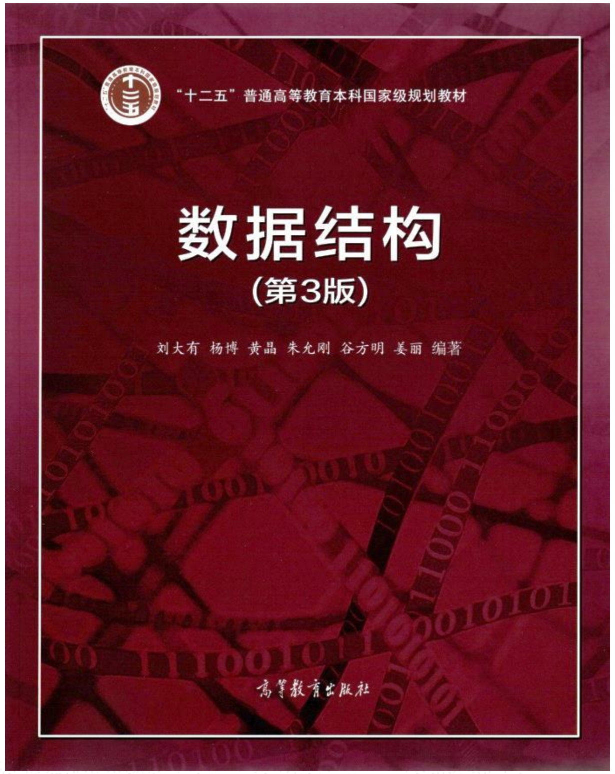


数组与矩阵

- 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

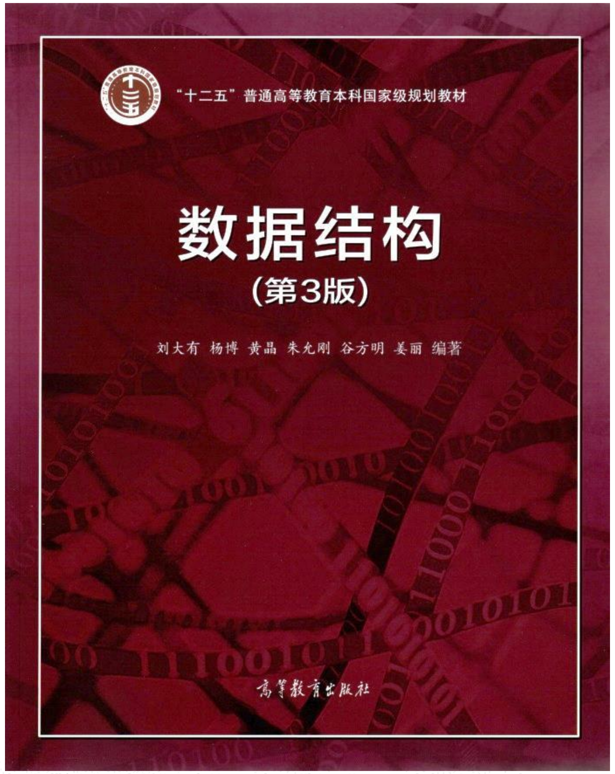
Last updated on 2022.9

朱允刚
zhuyungang@jlu.edu.cn

不要在该奋斗的年纪选择安逸

慕课自学内容

- 数组的存储和寻址
- 一维数组类
- 矩阵类



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 2022.9

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：

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

例子

- 已知数组A[3][5][11][3]

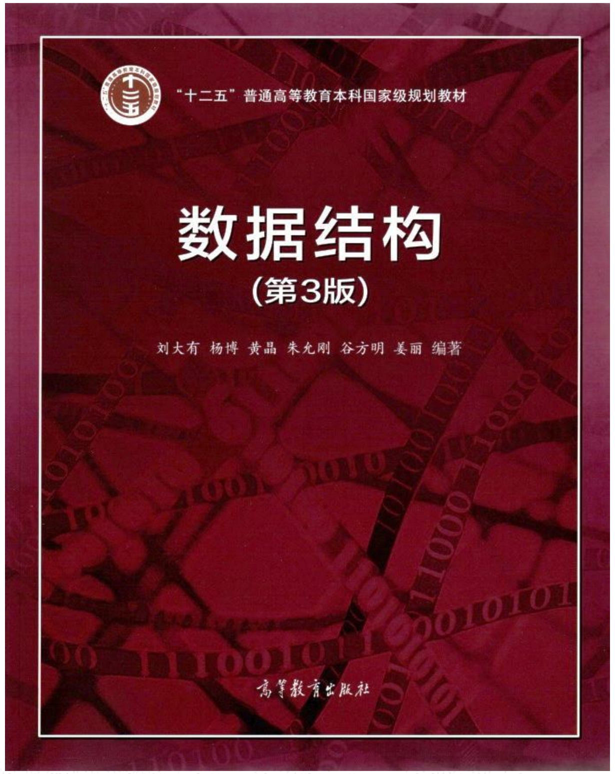
- 给出按行优先存储下的A[I][J][K][L]地址计算公式
- Loc(A)+(I*5*11*3+J*11*3+K*3+L)*C
- =Loc(A)+(165I+33J+3K+L)*C

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______.

$$
\begin{aligned}
&\mathbf{Loc(A[1][2][6][1])}\\
&\mathbf{=1000+4*(165}\boldsymbol{i}\mathbf{+33}\boldsymbol{j}\mathbf{+3}\boldsymbol{k}\mathbf{+}\boldsymbol{l}\mathbf{)}\\
&\mathbf{=1000+4*(165*1+33*2+3*6+1)}\\
&\mathbf{=1000+4*250}\\
&\mathbf{=2000}
\end{aligned}
$$

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

1、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≠j (1≤ i, j ≤ 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：矩阵的任意一个元素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

1、对角矩阵的压缩存储

- 采用一维数组d[n]来压缩存储对角矩阵，其中d[i-1]存储M(i, i)的值。

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

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

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

2、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**.

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

2、三角矩阵的压缩存储

方阵M是下三角矩阵，当且仅当$i<j$时有$M(i,j)=0$.

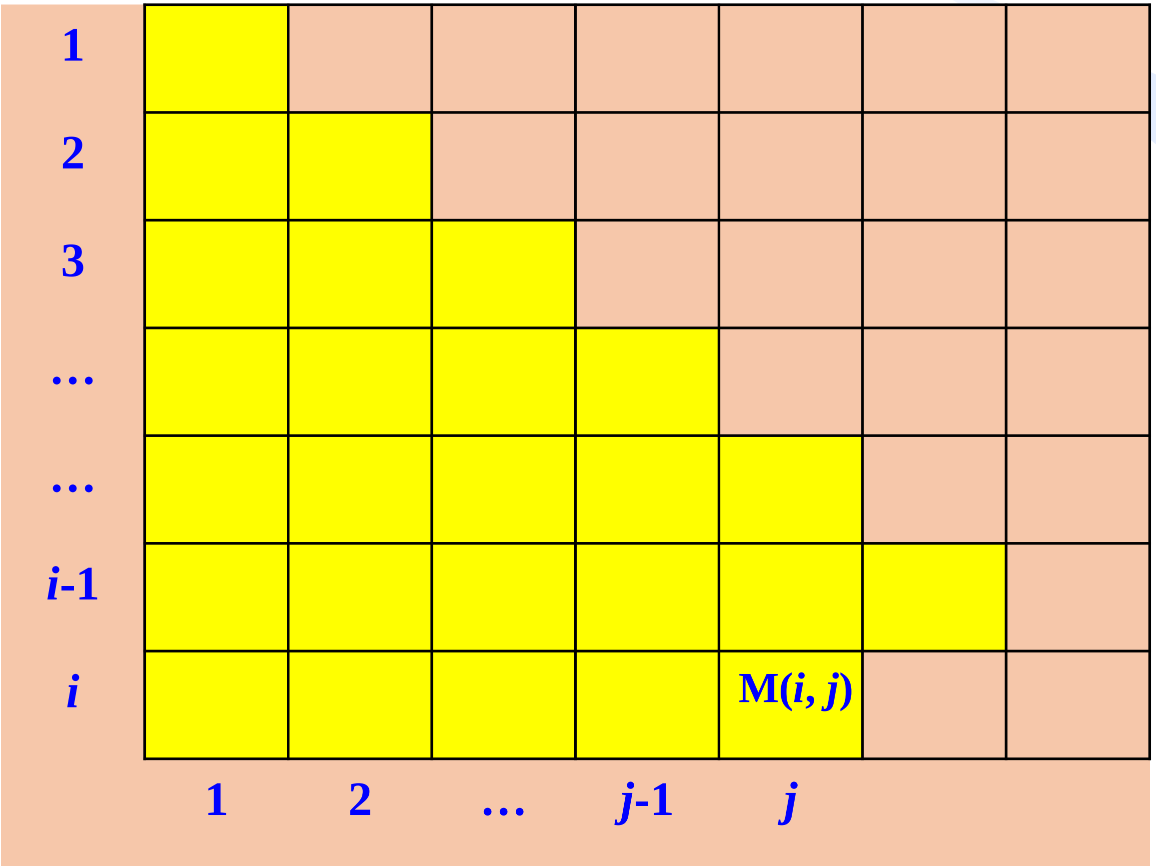
$$\mathbf{L}=\begin{bmatrix} l_{1,1} & & \cdots & & 0 \\ l_{2,1} & l_{2,2} & & (0) & \\ l_{3,1} & l_{3,2} & \ddots & & \vdots \\ \vdots & \vdots & \ddots & \ddots & \\ l_{n,1} & l_{n,2} & \ldots & l_{n,n-1} & l_{n,n} \end{bmatrix}$$

