C7 - Solution

A 我怎可将你与夏日比拟

难度	考点
1	二维数组，字符串

题目分析

建立二维字符数组，初始化每行为对应字符串，每组数据输出对应的字符串即可。

示例代码

```
#include <stdio.h>
char s[14][100] =
{
    "Shall I compare thee to a summer's day?",
    "Thou art more lovely and more temperate:",
    "Rough winds do shake the darling buds of May,",
    "And summer's lease hath all too short a date;",
    "Sometime too hot the eye of heaven shines,",
    "And often is his gold complexion dimm'd;",
    "And every fair from fair sometime declines,",
    "By chance or nature's changing course untrimm'd;",
    "But thy eternal summer shall not fade,",
    "Nor lose possession of that fair thou ow'st;",
    "Nor shall death brag thou wander'st in his shade,",
    "When in eternal lines to time thou grow'st:",
    "    So long as men can breathe or eyes can see,",
    "    So long lives this, and this gives life to thee."
};
int main()
{
    int n;
    while(~scanf("%d", &n))
        puts(s[n - 1]);
    return 0;
}
```

示例代码 - 2

利用指针数组实现。区别在于该方法指针指向的字符串为常量，不可修改。

```
#include <stdio.h>
char *s[14] =
{
    "Shall I compare thee to a summer's day?",
    "Thou art more lovely and more temperate:",
    "Rough winds do shake the darling buds of May,",
```

```
    "And summer's lease hath all too short a date;",
    "Sometime too hot the eye of heaven shines,",
    "And often is his gold complexion dimm'd;",
    "And every fair from fair sometime declines,",
    "By chance or nature's changing course untrimm'd;",
    "But thy eternal summer shall not fade,",
    "Nor lose possession of that fair thou ow'st;",
    "Nor shall death brag thou wander'st in his shade,",
    "When in eternal lines to time thou grow'st:",
    "    So long as men can breathe or eyes can see,",
    "    So long lives this, and this gives life to thee."
};
int main()
{
    int n;
    while(~scanf("%d", &n))
        puts(s[n - 1]);
    return 0;
}
```

B 小亮学矩阵加减法

难度	考点
1	二维数组 循环

题目分析

矩阵的加减法是两个矩阵对应元素的加减法。

需要注意的是计算结果可能会超出 `int` 范围，需要保证计算范围和输出数组都需要在 `long long` 范围内。

示例代码

```
#include <stdio.h>

int main()
{
    int a[50][50]={0};
    int b[50][50]={0};
    long long c[50][50]={0};
    int m,n,k;int s;
    scanf("%d%d%d",&m,&n,&s);
    for (int i = 0; i < m; ++i) {//矩阵A
        for (int j = 0; j < n; ++j) {
            scanf("%d", &a[i][j]);
        }
    }
    for (int i = 0; i < m; ++i) {//矩阵B
        for (int j = 0; j < n; ++j) {
            scanf("%d", &b[i][j]);
```

```
        }
    }

    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            if (s==1) c[i][j]=1ll*a[i][j]+b[i][j];//矩阵相加，注意1ll
            else c[i][j]=1ll*a[i][j]-b[i][j];//矩阵相减，注意1ll
        }
    }
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            printf("%lld ",c[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

C 寻找字符

难度	考点
2	指针，字符串函数

题目分析

利用 `strchr` 和 `strrchr` 函数搜索字符串 s 中字符 c 第一次出现和最后一次出现的位置。代码如下：

```
char *p = strchr(s, c);
char *q = strrchr(s, c);
```

根据指针 `p, q` 的值进行判断：

- 若 c 在 s 中未出现，则 `p, q` 均为空；
- 若 c 在 s 中恰好出现一次，则 `p, q` 相等，均指向 c 在 s 中唯一出现的位置，`p - s` 的值就是答案；
- 若 c 在 s 中出现不止一次，则 `p, q` 分别指向 c 在 s 中第一次出现的位置和最后一次出现的位置，`q - p` 就是答案。

分类讨论即可。

示例代码

```
#include <stdio.h>
#include <string.h>
char s[100005], c;
int main()
{
    gets(s);
    while(~(c = getchar()))
```

