A - 转啊转

难度	考点
1	浮点数，格式化输入输出

题目分析

简单签到题，考察基本的输入输出与 `math.h` 中函数的使用。平面向量的旋转公式已经给出，按照公式进行计算即可。注意 `math.h` 中的 `sin`，`cos` 函数参数均为弧度制。圆周率 π 需要保留至少 7 位小数才可满足精度要求（可以采用 `#define` 直接定义 π 的值，也可用 `acos(-1)` 来获取较为精确的 π 值，其中 `acos` 为 `math.h` 中的反余弦函数）

```c
#include <math.h>
#include <stdio.h>

int main()
{
    int x, y, degree;
    double theta, xr, yr;

    scanf("%d,%d,%d", &x, &y, &degree);
    // 用浮点数输入也可以，参考代码如下：
    // double x, y, degree;
    // scanf("%lf,%lf,%lf", &x, &y, &degree);
    theta = degree * acos(-1) / 180;      // 角度制转成弧度制
    xr = x * cos(theta) - y * sin(theta); // 按照公式计算
    yr = x * sin(theta) + y * cos(theta); // 按照公式计算
    printf("(%.2f,%.2f)\n", xr, yr);

    return 0;
}
```

B - 简单的日期计算

难度	考点
1	函数

题目分析

这道题基本上是 PPT 原题，只不过所求是下下月的某一天。已知本月 n 天，下月 m 天，本月第 x 天是星期 y，那么下下月第 z 天是星期 $(n-x+m+z+y) \bmod 7$。注意如果结果是 0 的话，应该输出 7。

示例程序

```c
#include <stdio.h>
#include <string.h>

int GetWeek(int n, int m, int x, int y, int z)
{
    int ans = (n - x + m + z + y) % 7;
    if (ans == 0) ans = 7;
    return ans;
}

int main()
{
    int n, m, x, y, z, ans;

    scanf("%d%d%d%d%d", &n, &m, &x, &y, &z);
    ans = GetWeek(n, m, x, y, z);
    printf("%d\n", ans);

    return 0;
}
```

C - 九次九日九重乘法表

难度	考点
2	二重循环、判断

题目分析

本题主要考察二重循环与判断的结合。

在二重循环中，若分别以变量 i, j 作为指示变量，则要注意循环范围分别是 $1 \leqslant i \leqslant 9, 1 \leqslant j \leqslant i$，且等式的左边应该是 `j * i` 形式，即 i 代表的是等式中的第二个乘数，j 代表第一个。

另外，注意每次外循环进行一次后换行。

示例代码

```c
#include <stdio.h>

int main()
{
    int l, r;
    int i, j, a;

    scanf("%d%d", &l, &r);
    for (i = 1; i <= 9; i++)
    {
        for (j = 1; j <= i; j++)
        {
            a = i * j;
            if (a < l || a > r)
            {
                printf("%d * %d = %d ", j, i, a);
            }
        }
        printf("\n");
    }

    return 0;
}
```

D - 旋转矢量法

难度	考点
2	库函数的使用

题目分析

如果适当的使用 `math.h` 头文件中的函数会大大简化运算，本题推荐大家使用的是 `atan2(double y,double x)` 函数。`atan2(double y,double x)` 可以直接接受一个点的 y 和 x坐标作为输入，输出结果即是题目中所要求的弧度。大家可以参阅 这个链接 获得 `math.h` 中各函数的具体使用方式。其中关于 `atan2(double y,double x)` 的具体返回值情况大家可以参考 这个链接。

同时再介绍一下本题的常见错误，如果你使用了 `atan(double x)` 函数，并计算 `y / x` 时要注意 x 的值可能会是 0。使用 `atan(double x), acos(double x), asin(double x)` 函数要注意他们的返回值的范围分别是 $[-\frac{\pi}{2},\frac{\pi}{2}]$，$[0,\pi]$，$[-\frac{\pi}{2},\frac{\pi}{2}]$，返回结果不能直接符合题目中的范围。

