<=n-(1<<j)+1; i++)
            F[i][j]=max(F[i][j-1], F[i+(1<<(j-1))][j-1]);

    return 0;
}
```



$0 \le j \le \log_2 n$

$1 \le i \le n-2^j+1$

$$F(i,j)=\begin{cases}\max\{F(i,j-1),F(i+2^{j-1},j-1)\}, & j>0\\ a[i], & j=0\end{cases}$$

时间复杂度
O(*n*log*n*)

```
const int maxn=1e5+10;
const int maxlogn=18;
int max(int a, int b){
    return (a>b)?a:b;
}
```

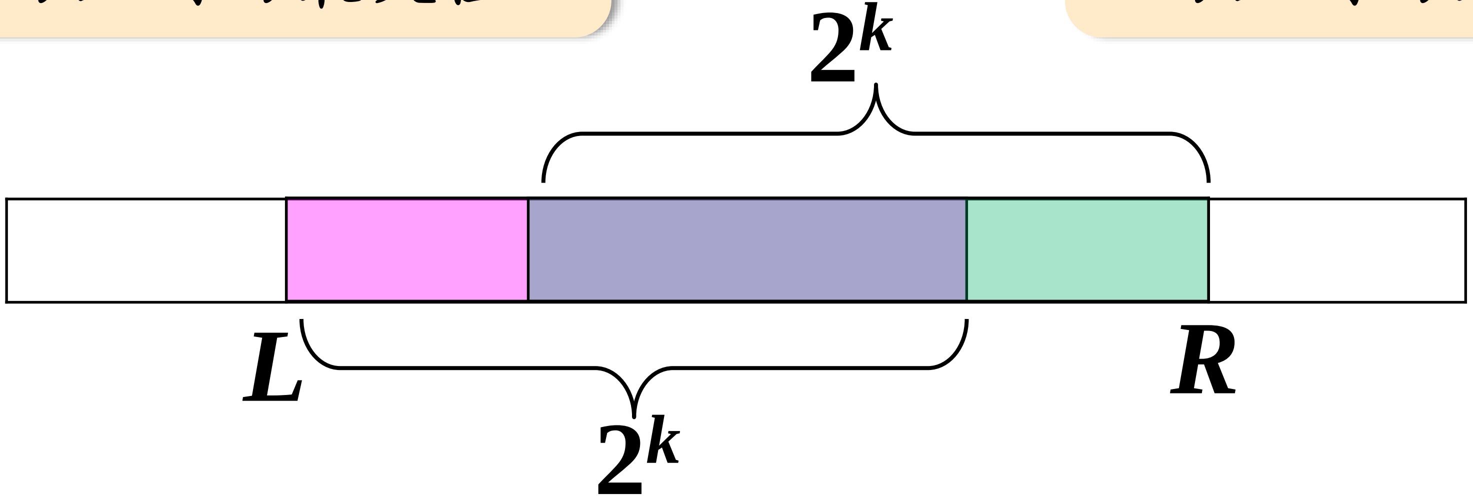
查询区间[L, R]的最值

- 令$F(i, j)$表示数组A中以下标i为起点的长度为2^j的区间的最大值，即子区间$[i, i+2^j-1]$的最大值。
- 令$k=\lfloor \log_2(R-L+1) \rfloor$
- 区间[L, R]的最大值= $max\{ F[L][k]$ ，$F[R-2^k+1][k]\}$

以L为起点、长度为2^k的区间的最大值

以R为终点、长度为2^k的区间的最大值

每次查询
O(1)



ST表 洛谷P3865

```
int main() {
    int n,m,L,R,a[maxn],F[maxn][maxlogn];
    scanf("%d %d", &n, &m);
    for (int i=1; i<=n; i++) {
        scanf("%d", &a[i]);
        F[i][0] = a[i];
    }
    int logn = log2(n);
    for (int j=1; j<=logn; j++)
        for (int i=1; i<=n-(1<<j)+1; i++)
            F[i][j]=max(F[i][j-1], F[i+(1<<(j-1))][j-1]);
    while(m--){
        scanf("%d %d", &L, &R);
        int k = log2(R - L + 1);
        int ans=max(F[L][k],F[R-(1<<k)+1][k]);
        printf("%d\n",ans);
    }
    return 0;
}
```

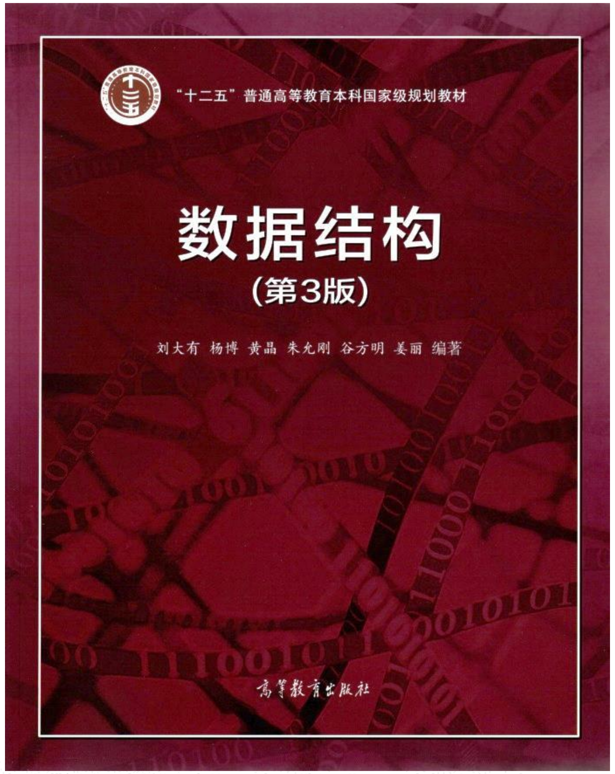
时间复杂度O(*nlogn*+*m*)

预处理O(*nlogn*)
每次区间查询O(1)

倍增思想

$max\{ F[L][k] , F[R-2^k+1][k] \}$