```
    {
        char *p = strchr(s, c), *q = strrchr(s, c);
        if(p == NULL)
            printf("-1\n");
        else if(p == q)
            printf("%d\n", p - s);
        else
            printf("%d\n", q - p);
    }
    return 0;
}
```

D One Last Kiss

难度	考点
2~3	子串查找函数`strstr`的使用，英文字符大小写转换

题目分析

根据题目要求，需要使用不定组输入语句读入多组字符串。对每组输入的字符串`str`，要在忽略大小写的条件下，寻找字符串`str`中`kiss`子串的出现次数，与最后一次出现时子串首字母在原字符串中的位置下标

题目要求忽略大小写，因此我们可以直接在读入字符串`str`后，把所有字符转成小写，然后直接使用`strstr`函数在字符串`str`中查找全小写的`kiss`子串的首地址值。

如果查找操作返回地址值为`NULL`说明目标子串不存在，如果返回地址值不为`NULL`，说明目标子串存在，记录本次查找到的子串首地址，同时将计数变量自增 1，然后从这次查到的位置的**后一个位置**开始，继续循环执行查找、计数累加、记录首地址的操作，直到查找失败返回`NULL`，此时记录下的最后一个地址就是`str`字符串中最后一个目标子串的首地址

因为一个数组在内存中是连续存储的，而数组首地址指向的是数组中下标为`0`的元素的位置，将数组中某个指定位置的地址值与数组首地址作差后，得到的结果就是两个位置的下标差，即指定位置在数组中的下标。所以将前面查到的目标子串最后一次出现位置的首地址减去`str`数组首地址，就得到了目标子串首字母在`str`数组中的位置下标。保险起见，推荐使用强制类型转换将结果类型转换为`int`再输出

示例代码

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

int main() {
    char str[1001] = {};
    while (scanf("%s", str) != EOF) {   //因为输入字符串中不包含空格，所以可以使用scanf读入每组的字符串
        //也可以使用while (gets(str) != NULL) {}
        int cnt = 0;
        for (int i = 0; str[i] != '\0'; ++i) {
```

```
            //使用ctype.h库中的tolower函数，当传入大写字母时返回小写字母的ASCII值，否则返回传入字符的原ASCII值
            str[i] = (char) tolower(str[i]);
            //也可以使用如下代码，利用ASCII值之差实现大小写转换
            //if ('A' <= str[i] && str[i] <= 'Z') {
            //    str[i] = (char) (str[i] - 'A' + 'a');
            //}
        }
        char *aim = NULL, *tmp = strstr(str, "kiss");
        while (tmp != NULL) {   //当找到目标子串时，累加计数并把该目标子串的首地址赋给指针变量aim
            cnt++;
            aim = tmp;
            tmp = strstr(aim + 1, "kiss");  //从成功匹配位置的下一个位置开始，再次查找目标子串
        }
        if (cnt > 0) {
            printf("%d %d\n", cnt, (int) (aim - str));
        } else {
            printf("0 -1\n");
        }
    }

    return 0;
}
```

E 面壁计划

难度	考点
3~4	字符串函数

题目分析

主要考察对于 $strlen$, $strcpy$, $strcmp$ 的函数运用。

本题的思路非常清晰，按照所给的步骤一步一步模拟即可。实现时，有一些细节需要注意，例如不要用变量定义数组的长度，记得字符串是以 `'\0'` 结尾的。

示例代码

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define mlen 1005

// 函数：将字符串进行逆序
void reverseStringAndAlphabets(char* str) {
    int start = 0;
    int end = strlen(str) - 1;
    // 首字符逆序
    while (start < end) {
```

```
        char temp = str[start];
        str[start] = str[end];
        str[end] = temp;
        start++;
        end--;
    }
}

// 函数：进行字符串加密
void encryptString(char* str) {
    int len = strlen(str);
    char copy1[mlen  + 1]; // 存储逆序后的字符串
    char newStr[mlen + 1]; // 存储新的长度为n的字符串
    char copy[2 * mlen + 1];//存储逆序后的字符串和源字符串
    char jc[mlen + 1];//奇串
    char oc[mlen + 1];//偶串
    strcpy(copy1, str); // 复制字符串
    reverseStringAndAlphabets(copy1); // 逆序后的字符串
    strcpy(copy, str);
    strcat(copy, copy1);//存放整体的字符串
    copy[len * 2] = '\0';
    for (int i = 0; i < strlen(copy); i++) {//奇串和偶串
        if (i % 2 == 0) {
            jc[i/2] = copy[i];
        }
        else {
            oc[(i-1)/2] = copy[i];
        }
    }
    jc[len] = '\0';
    oc[len] = '\0';
    if (strcmp(jc, oc) < 0)
    {
        strcpy(newStr, jc);
    }
    else
    {
        strcpy(newStr, oc);
    }
    strcpy(str, newStr); // 更新原始字符串内容
}
int main() {
    char s[1005];
    gets(s);
    int n;
    scanf("%d", &n);
    while (n--)
    {
        encryptString(s);

