E6 - Solution

A 字符串距离

难度	考点
1	字符串、strlen

题目分析

读入两个字符串后，可以先用 `strlen` 获取到某字符串的长度，然后再遍历每个位置，计算每个位置的“距离”加到总距离上，最后输出总距离。主要考察字符串的基本使用。

示例代码

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

// 字符数组和字符串，就像杯子和水，我们要先有足够大的杯子，再用杯子装水
char a[1005], b[1005];

// 计算两个 char 的绝对值
int myAbs(char a, char b) {
    // 三目简写
    return a - b > 0 ? a - b : b - a;
}

int main() {
    // 读入字符串
    scanf("%s", a);
    scanf("%s", b);
    int len = strlen(a), sum = 0;

    for(int i = 0; i < len ;i++)
        sum += myAbs(a[i], b[i]);
    printf("%d", sum);
}
```

B 努力学习的小懒獭

难度	考点
2	字符串

题目分析

按照与 A 题类似的方法进行遍历，在遍历的过程中我们可以维护一个变量 `cnt`，表示算上今天小懒獭连续学习的天数，则每一天对小懒獭总智力的贡献就是当天的 `cnt`，然后：

- 若当天是学习日，则 cnt + 1
- 若当天是偷懒日，则 cnt 重置为 `0`

示例代码

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

char str[1005];

int main() {
    scanf("%s", str);
    int len = strlen(str), ans = 0, cnt = 0;
    for(int i = 0; i < len ;i++) {
        if(str[i] == 'S') {
            cnt += 1;
            ans += cnt;
        } else
            cnt = 0;
    }
    printf("%d", ans);
    return 0;
}
```

`B` 三生万物2024

难度	考点
2	进制转换，输出控制

题目分析

题目大意：

将一个数字转化为三进制，按特定格式输出。

解题思路：

与二进制转换类似，不断除以 3 直至 0 ，并保存每次的余数，即为这一项的系数。注意，（1）去掉系数为 0 的项，（2）系数为 1 的项不输出系数 1 。

示例代码

```
#include <stdio.h>

int main() {
    int x;
    while (~scanf("%d", &x)) {
        int i = 0, first = 1;
        printf("%d =", x);
        while (x) {
            if (x % 3 != 0) {
                if (first == 0) {
                    printf("+");
                }
                first = 0;
                if (x % 3 == 1) {
                    printf(" 3^%d ", i);
                } else {
                    printf(" %d*3^%d ", x % 3, i);
                }
            }
            i++;
            x /= 3;
        }
        printf("\n");
    }

    return 0;
}
```

D 小懒獭与矩阵乘法

难度	考点
3	二维数组

题目分析

先用二维数组 a 和 b 记录两个矩阵的各个元素，然后用三层循环：第一层 i 从 1 到 n，第二层 j 从 1 到 k，前两层循环代表结果矩阵元素的位置；第三层 p 从 1 到 m，按照公式计算矩阵乘积结果第 i 行 j 列元素的值。

分析数据范围，不会超过 `int`。

示例代码

```
#include <stdio.h>

int a[105][105], b[105][105];
int ans[105][105];
```

```
int main() {
    int n, m, k;
    scanf("%d%d%d", &n, &m, &k);
    for(int i = 1; i <= n; i++) {
       for(int j = 1; j <= m; j++)
           scanf("%d", &a[i][j]);
    }

    for(int i = 1; i <= m; i++) {
        for(int j = 1; j <= k; j++)
            scanf("%d", &b[i][j]);
    }

    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= k; j++) {
            for(int p = 1; p <= m; p++) {
                ans[i][j] += a[i][p] * b[p][j];
            }
        }
    }

    for(int i = 1; i <= n; ++i) {
        for(int j = 1; j <= k; ++j) {
            printf("%d ", ans[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

E 摩卡与数独

难度	考点
4	二维数组

题目分析

本题就是判断一个数独是否是正确的数独局面，正确的做法是判断每一行、每一列、每一宫是不是只出现且只出现一次 1-9，有如下几种常见的错误做法：

- 判断每一行、每一列、每一宫的和都是 45
- 判断每一行、每一列、每一宫的和都是 45 且积是 362880（这个也是错的，因为符合这个条件的几个数除了 1 2 3 4 5 6 7 8 9 外还有 1 2 4 4 4 5 7 9 9），这种情况下的一组 hack 数据如下：

