CHAPTER 5

第5章 数组和稀疏矩阵

5.1 实战题解析

【实战 5.1】 POJ2189——最多围栏个数

时间限制：1000ms；内存限制：65536KB。

视频讲解

问题描述：一个条形牧场用 $p(1\leqslant p\leqslant 1000)$ 根柱子(编号分别为 $1\sim p$)分隔成等间距的围栏，每头牛只能在围栏中放牧，现有 $n(1\leqslant n\leqslant 1000)$ 头牛在放牧。请求出数量不超过 $c(0\leqslant c\leqslant 1000)$ 头牛的最大连续区域的围栏个数。

输入格式：第 1 行是 3 个整数 n、p 和 c。第 2 行到第 $n+1$ 行，每行包含一个整数 x，取值范围为 $1\sim p-1$，指定一头牛在柱子 x 和柱子 $x+1$ 的围栏中放牧。注意一个围栏中允许放牧多头牛。

输出格式：在第 1 行中输出一个整数，表示最多有 c 头牛的最大连续区域的围栏个数。

输入样例：

```
2 6 1
2
3
```

输出样例：

```
3
```

解：假设柱子 x 和柱子 $x+1$ 的围栏的编号为 x，设置一维数组 num[1..$p-1$]，num[i]记录围栏 i 中放牧的牛数量，题目样例的场景如图 5.1 所示，围栏的编号为 1～5，两头牛分别在围栏 2 和围栏 3 中放牧，求出一头牛的最大连续区域的围栏个数为 3。

本题就是枚举所有的连续区域，即 num[i..j]($i\leqslant j$)，其中包含的围栏个数为 $j-i+1$，求出放牧的牛数量(num[i]+…+num[j])，在所有小于或等

于 c 的牛数量中求最大的围栏个数，对应的时间复杂度为 $O(p^3)$。该方法会出现超时，改进方法是采用前缀和。

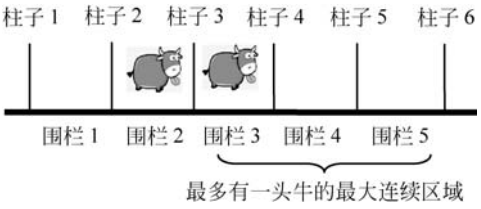


图 5.1　样例的场景

设置一维数组 sum，sum[i]表示 num 中的前 i 项元素和(称为前缀和)，即

$$sum[i]=num[1]+num[2]+\cdots+num[i]$$

可以在 $O(p)$的时间内求出 sum 数组，其过程是先置 sum 的所有元素为 0，再通过迭代方式求出 sum 的每个元素：

```
for(int i=1;i<=p;i++)
  sum[i]=sum[i-1]+num[i];
```

假设 $i\leqslant j$，有：

$$sum[i-1]=num[1]+num[2]+\cdots+num[i-1]$$
$$sum[j]=num[1]+num[2]+\cdots+num[i-1]+num[i]+\cdots+num[j]$$

用后式减前式：

$$sum[j]-num[i-1]=num[i]+\cdots+num[j]$$

其中，sum[j]−num[i−1]表示围栏 i 到围栏 j 的连续区域中放牧的牛数量，其中的围栏个数为 $j-i+1$，这样枚举所有的连续区域，即 num[$i..j$]($i\leqslant j$)，在所有小于或等于 c 的牛数量中求最大的围栏个数。对应的时间复杂度为 $O(p^2)$。

对应的程序如下：

```
#include<iostream>
#include<cstring>
const int MAXN=1010;
using namespace std;
int num[MAXN];                          //num[x]表示围栏 x 中放牧的牛的数量
int sum[MAXN];                          //存放前缀和
int main()
{ int n,p,c,x;
  while(~scanf("%d%d%d",&n,&p,&c))
  { memset(num,0,sizeof num);
    for(int i=0;i<n;i++)
    { scanf("%d",&x);
      num[x]++;
    }
    sum[0]=0;
    for(int i=1;i<=p;i++)               //求前缀和
```

```
            sum[i]=sum[i-1]+num[i];
        int ans=0;
        for(int i=1;i<=p-1;i++)                    //求题目要求的结果 ans
            for(int j=i;j<=p-1;j++)
            { if(sum[j]-sum[i-1]<=c)
                ans=max(ans,j-i+1);
            }
        printf("%d\n", ans);
    }
    return 0;
}
```

上述程序是 AC 代码,运行时间为 32ms,占用的空间为 96KB,编译语言为 C++。

【实战 5.2】 POJ3070——矩阵快速幂求 Fibonacci 数列

时间限制:1000ms;内存限制:65536KB。


视频讲解

问题描述:Fibonacci 数列是 $F_0=0, F_1=1, F_n=F_{n-1}+F_{n-2}(n\geqslant 2)$。例如,前 10 项 Fibonacci 数列是 0,1,1,2,3,5,8,13,21,34。