    }
    printf("%s", s);
    return 0;
}
```

示例代码 - 2

```
#include <stdio.h>
#include <string.h>
int main()
{
    char s[2001];
    int n;
    scanf("%s%d", s, &n);
    int len = strlen(s);
    while(n--)
    {
        for(int i = 0; i < len; ++i)
            s[2 * len - 1 - i] = s[i];
        char e[1001], o[1001];
        for(int i = 0; i < len; ++i)
        {
            e[i] = s[2 * i];
            o[i] = s[2 * i + 1];
        }
        e[len] = o[len] = '\0';
        strcpy(s, strcmp(e, o) > 0 ? o : e);
    }
    puts(s);
    return 0;
}
```

F 生成扫雷地图

难度	考点
3~4	二维数组

题目分析

本题思路较简单，代码只需要输入、处理、输出三部分即可完成目标。三部分代码均有多种实现方式，思路多种多样，以下为三部分代码的部分实现方式：

0. 代码开始部分

```
#include<stdio.h>
char a[103][103];
int b[10][3] = {{1, 1}, {1, 0}, {1, -1}, {0, -1}, {-1, -1}, {-1, 0}, {-1, 1}, {0, 1}};
//b用来简化读取周围8个空格的过程，利用8个if也可实现，形式不唯一
int n, m;
```

1. 输入部分

1.逐字符地输入

从“二维字符数组”出发，逐个读入字符的处理方式，注意处理掉换行符。

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

2.利用gets

一次读取一行，注意处理掉 n, m 之后的换行符，以及注意字符串要从角标为 1 的部分开始存储。

```
void input() {
    scanf("%d%d", &n, &m);
    getchar();
    for (int i = 1; i <= n; i++) {
        gets(a[i] + 1);
    }
}
```

3.利用%s

一次读取一行，注意字符串要从角标为 1 的部分开始存储，无需考虑换行符。

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

2. 处理部分

本题的所产生的数字均不会大于 8 。由于 `'0'` 到 `'8'` 在 ASCII 码中是连续的，所以在本题中直接对 `char` 类型的字符递增加一即可。

1.从每个空格出发，考虑这个空格会怎么样地受到周围的雷的影响

遍历二维数组。当读到雷的位置的时候跳过，当读到空格的位置时，找寻周围有几个雷。

```
void process() {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (a[i][j] != 'M') {
                for (int k = 0; k <= 7; k++) {
                    if (a[i + b[k][0]][j + b[k][1]] == 'M') {
                        a[i][j] += 1;
                    }
                }
            }
        }
    }
}
```

2.从每个雷出发，考虑这个雷会怎么样地对周围的空格产生影响

遍历二维数组。当读到空格的位置的时候跳过，当读到雷的位置时，给周围所有不是雷的空格的计数加一。

```
void process() {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (a[i][j] == 'M') {
                for (int k = 0; k <= 7; k++) {
                    if (a[i + b[k][0]][j + b[k][1]] != 'M') {
                        a[i + b[k][0]][j + b[k][1]] += 1;
                    }
                }
            }
        }
    }
}
```

3.输出部分

1.逐字符地输出

```
void output() {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            printf("%c", a[i][j]);
        }
        printf("\n");
    }
}
```

2.整体地输出字符串

参照前面的输入部分，注意要从 `a[i]+1` 开始输出，否则会产生问题。

```
void output() {
    for (int i = 1; i <= n; i++) {
        printf("%s\n",a[i]+1);
    }
}
```

示例代码

组装上面三部分的代码，每部分代码任选一种实现方式即可。题解为了使结构清晰使用了函数来实现，也可不使用函数。