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？

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

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



- **设元素M(*i*, *j*)前面有*k*个元素，可以计算出**
- $k = 1+2+\ldots+(i-1)+(j-1) = i(i-1)/2+(j-1)$
- **M(*i*, *j*)映射到d中所对应的元素是d[*k*].**
- $M(i, j) = d[k] = d[i(i-1)/2+(j-1)]$

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

3. 对称矩阵M的压缩存储方法：

- 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？

➢ **$i \ge j$，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*≥*j*)，和下三角矩阵压缩存储的映射公式一样

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

对于上三角元素M(*i*, *j*) (*i*<*j*)，元素值与下三角中的M(*j*, *i*)相同

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

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，
即总第51个元素，故M(6, 6) =N[50]

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]

4. 三对角矩阵M的压缩存储方法：

➢ 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。

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

4. 三对角矩阵M的压缩存储方法：

$$\begin{bmatrix} a_{1,1} & a_{1,2} & & & & \\ a_{2,1} & a_{2,2} & a_{2,3} & & 0 & \\ & a_{3,2} & a_{3,3} & a_{3,4} & & \\ & & \ddots & \ddots & \ddots & \\ & 0 & & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ & & & & a_{n,n-1} & a_{n,n} \end{bmatrix}$$

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

M(*i*, *j*)=d[*k*],

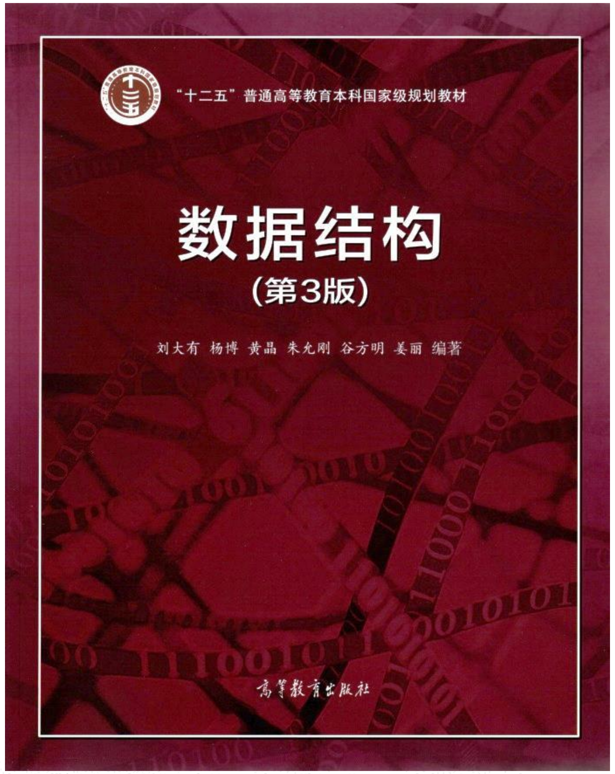
$$k = 2+(i-2)*3+(j-i)+1$$

$$=2i+j-3$$

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 2022.9

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
- 压缩作用：仅存储非零元素，节省空间。

➢对于矩阵 A_{m*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[6];
```

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

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

b

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

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

时间复杂度O(*nt*)

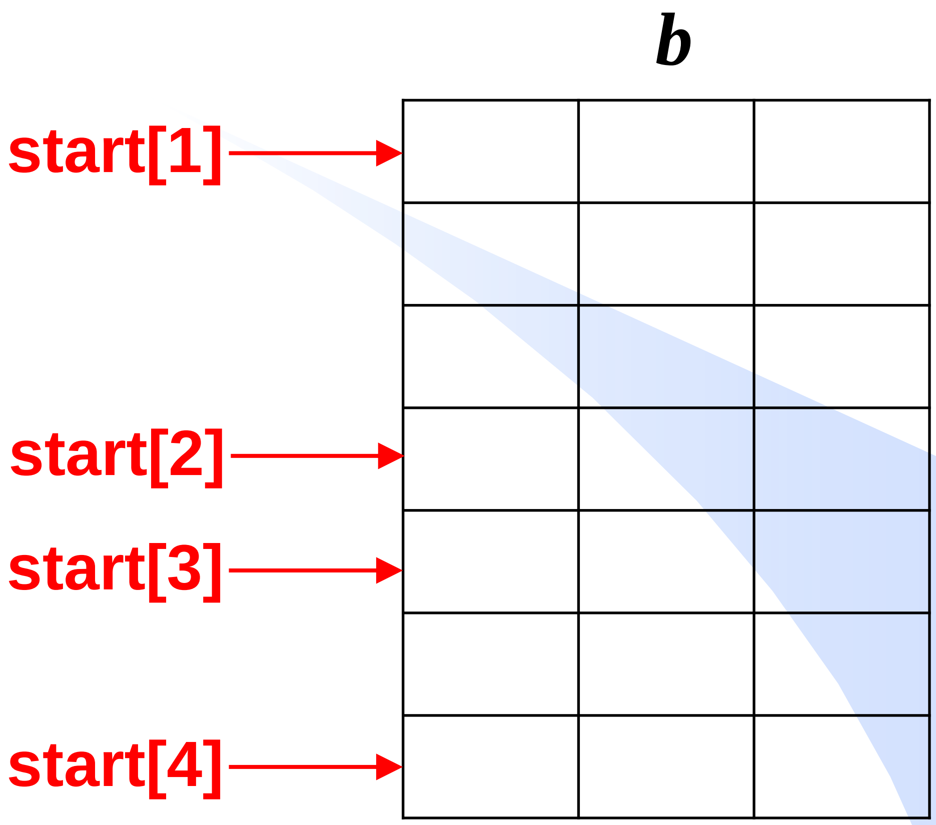
```
void Transpose(Triple a[], int m, int n, int t, Triple b[]) {
//将三元组表a表示的m行n列矩阵转置，保存在三元组表b中，a中非0元素个数为t
    int j = 0;                  //目的是填三元组b，j标识填b的第几位
    if (t == 0) return;         // a空
    for (int k = 1; k <= n; k++)  //填转置后的矩阵的第k行的元素
        for (int i = 0; i < t; i++)//扫描矩阵a,看哪些元素列号是k
            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
            }
}
```

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。

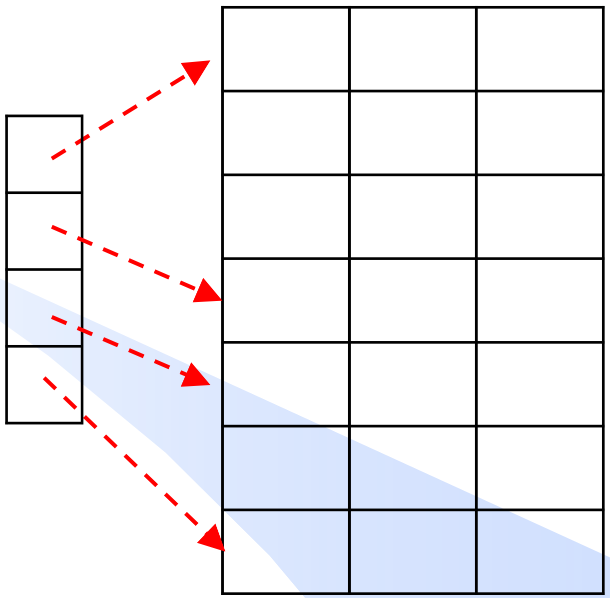
```
for (i = 1; i <= n; i++)
     rowsize[i] = 0;