Fibonacci 数列的另外一种公式是:

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \underbrace{\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}\cdots\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}}_{n\text{次}}$$

给定一个整数 n,请求出 F_n 的最后 4 位数字。

输入格式:输入包含多个测试用例,每个测试用例由单个包含整数 $n(0\leqslant n\leqslant 1000000000)$的行构成,以 $n=-1$ 表示结束。

输出格式:对于每个测试用例,输出一行包含 F_n 的最后 4 位数字,如果均为 0 则输出'0',否则忽略前导 0(也就是输出 F_n mod 10000)。

输入样例:

```
0
9
999999999
1000000000
-1
```

输出样例:

```
0
34
626
6875
```

参考博客:https://blog.csdn.net/u012350533/article/details/12460959 博客。

解:先介绍快速幂算法。如果计算 x^n,可以将 n 拆成二进制数,该二进制数的第 i 位的权为 2^{i-1}。例如当 $n=11$ 时,其二进制数是$[1011]_2$,即 $11=2^3\times 1+2^2\times 0+2^1\times 1+2^0\times 1$,因此可以将 x^{11} 转化为计算 $x^{2^3}\times x^{2^1}\times x^{2^0}$,也就是 $x^{11}=x^8\times x^2\times x^1$,如图 5.2 所示。

用 ans 存放计算结果，先置 ans＝1，base＝x(权)，当 $n>0$ 时循环，取 n 的末尾二进制位 d，若 $d=0$，则跳过；若 $d=1$，则执行 ans＊＝base，再执行 base＊＝base(向高位移一位的权)，并且将 n 右移一位。最后返回 ans。简单地说，base 为 x 的权，按 x、x^2、x^4、……递增，当对应二进制位为 1 时 ans 乘上相应的 base。该算法称为**快速幂算法**，求 x^n 的快速幂算法如下：

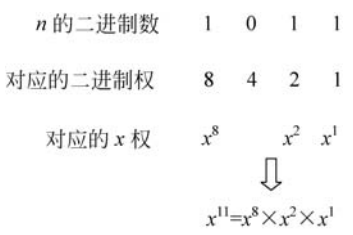


图 5.2　快速幂计算 x^{11}

```
double pow(double x,int n)
{ double ans=1.0,base=x;
  while (n!=0)
  { if ((n&1)==1)                    //遇到二进制位 1
      ans*=base;                     //求 ans
    base*=base;                      //权递增
    n>>=1;                           //n 右移一位
  }
  return ans;
}
```

将上述 x^n 中的 x 改为 $m\times m$ 矩阵 $\boldsymbol{A}$，对应快速幂就是**矩阵快速幂算法**。对于本题有 $m=2$，两个 2×2 的矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 相乘的公式如下：

$$\begin{bmatrix}a_{0,0} & a_{0,1}\\ a_{1,0} & a_{1,1}\end{bmatrix}\begin{bmatrix}b_{0,0} & b_{0,1}\\ b_{1,0} & b_{1,1}\end{bmatrix}=\begin{bmatrix}a_{0,0}b_{0,0}+a_{0,1}b_{1,0} & a_{0,0}b_{0,1}+a_{0,1}b_{1,1}\\ a_{1,0}b_{0,0}+a_{1,1}b_{1,0} & a_{1,0}b_{0,1}+a_{1,1}b_{1,1}\end{bmatrix}$$

注意：任何一个矩阵的 0 次幂为对应的单位矩阵，即

$$\begin{bmatrix}* & *\\ * & *\end{bmatrix}^0=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$$

首先置 ans 为单位矩阵 $\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$，$\boldsymbol{A}=\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}$，采用矩阵快速幂算法求出 ans＝$\boldsymbol{A}^n$，再取 ans 左上角元素(即 F_n)模 10000 得到最终结果。对应的程序如下：

```
#include<iostream>
#include<cstring>
using namespace std;
struct Matrix                        //表示 2×2 矩阵类型
{
  int data[2][2];
};
int n;
Matrix multiply(Matrix A,Matrix B)   //返回矩阵 A 和 B 相乘的结果
{ Matrix C;
  memset(C.data,0,sizeof(C.data));
  for(int i=0;i<2;i++)
    for(int j=0;j<2;j++)
      for(int k=0;k<2;k++)
      { C.data[i][j]+=A.data[i][k]*B.data[k][j];
```

```
            C.data[i][j]%=10000;
        }
    return C;
}
Matrix quick_pow(Matrix A,int n)                    //求矩阵 A 的 n 次幂的快速幂算法
{ Matrix ans;
  memset(ans.data,0,sizeof(ans.data));
  for(int i=0;i<2;i++)                              //置 ans 为单位矩阵
    ans.data[i][i]=1;
  while(n!=0)
  { if (n & 1)
      ans=multiply(ans,A);
    A=multiply(A,A);
    n>>=1;                                          //n 右移一位
  }
  return ans;
}
int main()
{ while(cin >> n && n!=-1)
  { if(n==0)
    { cout << 0 << endl;
      continue;
    }
    Matrix ans;
    ans.data[0][0]=0;                               //先置 ans 为初始矩阵
    ans.data[0][1]=1;
    ans.data[1][0]=1;
    ans.data[1][1]=1;
    ans=quick_pow(ans,n);                           //取 ans 左上角元素
    cout << ans.data[0][1]%10000 << endl;
  }
  return 0;
}
```

