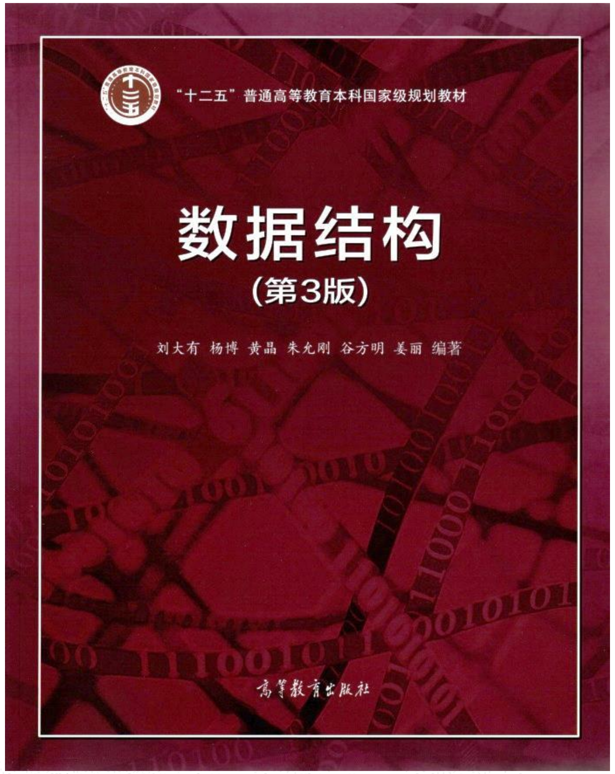


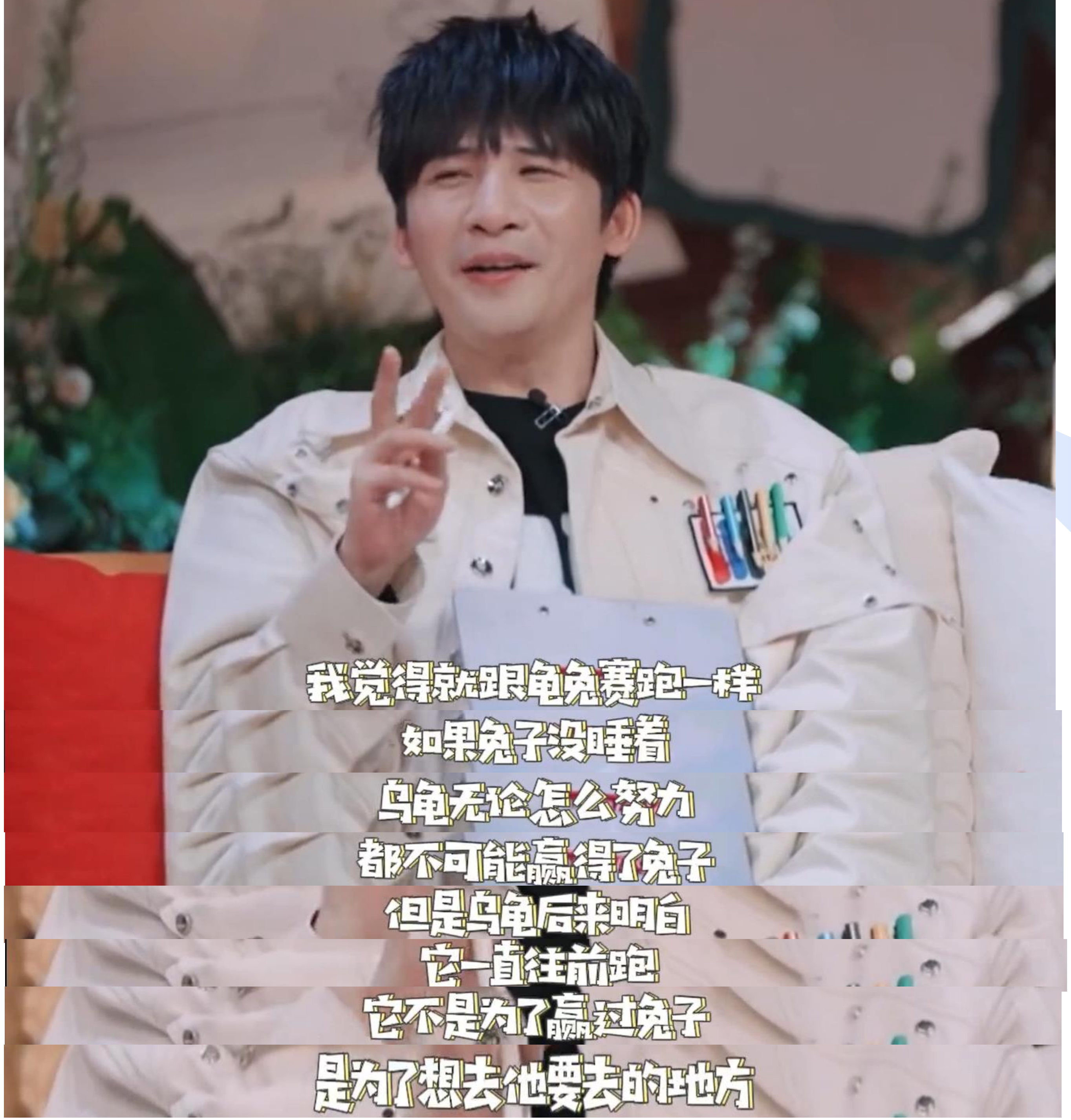
数组与矩阵

- 数组存储与寻址补充
- 特殊矩阵的压缩存储
- 三元组表
- 十字链表
- 动态规划初探
- 前缀和与差分数组
- 尺取法
- 其他问题选讲

数据之法
结构之美
算法之道

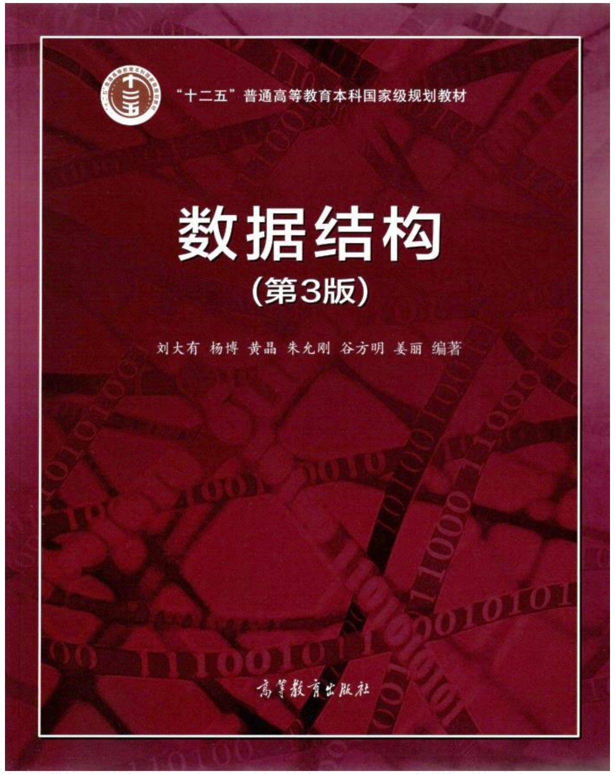
Last updated on 2023.9

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
- 前缀和与差分数组
- 尺取法
- 其他问题选讲

数据之法
结构之美
算法之道

zhuyungang@jlu.edu.cn

n 维数组

- 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：

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

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______.

Loc(A[1][2][6][1])

