第 5 章　数组与数据批量存储

计算机运算速度很快，一秒钟可以处理成千上万的数据。之前的例子都是读取一个数据后立刻对这些数据进行处理，然后再也不需要用到这些数据了；有时候，读入数据后还需要将这些数据保存下来，以便以后再次使用。如果保存个别几个数据，可以设立几个变量存储；但是如果要存储成千上万个数据，总不能定义成千上万个变量吧。

既然可以通过循环语句来重复执行结构类似的语句，也有办法一次定义一组成千上万个的相同类型的变量——使用数组，这样就可以把大量的数据存储下来，并随时使用。数组不仅可以存储输入的数据，还能存下运算过程中的“半成品”甚至答案，是 C++ 中非常重要的一部分。图 5-1 所示为本章思维导图。

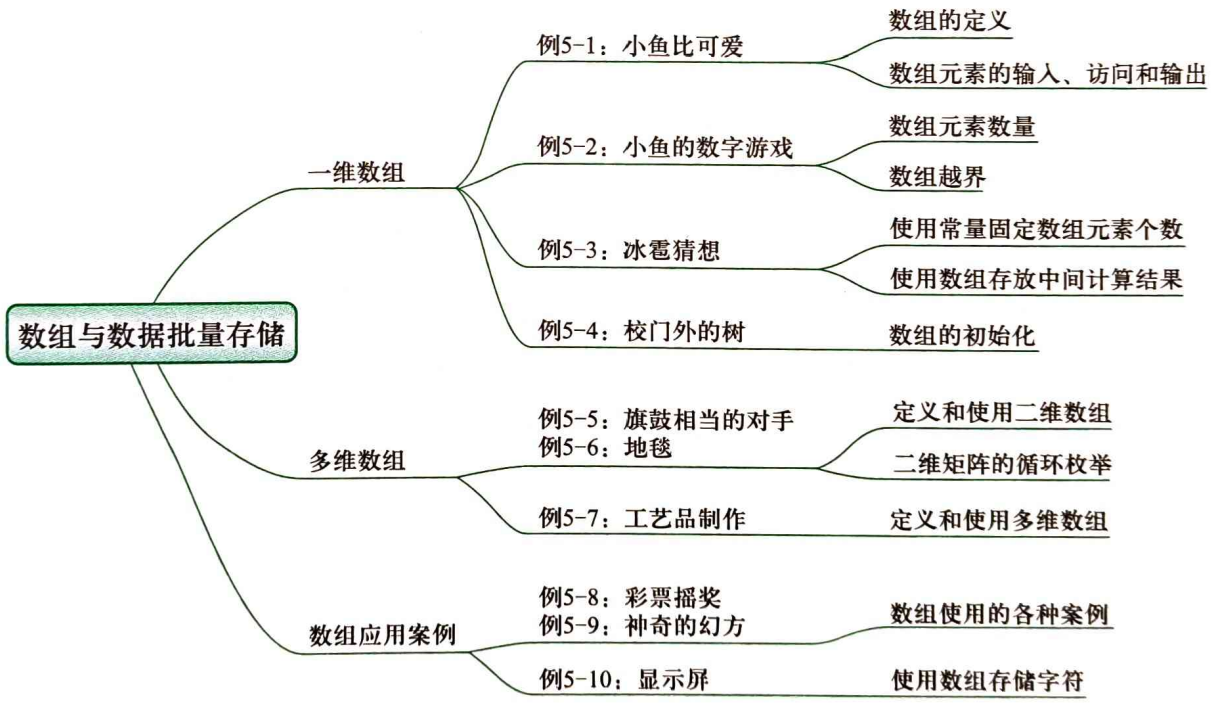

图 5-1　本章思维导图

5.1 一维数组

例 5-1　小鱼比可爱（洛谷 P1428）。n（$n \leqslant 100$）只小鱼最近参加了一个“比可爱”比赛，比

的是每条鱼的可爱程度。参赛的鱼被从左到右排成一排,头都朝向左边。每条鱼会有一个整数数值,表示这条鱼的可爱程度,而且可能存在可爱程度相同的两条鱼。由于所有的鱼头都朝向左边,它们在心里都在计算,在自己的眼力范围内有多少条鱼不如自己可爱呢。