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]++;
}
```

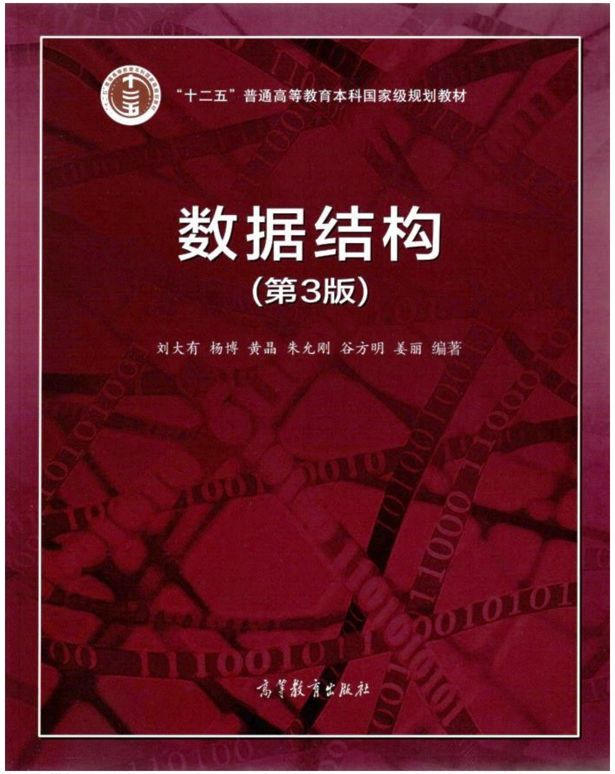


1	1	50
2	1	10
2	3	20
3	2	5
4	1	-30
4	3	6
4	4	5

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

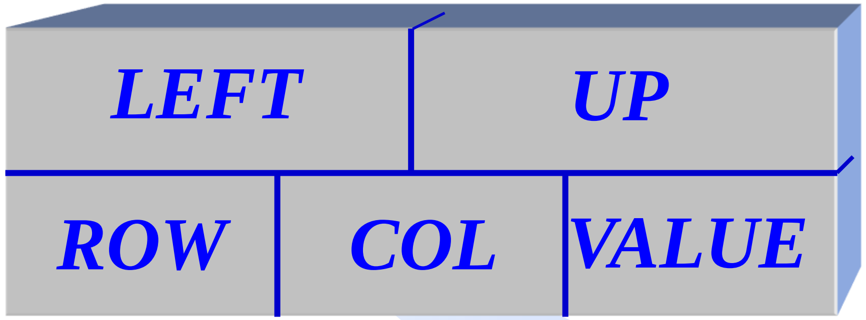
数据之法
结构之美
算法之道

Last updated on 2022.9

zhuyungang@jlu.edu.cn

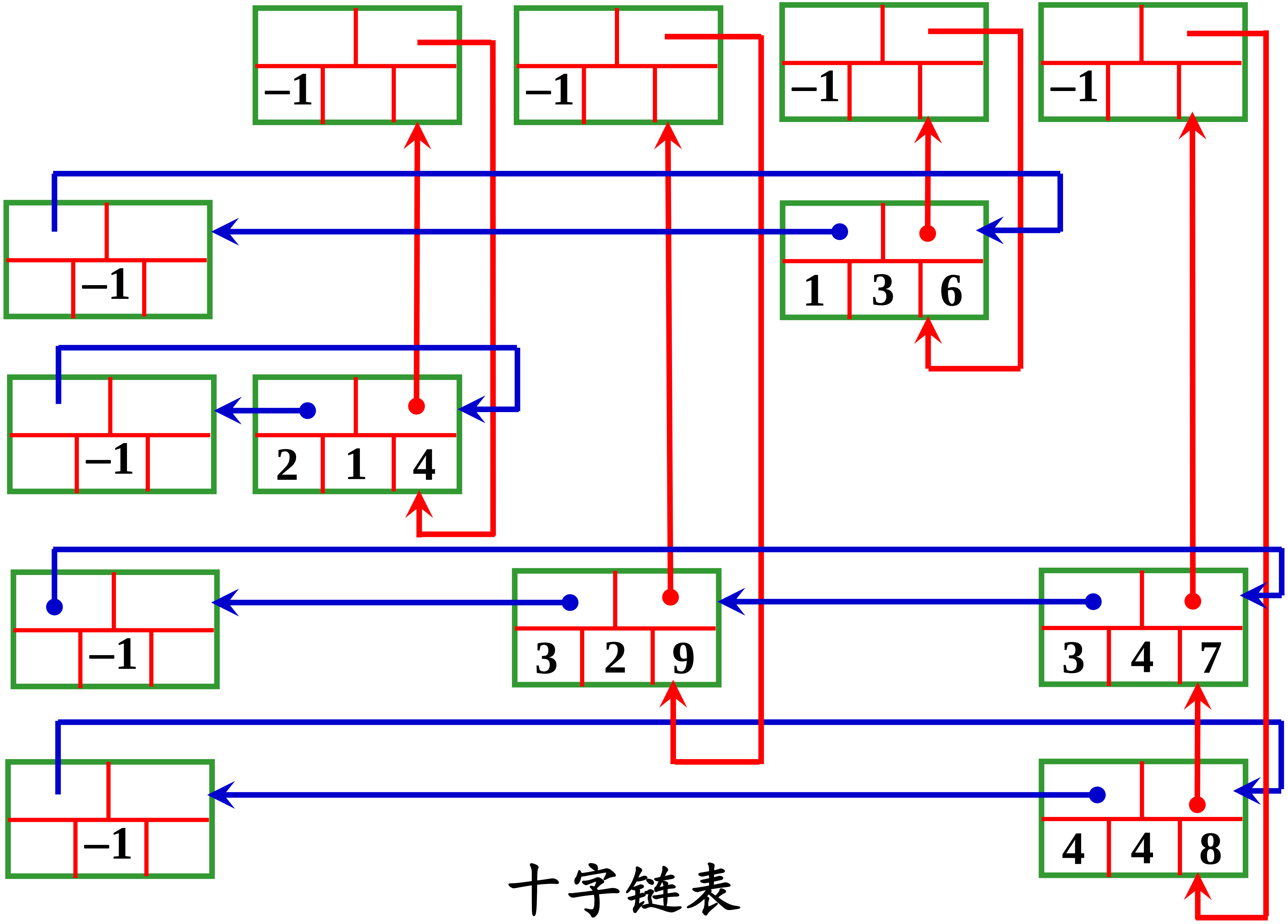
十字链表

矩阵的元素结构如右：分别表示该元素的左邻非零元素、上邻非零元素、所在的行、所在的列和它的值。



例：稀疏矩阵

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



十字链表

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

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，

- 每一行的哨位结点：

  BaseRow[i]， i=1, 2, …, m（m行）

- 每一列的哨位结点：

  BaseCol[j]， j=1, 2, …, n（n列）

LEFT		UP
ROW	COL	VAL

- 并且各行和各列的表头结点有如下特点：

  COL(BaseRow[i]) = -1

  ROW(BaseCol[j]) = -1

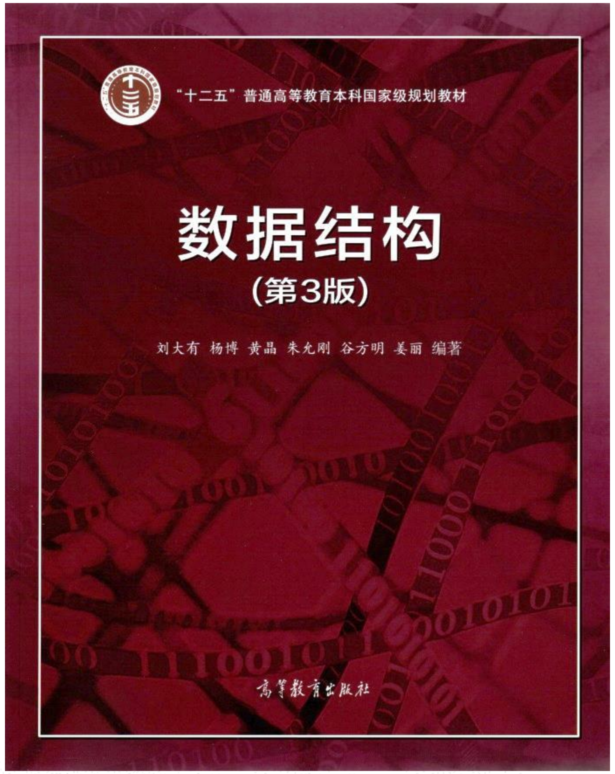
若某一行没有非零元素，则

LEFT(BASEROW[i])=BASEROW[i]

若某一列没有非零元素，则

UP(BASECOL[i])=BASECOL[i]

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 2022.9

zhuyungang@jlu.edu.cn

用数组或矩阵消除递归：动态规划

[例] 计算斐波那契数列*Fib(n)*

$$Fib(n)=\begin{cases} n, & n=0,1 \\ Fib(n-1)+Fib(n-2), & n>1 \end{cases}$$

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

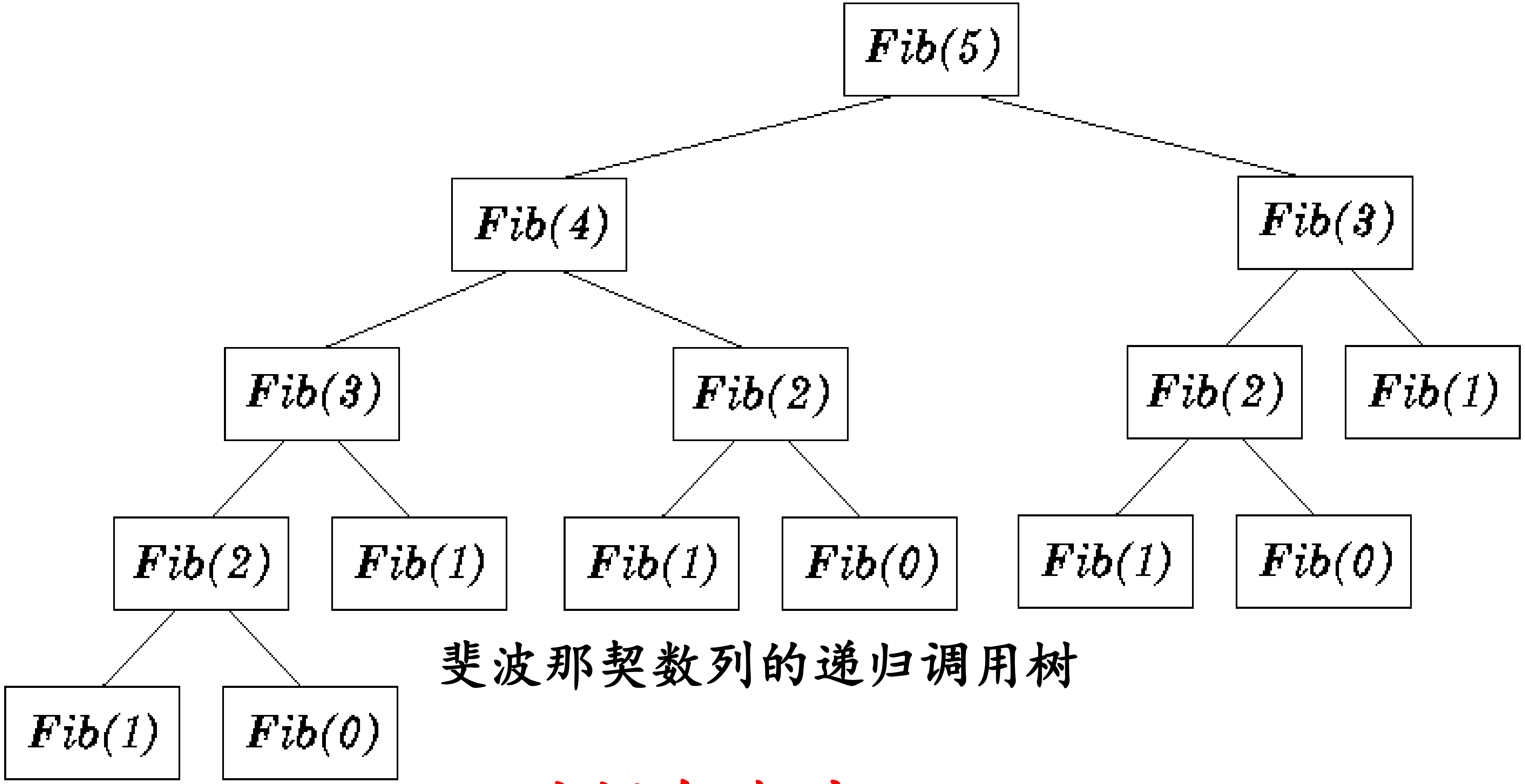
$T(n)=T(n-1)+T(n-2)+1$, $T(n-1)=T(n-2)+T(n-3)+1$

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

$$
\begin{aligned}
T(n) &\ge 2T(n-2) \\
&\ge 2^2T(n-4) \\
&\ge 2^3T(n-6) \\
&\ge 2^4T(n-8) \\
&\cdots \\
&\ge 2^kT(n-2k) \\
&\cdots \\
&\ge 2^{(n-1)/2}T(1) \\
&\ge 2^{(n-1)/2}
\end{aligned}
$$

$T(n)=\Theta(2^n)$

$\boxed{Fib(5)}$



斐波那契数列的递归调用树

时间复杂度$O(2^n)$

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，通过综合子问题的最优解来得到原问题的最优解。
- 将每个求解过的子问题的解记录下来，这样当下一次碰到同样的子问题时，就可以直接使用之前记录的结果，而不是重复计算。
- 往往可以通过“记忆化搜索”和“递推”来实现。

改进1：记忆化搜索