$=1000+4*(165i+33j+3k+l)$

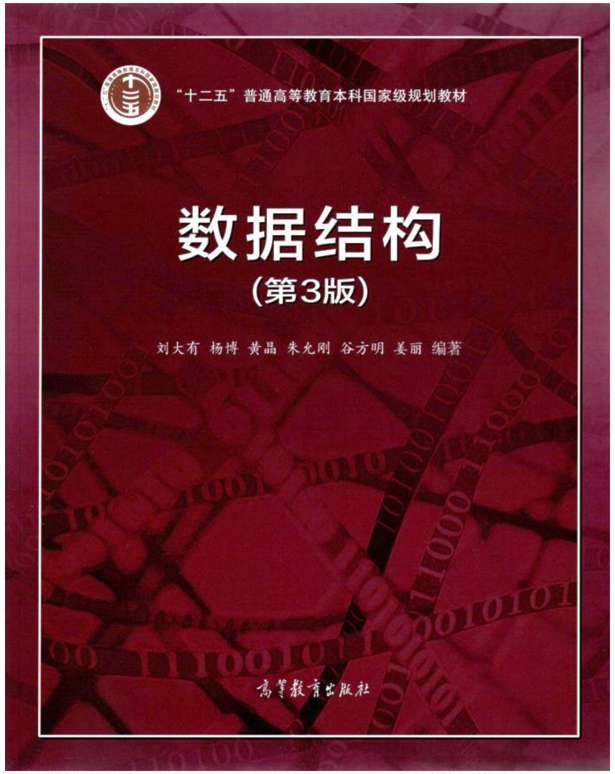
$=1000+4*(165*1+33*2+3*6+1)$

$=1000+4*250$

$=2000$

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
- 前缀和与差分数组
- 尺取法
- 其他问题选讲



数据之法
结构之美
算法之道

zhuyungang@jlu.edu.cn

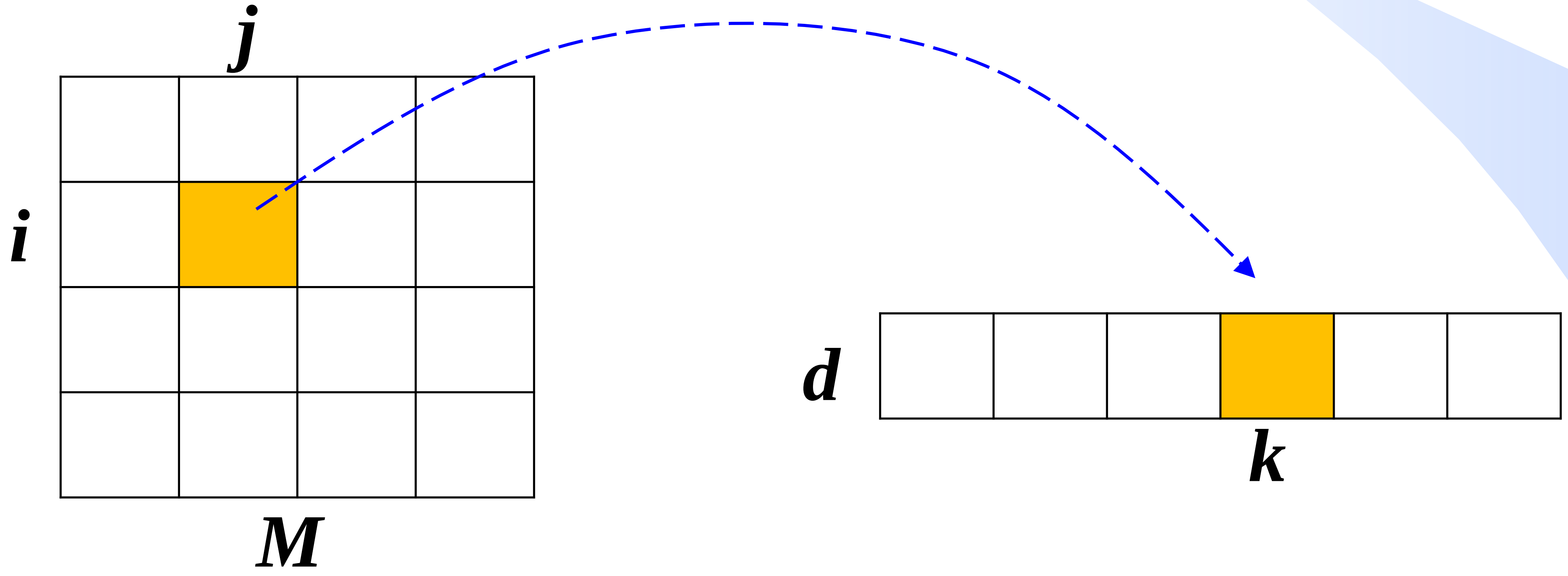
对角矩阵的压缩存储

- 若$n\times n$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$n\times n$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。