同时还有输出格式的问题，比如正数时前面要输出 `+` 号，每组数据后要换行，特别注意 `-1 0` 的角度不是 0。

```c
#include <math.h>
#include <stdio.h>

int main()
{
    int x, y;
    double ans, A;

    while (~scanf("%d %d", &x, &y))
    {
        ans = atan2(y, x);
        A = sqrt(x * x + y * y);
        if (y == 0 && x > 0)
            printf("x=%.2fcos(ct)\n", A);
        else if (y < 0)
            printf("x=%.2fcos(ct%.2f)\n", A, ans);
        else
            printf("x=%.2fcos(ct+%.2f)\n", A, ans);
    }

    return 0;
}
```

E - 龟速幂

难度	考点
3	自定义函数调用

题目分析

本题主要考察同学们调用已有函数完成程序的能力。

题目涉及到了一些新的概念（如欧拉函数与扩展欧拉定理），做题的时候可能会被迷惑。但题目提供了三个相关函数，我们只需要知道如何调用每个函数，就可以用简单的 `if-else` 判断完成题目的任务。

实际代码编写中，这样的思想是很常见的：特定的功能交给标准库或第三方库的函数实现，我们只专注于主要部分的代码编写，而不关注每个功能内部具体的实现方式。

示例代码

```c
#include <stdio.h>

int get_phi(int n);
int gcd(int a, int b);
int pow_mod(int a, int t, int p);

int main()
{
    int T;
    int a, t, p;
    int phi;

    scanf("%d", &T);
    while (T--)
    {
        scanf("%d%d%d", &a, &t, &p);
        phi = get_phi(p);
        if (gcd(a, p) == 1)
        {
            printf("Case #1: %d\n", pow_mod(a, t % phi, p));
        }
        else
        {
            if (t < phi)
            {
                printf("Case #2: %d\n", pow_mod(a, t, p));
            }
            else
            {
                printf("Case #3: %d\n", pow_mod(a, t % phi + phi, p));
            }
        }
    }

    return 0;
}
```

```
37
38  int pow_mod(int a, int t, int p)
39  {
40      int ans = 1;
41      while (t--)
42          ans = 1LL * ans * a % p;
43      return ans;
44  }
45
46  int gcd(int a, int b)
47  {
48      return b ? gcd(b, a % b) : a;
49  }
50
51  int get_phi(int n)
52  {
53      int phi = n, i;
54      for (i = 2; i * i <= n; i++)
55      {
56          if (n % i == 0)
57          {
58              while (n % i == 0)
59                  n /= i;
60              phi -= phi / i;
61          }
62      }
63      if (n > 1) phi -= phi / n;
64      return phi;
65  }
```

F - Air Is Recursive!

难度	考点
3	简单递归

题目分析

本题是一个较为清晰的递归，只要将题目所给的数学函数用 C 语言代码表述即可，但是需要同学理解递归的含义。 其中，既可以将 `a, b` 作为函数参数，又可以将其作为全局变量使用。

值得一提的是，本题来源于 Lisp 语言的发明者约翰·麦卡锡提出的“麦卡锡 91 函数”（McCarthy 91），其原始表述如下：

$$f(x) = \begin{cases} f(f(x+11)), & \text{if } x \leq 100, \\ x-10, & \text{if } x > 100. \end{cases}$$

有趣的是，这个函数等价于下面这个函数：

$$f(x) = \begin{cases} 91, & \text{if } x \leq 100, \\ x-10, & \text{if } x > 100. \end{cases}$$

两个函数的等价性可以通过数学归纳法进行证明（这里有维基百科上的证明）。同样，根据这个证明我们不难发现函数递归的次数是随输入线性增长的，而且中间结果不会特别大。

本题对其进行了扩展，但是性质并未改变，所以聪明的小朋友们会发现当 $x > a$ 时函数返回 $x - b$，否则返回 $a - b + 1$。

示例代码 1

```c
#include <stdio.h>

int a, b;

int air(int x)
{
    if (x <= a)
        return air(air(x + b + 1));
    else
        return x - b;
    // 也可以写成 return (x<=a) ? air(air(x+b+1)) : x-b;
}

int main()
{
    int x;

    while (~scanf("%d%d%d", &a, &b, &x)) printf("%d\n", air(x));

    return 0;
}
```