```
int F[N];    //初始值-1
F[0]=0; F[1]=1;
int Fib(int n){
    if(F[n]!=-1) return F[n];
    else F[n]=Fib(n-1)+Fib(n-2);
    return F[n];
}
```

将算过的值保存起来

改进2：递推

$$Fib(n)=\begin{cases} n, & n=0,1 \\ Fib(n-1)+Fib(n-2), & n>1 \end{cases}$$

```
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];
}
```

时间复杂性 O(n)

从前往后
当算到F[i]时，F[i-1]和F[i-2]已经算完了

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

计算斐波那契数列*Fib(n)*

$$Fib(n)=\begin{cases} n, & n=0,1 \\ Fib(n-1)+Fib(n-2), & n>1 \end{cases}$$

递归算法

从后往前

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

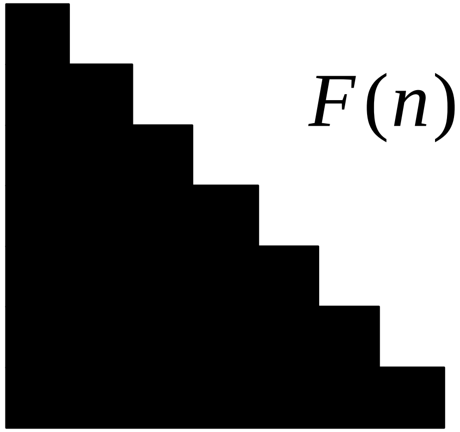
跳台阶问题

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。【华为、字节跳动、腾讯、百度、阿里、小米、拼多多、快手、京东面试题】

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

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。



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

➢如下递归关系：

$$C(n,k) = C(n-1,k) + C(n-1,k-1)$$

递归出口：

➢$C(n,k) = 0$， $k>n$.

➢$C(n,0)=1$.

➢$C(n,n)=1$.

$$C_n^k$$

递归算法

算法C (n, k)

IF $k > n$ **THEN RETURN** 0.

IF $n = k$ **OR** $k = 0$ **THEN RETURN** 1.

RETURN $C(n-1, k) + C(n-1, k-1)$ ▌

$$C(n, k) = C(n-1, k) + C(n-1, k-1)$$

递推算法

$$C(n, k) = C(n-1, k) + C(n-1, k-1)$$

	C[*n*-1][*k*-1]	C[*n*-1][*k*]	
		C[*n*][*k*]	

递推算法

	C[*n*-1][*k*-1]	C[*n*-1][*k*]	
		C[*n*][*k*]	

```
FOR i=0 TO n DO
    FOR j=0 TO k DO
    (     IF j > i THEN C[i][j] ← 0;
          ELSE IF j = 0 THEN C[i][j] ← 1;
          ELSE IF i = j THEN C[i][j] ← 1;
          ELSE C[i][j] ← C[i-1][j-1] + C[i-1][j];
    )
```

空间优化

	C[n-1][k-1]	C[n-1][k]	C[n-1][k+1]
		C[n][k]	C[n][k+1]