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

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

$$d=\begin{bmatrix} d_0 & d_1 & d_2 & d_3 \end{bmatrix}$$

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

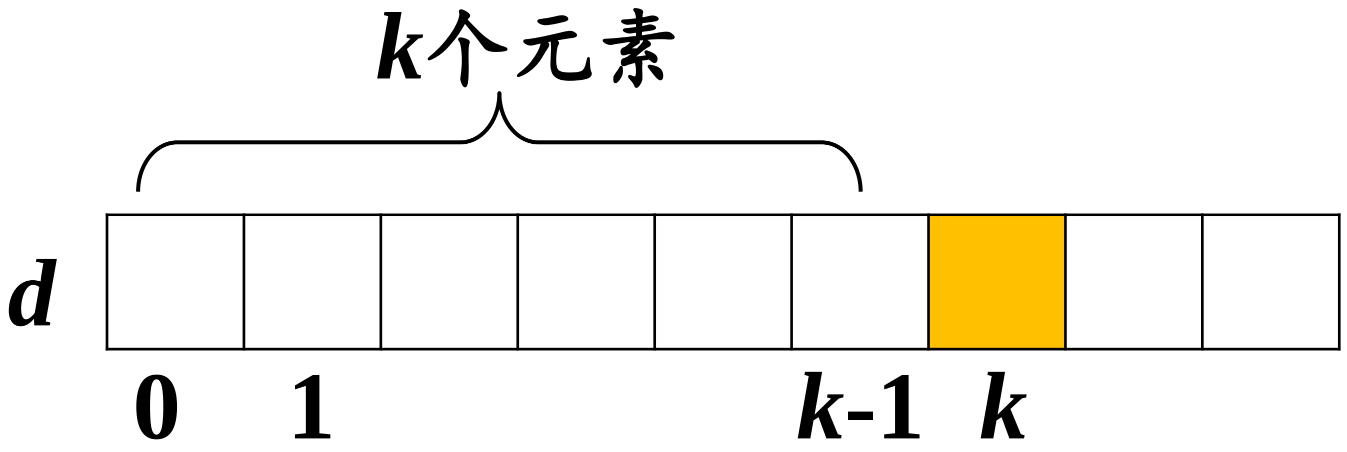
以下三角矩阵M为例，讨论其压缩存储方法：

- ➢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？

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

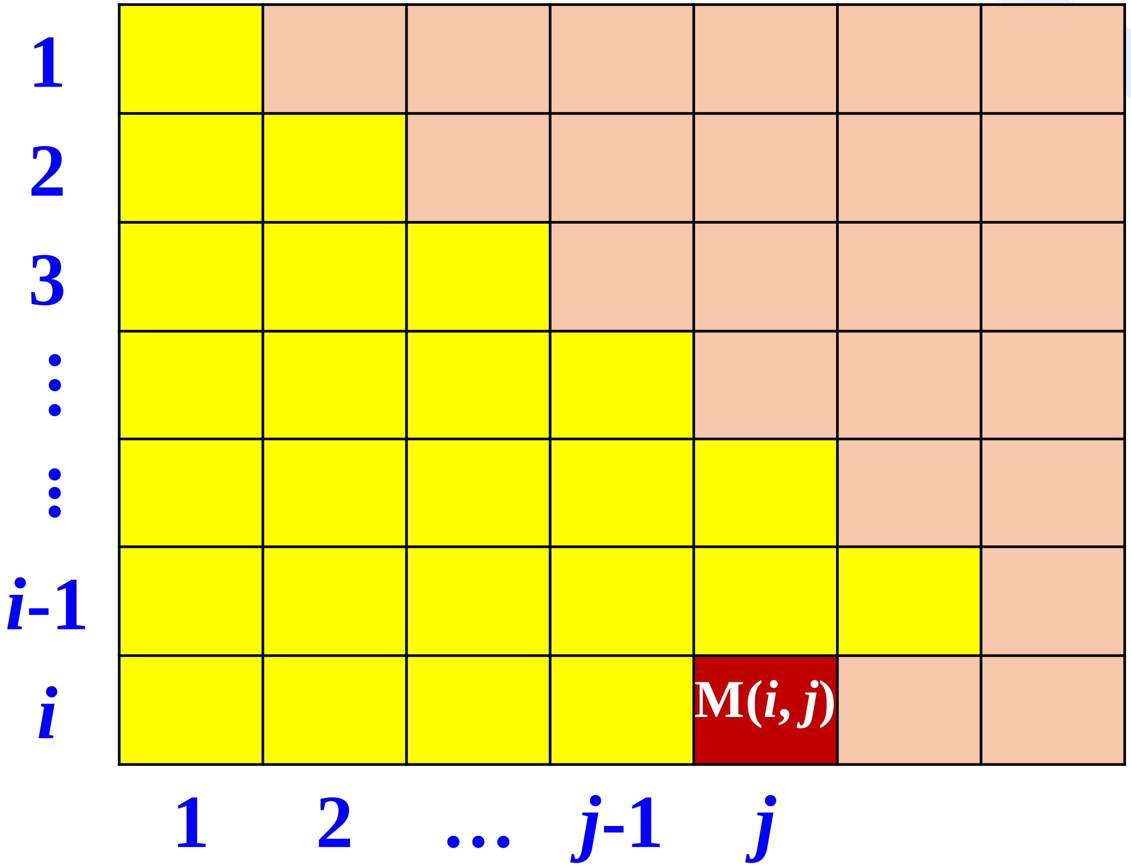
M(*i*, *j*)在数组d的什么位置

- 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)]

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

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？

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

对于对称矩阵中的下三角元素M(*i*, *j*) (*i*≥*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

- $i < j$, $\mathbf{M}(i, j)=\mathbf{M}(j, i)=\mathbf{d}[q]$，$q= j(j-1)/2 + (i-1)$

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