分析:对于每条小鱼来说,都要查找一下之前读入过的所有小鱼的数据,然后进行比较。所以读入的数据不能就这么丢了,必须要存储下来!可以使用**数组**来存下这些数据。

定义一维数组的方式是"数组类型 数组变量名称[元素个数];",代码如下:

```
#include<iostream>
using namespace std;
int main() {
    int a[110], n;
    cin >> n;
    for (int i = 0; i < n; i++) // 读入每条鱼的可爱值
        cin >> a[i];
    for (int i = 0; i < n; i++) { // 枚举n条鱼
        int cnt = 0;
        for (int j = i - 1; j >= 0; j--)  // 从第i个位置倒着往前找
            if (a[j] < a[i])
                cnt++; // 如果找到比第i条鱼没有比不上，计数器就增加1
        cout << cnt << ' ';
    }
    return 0;
}
```

程序第4行,定义了a数组,类型是int,数量是110个。这回就不是在操作台准备一个碗了,而是准备了110个碗排成一排,碗的编号从a[0]到a[109],共计110个,每个碗都可以盛放int类型的数据,方括号里的数字被称为数组下标。不是最多只有100条鱼吗,为什么要准备110个碗呢?因为多准备一点碗可以应对不时之需,所以数组空间多开10个比较好。

第一次for循环,i从0循环到n,每次读入a[i],把第0条小鱼的可爱值放到a[0]中,第1条小鱼的可爱值放到a[1]中,……第n-1条小鱼的可爱值放到a[n-1]中。这样就读入完毕了。

第二次for循环,还是从每条小鱼开始。从这条(第i条)小鱼的左边那条开始,一直到最开头的那条小鱼为止,每条(第j条)都和这条小鱼进行比较,如果发现不如自己的小鱼(a[j]<a[i]),计数器cnt就增加1。比较完毕就输出计数器的值。注意,每次都要清空计数器,而且不要漏掉输出之间的空格。

例5-2 小鱼的数字游戏(洛谷P1427)。小鱼最近被要求参加一个数字游戏,要求它记忆一串数字(个数不定,最多不超过100个,数字大小不超过$2^{32}-1$,最后以0结束),然后反着念出来(表示结束的数字0就不要念出来了)。这对小鱼的那点记忆力来说实在是太难了,请你帮小鱼编程解决这个问题。

分析:和上面差不多,一个一个读入数字,存入数组,从a[0]存到a[n-1](一共有n个数字),然后再从a[n-1]开始,逆序输出,直到输出到a[0]为止。代码如下:

```
#include<iostream>
using namespace std;
int main() {
    int n = 0, tmp, a[110];
    do {
        cin >> tmp; a[n] = tmp; n++; // 本行可以替代成 cin>>a[n++];
    } while (tmp != 0);
    n--; // 或者 --n;
    while (n--) // 不能写成 while (--n)
        cout << a[n] << ' ';
    return 0;
}
```

最初 n 从 0 开始，每次读入 tmp 变量，然后 a [n] 赋值成 tmp，然后 n 增加 1，直到 tmp 读到 0 为止。当要读入 100 个非零整数时，数组从 a [0] 一直读到 a [99]，此时 n 变为了 100。如果刚好只定义到了 a [100]，实际上可以使用的数组元素只有 a [0] 到 a [99]，没有 a [100]。如果这时读入最后的一个 0，就会访问 a [100]，这就会导致**数组越界**。数组越界一般来说不会导致运行时错误，但是经常会导致一些其他的后果（比如把其他无关变量的值给改掉了）。

读入后就要输出，需要将 n 减 1，因为刚刚进行了 n++，此时的 n 是指向下一个待存变量的位置，所以要减回去。接下来就是使用 while 语句，首先判断 n 是否为 0，然后对 n 减 1，接着依次输出 a [n] 的值。由于开始时 a [n] 是最后读入的 0，不需要输出，所以可以先自减，然后输出 a [n]。当判断 n 是 0 的时候，说明 a [0] 上一轮循环已经输出过了，这时就可以退出循环，结束程序。