```
const int maxn=1e5+10;
const int maxlogn=18;
int max(int a, int b){
    return (a>b)?a:b;
}
```



数组与矩阵

- 数组存储与寻址
- 特殊矩阵的压缩存储
- 三元组表
- 十字链表
- 动态规划初探
- **区间处理技巧**
  - 前缀和
  - 差分数组
  - ST表
  - **尺取法**
- 子集生成

数据之法
结构之美
算法之道

Last updated on 2023.9

zhuyungang@jlu.edu.cn

例：给定一个长度为n的正整数数组a和一个整数S，在这个数组中找出元素之和等于S的所有区间，输出区间的起点和终点位置。

示例：

数组a： 6 1 2 3 4 6 4 1 1 8 9，S=6

输出：

0 0

1 3

5 5

6 8

例：给定一个长度为n的**正整数**数组a和一个整数S，在这个数组中找出元素之和等于S的所有区间，输出区间的起点和终点位置。

暴力方案：

```
for(int i=0;i<n;i++)
    for(int j=i;j<n;j++){
        //看a[i]+…+a[j]是否等于S
    }
```

例：给定一个长度为*n*的正整数数组*a*和一个整数*S*，在这个数组中找出元素之和等于*S*的所有区间，输出区间的起点和终点位置。以*S*=9为例

5	2	3	6	2	7	1	5	3	2

例：给定一个长度为*n*的正整数数组*a*和一个整数*S*，在这个数组中找出元素之和等于*S*的所有区间，输出区间的起点和终点位置。以S=9为例

5	2	3	6	2	7	1	5	3	2

- 区间和＜S，区间右边扩1位（区间终点推进1位）
- 区间和≥S，区间左边缩1位（区间起点推进1位）

5	**2**	3	6	2	7	1	5	3	2

5	**2**	**3**	6	2	7	1	5	3	2

5	**2**	**3**	6	2	7	1	5	3	2

5	**2**	**3**	**6**	2	7	1	5	3	2

5	2	**3**	**6**	2	7	1	5	3	2

找到

5	2	3	**6**	2	7	1	5	3	2

5	2	3	**6**	**2**	7	1	5	3	2

5	2	3	**6**	**2**	**7**	1	5	3	2

5	2	3	6	**2**	**7**	1	5	3	2

找到

5	2	3	6	2	**7**	1	5	3	2

5	2	3	6	2	**7**	**1**	5	3	2

5	2	3	6	2	**7**	**1**	**5**	3	2

5	2	3	6	2	7	**1**	**5**	3	2

5	2	3	6	2	7	**1**	**5**	**3**	2

找到

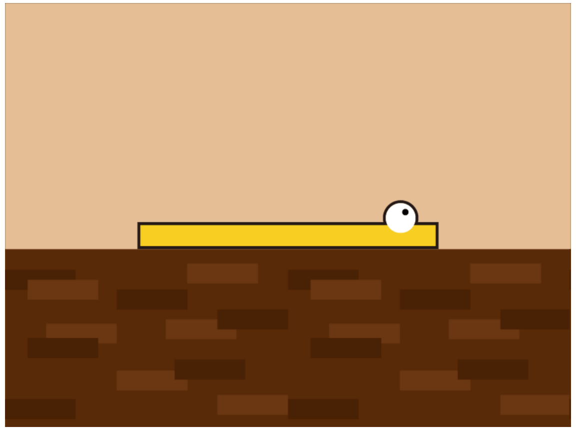
5	2	3	6	2	7	1	**5**	**3**	2

5	2	3	6	2	7	1	**5**	**3**	**2**

例：给定一个长度为n的正整数数组a和一个整数S，在这个数组中找出元素之和等于S的所有区间，输出区间的起点和终点位置。

时间复杂度$O(n)$

5	2	3	6	2	7	1	5	**3**	**2**



尺取法：维护两个指针（下标）指向区间的起点和终点，根据实际情况交替推进两个指针（区间的左右边界），直到得出答案。

应用举例

给定一个长度为*n*的正整数数组*a*和一个正整数*S*，在这个数组中找出元素之和大于等于S的最短区间，返回该区间长度。若不存在满足条件的区间，返回0。【华为、腾讯、字节跳动、谷歌、微软、苹果面试题LeetCode209、POJ3061】

示例：

输入：*a*=[2 3 1 1 4 3]，S=6

输出：2

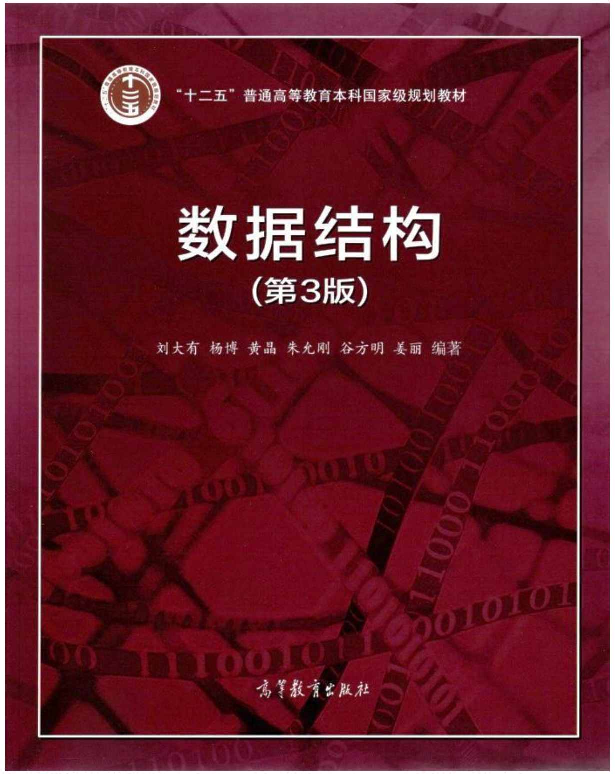
```
int minSubArrayLen(int S, int a[], int n){
    int left=0,right=0,sum=a[0],min=n+1;
    while(true){
        while(right<n-1 && sum<S)