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]

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

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

$$\begin{bmatrix} a_{1,1} & a_{1,2} & & & & & & \\ a_{2,1} & a_{2,2} & a_{2,3} & & & & & \\ & a_{3,2} & a_{3,3} & a_{3,4} & & & & \\ & & a_{4,3} & a_{4,4} & a_{4,5} & & & \\ & & & \cdots & \cdots & & & \\ & & & & a_{i,i-1} & a_{i,i} & a_{i,i+1} & \\ & & & & & \cdots & \cdots & \\ & & & & & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ & & & & & & a_{n,n-1} & a_{n,n} \end{bmatrix}$$

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

M(*i*, *j*)前面有*k*个元素

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

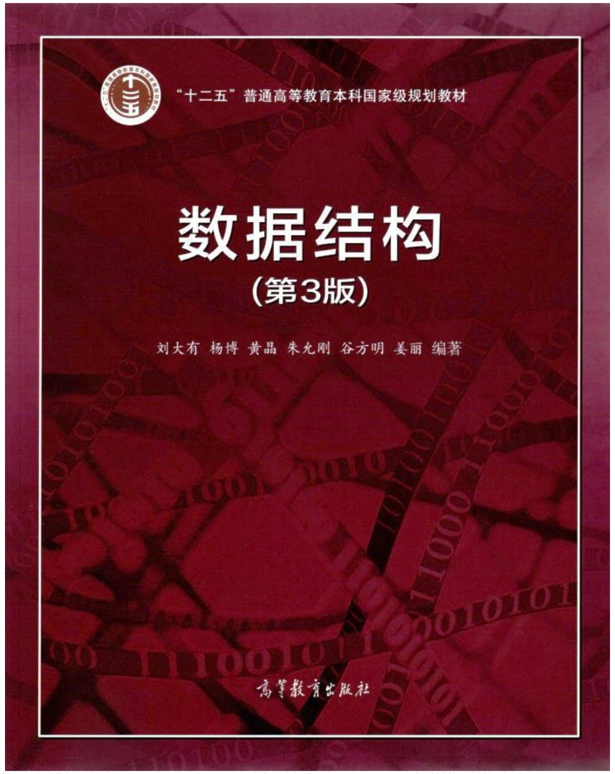
$=2i+j-3$

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

课下思考

有一个100阶的三对角矩阵M，其元素M(i, j) ($1 \le i, j \le 100$)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
- 前缀和与差分数组
- 尺取法
- 其他问题选讲