上述程序是 AC 代码,运行时间为 0ms,占用的空间为 172KB,编译语言为 C++。求 ans=x^n 的快速幂算法和求 ans=$\boldsymbol{A}^n$ 的矩阵快速幂算法的对应关系,如表 5.1 所示。

表 5.1　快速幂和矩阵快速幂算法的对应关系

比较项	快速幂算法	矩阵快速幂算法	说明
初始化	base=x ans=1	$\boldsymbol{A}$ 为 m 阶矩阵 ans 置为 m 阶单位矩阵	
n 的二进制位 d=0	跳过	跳过	
n 的二进制位 d=1	ans *=base base *=base	ans=ans×$\boldsymbol{A}$ $\boldsymbol{A}$=$\boldsymbol{A}$×$\boldsymbol{A}$	由 double 数乘法改为矩阵乘法


视频讲解

【实战 5.3】　HDU4920——稀疏矩阵乘法

时间限制:2000ms;内存限制:131072KB。

问题描述:给定两个 $n\times n$ 的矩阵 $\boldsymbol{a}$ 和 $\boldsymbol{b}$,求它们的积,结果矩阵中每个元素值模 3。

输入格式:输入包含多个测试用例,每个测试用例的第一行为 n($1\leqslant n\leqslant 800$);接下来

n 行，每行 n 个整数，表示矩阵 $\boldsymbol{a}$，第 i 行的第 j 个整数为 a_{ij}；类似地，再接下来的 n 行为矩阵 $\boldsymbol{b}$（$0 \leqslant a_{ij}, b_{ij} \leqslant 10^9$）。

输出格式：对于每个测试用例，输出 n 行，每行 n 个整数表示 $\boldsymbol{a} \times \boldsymbol{b}$ 的结果。

输入样例：

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

输出样例：

```
0
0 1
2 1
```

解：本题看起来十分平常，直接按矩阵乘法设计算法即可，但会出现超时。

因为题目要求结果矩阵中的每个元素值模 3，按求模性质，可以先对 $\boldsymbol{a}$、$\boldsymbol{b}$ 的元素值模 3，这样 $\boldsymbol{a}$、$\boldsymbol{b}$ 中较多的元素为 0，可以看成非严格意义上的稀疏矩阵。矩阵 $\boldsymbol{a}$ 和 $\boldsymbol{b}$ 的乘法运算的转换过程如图 5.3 所示。

```
memset(c,0,sizeof(c));
for (int i=0;i<n;i++)
    for (int j=0;j<n;j++)
        for (int k=0;k<n;k++)
            c[i][j]+=a[i][k]*b[k][j];
```

⇩ j、k 的两重循环改变顺序

```
memset(c,0,sizeof(c));
for (int i=0;i<n;i++)
    for (int k=0;k<n;k++)
        for (int j=0;j<n;j++)
            c[i][j]+=a[i][k]*b[k][j];
```

⇩ j、k 两个变量相交换

```
memset(c,0,sizeof(c));
for (int i=0;i<n;i++)
    for (int j=0;j<n;j++)
        for (int k=0;k<n;k++)
            c[i][k]+=a[i][j]*b[j][k];
```

图 5.3　求 $\boldsymbol{c}=\boldsymbol{a}\times\boldsymbol{b}$ 的过程的转换

例如 $n=2$ 时，计算 $\boldsymbol{c}=\boldsymbol{a}\times\boldsymbol{b}$ 的公式如下：

$$\boldsymbol{c}=\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix}\begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix}=\begin{bmatrix} a_{0,0}b_{0,0}+a_{0,1}b_{1,0} & a_{0,0}b_{0,1}+a_{0,1}b_{1,1} \\ a_{1,0}b_{0,0}+a_{1,1}b_{1,0} & a_{1,0}b_{0,1}+a_{1,1}b_{1,1} \end{bmatrix}$$

常规计算顺序如下：

```
c[0][0]+=a[0][0]*b[0][0]
c[0][0]+=a[0][1]*b[1][0]
c[0][1]+=a[0][0]*b[0][1]
c[0][1]+=a[0][1]*b[1][1]
c[1][0]+=a[1][0]*b[0][0]
c[1][0]+=a[1][1]*b[1][0]
c[1][1]+=a[1][0]*b[0][1]
c[1][1]+=a[1][1]*b[1][1]
```