示例代码 2

```c
#include <stdio.h>

int air(int a, int b, int x)
{
    return (x <= a) ? air(a, b, air(a, b, x + b + 1)) : x - b;
}

int main()
{
    int a, b, x;

    while (~scanf("%d%d%d", &a, &b, &x)) printf("%d\n", air(a, b, x));

    return 0;
}
```

示例代码 3

```c
#include <stdio.h>

int main()
{
    int a, b, x;

    while (~scanf("%d%d%d", &a, &b, &x)) printf("%d\n", (x <= a) ? a - b + 1 : x - b);

    return 0;
}
```

G - 多项式相加

难度	考点
4	简单数组应用

题目分析

对于本道题，如果多项式的每一项指数部分都在 $[0, 10^5]$ 这样的范围内，则我们只需要设置一个数组 `int coe[100005]`，每次读入的时候以指数部分作为数组下标，将系数记录进 `coe` 数组中即可完成任务。但本题中指数范围较大（`int` 范围，甚至可以是负数），直接采用上述方法进行计算将会导致数组越界，因此需要考虑其他方法。

注意到题目中的 $f(x), g(x)$ 都是以**严格降序**给出的，因此可以考虑这样一种做法：

1. 用两个变量 p, q 记录当前正在处理 $f(x), g(x)$ 从大到小的第 i 项。一开始时 $p = q = 1$。
2. 比较第 p 项和第 q 项的指数部分：
    - 若指数部分相同，说明这两项的系数部分在结果中应当相加，并将 p, q 都向后移动一位；
    - 若 p 对应的指数部分较大，说明只有 $f(x)$ 在结果的这一项中出现，并将 p 向后移动一位；
    - 若 q 对应的指数部分较大，说明只有 $g(x)$ 在结果的这一项中出现，并将 q 向后移动一位；

当 p, q 将 $f(x), g(x)$ 的每一项都计算之后，最后结果的多项式也被计算出来了。可以发现，通过该方法得到的多项式，其指数部分也是严格递减的。由于 p, q 只会移动最多 $n + m$ 次，因此总循环次数不超过 $n + m$ 次，可以在题目要求的时间范围内完成计算。

另外需要注意的是，本题的系数可能超出 `int` 范围，因此需要采用 `long long` 进行存储。

示例程序1

这种写法边计算边输出。

```
#include <stdio.h>

#define N (200000 + 5)

long long a[N], b[N];
long long s[N], t[N];

int main()
{
    int n, m;
    int p = 1, q = 1;
    int i;

    scanf("%d%d", &n, &m);
    for (i = 1; i <= n; i++)
        scanf("%lld%lld", &a[i], &s[i]);
    for (i = 1; i <= m; i++)
        scanf("%lld%lld", &b[i], &t[i]);

    while (p <= n || q <= m)
    {
        if ((p <= n && q <= m && s[p] > t[q]) || q > m)
```

```
        {
            // 只计算 p 的情况：p 对应的指数较小，或另一个数组已经扫描完了
            printf("%lld %lld ", a[p], s[p]);
            p++;
        }
        else if ((p <= n && q <= m && s[p] < t[q]) || p > n)
        {
            // 只计算 q 的情况：q 对应的指数较小，或另一个数组已经扫描完了
            printf("%lld %lld ", b[q], t[q]);
            q++;
        }
        else
        {
            // 合并系数的情况：能来到这个分支，一定是 pq 都没有扫描完，且 f 的第 p 项和 g 的第 y 项系数相等
            if (a[p] + b[q] != 0)
                printf("%lld %lld ", a[p] + b[q], t[q]);
            p++, q++;
        }
    }

    return 0;
}
```