```
FOR i=0 TO n DO
    FOR j=0 TO k DO
    (
        IF j > i THEN C[i][j] ← 0;
        ELSE IF j = 0 THEN C[i][j] ← 1;
        ELSE IF i = j THEN C[i][j] ← 1;
        ELSE C[i][j] ← C[i-1][j-1] + C[i-1][j];
    )
```

空间优化

1	2	3	4

空间优化

1	2	3	7

空间优化

1	2	5	7

空间优化

1	3	5	7

空间优化

1	3	5	12

空间优化

1	3	8	12

空间优化（滚动数组）

1	4	8	12

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、苹果、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】

-2	5	3	-6	4	-8	6

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

最大子数组和

算法1：遍历所有子数组。

算法MaxSubArraySum(*a*, *n*. MaxSum)

MaxSum ← -∞.

FOR *i* =0 **TO** *n*-1 **DO**

  FOR *j*=*i* **TO** *n*-1 **DO**

  (sum ← 0.

    FOR *k*=*i* **TO** *j* **DO** sum ← sum+ *a*[*k*].

    IF sum>MaxSum **THEN** MaxSum ← sum.

) ▌

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。

算法MaxSubArraySum(*a*, *n*. MaxSum)

MaxSum ← -∞.

FOR *i* =0 **TO** *n*-1 **DO**

(　　sum ← 0.

　　FOR *j*=*i* **TO** *n*-1 **DO**

　　(　　sum ← sum+ *a*[*j*].

　　　　IF sum>MaxSum **THEN** MaxSum ← sum.

　　)

) ▌

时间复杂度$O(n^2)$

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

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			

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。即以位置i结尾的最大子数组只有两种可能情况：

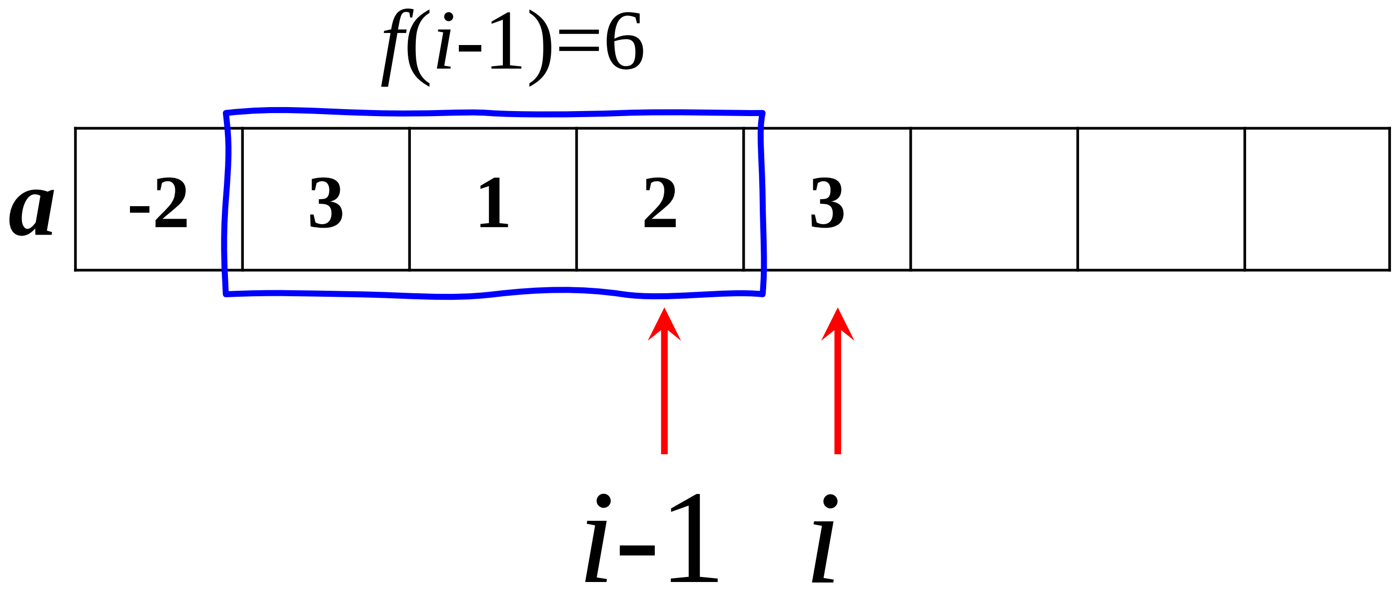
- ➢ 该子数组包含$a[i]$及其前面的若干元素，即以位置i-1结尾的最大子数组以及$a[i]$，此时最大和为$f(i-1)+a[i]$。
- ➢ 该子数组只有一个元素$a[i]$，最大和即$a[i]$。

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

最大子数组和

情况1：*f*(*i*-1)>0，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*]。

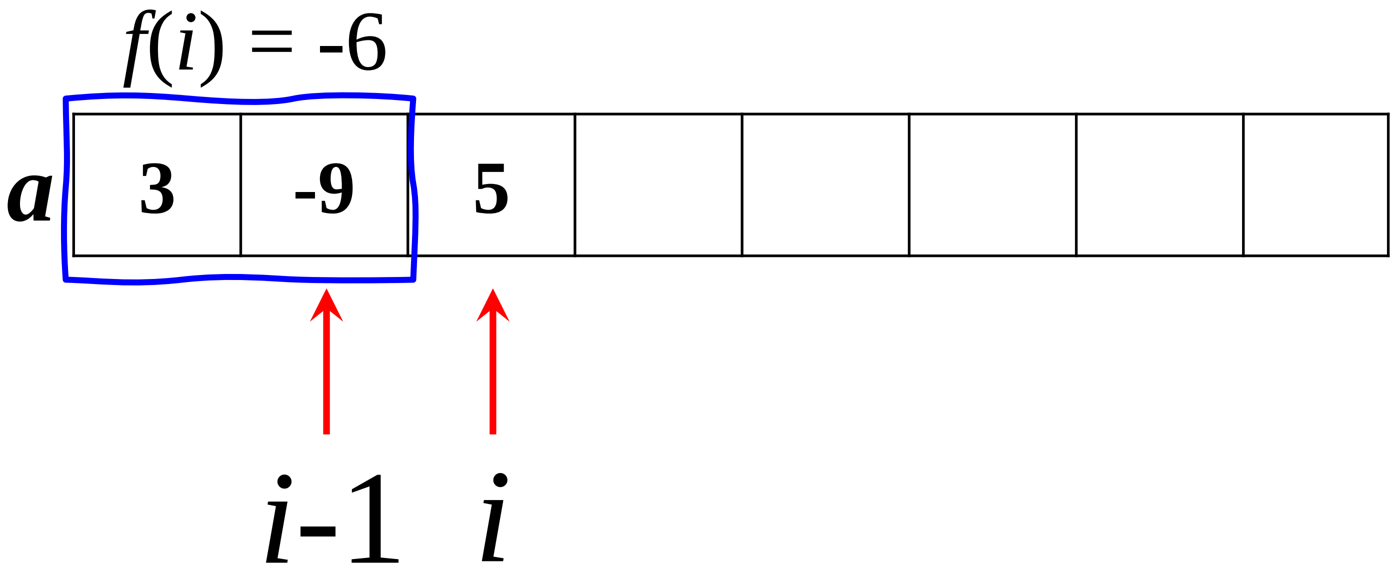
$$f(i)=f(i-1)+a[i],\qquad f(i-1)>0$$



最大子数组和

情况2： $f(i\text{-}1) \le 0$，则$a[i] + f(i\text{-}1) \le a[i]$，此时以位置$i$结尾的最大子数组就是$a[i]$自己。

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



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			

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

算法MaxSubArraySum(*a*, n. MaxSum)

时间复杂度O(n)
空间复杂度O(n)

f[0] ← *a*[0]. MaxSum ← *a*[0].
FOR *i* =1 TO *n*-1 DO
(　　IF f[*i*-1] > 0 THEN f[*i*] ← f[*i*-1]+ *a*[*i*].
**　　ELSE f[*i*] ← *a*[*i*].**
**　　IF f[*i*]>MaxSum THEN MaxSum ← f[*i*].**
) ∎

最大子数组和

算法4：用变量*f*代替数组*f*[*i*]。

			…	***a*[*i*]**			
				i			

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

算法MaxSubArraySum(*a*, *n*. MaxSum)

f **← *a*[0]. MaxSum ←*a*[0].**

FOR *i* =1 TO *n*-1 DO

(IF *f* > 0 THEN *f* ← *f*+*a*[*i*].

ELSE *f* ← *a*[i].

IF *f* > MaxSum THEN MaxSum ← *f*.

) ■

时间复杂度O(n)
空间复杂度O(1)

课下思考

不仅求最大子数组的和，还要求出该子数组（即该子数组开始位置和结束位置的下标）。

			…	*a*[*i*]			

i

课下思考

拓展：求最大子数组之和，同时输出该子数组的开始位置和结束位置的下标。

			…	*a*[*i*]			

i

算法MaxSubArraySum(*a*, n. MaxSum, MaxStart, MaxEnd)

f ← *a*[0]. MaxSum ←*a*[0]. start ← 0. end←0. MaxStart←0. MaxEnd←0.

FOR i =1 TO n-1 DO

(IF f > 0 THEN f ← f+*a*[i]. //起点是f(i-1)的起点

ELSE (f ← *a*[i]. start ←i.)

end ← i.

IF f>MaxSum THEN

(MaxSum←f. MaxStart←start. MaxEnd←end.)

)

最大子数组乘积

拓展：求最大子数组乘积。【阿里、百度、腾讯、美团、快手、京东、华为、新浪、字节、微软、谷歌面试题】

-2	4	0	3	0.5	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。**

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

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

最大子数组乘积

			…	***a*[*i*]**			

i

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

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

算法MaxSubArrayProduct (*a*, *n*. MaxProduct)

$f \leftarrow g \leftarrow a$[0]. MaxProduct $\leftarrow a$[0].

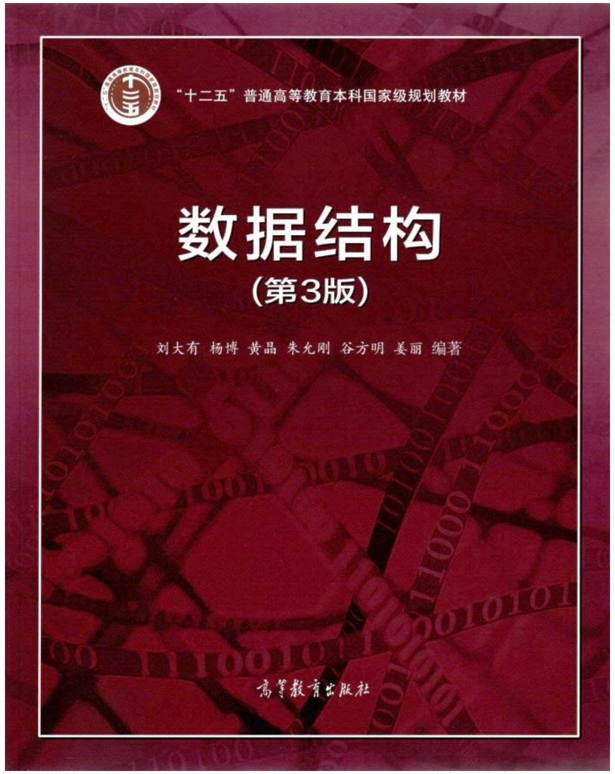
FOR *i* =1 TO *n*-1 DO

(　　$f \leftarrow$ max{ f* a[i], g*a[i], a[i] }.

**　　$g \leftarrow$ min{ f* a[i], g*a[i], a[i] }.**

**　　IF f > MaxProduct THEN MaxProduct $\leftarrow f$.**

) ▌



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 2022.10

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【2019级数据结构上机考试题】。

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

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。

暴力方法：

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

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)

➢ $sum_i = a_1+a_2+\ldots+a_{i-1}+a_i$

$= sum_{i-1}+a_i$

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

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。