例 5-3　冰雹猜想（洛谷 P5727）。给出一个正整数 n（$n \leqslant 100$），然后对这个数字一直进行下面的操作：如果这个数字是奇数，那么将其乘 3 再加 1，否则除以 2。经过若干次循环后，最终都会回到 1。经过验证很大的数字（7×10^{11}）都可以按照这样的方式比变成 1，所以称为“冰雹猜想”。例如当 n 是 20，变化的过程是 [20,10,5,16,8,4,2,1]。

根据给定的数字，验证这个猜想，并从最后的 1 开始，倒序输出整个变化序列。例如当输入“20”时，输出应当是“1 2 4 8 16 5 10 20”。已知变化次数不会超过 200 次。

分析：按照题目的要求计算数字，并像上一个例子一样依次将计算结果存入数组，直到计算到了 1 为止。最后使用 for 循环倒序输出。while 语句和 do-while 只是在判定循环条件的位置不一样，是可以相互转换的。代码如下：

```
#include<iostream>
using namespace std;
#define MAXN 205
int main() {
    int n, num = 0, a[MAXN];
    cin >> n;
    while (n != 1) {
        a[num] = n; num++; // 本行可以替代成 a[num++]=n;
```

```
            if (n % 2 == 0)n /= 2;
            else n = 3 * n + 1;
        }
        a[num] = 1; // 将最后的1加入数组
        for (int i = num; i >= 0; i--) // 倒序输出
            cout << a[i] << ' ';
        return 0;
    }
```

首先使用宏定义,将 MAXN 定义为 205。然后,依然定义一个 a 数组,一共有 MAXN 个元素,也就是 205 个元素。这么做的目的是:在要定义多个个数相同的数组时,如果需要调整数组大小,只要调整 MAXN 定义的数字即可,而不需要逐一调整。在定义数组时,个数应当是一个确定的正整数,这个正整数可以是直接数字的形式(如 a [205])、表达式(如 a [200+5])、宏定义或者 const 整数常量(如 a [MAXN]),但是定义个数不可以是一个变量,比如定义成 a [n] 是不可以的,因为 n 不是固定的常量。

for 循环中 i 从 num 开始循环,每次减少 1,直到它小于 0 终止循环。这里特别要注意的是,如果使用倒序循环,那么不要习惯性地录入成 i++,要不然就成为死循环了。此外,变量 num 因为是在主函数中定义的,所以使用前必须初始化,否则运行时就会出问题。

例 5-4 校门外的树(洛谷 P1047,NOIP2005 普及组)。有一条长度为 L 的马路上有一排树,每两棵相邻的树之间的间隔都是 1m。可以把马路看成一个数轴,马路的一端在数轴 0 的位置,另一端在 L 的位置;数轴上的每个整数点,即 0,1,2, …,L,都种有一棵树。

由于马路上有一些区域要用来建地铁。这些区域用它们在数轴上的起始点和终止点表示。已知任一区域的起始点和终止点的坐标(a,b)都是整数,区域之间可能有重合的部分。现在要把这些区域中的树(包括区域端点处的两棵树)移走。你的任务是计算将这些树都移走后,马路上还有多少棵树。

分析:建立一个数组,模拟每一个点的树是否还在,如果这个点的树还在,则该数组元素的值为 0,否则就是 1。最开始所有数组元素都是 0。每次移掉树木,将这一段从左到右的所有元素都设置为 1。最后,循环枚举路上的每一个点,统计有多少个点是 0 即可,代码如下:

```
#include <iostream>
// #include <cstring>
using namespace std;
int main() {
    int l, m, tree[10010] = {0}, a, b, s = 0;
    cin >> l >> m;
    // memset(tree, 0, sizeof(tree));
    for (int i = 0; i < m; i++) {
        cin >> a >> b;
        for (int j = a; j <= b; j++)
            tree[j] = 1;
    }
```

```
    for (int i = 0; i <= 1; i++)
        if (tree[i] == 0) s++;
    cout << s << endl;
    return(0);
}
```