```
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0

1 2 4 4 4 5 7 9 9
4 4 5 7 9 9 1 2 4
7 9 9 1 2 4 4 4 5
2 4 4 4 5 7 9 9 1
4 5 7 9 9 1 2 4 4
9 9 1 2 4 4 4 5 7
4 4 4 5 7 9 9 1 2
5 7 9 9 1 2 4 4 4
9 1 2 4 4 4 5 7 9
```

- 没有跟原数对比，这种情况的一组 hack 数据如下：

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

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

示例代码

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

int a[10][10]; // 表示数独的初始填充数据
int b[10][10]; // 数独的当前验证数据
int vis[10];   // 表示当前行的验证情况

void clear() {
    memset(vis, 0, sizeof(vis));
}

int main() {
    int c;
    scanf("%d", &c);

    while (c--) {
        for (int i = 1; i <= 9; i++) {
```

```
        for (int j = 1; j <= 9; j++) {
            scanf("%d", &a[i][j]);
        }
    }

    int flag = 1; // 用 flag 标记是否验证通过

    // 验证每行的数字是否重复
    for (int i = 1; i <= 9; i++) {
        clear();
        for (int j = 1; j <= 9; j++) {
            vis[b[i][j]] = 1;
        }
        for (int i = 1; i <= 9; i++) {
            if (!vis[i]) {
                flag = 0;
            }
        }
    }

    // 验证每列
    for (int j = 1; j <= 9; j++) {
        clear();
        for (int i = 1; i <= 9; i++) {
            vis[b[i][j]] = 1;
        }
        for (int i = 1; i <= 9; i++) {
            if (!vis[i]) {
                flag = 0;
            }
        }
    }

    // 验证每个小九宫格
    for (int k = 1; k <= 9; k++) {
        int x = (k - 1) / 3 * 3 + 1;
        int y = (k - 1) % 3 * 3 + 1;
        clear();
        for (int i = x; i <= x + 2; i++) {
            for (int j = y; j <= y + 2; j++) {
                vis[b[i][j]] = 1;
            }
        }
        for (int i = 1; i <= 9; i++) {
            if (!vis[i]) {
                flag = 0;
            }
        }
    }

    if(flag != 0)
        printf("Moca finish this sudoku perfectly!\n");
    else
        printf("Moca is so careless!\n");
}
```

```
    return 0;
}
```

因为把所有括号都写上了所以看上去可能比较繁琐，但是这道题的思路还是清晰明确的。

F 多项式导数计算

难度	考点
4	字符串

题目分析

显然，多项式是由单项式组成的，而单项式之间由 `+` 分开，因此我们可以分别列出每个单项式，并且获得对应单项式的系数和指数，进行计算就可以啦！

为了方便写代码，我们可以在字符串的末尾补充一个 `+`，这样每当遇到一个 `+`，就提取一个单项式计算。

本题中还要注意许多细节：系数、指数为 1 的时候应该省略，但是常数项的 1 不能省略；导数系数为 0 即常数项的导数应该省略，但如果只有常数项就要输出 0 ......同学们写代码的时候一定要注意这些细节，调试时也可以尝试这些特殊输入。

示例代码 1

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
char tmp[20];
char s[1008611];
int isPrint;
void calc();
int main()
{
    scanf("%s", s);
    s[strlen(s)] = '+';
    printf("f'(x)=");
    for(int i = 5, len = strlen(s), last = 5;i < len;i++)
        if('+' == s[i])
        {
            s[i] = 0;
            strcpy(tmp, s + last);
            calc();
            last = i + 1;
        }
    if(!isPrint)
        printf("0");
    return 0;
}
void calc()
{
```