示例程序2

这种写法将计算结果先存下来，最后一起输出。

```
#include <stdio.h>

long long poly_f_coe[100010];
long long poly_f_pow[100010];
long long poly_g_coe[100010];
long long poly_g_pow[100010];

long long poly_ans_coe[200020];
long long poly_ans_pow[200020];

int main()
{
    int n, m, i;
    int current_i = 0;
    int current_j = 0;
    int current_k = 0;

    scanf("%d%d", &n, &m);

    for (i = 0; i < n; i++)
    {
        scanf("%lld%lld", &poly_f_coe[i], &poly_f_pow[i]);
    }
    for (i = 0; i < m; i++)
    {
        scanf("%lld%lld", &poly_g_coe[i], &poly_g_pow[i]);
    }

```

```
    while (current_i < n || current_j < m)
    {
        if (current_i < n)
        {
            if (current_j < m)
            {
                if (poly_f_pow[current_i] > poly_g_pow[current_j])
                {
                    poly_ans_coe[current_k] = poly_f_coe[current_i];
                    poly_ans_pow[current_k] = poly_f_pow[current_i];
                    current_k++;
                    current_i++;
                }
                else if (poly_f_pow[current_i] < poly_g_pow[current_j])
                {
                    poly_ans_coe[current_k] = poly_g_coe[current_j];
                    poly_ans_pow[current_k] = poly_g_pow[current_j];
                    current_k++;
                    current_j++;
                }
                else
                {
                    poly_ans_coe[current_k] = poly_f_coe[current_i] +
poly_g_coe[current_j];
                    poly_ans_pow[current_k] = poly_f_pow[current_i];
                    current_k++;
                    current_i++;
                    current_j++;
                }
            }
            else
            {
                poly_ans_coe[current_k] = poly_f_coe[current_i];
                poly_ans_pow[current_k] = poly_f_pow[current_i];
                current_k++;
                current_i++;
            }
        }
        else
        {
            poly_ans_coe[current_k] = poly_g_coe[current_j];
            poly_ans_pow[current_k] = poly_g_pow[current_j];
            current_k++;
            current_j++;
        }
    }

    for (i = 0; i < current_k; i++)
    {
        if (poly_ans_coe[i] != 0)
        {
            if (i != current_k - 1)
            {
                printf("%d %d ", poly_ans_coe[i], poly_ans_pow[i]);
            }
            else
            {
                printf("%d %d\n", poly_ans_coe[i], poly_ans_pow[i]);
```

```
            }
        }
    }

    return 0;
}
```

H - 拆数字

难度	考点
5	递归，数据范围

题目分析

- 首先应该想到“怎么拆”：对于一个待拆的数字 x，每次找到一个最大的数字 $y = 3^k$ 满足 $y \le x$，在 x 中减去 y，x 变成 $x - y$；循环下去直至 x 为 0。
- 然后发现拆出来的幂数可能也需要拆，但是用来拆幂数的循环语句应该放在哪里呢？观察样例发现：幂数的表示应该出现在整个数的表示之间（某个括号之内）。也就是说，“打印幂数的表示的字符”这个操作，是嵌在“打印整个数的表示的字符”这个操作之间的。这类存在“嵌套”的操作，可以考虑用递归实现。
- 易错点：注意数据范围，在计算过程中可能出现超过 `int` 类型最大值的数字。$3^{19} = 1,162,261,467$，`int` 最大值 $2,147,483,647$，$3^{20} = 3,486,784,401$，$3^{19} < \text{INT\_MAX} < 3^{20}$。若 $x > 3^{19}$，则 `split` 函数里的 `for` 循环第 20 轮结束第 21 轮开始时，会执行 `j * 3 <= x` 这一判断语句，若 `j` 是 `int` 类型，`j * 3` 将发生溢出，造成错误，（OJ 评测结果可能是 `WA` `RE` `TLE`）。

示例代码

```c
#include <stdio.h>

void split(int x)
{
    int i;
    long long j;
    while (x)
    {
        for (i = 0, j = 1; j * 3 <= 1LL * x; j *= 3, i++);
        if (j <= 3)
            printf("%d", j);
        else
        {
            printf("3(");
            split(i);
            printf(")");
        }
        x -= j;
        if (x) printf("+");
    }
}

int main()
{
    int n;

    scanf("%d", &n);
    split(n);

    return 0;
```

```
31 }
```