在之前的章节中提到过，在函数体（比如 main 函数）中定义的变量，必须要初始化才能直接引用，否则不知道变量里面装的都是些什么。数组也需要**数组初始化**，只不过数组不是单个元素，而是由很多个元素组成。数组初始化的方法也很简单，在定义数组后面加上 ={0} 就可以将整个数组初始化为 0；如果是 int a[10010]={1}，那么可不是将数组全部初始化为 1，而是将 a[0] 赋值成 1，后面剩余的都初始化为 0；还可以初始化为 int a[10010]={5,2,1}，这样的话，a[0]、a [1]、a [2] 分别就是 5、2、1，后面剩余的还是 0。

除了在定义数组时可以初始化，数组还可以随时使用 memset 函数（需要 cstring 头文件），将数组内的全部“填充”成 0，以达成初始化的效果。方法是 memset（数组名称，0，sizeof（数组名称））。如果数组是 int 类型，那么中间的一项只有是 0 或者 -1 的时候，数组的每一项值才会被变成 0 或者 -1，其他的值并不能达成你所想象的效果，具体原因不在这里深究。

5.2 多维数组

例 5-5　旗鼓相当的对手（洛谷 P5728）。现有 N（$N \leqslant 1000$）名同学参加了期末考试，并且获得了每名同学的信息：语文、数学、英语成绩（均为不超过 150 的自然数）。如果某对学生 <i,j> 的每一科成绩的分差都不大于 5，且总分分差不大于 10，那么这对学生就是“旗鼓相当的对手”。现在想知道这些同学中，有几对“旗鼓相当的对手”？同样一个人可能会和其他好几名同学结对。

分析：如果说一维数组是一排容器，那么多维数组就是一片容器的矩阵了。可以建立一个 3 × MAXN 的二维数组 a，这样相当于有一个 3 行 MAXN 的表格可以用来存放东西了。其实二维数组的本质也是一个一维数组，只是这个一维数组的每一个元素不是一个 int，而还是一个一维数组。

和一维数组类似，定义 a [3] [MAXN] 后，可以用 a [i] [j] 来访问第 i 行、第 j 列的元素。需要注意的是，i 的范围是 0 到 2，没有 3。不过，考虑到第一维的数组下标无论怎么样都没有越界的可能，所以可以不用考虑余量，也就不需要多定义了。a [0] [i] 存储了第 i 名同学的语文成绩，a [1] [i] 存储了他的数学成绩，a [2] [i] 存储了他的英语成绩。使用数组记录下读入的每名同学的所有信息，然后枚举任意一对同学来判断这一对同学是不是“旗鼓相当的对手”即可。如果发现符合条件，就在答案变量中增加 1。

```
#include <iostream>
#include <cmath>
#define MAXN 1024
using namespace std;
int main() {
```

```
    int n, a[3][MAXN], ans = 0;
    cin >> n;
    for (int i = 1; i <= n; i++)
        cin >> a[0][i] >> a[1][i] >> a[2][i];
    for (int i = 1; i <= n - 1; i++) // 枚举第一个学生 i
        for (int j = i + 1; j <= n; j++) // 枚举第二个学生 j
            if (abs(a[0][i] - a[0][j]) <= 5
                && abs(a[1][i] - a[1][j]) <= 5
                && abs(a[2][i] - a[2][j]) <= 5
                && abs(a[0][i]+a[1][i]+a[2][i]-a[0][j]-a[1][j]-a[2][j])<=10
            )
                ans++; // 如果这两名学生是旗鼓相当的对手，那么 ans++
    cout << ans << endl;
    return 0;
}
```

在程序中，由于要枚举所有两个不一样的对象，可以使用两重循环——外层循环变量 i 从 1 枚举到 n-1，内层循环变量 j 从 i+1 枚举到 n，这样可以保证 i<j 并且可以不重复、不遗漏地比较所有两个元素了。