```
const int N=1e5+10;
void IntervalSum(int a[],int n,int m){
    int sum[N],i,j;
    sum[0]=0;
    for(int i=1;i<=n;i++)
        sum[i]=sum[i-1]+a[i];
    while(m--){
        scanf("%d %d",&i,&j);
        printf("%d\n",sum[j]-sum[i-1]);
    }
}
```

O(n)

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。

例：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。输出最后的数组。

示例：

数组a：　6 10 20 29 35 50 60 70 80 90

操作

2 5 8　　6 10 28 37 43 58 60 70 80 90

3 6 10　　6 10 28 47 53 68 70 70 80 90

1 7 2　　6 12 30 49 55 70 72 72 80 90

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;
}
```

O(nm)

差分数组 $diff[i]=a[i]-a[i-1]$

	0	1	2	3	4	5	6	7
a	1	2	5	6	9	8	10	7
diff								

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

	0	1	2	3	4	5	6	7
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+1<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*，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*。

```
const int N=1e5+10;
void IntervalIncrement(int a[],int n,int m){
    int diff[N],i,j;
    diff[0]=a[0];    //计算差分数组
    for(int i=1;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+1<n) diff[j+1]-=d;
    }
    a[0]=diff[0];    //利用diff反推（恢复）数组a
    for(int i=1; i<n; i++) a[i]=a[i-1]+diff[i];
}
```

O(n)

差分数组技巧适用场合：频繁对数组的某个区间的元素进行增减。

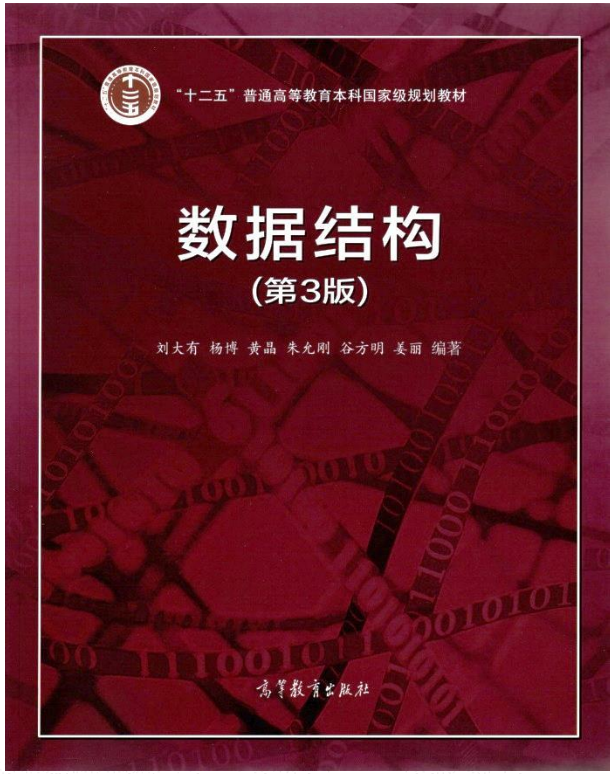
例：有*n*个航班，分别从1到*n*进行编号。有一份航班预订表bookings，表中第 *k* 条预订记录 bookings[*k*] = [*i*, *j*, *d*] 意味着在从 *i* 到 *j* 的每个航班上预订了 *d* 个座位。请你返回一个长度为 *n* 的数组ans，里面的元素是每个航班预定的座位总数。【华为、字节跳动、支付宝、微软、苹果、谷歌面试题】。

示例输入：bookings = [[1,2,10],[2,3,20],[2,5,25]], n = 5

示例输出：[10, 55, 45, 25, 25]

解释：

航班编号	1	2	3	4	5
预订记录 1 :	10	10			
预订记录 2 :		20	20		
预订记录 3 :		25	25	25	25
总座位数:	10	55	45	25	25