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

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

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

转置后

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

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

$O(nt)$

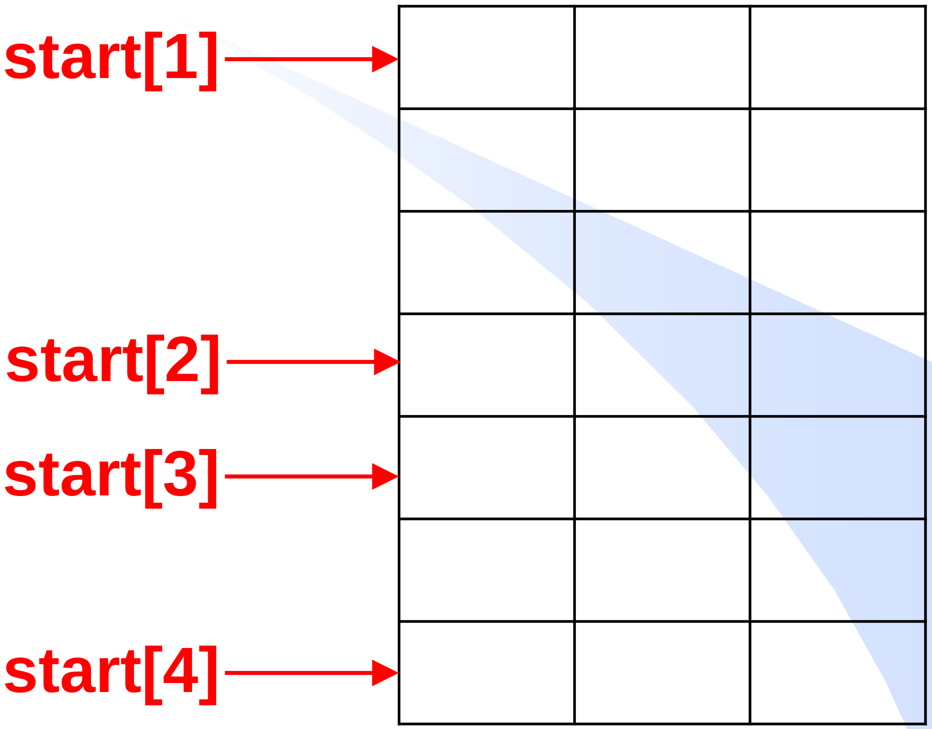
快速转置算法

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

b



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。

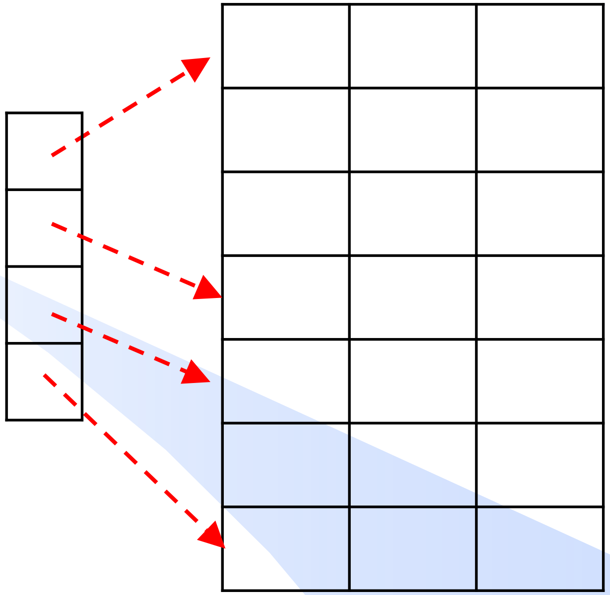
```
for (i = 1; i <= n; i++)
    rowsize[i] = 0;

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]++;
}
```