转换后的计算顺序如下：

```
c[0][0]+=a[0][0]*b[0][0]
c[0][1]+=a[0][0]*b[0][1]
c[0][0]+=a[0][1]*b[1][0]
c[0][1]+=a[0][1]*b[1][1]
c[1][0]+=a[1][0]*b[0][0]
```

```
c[1][1]+=a[1][0]*b[0][1]
c[1][0]+=a[1][1]*b[1][0]
c[1][1]+=a[1][1]*b[1][1]
```

之所以做这样的转换是为了提高性能,一方面利用了程序局部性原理,a 中元素是按行优先存储的,而按 i、j 循环处理 $a[i][j]$也是按行优先访问 a 中的元素;另外一方面,当 $a[i][j]=0$ 时无论 $b[j][k]$是多少都不必做乘法运算了,可以优化这种情况的计算。对应的程序如下:

```
#include <stdio.h>
using namespace std;
const int MAXN=805;
int a[MAXN][MAXN],b[MAXN][MAXN],c[MAXN][MAXN];
int main()
{ int n,i,j,k;
  while (~scanf("%d",&n))
  { for(i=0;i<n;i++)
      for(j=0;j<n;j++)
      { scanf("%d",&a[i][j]);
        a[i][j]%=3;
        c[i][j]=0;
      }
    for(i=0;i<n;i++)
      for(j=0;j<n;j++)
      { scanf("%d",&b[i][j]);
        b[i][j]%=3;
      }
    for(i=0;i<n;i++)                    //计算矩阵 c
      for(j=0;j<n;j++)
      { if(!a[i][j])
        continue;
        for(k=0;k<n;k++)
          c[i][k]=c[i][k]+a[i][j]*b[j][k];
      }
    for(i=0;i<n;i++)                    //输出结果
    { for(j=0;j<n;j++)
        if(j==n-1)
          printf("%d\n",c[i][j]%3);
        else
          printf("%d ",c[i][j]%3);
    }
  }
  return 0;
}
```

上述程序是 AC 代码,运行时间为 1154ms,占用的空间为 9320KB,编译语言为 C++。

可以进一步优化计算矩阵 c 的过程,也就是说根据 $a[i][j]$取值 0、1、2 做相应计算,当 $a[i][j]=0$ 时跳过;当 $a[i][j]=1$ 时,$c[i][k]+=b[j][k]$($0\leqslant k<n$);当 $a[i][j]=2$ 时,$c[i][k]+=2*b[j][k]$($0\leqslant k<n$)。对应的程序如下:

```
#include <stdio.h>
using namespace std;
const int MAXN=805;
int a[MAXN][MAXN],b[MAXN][MAXN],c[MAXN][MAXN];
int main()
{ int n,i,j,k;
  while(~scanf("%d",&n))
  { for(i=0;i<n;i++)
      for(j=0;j<n;j++)
      { scanf("%d",&a[i][j]);
        a[i][j]%=3;
        c[i][j]=0;
      }
    for(i=0;i<n;i++)
      for(j=0;j<n;j++)
      { scanf("%d",&b[i][j]);
        b[i][j]%=3;
      }
    for(int i=0;i<n;i++)                    //计算矩阵 c
      for(int j=0;j<n;j++)
      { if(a[i][j]==1)
          for (int k=0;k<n;k++)
            c[i][k]+=b[j][k];
          else if(a[i][j]==2)
            for(int j=0;j<n; j++ )
              c[i][k]+=(b[j][k]<<1);
      }
    for(i=0;i<n;i++)                        //输出结果
    { for(j=0;j<n;j++)
        if(j==n-1)
          printf("%d\n",c[i][j]%3);
        else
          printf("%d ",c[i][j]%3);
    }
  }
  return 0;
}
```

上述程序是 AC 代码，运行时间为 967ms，占用的空间为 9316KB，编译语言为 C++。

5.2　在线编程题解析


视频讲解

5.2.1　LeetCode48——旋转图像

问题描述：给定一个 $n \times n$ 的二维矩阵表示一个图像。将图像顺时针旋转 90°。注意必须在原地旋转图像，这意味着需要直接修改输入的二维矩阵。请不要使用另一个矩阵来旋转图像。例如，给定 matrix={{1,2,3},{4,5,6},{7,8,9}}，原地旋转输入矩阵，使其变为{{7,4,1},{8,5,2},{9,6,3}}。要求设计如下函数：

```
class Solution {
public:
  void rotate(vector<vector<int>>& matrix)
  {  ...  }
};
```