```
    if(NULL == strchr(tmp, 'x'))
        return;
    int a = 0, b = 0, i = 0;
    if('x' == tmp[0])
        a = 1;
    else
        while(isdigit(tmp[i]))
            a = a * 10 + tmp[i++] - '0';
    i += 2;
    if(strlen(tmp) <= i)
        b = 1;
    else
        while(isdigit(tmp[i]))
            b = b * 10 + tmp[i++] - '0';
    a *= b--;
    if(isPrint++)
        printf("+");
    if(1 != a||0 == b)
        printf("%d", a);
    if(0 < b)
        printf("x");
    if(1 < b)
        printf("^%d", b);
}
```

示例代码 2

`scanf` 的功能相当强大，此代码给出一种读入方式，感兴趣的同学可以了解。

```
#include <stdio.h>

int a, b, t;
char s[64];

int main() {
    scanf("f(x)=");
    printf("f'(x)=");
    s[0] = '+';
    while (scanf("%[+]", s) != EOF) { // 尝试读入 +
        a = 1;
        scanf("%d", &a); // 尝试读入 a ，如果匹配不到数字，则 a 的值仍为 1
        if (scanf("%[x]", s) == 1) { // 尝试读入 x
            b = 1;
            scanf("^%d", &b); // 尝试读入 ^b ，如果匹配不到 ^ ，则 b 的值仍为 1
        } else { break; } // 读入不到 x 时为常数项，其导数为 0 ，且由题知此项为末项，可以结束循环
        a = a * b, b -= 1;
        if (a != 0) {
            if (t++ != 0) { printf("+"); } // 第一个输出前面不带 +
            if (b == 0 || a != 1) { printf("%d", a); } // 输出系数
            if (b != 0) { printf("x"); } // 非常数项输出 x
            if (b > 1) { printf("^%d", b); } // 指数不为 1 时才输出
        }
    }
    if (t == 0) { printf("0"); } //什么都没输出（即输入为常数）时输出 0
```

```
    return 0;
}
```

G 某程设助教大战高代

难度	考点
4	高斯消元法，二维数组

题目分析

本题模拟量大，推荐将每个步骤封装起来，避免循环嵌套次数过多，不好 debug。

本题最大坑点在于：判断一个浮点数是否为 0，不能直接判断，而是应该使用 `fabs` 函数。如果你直接判断，会得到 `WA 0.4` 的结果。

还有一个可能的坑点，最后输出时，应该 `printf("%.0f ", a[i][j]);` 而不是 `printf("%d", (int)a[i][j]);`，因为经过多次计算，可能会出现 `1.9999` 这样的轻微的精度丢失，直接转换为 `int` 会把小数部分截断。

示例代码

```
#include <stdio.h>
#include <math.h>
double a[105][105];
int n;
void swap(int x, int y) // 交换第x行与第y行
{
    double tmp;
    for (int i = 0; i < n + 1; i++)
    {
        tmp = a[x][i];
        a[x][i] = a[y][i];
        a[y][i] = tmp;
    }
}

void mult(int x, double m) // 把第x行全部乘以m
{
    for (int i = 0; i < n + 1; i++)
    {
        a[x][i] *= m;
    }
}

void add_mult(int x, int y, double m) // 把第y行的m倍给加到第x行上
{
    for (int i = 0; i < n + 1; i++)
    {
        a[x][i] += a[y][i] * m;
    }
}
```