I - III

难度	考点
6	递归、函数调用

题目分析

题目中给出了当前汉诺塔上每个圆盘的位置，要求移动到第三根柱子上，我们来考虑这样一个函数 `move(a, from, to)` 表示将 a 从起点 $from$ 移动到终点 to，我们为了移动 a，则需要将所有比 a 小的圆盘移动到除了 $from, to$ 之外的第三根柱子上，然后才能移动 a。所以将 a 从 $from$ 移动到 to 可以分为这两步：

1. 将所有比 a 小的圆盘移动到不是 $from$ 也不是 to 的柱子上；
2. 移动 a。

在移动的过程中，a 的起点可能会发生变化，我们用一个数组 $start$ 来记录。

```c
#include <stdio.h>

int start[100]; // 记录每个圆盘所在的位置
int cnt;        // 记录移动的步数

// move函数的作用是将a从起点from移动到终点to
void move(int a, int from, int to)
{
    int i;
    if (from == to)
        return;                 // 如果起点和终点重合，则不用移动
    for (i = a - 1; i > 0; i--) // 将比a小的所有圆盘移动到不是起点也不是终点的柱子上
        move(i, start[i], 6 - from - to);
    // 移动a
    printf("%d:%d-->%d\n", a, from, to);
    start[a] = to;
    cnt++;
}

int main()
{
    int n, i;

    scanf("%d", &n);
    for (i = 1; i <= n; i++)
        scanf("%d", &start[i]);
    for (i = n; i > 0; i--) // 从大到小依次移动圆盘
        move(i, start[i], 3);
    printf("%d", cnt);

    return 0;
}
```

J - 机动冈达战士

难度	考点
6	一维数组/二维数组的应用

题目分析

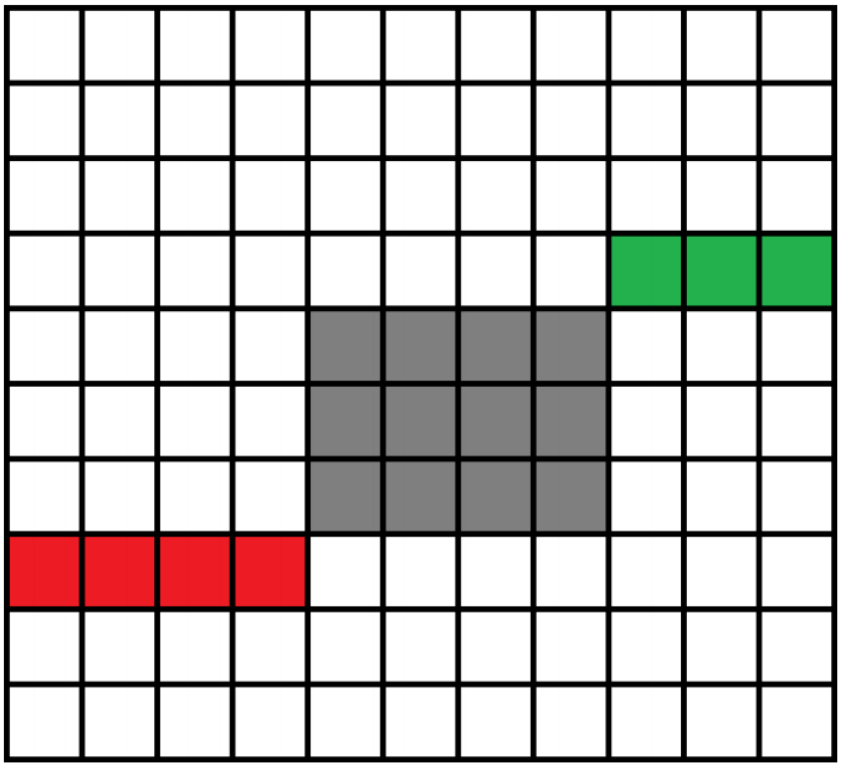
本题需要用到二维数组，如果还不熟悉二维数组的使用，可以使用下面的操作使用一维数组模拟二维数组：