解：这里要求只能在原数组中做旋转操作，从中找到图像旋转的规律。实现该操作的步骤如下：

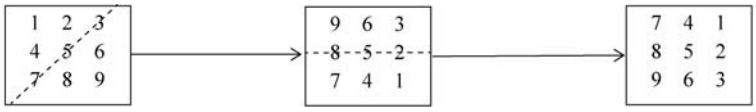
① 按左下—右上对角线交换所有对角元素。

② 将第 i 行($0 \leqslant i < n/2$)的所有元素与倒数第 i 行交换。

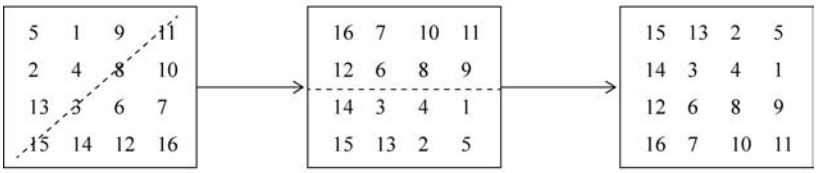
例如，两个示例的旋转过程如图 5.4 所示。对应的算法如下：

```
class Solution
{
public:
  void rotate(vector<vector<int>>& matrix)
  { int n=matrix.size();                    //行、列数为 n
    for (int i=0;i<n;i++)                   //步骤①
      for (int j=0;j<n-i;j++)
        swap(matrix[i][j],matrix[n-j-1][n-i-1]);
    for (int i=0;i<n/2;i++)                 //步骤②
        for (int j=0;j<n;j++)
          swap(matrix[i][j],matrix[n-i-1][j]);
  }
};
```

上述程序是 AC 代码，运行时间为 0ms，占用的空间为 6.5MB，编译语言为 C++。



(a) 示例1的转换过程



(b) 示例2的转换过程

图 5.4　两个示例的旋转过程

视频讲解

5.2.2　HDU1575——方阵 A 的迹

时间限制：1000ms；内存限制：32768KB。

问题描述：$\boldsymbol{A}$ 为一个方阵，则 Tr A 表示 $\boldsymbol{A}$ 的迹(就是主对角线上各项的和)，现要求 Tr(A^k)%9973。

输入格式：数据的第一行是一个 t，表示有 t 组数据。每组数据的第一行是 $n(2\leqslant n\leqslant 10)$ 和 $k(2\leqslant k<10^9)$两个整数。接下来有 n 行，每行有 n 个整数，每个整数的范围是[0,9]，表示方阵 ***A*** 的内容。

输出格式：对应每组数据，输出 Tr(A^k)%9973。

输入样例：

```
2
2 2
1 0
0 1
3 99999999
1 2 3
4 5 6
7 8 9
```

输出样例：

```
2
2686
```

解：直接采用本书第 5 章中实战 5.2 的矩阵快速幂方法求 res=A^k，再用 ans 累加 res 主对角线上的元素和(模 9973)，最后输出 ans。对应的程序如下：

```
#include <iostream>
#include <cstring>
using namespace std;
const int MOD=9973;
int n,k;
struct Matrix                                   //矩阵类型
{
  int data[10][10];
} res,A;
void init()                                     //初始化 res 和 A 矩阵
{ for(int i=0;i<n;i++)
    for(int j=0;j<n;j++)
    { res.data[i][j]=0;
      scanf("%d",&A.data[i][j]);                //A 为输入的矩阵
      if(i==j)
        res.data[i][j]=1;                       //将 res 置为 n 阶单位矩阵
    }
}
Matrix multiply(Matrix A,Matrix B)              //返回矩阵 A 和 B 相乘的结果
{ Matrix C;
  memset(C.data,0,sizeof(C.data));
  for(int i=0;i<n;i++)
    for(int j=0;j<n;j++)
      for(int k=0;k<n;k++)
        C.data[i][j]=(C.data[i][j]+A.data[i][k]*B.data[k][j])%MOD;
  return C;
}
```

```
void quick_pow()                                    //求矩阵 A 的 k 次幂 res(矩阵快速幂)
{ while(k!=0)
  { if (k & 1)
      res=multiply(res,A);
    A=multiply(A,A);
    k>>=1;                                          //k 右移一位
  }
}
int main()
{ int t;
  cin >> t;
  while(t--)
  { scanf("%d%d",&n,&k);
    init();                                         //初始化 res 和 A
    quick_pow();                                    //用快速幂方法求 A 矩阵的 k 次幂
    int ans=0;
    for(int i=0;i<n;i++)
      ans=(ans+res.data[i][i]) % MOD;
    printf("%d\n",ans);
  }
  return 0;
}
```

上述程序是 AC 代码,运行时间为 15ms,占用的空间为 1804KB,编译语言为 C++。


视频讲解

5.2.3 HDU1559——最大子矩阵

时间限制:10000ms;内存限制:32768KB。

问题描述:给出一个 $m \times n$ 的整数矩阵,在上面找一个 $x \times y$ 的子矩阵,使子矩阵中所有元素的和最大。

输入格式:输入数据的第一行为一个正整数 t,表示有 t 组测试数据。每一组测试数据的第一行为 4 个正整数 m、n、x、y($0<m$,$n<1000$,$0<x \leqslant m$,$0<y \leqslant n$),表示给定的矩形有 m 行 n 列。接下来是这个矩阵,有 m 行,每行有 n 个不大于 1000 的正整数。