```
void solve()
{
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n + 1; j++)
        {
            scanf("%lf", &a[i][j]);
        }
    }
    for (int i = 0; i < n; i++)
    {
        if (fabs(a[i][i]) < 1e-7) // 不要直接判等，而是用fabs
        {
            for (int j = i + 1; j < n; j++)
            {
                if (fabs(a[j][i]) > 1e-7)
                {
                    swap(i, j);
                    break;
                }
            }
        }
        mult(i, 1 / a[i][i]);
        for (int j = 0; j < n; j++)
        {
            if (j == i)
                continue;
            add_mult(j, i, -a[j][i]);
        }
    }
    for (int i = 0; i < n; i++)
    {
        printf("%.0f ", a[i][n]);
    }
    printf("\n");
}
int main(void)
{
    int t;
    scanf("%d", &t);
    while (t--)
    {
        solve();
    }
    return 0;
}
```

H 寻找因数 (hard version)

难度	考点
5	质数，简单数论

题目分析

对于正整数 m，对其进行质因数分解，我们可以得到类似这样的式子：

$m = p_1^{c_1} p_2^{c_2} \cdots p_n^{c_n}$

那么 m 的正因数有多少个呢？注意到对于 m 的因数，可以看作是从 m 的质因数中取出若干个进行相乘得到。对于 p_1，可以取 0 到 c_1 个进行相乘。对于 p_2，可以取 0 到 c_2 个进行相乘，以此类推。最终我们得到 m 的因数个数为：

$\prod_{i=1}^{n}(c_i+1)$

那么怎么对于 m 进行质因数分解呢？我们只需对 a_1 到 a_n 分别进行质因数分解，然后把对于它们分解出来的结果组合起来即可。更具体地，我们把它们的质因数全部放到一个数组里，然后将这个数组从小到大排好序，我们就能知道不同的质因数分别出现了多少次了。又发现 10 个不超过 10^9 的数的质因数总个数不可能多于 320 个，因此可以使用冒泡排序。

坑点1：1 的正因数只有它本身一个。因此模拟的时候可能需要对于这个情况进行特殊判断。注意不要将数组里的全部数相乘之后再判断这个结果是否是 1，因为会溢出，而本题数据点很不怀好意的加入了相乘溢出后结果恰好为 1 的数据。

坑点2：本题的结果可能会超过 `int` 范围，例如我可以将最小的 45 个质数分配到 10 个不超过 10^9 的正整数中，此时答案为 2^{45}，超过了 `int` 最大的表示范围，但不会超过 `long long` 的。因此记得使用 `long long`。

示例代码

```c
#include <stdio.h>
#include <stdlib.h>
int p[1024], top;
void bubble(int a[], int len)
{
    for (int i = 0; i < len; i++)
    {
        for (int j = len - 1; j > i; j--)
        {
            if (a[j - 1] > a[j])
            {
                int tmp = a[j - 1];
                a[j - 1] = a[j];
                a[j] = tmp;
            }
        }
    }
}
void solve()
{
    int n;
    int arr[10];
```

```c
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
    }
    top = 0;
    for (int i = 0; i < n; i++)
    {
        for (int a = 2; a * a <= arr[i]; a++)
        {
            if (arr[i] % a)
                continue;
            while (arr[i] % a == 0)
            {
                p[top++] = a;
                arr[i] /= a;
            }
        }
        if (arr[i] != 1)
            p[top++] = arr[i];
    }
    if (top == 0) // m 不能被分解出任何质因数，说明 m 等于 1
    {
        printf("1\n");
        return;
    }
    bubble(p, top);
    long long ans = 1; // 记得开long long
    int cnt = 1;
    for (int i = 1; i < top; i++)
    {
        if (p[i] == p[i - 1])
        {
            cnt++;
        }
        else
        {
            ans *= (cnt + 1);
            cnt = 1;
        }
    }
    ans *= (cnt + 1);
    printf("%lld\n", ans);
}
int main(void)
{
    int t;
    scanf("%d", &t);
    while (t--)
    {
        solve();
    }

    return 0;
}
```

示例代码 (qsort 版本，最后一节课会学，强烈推荐)