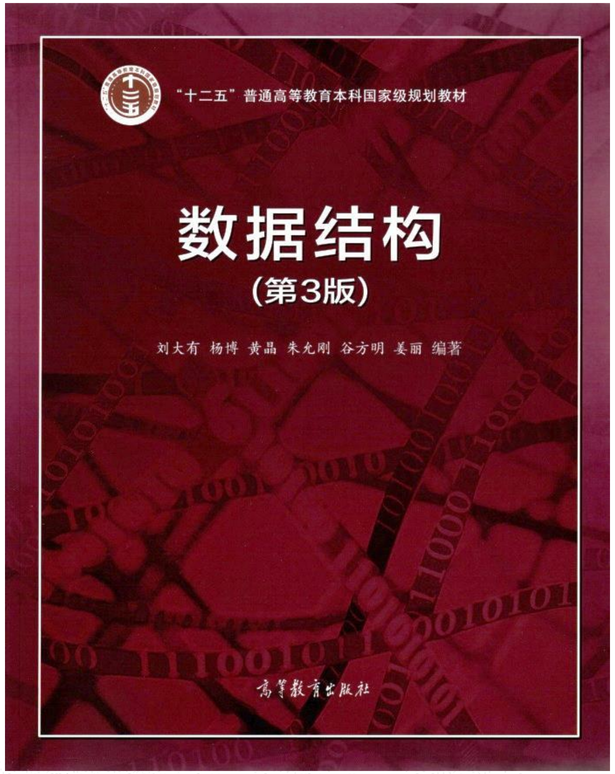


1	1	50
2	1	10
2	3	20
3	2	5
4	1	-30
4	3	6
4	4	5

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

- 数组存储与寻址补充
- 特殊矩阵的压缩存储
- 三元组表
- **十字链表**
- 动态规划初探
- 前缀和与差分数组
- 尺取法
- 其他问题选讲

数据之法
结构之美
算法之道

zhuyungang@jlu.edu.cn

十字链表

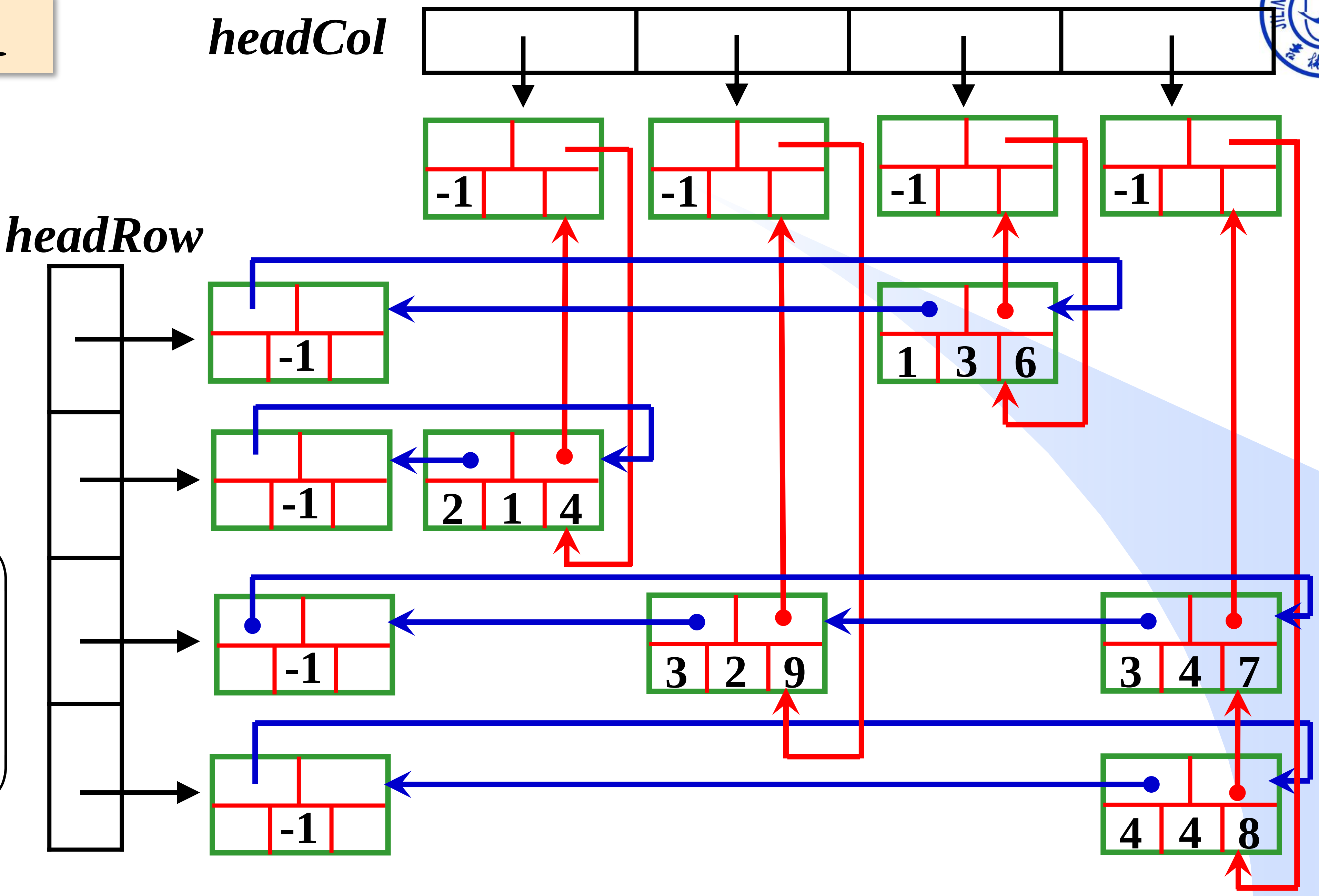
$$
\begin{array}{c|cccc}
 & 1 & 2 & 3 & 4 \\
\hline
1 & 0 & 0 & 6 & 0 \\
2 & 4 & 0 & 0 & 0 \\
3 & 0 & 9 & 0 & 7 \\
4 & 0 & 0 & 0 & 8
\end{array}
$$