输出格式:对于每组数据,输出一个整数,表示子矩阵的最大和。

输入样例:

```
1
4 5 2 2
3    361  649  676  588
992  762  156  993  169
662  3 4  638  8 9  543
525  165  254  809  280
```

输出样例:

```
2474
```

解:用 a 数组存放 $m \times n$ 的整数矩阵,为了简单,$a[1..m][1..n]$存放有效数据,其他元素为 0。设计一个二维数组 sum,sum$[i][j]$存放(1,1)~(i,j)矩阵的所有元素和(该矩阵

的左上角为(1,1)、右下角为(i,j))，其中行、列下标为 0 的元素值均为 0，类似于前缀和。可以通过如下循环求出 sum 数组：

```
for(int i=1;i<=n;i++)                    //求(1,1)~(i,j)矩阵的元素和
  for(int j=1;j<=m;j++)
    sum[i][j]=sum[i-1][j]+sum[i][j-1]+a[i][j]-sum[i-1][j-1];
```

如图 5.5 所示，任意一个右下角为(i,j)的 $x \times y$ 的子矩阵(宽度为 x、高度为 y)的所有元素和表示为 sum[i][j]−sum[$i-x$][j]−sum[i][$j-y$]+sum[$i-x$][$j-y$]。通过遍历求所有这样子矩阵的元素和，比较求出最大元素和 ans，最后输出 ans。对应的程序如下：

```
#include <iostream>
#include <algorithm>
#include <cstring>
using namespace std;
const int MAXN=1007;
int t,m,n,x,y;
int a[MAXN][MAXN];
int sum[MAXN][MAXN];
int main()
{ scanf("%d",&t);
  while(t--)
  { memset(sum,0,sizeof(sum));
    scanf("%d%d%d%d",&n,&m,&x,&y);
    int ans=0;                           //存放 x×y 子矩阵的最大元素和
    for(int i=1;i<=n;i++)                //输入 a[i][j]
      for(int j=1;j<=m;j++)
        scanf("%d",&a[i][j]);
    for(int i=1;i<=n;i++)                //求(1,1)~(i,j)矩阵的元素和
      for(int j=1;j<=m;j++)
        sum[i][j]=sum[i-1][j]+sum[i][j-1]+a[i][j]-sum[i-1][j-1];
    for(int i=x;i<=n;i++)                //求 ans
      for (int j=y;j<=m;j++)
        ans=max(ans,sum[i][j]-sum[i-x][j]-sum[i][j-y]+sum[i-x][j-y]);
    printf("%d\n",ans);                  //输出 ans
  }
  return 0;
}
```

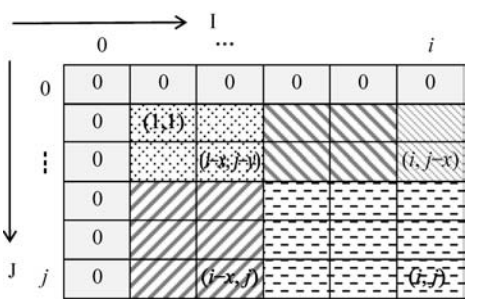


图 5.5　求($i-x$,$j-y$)~(i,j)矩阵和

上述程序是 AC 代码,运行时间为 249ms,占用的空间为 9656KB,编译语言为 C++。当然也可以将 3 个 for 循环语言合并起来:

```
int ans=0;
for (int i=1;i<=n;i++)
  for (int j=1;j<=m;j++)
  { scanf("%d",&a[i][j]);
    sum[i][j]=sum[i-1][j]+sum[i][j-1]+a[i][j]-sum[i-1][j-1];
    if (i>=x && j>=y)
      ans=max(ans,sum[i][j]-sum[i-x][j]-sum[i][j-y]+sum[i-x][j-y]);
  }
```

上述采用类似前缀和的方法,对于每个测试用例,求解的时间复杂度为 $O(mn)$,如果采用直接枚举所有子矩阵的方法,求解的时间复杂度为 $O(m^2n^2)$。


视频讲解

5.2.4 POJ3213——矩阵乘法问题

时间限制:5000ms;内存限制:131072KB。

问题描述:USTC 最近开发了并行矩阵乘法机器 PM3,用于非常大的矩阵乘法运算。给定两个矩阵 ***A*** 和 ***B***,其中 ***A*** 是 $n\times p$ 矩阵,***B*** 是 $p\times m$ 矩阵,PM3 可以在 $O(p(n+p+m))$ 时间内计算矩阵 $\boldsymbol{C}=\boldsymbol{A}\times\boldsymbol{B}$。然而,PM3 的开发人员很快发现了一个小问题:PM3 可能出错,而且每当出现错误时,结果矩阵 ***C*** 只包含一个不正确的元素。

开发人员在 PM3 给出矩阵 ***C*** 之后检查并纠正它。他们认为这是一项简单的任务,因为最多会有一个不正确的元素。请编写一个程序来检查和纠正 PM3 计算的结果。