```
#include<stdio.h>
char a[103][103];
int b[10][3] = {{1, 1}, {1, 0}, {1, -1}, {0, -1}, {-1, -1}, {-1, 0}, {-1, 1}, {0,
1}};
int n, m;
void input() {
    scanf("%d%d", &n, &m);
    getchar();
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            scanf("%c", &a[i][j]);
        }
        getchar();
    }
}//输入
void process() {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (a[i][j] != 'M') {
                for (int k = 0; k <= 7; k++) {
                    if (a[i + b[k][0]][j + b[k][1]] == 'M') {
                        a[i][j] += 1;
                    }
                }
            }
        }
    }
}//处理
void output() {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            printf("%c", a[i][j]);
        }
        printf("\n");
    }
}//输出
int main() {
    input();
    process();
    output();
    return 0;
}//主程序，依次调用三部分
```

G Ex-GCD

难度	考点
3~4	指针，递归

题目分析

按照伪代码和Hint写出递归函数即可。

示例代码 - 1

```
#include<stdio.h>
void ExGCD(int a, int b, int *d, int *x, int *y)
{
    if(b == 0)
    {
        *d = a, *x = 1, *y = 0;
        return;
    }
    int dd, xx, yy;
    ExGCD(b, a % b, &dd, &xx, &yy);
    *d = dd, *x = yy, *y = xx - a / b * yy;
}
int main()
{
    int a, b, d, x, y;
    scanf("%d%d", &a, &b);
    ExGCD(a, b, &d, &x, &y);
    printf("%d = %d*(%d) + %d*(%d)", d, a, x, b, y);
    return 0;
}
```

示例代码 - 2

```
#include <stdio.h>
void Exgcd(int a, int b, int *d, int *x, int *y)
{
    if(b == 0) *d = a, *x = 1, *y = 0;
    else Exgcd(b, a % b, d, y, x), *y -= a / b * *x;
}
int main()
{
    int a, b, d, x, y;
    scanf("%d%d", &a, &b);
    Exgcd(a, b, &d, &x, &y);
    printf("%d = %d*(%d) + %d*(%d)\n", d, a, x, b, y);
    return 0;
}
```

示例代码 - 3

```c
#include<stdio.h>
int ExGCD(int a, int b, int *x, int *y)
{
    if(b == 0)
    {
        *x = 1;
        *y = 0;
        return a;
    }
    else
    {
        int d = ExGCD(b, a % b, y, x);
        *y -= a / b * *x;
        return d;
    }
}
int main()
{
    int a, b, x, y;
    scanf("%d%d", &a, &b);
    int d = ExGCD(a, b, &x, &y);
    printf("%d = %d*(%d) + %d*(%d)", d, a, x, b, y);
    return 0;
}
```

H Cirno 的完美函数教室

难度	考点
4	递归，剪枝

问题分析

本题如果使用递归，需要剪枝，存储计算过的函数值，否则递归次数过多会导致 TLE。

当然，也可以使用递推，要注意递推的关系式和更新的顺序。

参考代码 #1

```c
#include <stdio.h>

int f[30][30][30];

int B(int a, int b, int c)
{
    if (a <= 0 || b <= 0 || c <= 0)
        return 1;
    else if (a > 25 || b > 25 || c > 25)
        return B(25, 25, 25);
```

```
    else
    {
        if (f[a][b][c])
            return f[a][b][c];
        else if (a < b && b < c)
            f[a][b][c] = B(a, b, c - 1) + B(a, b - 1, c - 1) - B(a, b - 1, c);
        else
            f[a][b][c] = B(a - 1, b, c) + B(a - 1, b - 1, c) + B(a - 1, b, c - 1)
- B(a - 1, b - 1, c - 1);
    }
    return f[a][b][c];
}

int main()
{
    int T, a, b, c;
    scanf("%d", &T);
    while (T--)
    {
        scanf("%d%d%d", &a, &b, &c);
        printf("%d\n", B(a, b, c));
    }
    return 0;
}
```

参考代码 #2

```
#include <stdio.h>

int main() {
    int B[26][26][26];
    for (int i = 0; i < 26; ++i) {
        for (int j = 0; j < 26; ++j) {
            for (int k = 0; k < 26; ++k) {
                if (i == 0 || j == 0 || k == 0) {
                    B[i][j][k] = 1;
                } else {
                    if (i < j && j < k) {
                        B[i][j][k] = B[i][j][k - 1] + B[i][j - 1][k - 1] - B[i][j
- 1][k];
                    } else {
                        B[i][j][k] = B[i - 1][j][k] + B[i - 1][j - 1][k] + B[i -
1][j][k - 1] - B[i - 1][j - 1][k - 1];
                    }
                }
            }
        }
    }
    int T, a, b, c;
    scanf("%d", &T);
    for (int i = 0; i < T; ++i) {
        scanf("%d%d%d", &a, &b, &c);
        if (a <= 0 || b <= 0 || c <= 0) {
            a = b = c = 0;
        } else if (a > 25 || b > 25 || c > 25) {
```