```c
#include <stdio.h>
#include <stdlib.h>
int cmp(const void *a, const void *b)
{
    int x, y;
    x = *(int *)a;
    y = *(int *)b;
    return (x > y) - (x < y);
}
int p[1024], top;
void solve()
{
    int n;
    int arr[10];
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &arr[i]);
    }
    top = 0;
    for (int i = 0; i < n; i++)
    {
        for (int a = 2; a * a <= arr[i]; a++)
        {
            if (arr[i] % a)
                continue;
            while (arr[i] % a == 0)
            {
                p[top++] = a;
                arr[i] /= a;
            }
        }
        if (arr[i] != 1)
            p[top++] = arr[i];
    }
    if (top == 0) // m 不能被分解出任何质因数，说明 m 等于 1
    {
        printf("1\n");
        return;
    }
    qsort(p, top, sizeof(p[0]), cmp);
    long long ans = 1; // 记得开long long
    int cnt = 1;
    for (int i = 1; i < top; i++)
    {
        if (p[i] == p[i - 1])
        {
            cnt++;
        }
        else
        {
            ans *= (cnt + 1);
            cnt = 1;
```

```
        }
    }
    ans *= (cnt + 1);
    printf("%lld\n", ans);
}
int main(void)
{
    int t;
    scanf("%d", &t);
    while (t--)
    {
        solve();
    }

    return 0;
}
```

I ddz 的开灯游戏

难度	考点
6	找规律、二分

题目分析

首先打表看看有没有规律

k	结果	k	结果	k	结果
1	2	8	11	15	19
2	3	9	12	16	20
3	5	10	13	17	21
4	6	11	14	18	22
5	7	12	15	19	23
6	8	13	17	20	24
7	10	14	18	21	26

发现结果序列是将自然数中的完全平方数去掉之后的序列（在拓展知识中给出证明），易验证结果 n 是满足 $n-\lfloor\sqrt{n}\rfloor=k$ 的最小的 n ，对此式做二分即可得到结果（详见示例代码一）。当然对于没有完全平方数的自然数数列我们有结论 $n=\lfloor k+\sqrt{k}+0.5\rfloor$ ，直接计算此式也可（详见示例代码二）。

示例代码一

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

void work() {
    long long k, mid, l = 0, r = 2000000000000000000ll;
    scanf("%lld", &k);
    while (l < r) {
        mid = (l + r) / 2;
        if (mid - (long long) floor(sqrt((double) mid)) < k) {
            l = mid + 1;
        } else {
            r = mid;
        }
    }
    printf("%lld\n", l);
}

int main() {
    int t;
    scanf("%d", &t);
    while(t--) work();
    return 0;
}
```

示例代码二

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

int main(){
    int t;
    scanf("%d", &t);
    while(t--){
        long long k;
        scanf("%lld", &k);
        printf("%lld\n", k + (int)(sqrt(k) + 0.5));
    }
    return 0;
}
```

拓展知识

首先对于第 x 盏灯来说，它最终是开还是关与灯总数 n 无关，因为只有所有 $i \le x$ 的操作对第 x 盏灯有效。而第 x 盏灯被操作的次数取决于 x 的因数个数，完全平方数的因数个数是奇数，其余的数的因数个数都是偶数。所以所有完全平方数对应的灯被操作奇数次，最终被关闭，其余的数对应的灯最终是打开状态。所以结果序列是将自然数中的完全平方数去掉之后的序列。

J 扫雷问题01

难度	考点
7	前缀和、二维前缀和

题目分析

本题考点较为普通，但思考量与代码量均较大，但得 0.5 分还是很容易的。

对于每个询问，关注其中每个雷对答案的贡献值。

可以将所有的雷分为三种：在内部，在角上，在边上。

对于内部的雷，每个雷贡献值为周围八格非雷格的个数。对于所有内部雷的贡献度，可以预处理一个二维前缀和，然后在询问时以 $O(1)$ 时间复杂度求得结果。

对于角部的雷，由于其最多只有 4 个，其贡献值直接枚举即可。

对于边上的雷，以左侧列举例，其贡献度为周围八格非雷格的个数减去左侧三格非雷格的个数。对于所有左侧雷的贡献度，可以预处理一个前缀和，然后在询问时以 $O(1)$ 时间复杂度求得结果。同理，上侧、下侧、右侧也要如此处理。

因此，最终要预处理四个前缀和和一个二维前缀和，以保证在询问时以 $O(1)$ 时间复杂度得到结果。

总时间复杂度：$O(nm+q)$

有以下两个容易出现问题的地方：

1. 并不是所有询问角部的雷都有 4 个，有时只有 2 个，有时只有 1 个。
2. 并不是所有时候边上/内部的雷都存在，并且不存在时各会有多种情况，注意用前缀和计算出的价值不能为负数。

示例代码

std:

