一、数组

基本概念

特点：顺序存储，每个元素大小，类型相同，元素有限

高维数组可以转化为一维数组

高维数组存放次序：按行优先或者按列优先

按行优先的寻址公式：

1. 二维数组a[m] [n]:   Loc(a[i] [j]) = Loc(a[0] [0]) + (i*n+j) * C
2. 三维数组a[m] [n] [p]: Loc(a[i] [j] [k]) = Loc(a[0] [0] [0]) + (i*n *p +j * p + k) * C
3. **a[i1] [i2]...[in] =**$\boldsymbol{Loc}(0,\cdots,0)+\{\sum_{k=1}^{n-1}(\boldsymbol{i}_k\times\prod_{p=k+1}^{n}\boldsymbol{m}_p)+\boldsymbol{i}_n\}\times\boldsymbol{C}$

按列优先的寻址公式：

1. 二维数组a[m] [n]:   Loc(a[i] [j]) = Loc(a[0] [0]) + (j*m+i) * C
2. 三维数组a[m] [n] [p]: Loc(a[i] [j] [k]) = Loc(a[0] [0] [0]) + (k*m *n +j * m + i) * C
3. **a[i1] [i2]...[in]=** $=\boldsymbol{Loc}(0,\cdots,0)+\{\sum_{k=1}^{n-1}(\boldsymbol{i}_k\times\prod_{p=k+1}^{n}\boldsymbol{m}_p)+\boldsymbol{i}_n\}\times\boldsymbol{C}$