```
            a = b = c = 25;
        }
        printf("%d\n", B[a][b][c]);
    }

    return 0;
}
```

I 拔刀

难度	考点
5	二维数组，递归

题目分析

本题是一个相对标准的递归问题，递归中，若当前位置为1，则进入其相邻四个元素判断（需要考虑向上的情况），否则返回当前行序号。

示例代码

```
int graph[1005][1005];
int used[1005][1005];
int m, n;
int max=-1;
void dfs(int i, int j);
int main(void)
{
    int i, j;
    scanf("%d%d", &m, &n);
    for (i = 0; i < m; i++)
    {
        for (j = 0; j < n; j++)
        {
            scanf("%d", &graph[i][j]);
        }
    }
    for (i = 0; i < n; i++)
    {
        if(graph[0][i]==1)
            dfs(0, i);
    }
    printf("%d", max + 1);
    return 0;
}
void dfs(int i, int j)
{
    if (i < 0 || j < 0 || i >= m || j >= n || used[i][j] || graph[i][j] == 0)
        return;
    used[i][j] = 1;
    dfs(i, j + 1);
```

```
    dfs(i, j - 1);
    dfs(i + 1, j);
    dfs(i - 1, j);
    max = MAX(max, i);
}
```

示例代码 - 2

```
#include <stdio.h>
int a[1005][1005], ans;
void f(int i, int j)
{
    if(!a[i][j]) return;
    a[i][j] = 0;
    if(i > ans) ans = i;
    f(i - 1, j);
    f(i + 1, j);
    f(i, j - 1);
    f(i, j + 1);
}
int main()
{
    int h, w;
    scanf("%d%d", &h, &w);
    for(int i = 1; i <= h; ++i)
        for(int j = 1; j <= w; ++j)
            scanf("%d", &a[i][j]);
    for(int j = 1; j <= w; ++j) f(1, j);
    printf("%d", ans);
    return 0;
}
```

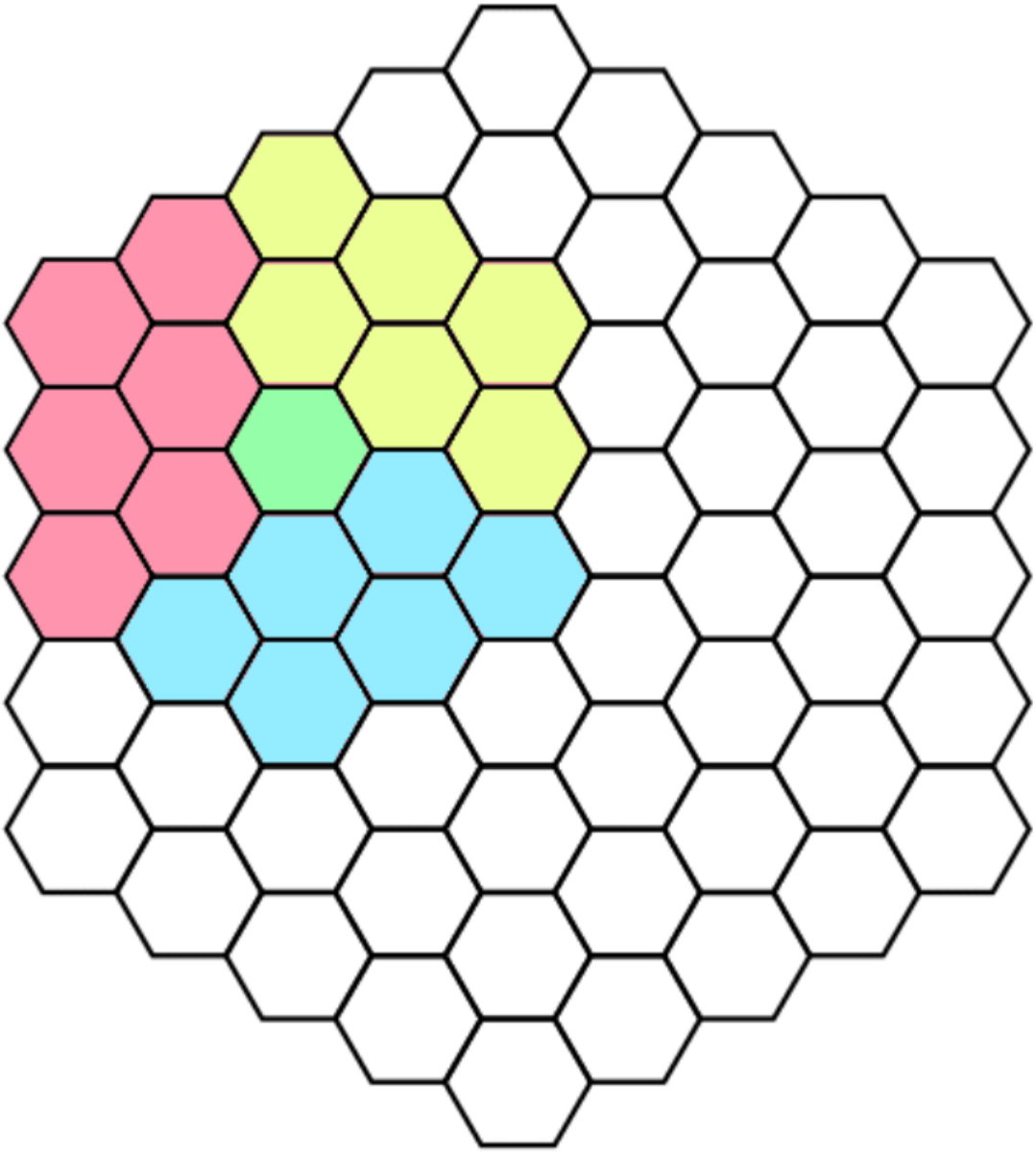
J De想成为六边形站侍

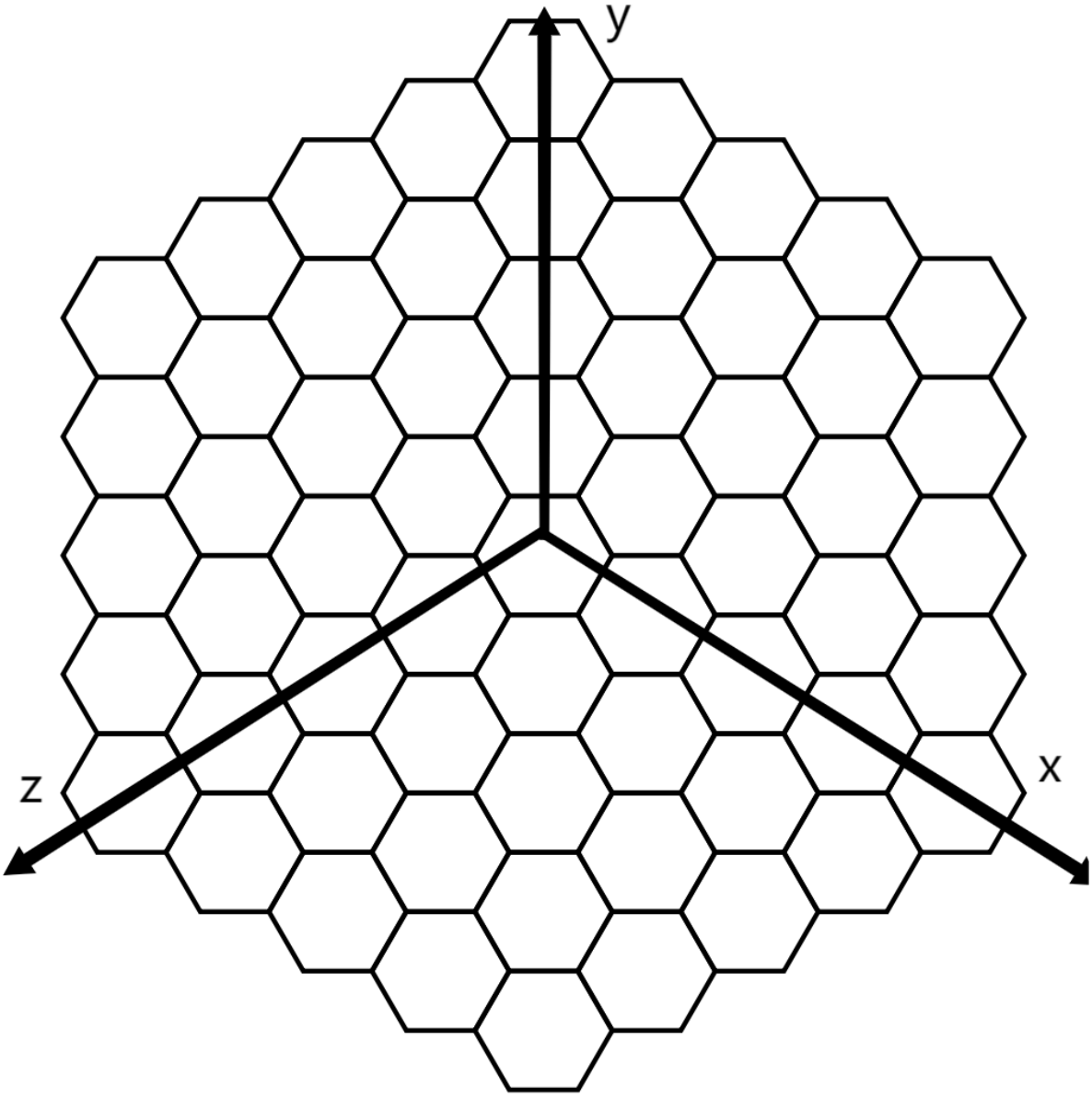