输入格式:输入的第一行是 3 个整数 n、p 和 m($0<n,p,m\leqslant 1000$),它们表示 ***A*** 和 ***B*** 的维数。然后跟随 n 行,每行有 p 个整数,以行序优先给出 ***A*** 的元素。之后是 ***B*** 和 ***C*** 的元素以相同的方式给出。

A 和 ***B*** 的元素以 1000 的绝对值为界,***C*** 的界限是 2000000000。

输出格式:如果 ***C*** 不包含不正确的元素,请打印“Yes”,否则打印“No”后跟另外两行,第一行上有两个整数 r 和 c,第二行上有另一个整数 v,表示 ***C*** 中行 r 列 c 的元素应该校正为 v。

输入样例:

```
2 3 2
1 2 -1
3 -1 0
-1 0
0 2
1 3
-2 -1
-3 -2
```

输出样例:

```
No
1 2
1
```

解：采用二维数组 a、b、c 分别存放 $\boldsymbol{A}$、$\boldsymbol{B}$、$\boldsymbol{C}$ 矩阵，$\boldsymbol{C}$ 为给出的 $\boldsymbol{A}\times\boldsymbol{B}$ 结果，可能会有错，最多只有一个元素出错。

如果按照矩阵乘法求出 $\boldsymbol{c}=\boldsymbol{A}\times\boldsymbol{B}$，再将 $\boldsymbol{C}$ 与 $\boldsymbol{c}$ 的元素一一比较找到出错的元素，花费的时间较多(时间复杂度为 $O(n^3)$)。通过一个示例看改进方法(时间复杂度为 $O(n^2)$)，例如：

$$\boldsymbol{a}=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{bmatrix},\quad \boldsymbol{b}=\begin{bmatrix}10 & 20 & 30\\ 40 & 50 & 60\\ 70 & 80 & 90\end{bmatrix}$$

求 $\boldsymbol{c}=\boldsymbol{a}\times\boldsymbol{b}$ 的公式如下：

$$c[i][j]=\sum_{l=0}^{2}a[i][l]\times b[l][j]$$

求 $\boldsymbol{c}$ 的各个行的元素和如下：

$$c[0][0]=1\times10+2\times40+3\times70$$
$$c[0][1]=1\times20+2\times50+3\times80$$
$$c[0][2]=1\times30+2\times60+3\times90$$

$\boldsymbol{c}$ 的第 0 行所有元素之和$=c[0][0]+c[0][1]+c[0][2]=1\times(10+20+30)+2\times(40+50+60)+3\times(70+80+90)$

$$c[1][0]=4\times10+5\times40+6\times70$$
$$c[1][1]=4\times20+5\times50+6\times80$$
$$c[1][2]=4\times30+5\times60+6\times90$$

$\boldsymbol{c}$ 的第 1 行所有元素之和$=c[1][0]+c[1][1]+c[1][2]=4\times(10+20+30)+5\times(40+50+60)+6\times(70+80+90)$

$$c[2][0]=7\times10+8\times40+9\times70$$
$$c[2][1]=7\times20+8\times50+9\times80$$
$$c[2][2]=7\times30+8\times60+9\times90$$

$\boldsymbol{c}$ 的第 2 行所有元素之和$=c[2][0]+c[2][1]+c[2][2]=7\times(10+20+30)+8\times(40+50+60)+9\times(70+80+90)$

从中看出，$\boldsymbol{c}$ 的第 i 行元素和 tmp$=\sum_{j=0}^{m-1}a[i][j]\times$($\boldsymbol{b}$ 数组的第 j 行元素和)

所以设置两个一维数组 b_row 和 c_row，分别累计 $\boldsymbol{b}$、$\boldsymbol{c}$ 数组的每一行元素和。不必计算出 $\boldsymbol{c}$ 数组，由 tmp$+=a[i][j]*b$_row$[j]$($0\leqslant j\leqslant m-1$)求出结果矩阵的第 i 行元素和，当出现 tmp$\neq c$_row$[i]$时，说明出错元素在 $\boldsymbol{C}$ 的第 i 行，再采用常规方法在 $\boldsymbol{C}$ 的第 i 行中查找出错元素的位置。