举例：

A[0:2,0:4,0:10,0:2]，A[I] [J] [K] [L] 地址计算公式

按行优先：

Loc(A)+(165I+33J+3K+L)*C

按列优先：

Loc(A)+ (165L+15K+3J+I)*C

二、矩阵

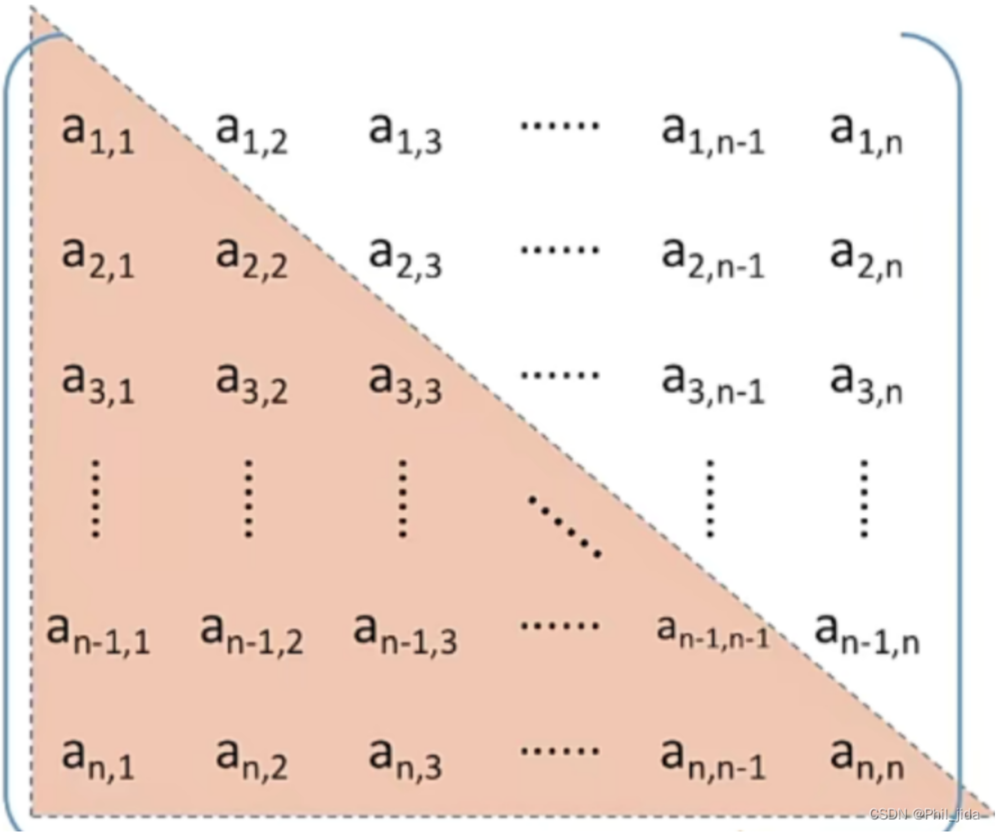
1、矩阵的乘法操作

思路：三重for循环实现

```c
//矩阵乘法
void mul(int a[][maxsize], int b[][maxsize], int ans[][maxsize],int a_m, int a_n,
int b_m, int b_n){//a_m,a_n为a的行数与列数，b_m,b_n为b的行数与列数
    int i,j,k;
    for(i=0; i < a_m; i++){ //三重for循环
        for(j=0; j < b_n; j++){
            for(k=0; k < a_n; k++){
                ans[i][j] += a[i][k] * b[k][j];
            }
        }
    }
} //O(n^2*m)
```

2、特殊矩阵的压缩存储

特殊矩阵：(转化为一阶矩阵存储都是下标从0开始)

1. 对称矩阵的压缩存储　　**(注意是1开头)**
    - 

- **一共N(N+1)/2元素**
- 行优先存储：**掌握自己推导**
    - **i>=j d[k] = i(i-1)/2 + (j-1)** 下三角区域
    - **i < j, d[q] = j(j-1)/2 + (i-1)** 上三角区域

2. 三角矩阵
    - 上三角矩阵 i<j M(i,j) = 0
        - 寻址方式：
            - 行优先：k = n + n-1 + n-i+2 + (j-i)
            - 列优先： k = 1+2+...+(j-1)+(i-1)

    - 下三角矩阵i>j M(i,j) = 0
        - 寻址方式：
            - 行优先：k = 1+2+...+(i-1)+(j-1)
            - 列优先：k = n + n-1 + n-j+2 + (i-j)

3. 对角矩阵
    - 三对角矩阵（带状矩阵）的压缩存储
        - $$\begin{pmatrix} a_{1,1} & a_{1,2} & 0 & \cdots & 0 & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & 0 & 0 \\ 0 & a_{3,2} & a_{3,3} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & 0 & \cdots & a_{n,n-1} & a_{n,n} \end{pmatrix}$$

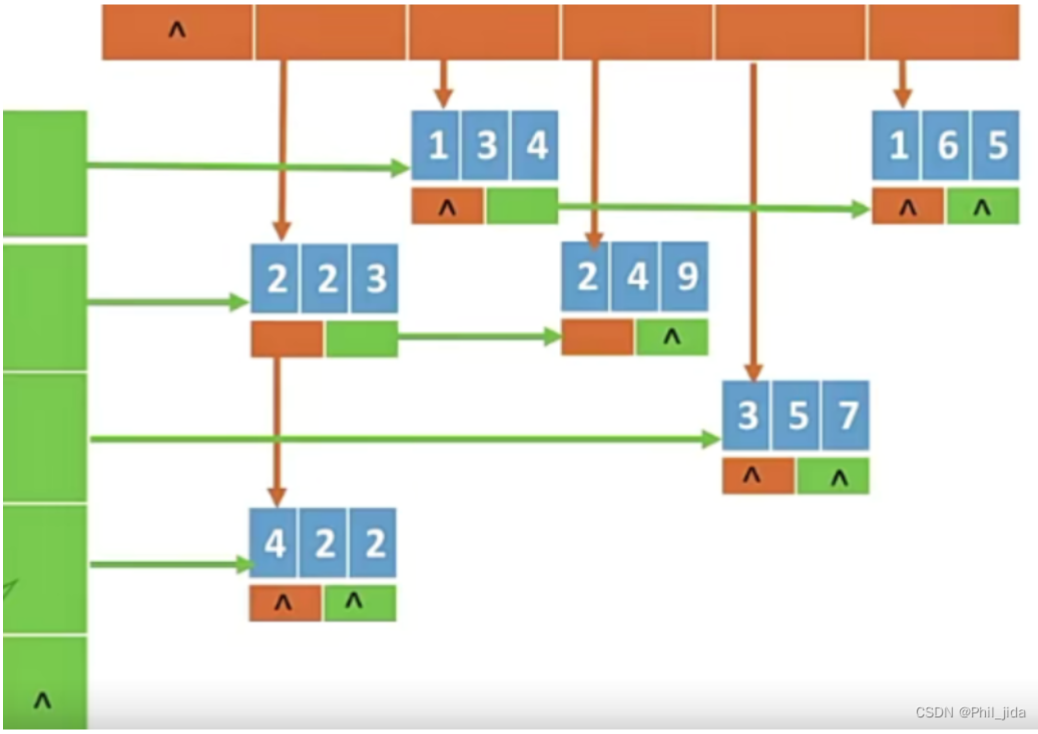
        - |i-j|>1时，有ai,j = 0(1<=i,j<=n)
        - 行优先
            - 前 i-1 行共有3(i-1)-1个元素
            - ai,j是第 i 行第j-i+2个元素
            - ai,j为2i+j-2个元素， 一维数组是从0开始的 k = 2i+j-3
            - 第k+个元素 计算第几行第几列
                - 3(i-1)-1 < k+1 <= 3i-1 ==> i >= (k+2)/3 向上取整得第几行
    - 对于一个n*n的矩阵，最多只有n个非0元素，只需存储n个对角元素信息即可。直接采用一维数组d[i]存储M(i,i)的值