难度	考点
6	二维数组，差分

题目分析

首先，我们基本可以确定这道题需要使用差分，对于有规则的平面图形，我们打算使用二维差分，尝试将一个六边形的整体加法用一个差分数组表示

具体方法（其中一种方法）如下：





我们将一个正六边形分为三个部分，如图中红黄蓝部分，那么易知对于其中一块，我们可以以其边所平行的两个轴来将整个图形及平面移为“平面直角坐标系”，并且每个点表示方法唯一

比如对于黄色区域如果我们要 `+1` ，那么相当于是二维差分中修改：

`c[-2][1]++, c[1][1]--, c[-2][3]--, c[1][3]++` （表示方法依个人实现方式不同而不尽相同）

因此，我们只需要维护三个二位差分即可，另外，注意一下超出边界的点不需要管

- 中心块我们单独处理
- 黄色区域轴 `x - y`
- 红色区域轴 `y - z`
- 蓝色区域轴 `x - z`

示例代码

~~（有点长，大家的实现肯定有非常短的）~~

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

#define ll long long
#define max(a, b) ((a) > (b) ? (a) : (b))
#define min(a, b) ((a) < (b) ? (a) : (b))
#define rep(i, a, b) for(int i = a; i <= b; i++)
#define per(i, a, b) for(int i = a; i >= b; i--)

#define maxn 805

int pre[3][maxn << 1][maxn << 1], node[maxn << 1][maxn << 1];
int n, test, lim;
int X, Y, Z, r, k;

void update(int X, int Y, int Z, int r, int k) {
    pre[0][min(lim, X - Z + n + 1)][min(lim, (Y + 1) - Z + n + 1)] += k;
    pre[0][min(lim, (X + r) - Z + n + 1)][min(lim, (Y + 1) - Z + n + 1)] -= k;
    pre[0][min(lim, X - Z + n + 1)][min(lim, (Y + r) - Z + n + 1)] -= k;
    pre[0][min(lim, (X + r) - Z + n + 1)][min(lim, (Y + r) - Z + n + 1)] += k;

    pre[1][min(lim, (X + 1) - Y + n + 1)][min(lim, Z - Y + n + 1)] += k;
    pre[1][min(lim, (X + r) - Y + n + 1)][min(lim, Z - Y + n + 1)] -= k;
    pre[1][min(lim, (X + 1) - Y + n + 1)][min(lim, (Z + r) - Y + n + 1)] -= k;