对应的程序如下：

```
#include <iostream>
using namespace std;
const int N=1001;
int a[N][N];
int b[N][N];
int c[N][N];
```

```
int c_row[N];
int b_row[N];
int main()
{ int n,m,p;
  while(~scanf("%d%d%d", &n,&m,&p))
  { for (int i=0;i<n;i++)                    //输入 a
      for(int j=0;j<m;j++)
        scanf("%d",&a[i][j]);
    for (int i=0;i<m;i++)                    //输入 b
      for(int j=0;j<p;j++)
        scanf("%d",&b[i][j]);
    for (int i=0;i<n;i++)                    //输入 c
      for(int j=0;j<p;j++)
        scanf("%d",&c[i][j]);
    for(int i=0;i<m;i++)                     //求出 b 的每行元素和
    { b_row[i]=0;
      for (int j=0;j<p;j++)
        b_row[i]+=b[i][j];
    }
    for (int i=0;i<n;i++)                    //求出 c 的每行元素和
    { c_row[i]=0;
      for(int j=0;j<p;j++)
        c_row[i]+=c[i][j];
    }
    int i;
    for (i=0;i<n;i++)                        //找出错行 i
    { int tmp=0;
      for (int j=0;j<m;j++)
        tmp+=a[i][j]*b_row[j];
      if (tmp!=c_row[i])
        break;
    }
    if (i==n)                                //没有错误
      printf("Yes\n");
    else
    { printf("No\n");                        //第 i 行错误
      for(int j=0;j<p;j++)
      { int res=0;
        for(int k=0;k<m;k++)
          res+=a[i][k]*b[k][j];
        if(res!=c[i][j])
        { printf("%d %d\n",i+1,j+1);
          printf("%d\n",res);
          break;
        }
      }
    }
  }
}
```

上述程序是 AC 代码,运行时间为 1188ms,占用的空间为 8736KB,编译语言为 C++。

5.2.5　POJ3292——求 H 半素数的个数


视频讲解

时间限制：1000ms；内存限制：65536KB。

问题描述：H 数是一个正数，为 4 的整数倍加 1，例如 1、5、9、13、17、21 等都是 H 数。H 数的乘法运算是闭合的。

与常规整数一样可以将 H 数划分为单元、H 素数和 H 合数。1 是唯一的单元，如果一个 H 数 h 不是单元，并且它仅有一种方式表示为两个 H 数的乘积，即 $1\times h$，则 h 为 H 素数，其余为 H 合数。例如，前几个 H 合数是 $5\times5=25$、$5\times9=45$、$5\times13=65$、$9\times9=81$、$5\times17=85$。

请编程累计 H 半素数的个数。H 半素数是一个 H 数，它正好是两个 H 素数的乘积，这两个 H 素数可以相等或不同。在上面的示例中，5 个数字均为 H 半素数，而 $125=5\times5\times5$ 不是 H 半素数，因为它是 3 个 H 素数的乘积。

输入格式：输入的每一行包含一个 H 数 h（$h\leqslant1000001$）。输入的最后一行包含 0，并且不需要处理此行。

输出格式：对于每个输入的 H 数 h，输出一行，包含 h 和 $1\sim h$（含 1 和 h）的 H 半素数个数，以空格分隔。

输入样例：

```
21
85
789
0
```

输出样例：

```
21 0
85 5
789 62
```

参考博客：https://blog.csdn.net/qq_36651153/article/details/77740460

解：置最大整数 MAXN 为 1000005，设计 solve()算法求出 1～MAXN 的 H 半素数个数的前缀和数组 a（采用类似第 1 章中 1.7 节的在线编程题 3 的素数筛选法）。对于给定的 h，$a[h]$就是 $1\sim h$ 的 H 半素数个数，直接输出即可。对应的程序如下：

```
#include <iostream>
#include <cstring>
using namespace std;
const int MAXN=1000005;
int prime[MAXN];                          //用于存放所有的 H 素数
bool isprime[MAXN];                       //isprime[i]为 true 表示整数 i 为 H 素数
int a[MAXN];
void solve()                              //求 H 半素数个数的前缀和数组 a
{ int k=0;
  memset(isprime,true,sizeof(isprime));
  memset(a,0,sizeof(a));
  for(int i=5;i<MAXN;i+=4)                //求 5～MAXN 的 H 素数
```

```
    { if(isprime[i])                          //i 为 H 素数
      { prime[k++]=i;                         //将 H 素数 i 添加到 prime 中
        for(int j=2*i;j<MAXN;j+=i)            //置所有 i 的倍数的 H 数为非 H 素数
        { if((j-1)%4==0)                      //若 j 是 H 数
            isprime[j]=false;                 //j 不是 H 素数,而是 H 合数
        }
      }
    }
    for(int i=0;i<k;i++)                      //k 为小于 MAXN 的 H 素数的个数
    { if(prime[i]>1000) break;                //h≤1000001,其最小因子不可能大于 1000
      for(int j=i;j<k;j++)
      { int tmp=prime[i]*prime[j];            //tmp 是两个 H 素数的乘积
        if(tmp>MAXN) break;
        a[tmp]=1;                             //表示 tmp 是一个 H 半素数
      }
    }
    for(int i=0;i<MAXN;i++)                   //a 作为前缀和
      a[i]+=a[i-1];
}
int main()
{ solve();
  int h;
  while(~scanf("%d",&h) && h)
    printf("%d %d\n",h,a[h]);
}
```

上述程序是 AC 代码,运行时间为 32ms,占用的空间为 5340KB,编译语言为 C++。