```
#include<stdio.h>
int ray[900010];
int left[900010];//雷对其左侧格的贡献度
int midinlr[900010];// 雷对其当列的贡献度
int right[900010];//雷对其右侧格的贡献度
int up[900010];//雷对其上侧格的贡献度
int midinud[900010];// 雷对其当行的贡献度
int down[900010];//雷对其下侧格的贡献度
int sum0[900010];//雷对周围8格的贡献度
signed main()
{
    int n,m,q,i,j,N,M,sum,x1,y1,x2,y2,ans,p1,q1,p2,q2;
    scanf("%d%d%d",&n,&m,&q);
    N=n+2;
    M=m+2;
    for(i=0;i<N;i++)
    {
        for(j=0;j<M;j++)
        {
```

```
            if(i==0||i==N-1||j==0||j==M-1)
            {
                ray[i*M+j]=1;
                continue;
            }
            scanf("%d",&ray[i*M+j]);
        }
    }
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=m;j++)
        {
            if(ray[i*M+j]==0)continue;
            //赋left
            sum=0;
            if(ray[(i-1)*M+(j-1)]==0)sum++;
            if(ray[i*M+(j-1)]==0)sum++;
            if(ray[(i+1)*M+(j-1)]==0)sum++;
            left[i*M+j]=sum;
            //赋right
            sum=0;
            if(ray[(i-1)*M+(j+1)]==0)sum++;
            if(ray[i*M+(j+1)]==0)sum++;
            if(ray[(i+1)*M+(j+1)]==0)sum++;
            right[i*M+j]=sum;
            //赋midinlr
            sum=0;
            if(ray[(i-1)*M+j]==0)sum++;
            if(ray[(i+1)*M+j]==0)sum++;
            midinlr[i*M+j]=sum;
            //赋up
            sum=0;
            if(ray[(i-1)*M+(j-1)]==0)sum++;
            if(ray[(i-1)*M+j]==0)sum++;
            if(ray[(i-1)*M+(j+1)]==0)sum++;
            up[i*M+j]=sum;
            //赋down
            sum=0;
            if(ray[(i+1)*M+(j-1)]==0)sum++;
            if(ray[(i+1)*M+j]==0)sum++;
            if(ray[(i+1)*M+(j+1)]==0)sum++;
            down[i*M+j]=sum;
            //赋midinud
            sum=0;
            if(ray[i*M+(j-1)]==0)sum++;
            if(ray[i*M+(j+1)]==0)sum++;
            midinud[i*M+j]=sum;
            //赋sum0
            sum0[i*M+j]=up[i*M+j]+down[i*M+j]+midinud[i*M+j];
            //可以换种写法，另一种写法代码长度更短，二者等价
        }
    }
    //求left right midinlr的前缀和
    for(j=1;j<=m;j++)
    {
        for(i=1;i<=n;i++)
```