    pre[1][min(lim, (X + r) - Y + n + 1)][min(lim, (Z + r) - Y + n + 1)] += k;

    pre[2][min(lim, Y - X + n + 1)][min(lim, (Z + 1) - X + n + 1)] += k;
    pre[2][min(lim, Y - X + n + 1)][min(lim, (Z + r) - X + n + 1)] -= k;
    pre[2][min(lim, (Y + r) - X + n + 1)][min(lim, (Z + 1) - X + n + 1)] -= k;
    pre[2][min(lim, (Y + r) - X + n + 1)][min(lim, (Z + r) - X + n + 1)] += k;
}

void push_up() {
    rep(i, 1, lim)
        rep(j, 1, lim) {
            pre[0][i][j] += pre[0][i - 1][j] + pre[0][i][j - 1] - pre[0][i - 1][j
- 1];
            pre[1][i][j] += pre[1][i - 1][j] + pre[1][i][j - 1] - pre[1][i - 1][j
- 1];
            pre[2][i][j] += pre[2][i - 1][j] + pre[2][i][j - 1] - pre[2][i - 1][j
- 1];
        }
}

int get(int X, int Y, int Z) {
    return node[X - Z + n + 1][Y - Z + n + 1] +
        pre[0][X - Z + n + 1][Y - Z + n + 1] +
        pre[1][X - Y + n + 1][Z - Y + n + 1] +
        pre[2][Y - X + n + 1][Z - X + n + 1];
```

```
}

int main() {
    scanf("%d %d", &n, &test);
    lim = (n << 1 | 1);
    while(test--) {
        scanf("%d %d %d %d %d", &X, &Y, &Z, &r, &k);
        node[X - Z + n + 1][Y - Z + n + 1] += k;
        update(X, Y, Z, r, k);
    }
    push_up();

    per(i, n - 1, 0)
        per(j, n - 1, 1 + i - n)
            printf("%d ", get(0, j, i));
    per(i, -1, 1 - n)
        per(j, n - 1 + i, 1 - n)
            printf("%d ", get(0, j, i));
    // system("pause");
    return 0;
}
```