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 2022.9

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

$\underbrace{2\quad 7}_{\text{找到}}$

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

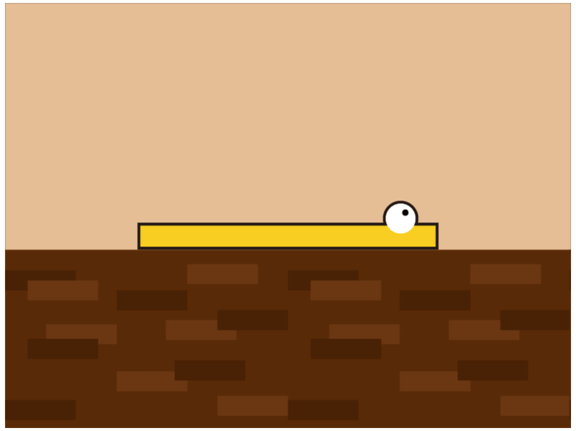
5	2	3	6	2	7	1	5	3	2

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。

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】

示例：

输入：a=[2 3 1 1 4 3]，S=6

输出：2

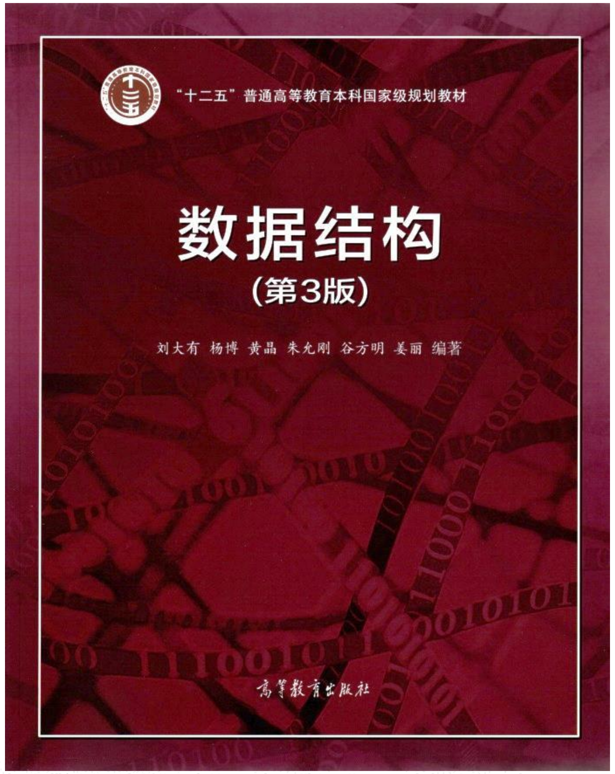
```
int MinInterval(int a[], int n, int S){
    int s=0, t=0, sum=a[0], min=n+1;
    while(true){
        while(t<n-1 && sum<S)
            sum+=a[++t];
        if(sum<S) break;
        if(t-s+1<min) min=t-s+1;
        sum-=a[s++];
    }
    if(min==n+1)  min=0;
    return min;
}
```

如果**sum>=S**，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

如果允许数组a中有负数，找和大于等于S的最短区间，该问题还有O(n)的算法么？【难度级别★★★★☆ 】

提示：前缀和+单调递增队列



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
- **子集生成**

数据之法
结构之美
算法之道

Last updated on 2022.9

zhuyungang@jlu.edu.cn

子集生成

求一个集合的幂集，如集合{a, b, c, d}的幂集是

```
{ }
{ 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 }
```

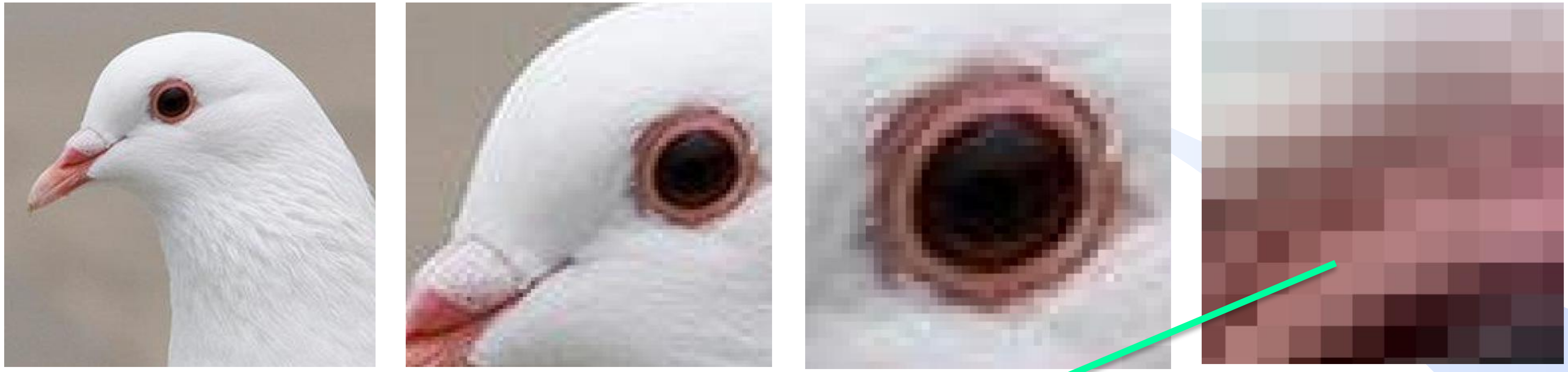
a	b	c	d

```cpp
void addone(int a[],int n)
{
	int i=0;
	while(a[i]!=0)
	{
		a[i]=0;
		i++;
	}
	a[i]=1;
}
```

```cpp
int main()
{
    char s[4]={'a','b','c','d'};
    powerset(s,4);
    return 0;
}
```

```cpp
void powerset(char s[],int n)
{
    int i;
    int a[100]={0};
    while (a[n]==0)
    {
        cout<<"{ ";
        for(i=0;i<n;i++)
        {
            if(a[i]==1) cout<<s[i]<<' ';
        }
        cout<<'}'<<endl;
        addone(a,n);
    }
}
```