4. 稀疏矩阵

-
    - $$\begin{pmatrix} 0 & 0 & 4 & 0 & 0 & 5 \\ 0 & 3 & 0 & 9 & 0 & 0 \\ 0 & 0 & 0 & 0 & 7 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

- 三元组 <行，列，值>定义一个新的结构体
i（行）	j（列）	v（值）
1	3	4
1	6	5
2	2	3
2	4	9
3	5	7
4	2	2

- 十字链表 定义一个新的结构体
    - 

- 

三、字符串

1、朴素模式匹配

思路：

- 不断比较，如果相同就继续比较，如果不同就重新比较，下标关系为：

```
i = i-j+1;              //i-j表示i回到起始前的一个位置      +1表示下一个子串的起始位置
j = 0;                  //j重新回到0
```

- 如果最后j等于模式串长度，说明匹配成功，返回 i-tlen,匹配成功的起始位置，不相等，匹配失败

代码：

```
//1、朴素模式匹配
int NaiveMatch(char *s, char *t){   //s为主串， t为模式串
    int lens = strlen(s), lent = strlen(t);
    int i=0,j=0;
    while(i < lens && j < lent){
        if(s[i] == t[j]){           //匹配成功，继续匹配
            i++;
            j++;
        }
        else{                       //匹配失败，模式串从头开始匹配，主串
            i = i-j+1;              //i-j表示i回到起始前的一个位置      +1表示下一个子串的起始位置
            j = 0;                  //j重新回到0
        }
    }
    if(j == lent) return i-lent;
    else return 0;
}
```

2、KMP算法

思路：

- 关键是求失败函数
  - 不断迭代找到当前i所指的值与s[k+]所指的值相同的下标
  - 如果s[i]==s[k+1],匹配上了，失败函数为f[i]的值就为k+1，否则为-1
- KMP算法前面与朴素模式匹配一样，在匹配成功时或者j==0时，同时+1
- 如果匹配失败，下一个j的值为失败函数对应的值

代码：

```
//关键：失败函数
void Fail(char *s, int f[]){
    int len = strlen(s);
    f[0] = -1;
    int i=1,k=0;
    for(i=1; i < len; i++){
        k = f[i-1];                 //k指向当前位置的前一个元素，前k项与第i-1往前找k
项相同，如果第k+1项与第j项不同
        while(s[i]!=s[k+1] && k>=0){
            k = f[k];               //迭代求下一个位置，保证k不越界
        }
        if(s[i] == s[k+1]){         //如果存在，就加1
            f[i] = k+1;
        }
        else{                       //不存在，赋值为-1
            f[i] = -1;
        }
    }
}

int KMP(char *s, char *t){          //s为主串，t为模式串
    int lens = strlen(s), lent = strlen(t);
    int f[lent];
    int i=0,j=0;

    Fail(t,f);                      //求得失败函数
    while(i<lens && j < lent){
        if(j==0 || s[i] == t[j]){
            i++;
            j++;
        }
        else{
            j = f[j-1]+1;
        }
    }
    if(j == lent) return i-lent;
    else return 0;
}
```