示例代码 - 2

```
#include <stdio.h>
#define max(a,b) ((a)>(b)?(a):(b))
#define min(a,b) ((a)<(b)?(a):(b))
int xy[1601][1601], yz[1601][1601], zx[1601][1601];
int (*A)[1601] = (int(*)[1601])&xy[800][800]; //偏移中心点，实现负数下标
int (*B)[1601] = (int(*)[1601])&yz[800][800];
int (*C)[1601] = (int(*)[1601])&zx[800][800];
int main()
{
    int n, m;
    scanf("%d%d", &n, &m);
    //维护差分数组
    while(m--)
    {
        int x, y, z, r, k, i, j;
        scanf("%d%d%d%d%d", &x, &y, &z, &r, &k);
        if(z == 0)
            i = x, j = y;
        else if(x == 0)
            i = -z, j = y - z;
        else
            i = x - z, j = -z;
        A[i][j] += k;
        A[min(i + r, n)][j] -= k;
        A[i][min(j + r, n)] -= k;
        A[min(i + r, n)][min(j + r, n)] += k;
        if(x == 0)
            i = y, j = z;
        else if(y == 0)
            i = -x, j = z - x;
        else
```

```
            i = y - x, j = -x;
        B[i][j + 1] += k;
        B[min(i + r, n)][j + 1] -= k;
        B[i][min(j + r, n)] -= k;
        B[min(i + r, n)][min(j + r, n)] += k;
        if(y == 0)
            i = z, j = x;
        else if(z == 0)
            i = -y, j = x - y;
        else
            i = z - y, j = -y;
        C[i + 1][j + 1] += k;
        C[min(i + r, n)][j + 1] -= k;
        C[i + 1][min(j + r, n)] -= k;
        C[min(i + r, n)][min(j + r, n)] += k;
    }
    //计算二维前缀和
    for(int i = -n + 1; i <= n; ++i)
    {
        for(int j = -n + 1; j <= n; ++j)
        {
            A[i][j] += A[i][j - 1] + A[i - 1][j] - A[i - 1][j - 1];
            B[i][j] += B[i][j - 1] + B[i - 1][j] - B[i - 1][j - 1];
            C[i][j] += C[i][j - 1] + C[i - 1][j] - C[i - 1][j - 1];
        }
    }
    //输出结果
    for(int i = -n + 1; i < n; ++i)
        for(int j = -n + 1 + max(i, 0); j < n + min(i, 0); ++j)
            printf("%d ", A[i][i - j] + B[-j][-i] + C[j - i][j]);
    return 0;
}
```

- End -