- 设二维数组为 `f[n][m]`，一维数组为 `g[n * m]`，则有 `f[i][j] = g[i * m + j]`。

西诺无法到达目标点当且仅当冈妈选择的子矩阵覆盖了一整行或一整列或 $(1,1)$ 或 (n,m)。

下面我们将一路上击败的敌方机动战士的战斗力之和简称为**该条路径的权值**。

我们将矩阵画成一个方格图以更形象地阐述，如下图所示。设灰色区域为被禁止通过的区域，那么所有合法路径一定至少经过了一个红色格子或绿色格子，同时至少经过了一个红色格子或绿色格子的路径也是合法的。



因此答案即为至少经过了一个红色格子或绿色格子的路径的权值最小值。

我们可以维护从 $(1,1)$ 到 (i,j) 的所有路径的权值最小值 $f_{i,j}$，有以下递推式：

$$f_{i,j}=a_{i,j}+\begin{cases}0, & i=1,j=1\\ f_{i-1,j}, & i>1,j=1\\ f_{i,j-1}, & i=1,j>1\\ \min(f_{i-1,j},f_{i,j-1}), & i>1,j>1\end{cases}$$

以及从 (i,j) 到 (n,m) 的所有路径的权值最小值 $g_{i,j}$，有以下递推式：

$$g_{i,j}=a_{i,j}+\begin{cases}0, & i=n,j=m\\ g_{i+1,j}, & i<n,j=m\\ g_{i,j+1}, & i=n,j<m\\ \min(g_{i+1,j},g_{i,j+1}), & i<n,j<m\end{cases}$$

设冈妈选择的子矩阵的左上角为 (x_1,y_1)，右下角为 (x_2,y_2)，则答案即为：

$$\min(\min_{i=1}^{y_1-1}(f_{x_2+1,i}+g_{x_2+1,i}-a_{x_2+1,i}),\min_{i=y_2+1}^{m}(f_{x_1-1,i}+g_{x_1-1,i}-a_{x_1-1,i}))$$

我们维护每一行的前缀最小值 $pre_{i,j}$ 和后缀最小值 $suf_{i,j}$，定义如下：

$$pre_{i,j}=\min_{k=1}^{j}(f_{i,k}+g_{i,k}-a_{i,k})$$
$$suf_{i,j}=\min_{k=j}^{m}(f_{i,k}+g_{i,k}-a_{i,k})$$

则答案即为：

$$\min(pre_{x_2+1,y_1-1},suf_{x_1-1,y_2+1})$$

对于每组数据，时间复杂度为 $O(nm+q)$。

示例程序1

使用一维数组模拟二维数组。

```
#include <stdio.h>

#define maxn 205 * 205
#define min(a, b) (a < b ? a : b)
#define id(i, j) ((i)*m + j) // 注意宏定义的二义性

typedef long long ll;

int main()
{
    int a[maxn], t, n, m, q, i, j, x1, y1, x2, y2;
    ll f[maxn], g[maxn], pre[maxn], suf[maxn], ans;

    scanf("%d", &t);
    while (t--)
    {
        scanf("%d%d%d", &n, &m, &q);
        for (i = 1; i <= n; i++)
        {
            for (j = 1; j <= m; j++)
            {
                scanf("%d", &a[id(i, j)]);
                f[id(i, j)] = g[id(i, j)] = 1e18;
            }
        }
        for (i = 1; i <= n; i++)
        {
            for (j = 1; j <= m; j++)
            {
                if (i == 1 && j == 1) f[id(i, j)] = 0;
```

```
                if (i - 1 > 0) f[id(i, j)] = min(f[id(i, j)], f[id(i - 1, j)] + a[id(i, j)]);
                if (j - 1 > 0) f[id(i, j)] = min(f[id(i, j)], f[id(i, j - 1)] + a[id(i, j)]);
            }
        }
        for (i = n; i; i--)
        {
            for (j = m; j; j--)
            {
                if (i == n && j == m) g[id(i, j)] = 0;
                if (i + 1 <= n) g[id(i, j)] = min(g[id(i, j)], g[id(i + 1, j)] + a[id(i, j)]);
                if (j + 1 <= m) g[id(i, j)] = min(g[id(i, j)], g[id(i, j + 1)] + a[id(i, j)]);
            }
        }
        for (i = 1; i <= n; i++)
        {
            for (j = 1; j <= m; j++)
            {
                if (j == 1)
                    pre[id(i, j)] = f[id(i, j)] + g[id(i, j)] - a[id(i, j)];
                else
                    pre[id(i, j)] = min(pre[id(i, j - 1)], f[id(i, j)] + g[id(i, j)] - a[id(i, j)]);
            }
            for (j = m; j; j--)
            {
                if (j == m)
                    suf[id(i, j)] = f[id(i, j)] + g[id(i, j)] - a[id(i, j)];
                else
                    suf[id(i, j)] = min(suf[id(i, j + 1)], f[id(i, j)] + g[id(i, j)] - a[id(i, j)]);
            }
        }
        while (q--)
        {
            scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
            if (x2 - x1 + 1 == n || y2 - y1 + 1 == m || (x1 == 1 && y1 == 1) || (x2 == n && y2 == m))
            {
                puts("-1");
                continue;
            }
            ans = 1e18;
            if (x2 < n && y1 > 1) ans = min(ans, pre[id(x2 + 1, y1 - 1)]);
            if (x1 > 1 && y2 < m) ans = min(ans, suf[id(x1 - 1, y2 + 1)]);
            printf("%lld\n", ans);
        }
    }

    return 0;
}
```