例 5-6 地毯(P3397，By 阮行止)。在 $n \times n$ 的格子上有 m 块地毯($n \leqslant 50, m \leqslant 100$)，给出每块地毯的两个对角的坐标$(x_1, y_1)$和$(x_2, y_2)$。问：最后格子每个点被多少块地毯覆盖。

分析：同样建立一个不小于 50×50 的二维数组 a[MAXN][MAXN]来模拟地毯的情况。这个二维数组相当于一个棋盘，里面有很多个小格子，每个小格子存储一个数字，记录这个小格子上面堆放了几块地毯。

当有一块左上角为(x_1, y_1)，右下角为(x_2, y_2)的地毯覆盖时，a[i][j](其中 $x_1 \leqslant i \leqslant x_2$ 且 $y_1 \leqslant j \leqslant y_2$)中的每一个元素都在这块地毯的覆盖范围中，因此对这一片范围中的每一个元素增加 1(使用双重循环)。最后使用双重循环枚举所有格子，输出每个格子的地毯覆盖数量。

```
#include <iostream>
#include <cstdio>
#include <cstring>
#define MAXN 55
using namespace std;
int main() {
    int n, m, a[MAXN][MAXN];
    memset(a, 0, sizeof(a));
    cin >> n >> m;
    for (int i = 1; i <= m; i++) {
        int x1, y1, x2, y2;
        cin >> x1 >> y1 >> x2 >> y2;
        for (int j = x1; j <= x2; j++)
            for (int k = y1; k <= y2; k++)
```

```
                a[j][k]++;
            //对于每块地毯覆盖的所有格子，把其值++，代表被覆盖了一次
    }
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            cout << a[i][j] << (j == n ? '\n' :  ' ');
            //输出这个二维数组每个位置的值
    return 0;
}
```

原题数据范围较大，需要使用二维前缀和的思路，感兴趣的同学可以查阅相关资料。

例 5-7　工艺品制作（洛谷 P5729）。现有一个长、宽、高分别为 w、x、h（$1 \leqslant w,x,h \leqslant 20$）的实心玻璃立方体，可以认为是由 $1 \times 1 \times 1$ 的数个小方块组成的，每个小方块都有一个坐标（i,j,k）。现在需要进行 q（$q \leqslant 100$）次切割。每次切割给出（x_1,y_1,z_1）和（x_2,y_2,z_2）这 6 个参数，保证 $x_1 \leqslant x_2, y_1 \leqslant y_2, z_1 \leqslant z_2$；每次切割时，使用激光工具切出一个立方体空洞，空洞的壁平行于立方体的面，空洞的对角点就是给出的切割参数的两个点。

换句话说，所有满足 $x_1 \leqslant i \leqslant x_2, y_1 \leqslant j \leqslant y_2, z_1 \leqslant k \leqslant z_2$ 的小方块（i,j,k）的点都会被激光蒸发。例如有一个 $4 \times 4 \times 4$ 的大方块，其体积为 64；给出参数（1,1,1）和（2,2,2）时，中间的 8 块小方块就会被蒸发，剩下 56 个小方块。现在想知道经过所有切割操作后，剩下的工艺品还剩下多少个小方块的体积？

分析：C++ 中的数组不仅可以支持二维数组，还能建立更多维数的数组。在本例中，建立了一个三维数组 v[22][22][22] 来模拟立方体的每一个小方块的情况，记录每个小方块是否还存在。每次操作就是把三维各在一个区间内的小方块全部消除，类似于上一个例题的覆盖地毯，把这个三维数组三维在一个区间内的值全部赋值为 0。当进行完所有操作之后，使用三重循环枚举每个小方块，看看每个位置是否为 1 即可，如果是 1，答案就增加 1，最后输出答案。

```
#include <iostream>
#include <cstdio>
using namespace std;
int main() {
    int v[22][22][22], w, x, h, q, x1, x2, y1, y2, z1, z2, ans;
    cin >> w >> x >> h >> q;
    for (int i = 1; i <= w; i++)
        for (int j = 1; j <= x; j++)
            for (int k = 1; k <= h; k++)
                v[i][j][k] = 1;
            //先把三维数组的每个位置赋值为 1
    while (q--) {
        cin >> x1 >> y1 >> z1 >> x2 >> y2 >> z2;
        for (int i = x1; i <= x2; i++)
            for (int j = y1; j <= y2; j++)
```

```
                for (int k = z1; k <= z2; k++)
                    v[i][j][k] = 0;
                // 对每个操作，把删掉的小方块所对应的数组位置赋值为 0
    }
    for (int i = 1; i <= w; i++)
        for (int j = 1; j <= x; j++)
            for (int k = 1; k <= h; k++)
                ans += v[i][j][k];
            // 所有操作结束之后，对每个小方块看一下是 1 还是 0，计算答案
    cout << ans << endl;
    return 0;
}
```