            sum+=a[++right];      //右扩一位
        if(sum<S) break;          //扩到头时区间和小于S
        int len=right-left+1;     //找到了一个和≥S的区间
        if(len<min) min=len;
        sum-=a[left++];           //左缩一位
    }
    if(min==n+1)
        min=0;
    return min;
}
```

如果找到sum ≥ S的区间，则需要做两件事

① 找到了满足条件的区间，与当前最短区间比较

② 区间左边缩一位

课下思考：若数组a中有负数，上述算法还行么？



数组与矩阵

- 数组存储与寻址
- 特殊矩阵的压缩存储
- 三元组表
- 十字链表
- 动态规划初探
- 区间问题处理技巧
- **子集生成**

数据之法
结构之美
算法之道

Last updated on 2023.9

zhuyungang@jlu.edu.cn

子集生成

输出一个集合的幂集，如集合{*a*, *b*, *c*, *d*}的幂集如下。【百度、字节跳动、华为、快手、谷歌面试题】

{ }
{ a }
{ b }
{ a b }
{ c }
{ a c }
{ b c }
{ a b c }
{ d }
{ a d }
{ b d }
{ a b d }
{ c d }
{ a c d }
{ b c d }
{ a b c d }

a	***b***	***c***	***d***

课下阅读

```c
void PowerSet(char s[], int n){
    int a[maxsize] = { 0 };
    while (a[n] == 0) {
        printf("{ ");
        for(int i=0; i<n; i++)
            if (a[i] == 1)
                printf("%c ", s[i]);
        printf("}\n");
        AddOne(a, n);
    }
}
```

```c
const int maxsize = 10;
void AddOne(int a[], int n) {
    int i = 0;
    while (a[i] != 0)
        a[i++] = 0;
    a[i] = 1;
}
```

```c
int main() {
    char s[maxsize]={'a','b','c','d',
'e','f','g'};
    int n=4;
    PowerSet(s, n);
    return 0;
}
```

类似题目：洛谷B3622

```
void PowerSet(int n) {
    int a[maxsize] = { 0 };
    while (a[n] == 0) {
        for (int i=n-1; i>=0; i--)
            if (a[i] == 1)
                printf("Y");
            else printf("N");
        printf("\n");
        addOne(a, n);
    }
}
```

```
const int maxsize = 15;
void AddOne(int a[], int n) {
    int i = 0;
    while (a[i] != 0)
        a[i++] = 0;
    a[i] = 1;
}
```

```
int main() {
    int n;
    scanf("%d", &n);
    PowerSet(n);
    return 0;
}
```

矩阵应用-图像处理

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & a_{m,3} & \cdots & a_{m,n} \end{bmatrix}$$

黑白图像：灰度值
彩色图像：RGB值