示例程序2

直接使用二维数组。

```
#include <stdio.h>

#define maxn 205
#define min(a, b) (a < b ? a : b)

typedef long long ll;

int main()
{
    int a[maxn][maxn], t, n, m, q, i, j, x1, y1, x2, y2;
    ll f[maxn][maxn], g[maxn][maxn], pre[maxn][maxn], suf[maxn][maxn], ans;

    scanf("%d", &t);
    while (t--)
    {
        scanf("%d%d%d", &n, &m, &q);
        for (i = 1; i <= n; i++)
        {
            for (j = 1; j <= m; j++)
            {
                scanf("%d", &a[i][j]);
                f[i][j] = g[i][j] = 1e18;
            }
        }
        for (i = 1; i <= n; i++)
        {
            for (j = 1; j <= m; j++)
            {
                if (i == 1 && j == 1) f[i][j] = 0;
                if (i - 1 > 0) f[i][j] = min(f[i][j], f[i - 1][j] + a[i]
[j]);
                if (j - 1 > 0) f[i][j] = min(f[i][j], f[i][j - 1] + a[i]
[j]);
            }
        }
        for (i = n; i; i--)
        {
            for (j = m; j; j--)
            {
                if (i == n && j == m) g[i][j] = 0;
                if (i + 1 <= n) g[i][j] = min(g[i][j], g[i + 1][j] + a[i]
[j]);
                if (j + 1 <= m) g[i][j] = min(g[i][j], g[i][j + 1] + a[i]
[j]);
            }
        }
        for (i = 1; i <= n; i++)
        {
            for (j = 1; j <= m; j++)
            {
                if (j == 1)
                    pre[i][j] = f[i][j] + g[i][j] - a[i][j];
                else
```

```
                    pre[i][j] = min(pre[i][j - 1], f[i][j] + g[i][j] - a[i]
[j]);
            }
            for (j = m; j; j--)
            {
                if (j == m)
                    suf[i][j] = f[i][j] + g[i][j] - a[i][j];
                else
                    suf[i][j] = min(suf[i][j + 1], f[i][j] + g[i][j] - a[i]
[j]);
            }
        }
        while (q--)
        {
            scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
            if (x2 - x1 + 1 == n || y2 - y1 + 1 == m || (x1 == 1 && y1 == 1)
|| (x2 == n && y2 == m))
            {
                puts("-1");
                continue;
            }
            ans = 1e18;
            if (x2 < n && y1 > 1) ans = min(ans, pre[x2 + 1][y1 - 1]);
            if (x1 > 1 && y2 < m) ans = min(ans, suf[x1 - 1][y2 + 1]);
            printf("%lld\n", ans);
        }
    }

    return 0;
}
```