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

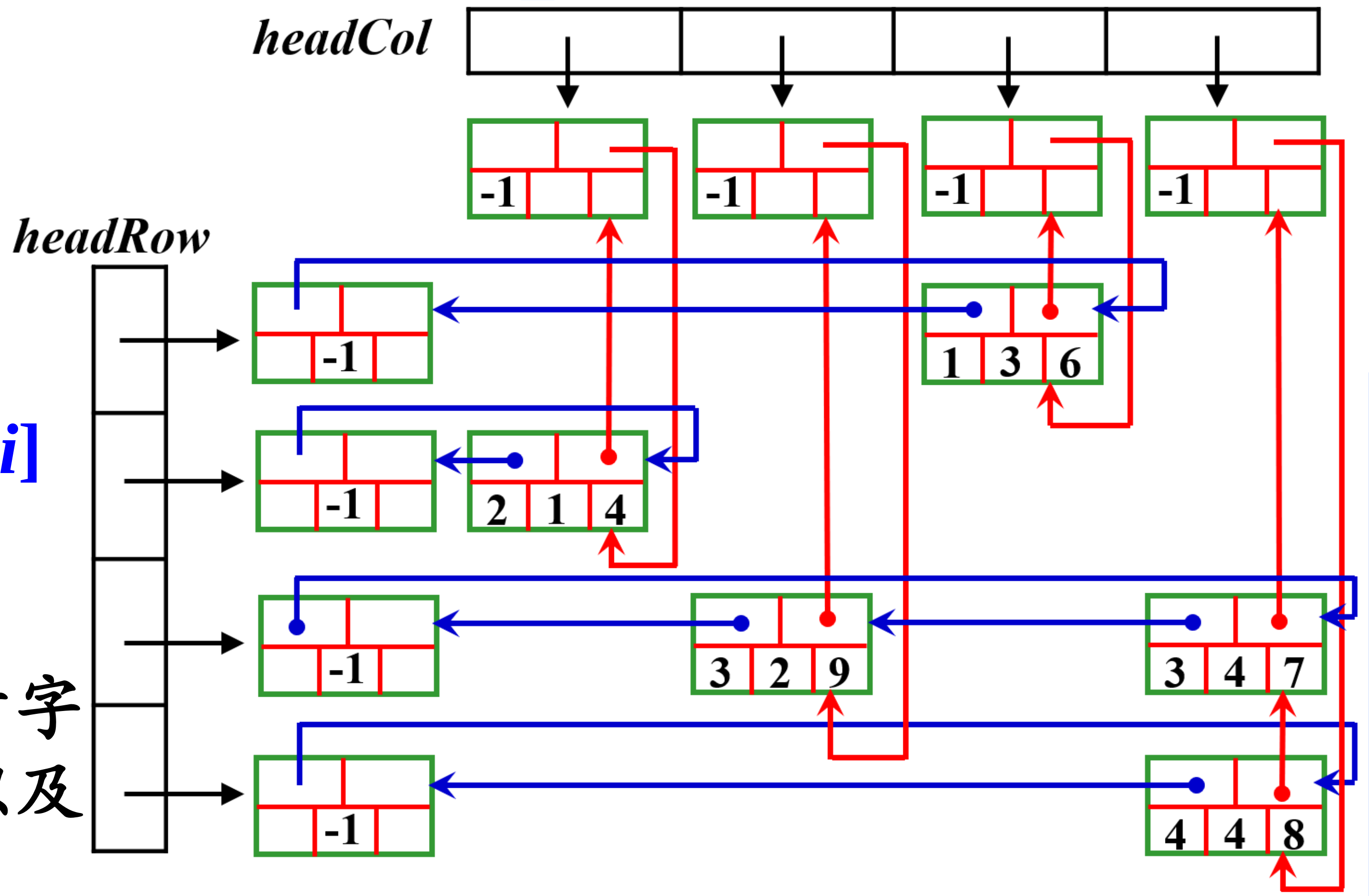


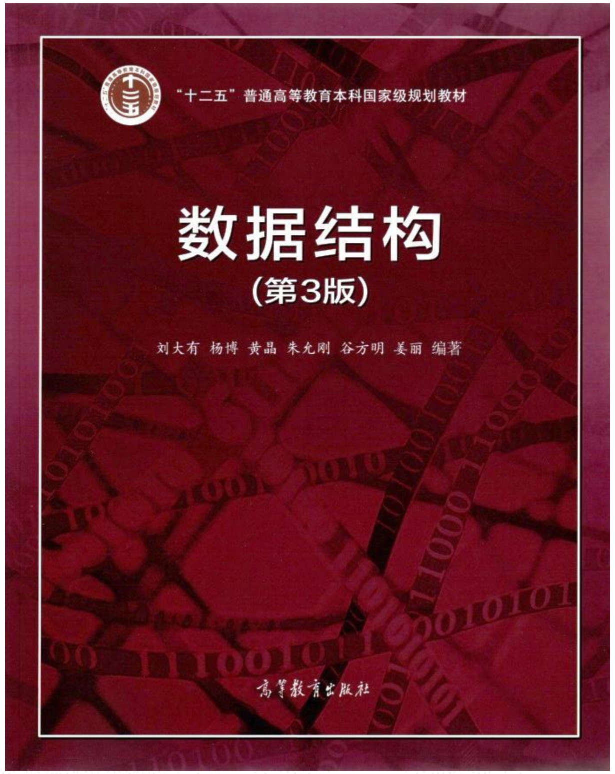
$$
\begin{array}{c c}
 & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{pmatrix} 0 & 0 & 6 & 0 \\ 4 & 0 & 0 & 0 \\ 0 & 9 & 0 & 7 \\ 0 & 0 & 0 & 8 \end{pmatrix}
\end{array}
$$

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

- 数组存储与寻址补充
- 特殊矩阵的压缩存储
- 三元组表
- 十字链表
- **动态规划初探**
- 前缀和与差分数组
- 尺取法
- 其他问题选讲

数据之法
结构之美
算法之道

Last updated on 2023.9

zhuyungang@jlu.edu.cn

动态规划

[例] 计算斐波那契数列*Fib(n)*

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

```
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