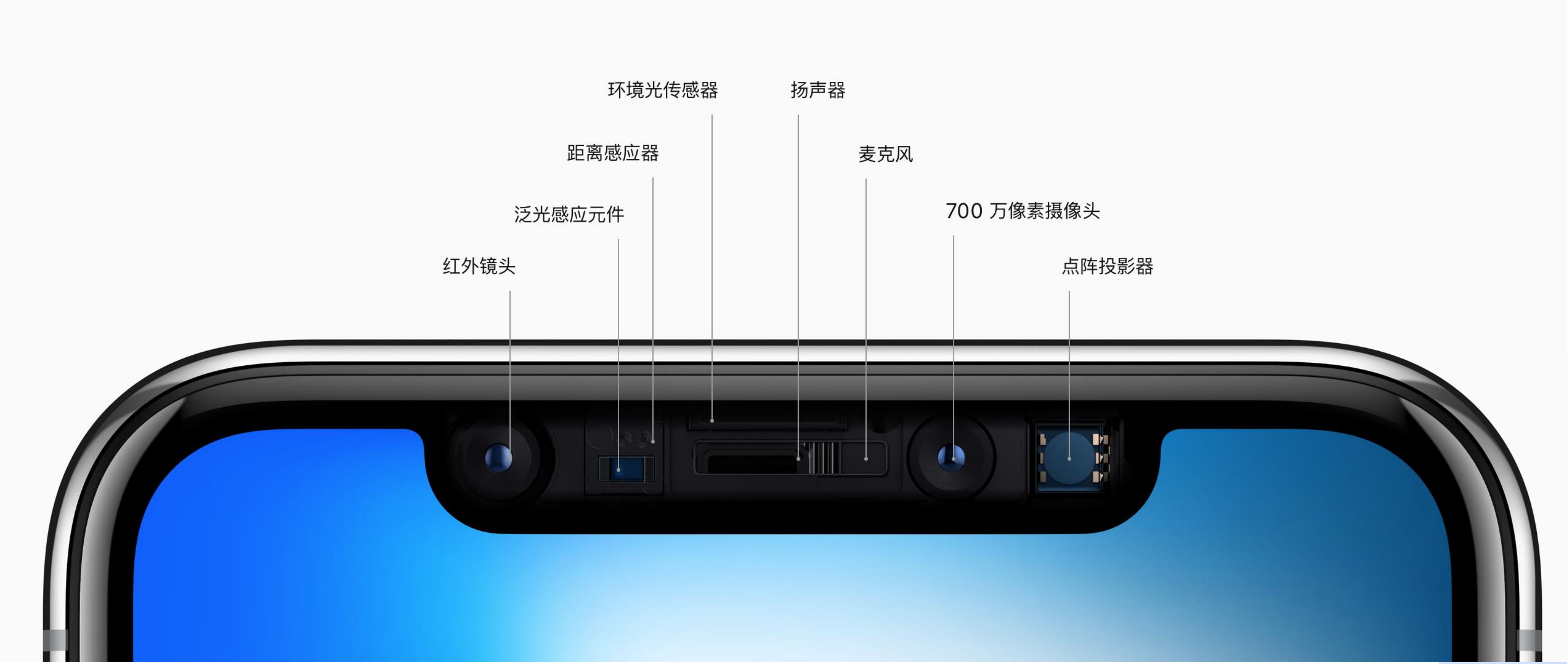
矩阵应用-图像处理



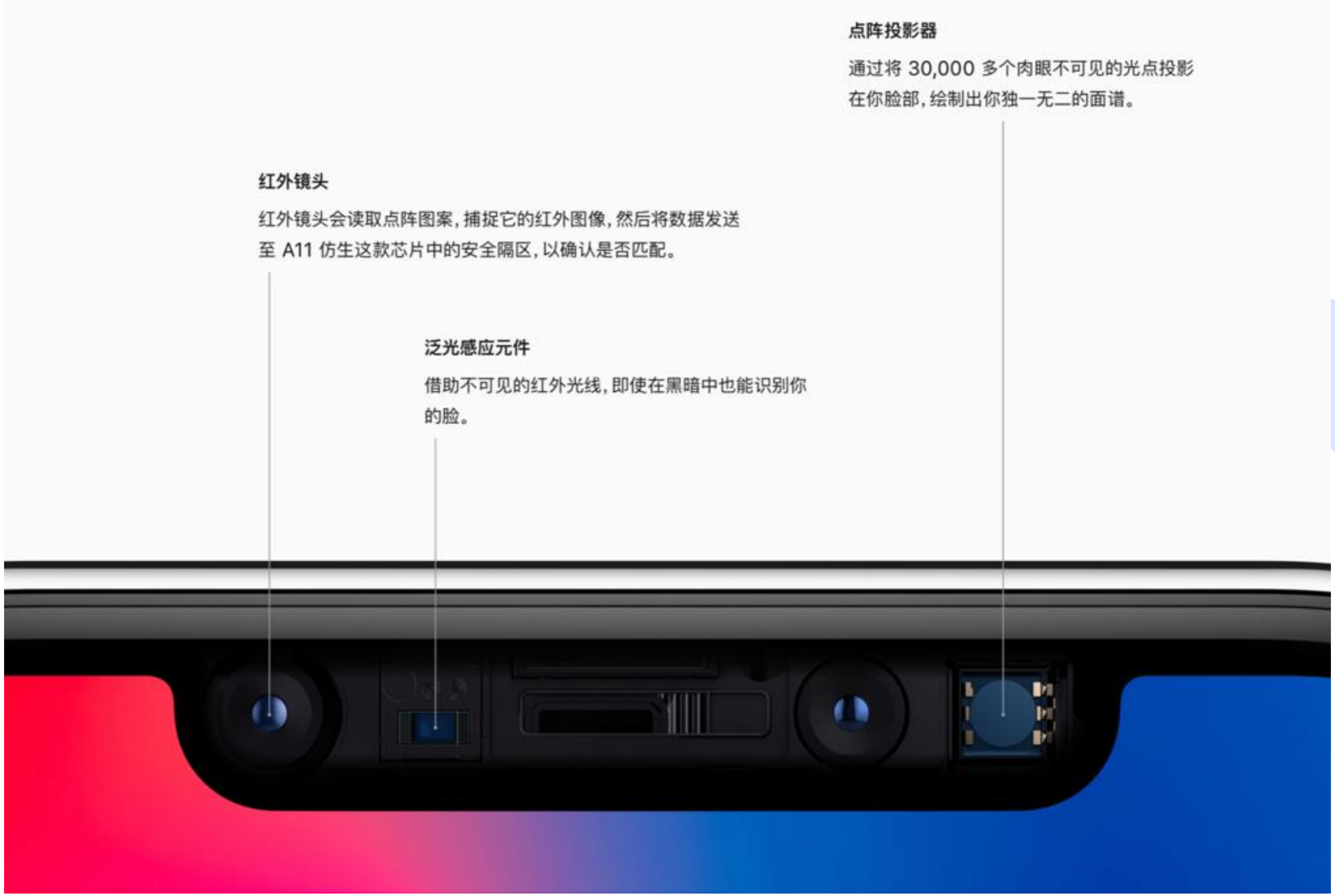
$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & a_{m,3} & \cdots & a_{m,n} \end{bmatrix}$$

黑白图像：灰度值
彩色图像：**RGB**值
景深信息：存储像素点距离摄像头的距离

iPhone Face ID



iPhone Face ID

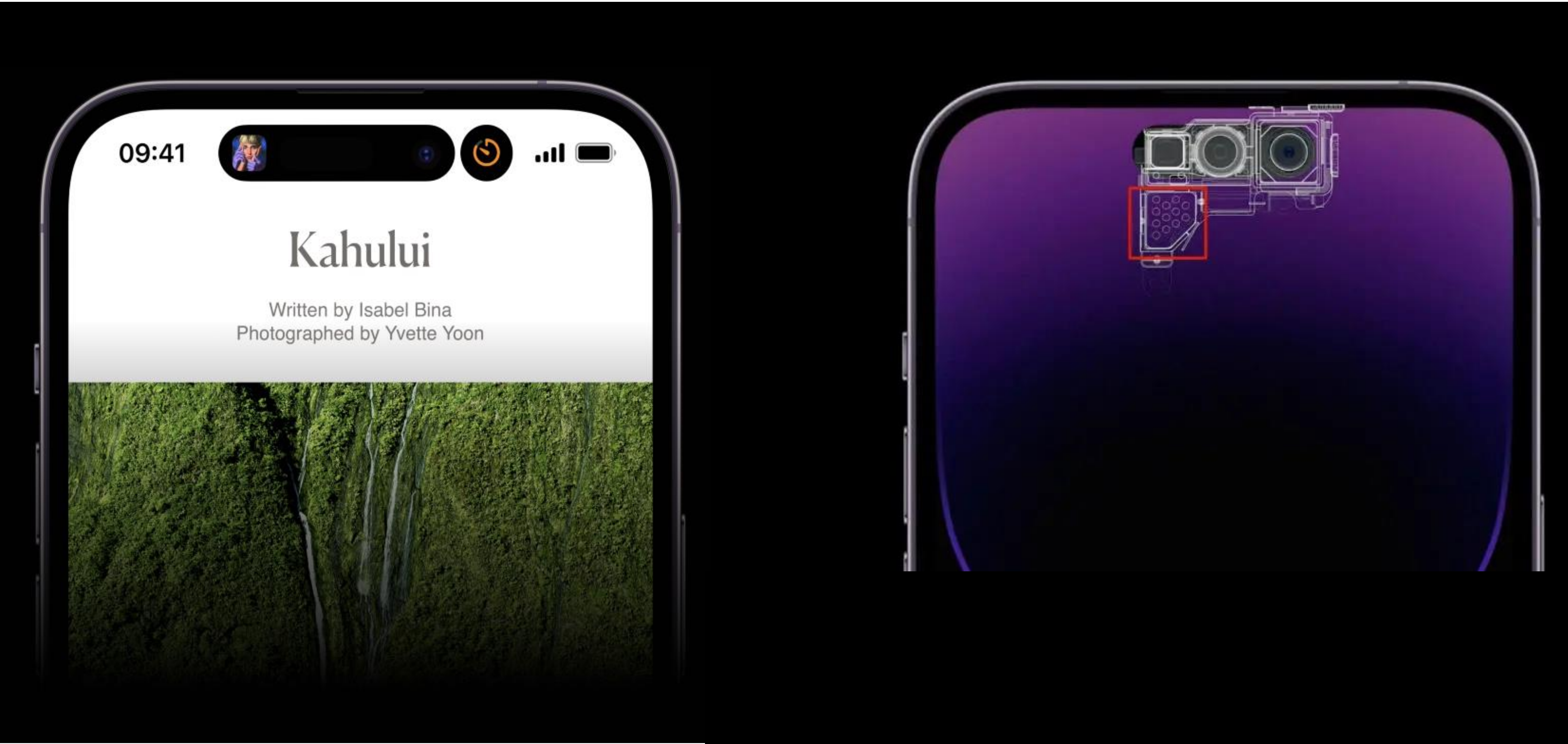


iPhone Face ID

iPhone 14 Pro

灵动岛 Dynamic Island

iPhone 14 Pro




吉林大学
JILIN UNIVER


鼎新圖書館

图像淡入淡出效果

A1

A2

令$\mathbf{A}(t) = (\mathbf{1}-t)*\mathbf{A1} + t*\mathbf{A2}$

*t*从0变成1的过程中，所生成的一系列矩阵，形成A1到A2的淡入淡出

自愿性质OJ练习题

- ✓ LeetCode 70（跳台阶）
- ✓ LeetCode 53（最大子数组和）
- ✓ HDU 1003（最大子数组和及其起止下标）
- ✓ LeetCode 1109（差分数组）
- ✓ POJ 3061、LeetCode 209（和≥S的最短区间）