```
        {
            left[i*M+j]=left[i*M+j]+left[(i-1)*M+j];
            right[i*M+j]=right[i*M+j]+right[(i-1)*M+j];
            midinlr[i*M+j]=midinlr[i*M+j]+midinlr[(i-1)*M+j];
        }
     }
    //求up down midinud的前缀和
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=m;j++)
        {
            up[i*M+j]=up[i*M+j]+up[i*M+(j-1)];
            down[i*M+j]=down[i*M+j]+down[i*M+(j-1)];
            midinud[i*M+j]=midinud[i*M+j]+midinud[i*M+(j-1)];
        }
    }
    //求sum0的二维前缀和
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=m;j++)
        {
            sum0[i*M+j]=sum0[i*M+(j-1)]+sum0[(i-1)*M+j]-sum0[(i-1)*M+(j-
1)]+sum0[i*M+j];
        }
    }
    //最终求和
    while(q--)
    {
        scanf("%d%d%d%d",&x1,&y1,&x2,&y2);
        p1=x1+1;
        q1=y1+1;
        p2=x2-1;
        q2=y2-1;
        //内部ans
        if(p1<=p2&&q1<=q2)
        {
            ans=sum0[p2*M+q2]-sum0[(p1-1)*M+q2]-sum0[p2*M+(q1-1)]+sum0[(p1-1)*M+
(q1-1)];
        }
        else
        {
            ans=0;
        }
        //左右列ans
        if(x1+2<=x2)
        {
            ans+=midinlr[(x2-1)*M+y1]-midinlr[x1*M+y1];
            if(y1!=y2)
            {
                ans+=midinlr[(x2-1)*M+y2]-midinlr[x1*M+y2];
                ans+=right[(x2-1)*M+y1]-right[x1*M+y1];
                ans+=left[(x2-1)*M+y2]-left[x1*M+y2];
            }
        }
        //上下行ans
        if(y1+2<=y2)
```

```
{
    ans+=midinud[x1*M+(y2-1)]-midinud[x1*M+y1];
    if(x1!=x2)
    {
        ans+=midinud[x2*M+(y2-1)]-midinud[x2*M+y1];
        ans+=down[x1*M+(y2-1)]-down[x1*M+y1];
        ans+=up[x2*M+(y2-1)]-up[x2*M+y1];
    }
}
//四个角
if(x1!=x2&&y1!=y2)
{
    if(ray[x1*M+y1]==1)
    {
        sum=0;
        if(ray[(x1+1)*M+y1]==0)sum++;
        if(ray[(x1+1)*M+(y1+1)]==0)sum++;
        if(ray[x1*M+(y1+1)]==0)sum++;
        ans+=sum;
    }
    if(ray[x2*M+y1]==1)
    {
        sum=0;
        if(ray[(x2-1)*M+y1]==0)sum++;
        if(ray[(x2-1)*M+(y1+1)]==0)sum++;
        if(ray[x2*M+(y1+1)]==0)sum++;
        ans+=sum;
    }
    if(ray[x1*M+y2]==1)
    {
        sum=0;
        if(ray[(x1+1)*M+y2]==0)sum++;
        if(ray[(x1+1)*M+(y2-1)]==0)sum++;
        if(ray[x1*M+(y2-1)]==0)sum++;
        ans+=sum;
    }
    if(ray[x2*M+y2]==1)
    {
        sum=0;
        if(ray[(x2-1)*M+y2]==0)sum++;
        if(ray[(x2-1)*M+(y2-1)]==0)sum++;
        if(ray[x2*M+(y2-1)]==0)sum++;
        ans+=sum;
    }
}
else if(y1!=y2)//x1==x2
{
    if(ray[x1*M+y1]==1)
    {
        sum=0;
        if(ray[x1*M+(y1+1)]==0)sum++;
        ans+=sum;
    }
    if(ray[x1*M+y2]==1)
    {
        sum=0;
```

```
                if(ray[x1*M+(y2-1)]==0)sum++;
                ans+=sum;
            }
         }
        else if(x1!=x2)//y1==y2
        {
            if(ray[x1*M+y1]==1)
            {
                sum=0;
                if(ray[(x1+1)*M+y1]==0)sum++;
                ans+=sum;
            }
            if(ray[x2*M+y1]==1)
            {
                sum=0;
                if(ray[(x2-1)*M+y1]==0)sum++;
                ans+=sum;
            }
        }
        printf("%d\n",ans);
    }
    return 0;
}
```