```
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);
}
```

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

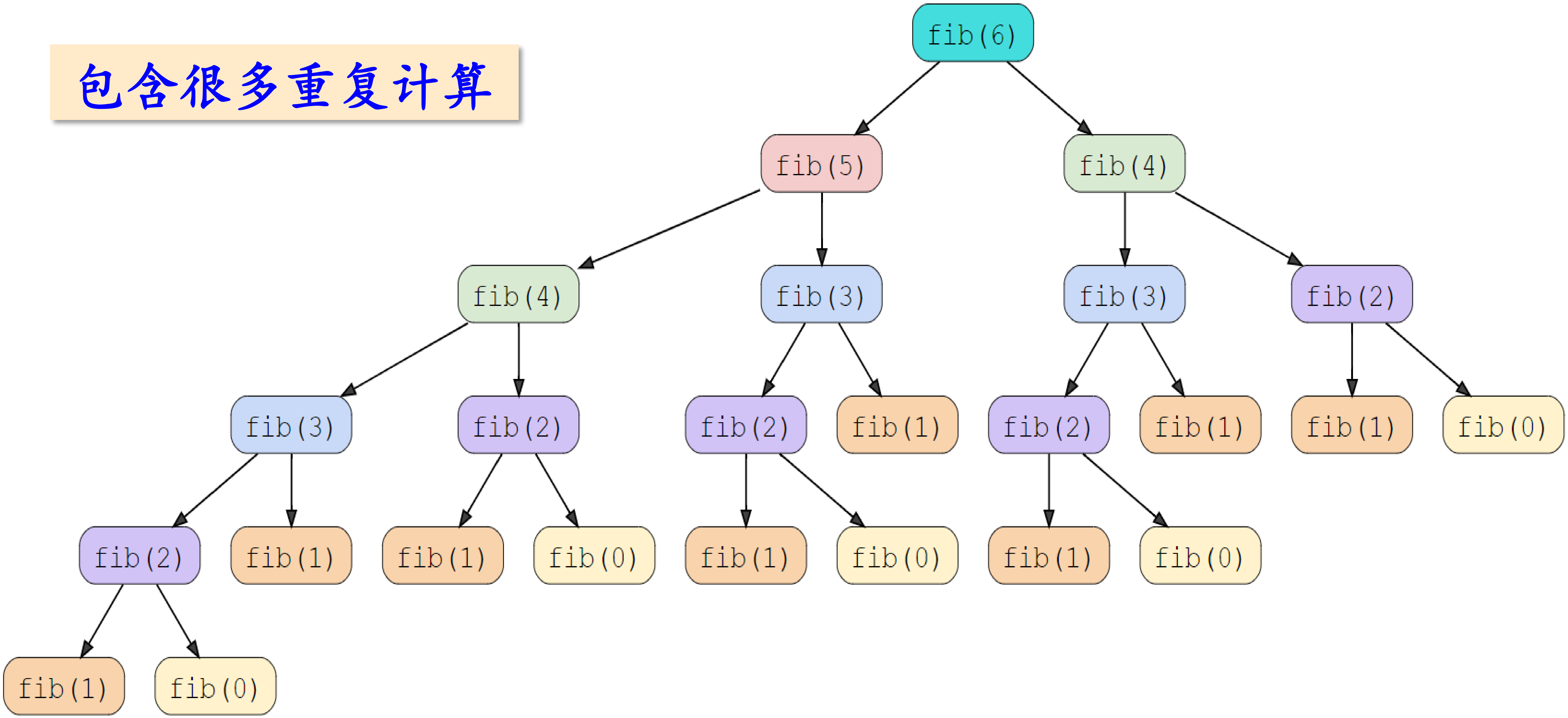
T(*n*-1)=T(*n*-2)+T(*n*-3)+1

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

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

时间效率的原因

包含很多重复计算



动态规划基本思想

- 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。

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];
}
```

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

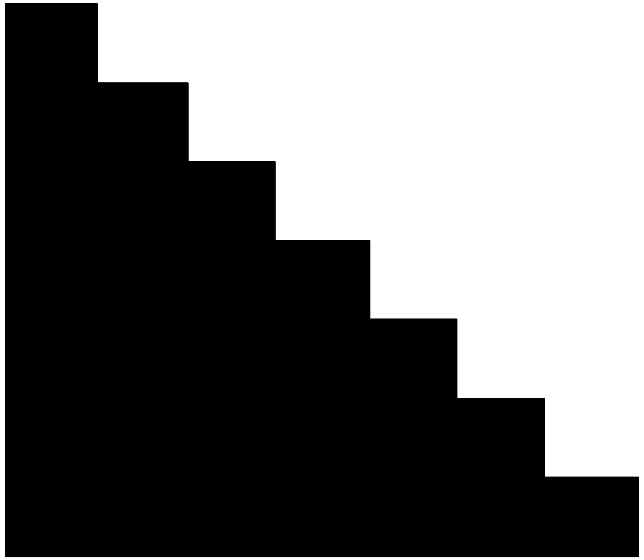
从前往后
当算到F[*i*]时，F[*i*-1]和F[*i*-2]已经算完了

时间复杂性 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

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。

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