5.3 数组应用案例

例 5-8 彩票摇奖(洛谷 P2550,安徽省队选拔 2001)。某种彩票的规则如下：

1) 每张彩票上印有 7 个各不相同的号码,且这些号码的取值范围为 1~33。

2) 每次在兑奖前都会公布一个由 7 个各不相同的号码构成的中奖号码。

3) 共设置 7 个奖项,特等奖和一等奖至六等奖。兑奖时并不考虑彩票上的号码和中奖号码中的各个号码出现的位置。兑奖规则如下：

特等奖:要求彩票上的 7 个号码都出现在中奖号码中。

一等奖:要求彩票上有 6 个号码出现在中奖号码中。

二等奖:要求彩票上有 5 个号码出现在中奖号码中。

三等奖:要求彩票上有 4 个号码出现在中奖号码中。

四等奖:要求彩票上有 3 个号码出现在中奖号码中。

五等奖:要求彩票上有 2 个号码出现在中奖号码中。

六等奖:要求彩票上有 1 个号码出现在中奖号码中。

输入 $n(n\leqslant 1000)$、已知中奖号码和小明买的 n 张彩票的号码,请写一个程序帮助小明判断他买的彩票的中奖情况,也就是特等奖、一等奖到六等奖分别中了几次。

分析:首先使用数组 a 读入 7 个中奖号码。外层循环是枚举每一张彩票,用 ans 变量来统计这张彩票上的数字是否和某个中奖号码一致,将所有中奖号码对比一下是否相等,如果发现相等,ans 就增加 1。统计完一张后就知道彩票有几个号码和中奖号码相等了,这时就可以使用另外一个数组 num 记录有几个数字和中奖号码相同。最后依次输出中了 7 个数字、6 个数字……一直到 1 个数字的次数,注意输出的顺序。

```
#include <iostream>
using namespace std;
int main() {
    int n, a[10], num[10] = {0};
    cin >> n;
```

```
    for (int i = 1; i <= 7; i++) cin >> a[i]; // 创建一个数组存下中奖号码
    while (n--) { // 用 for 也可以
        int ans = 0;
        for (int i = 1; i <= 7; i++) {
            int x;
            cin >> x;
            for (int j = 1; j <= 7; j++) // 每次比较每个号码是否为中奖号码
                if (a[j] == x)
                    ans++;
        }
        num[ans]++;
    }
    for (int i = 7; i; i--)
        cout << num[i] << (i == 1 ? '\n' : ' '); /* 输出答案，注意最后一个要加换
行而不是空格 */
    return 0;
}
```

例 5-9　神奇的幻方(洛谷 P2615,NOIP2015 提高组)。幻方是一种很神奇的 $N\times N$ 矩阵:它由数字 1,2,3, …,$N\times N$ 构成,且每行、每列及两条对角线上的数字之和都相同。当 N 为奇数时,可以通过以下方法构建一个幻方:首先将 1 写在第一行的中间;之后,按如下方式从小到大依次填写每个数 K(K=2,3, …,$N\times N$)。

1) 若(K-1)在第一行但不在最后一列,则将 K 填在最后一行,(K-1)所在列的右一列。

2) 若(K-1)在最后一列但不在第一行,则将 K 填在第一列,(K-1)所在行的上一行。

3) 若(K-1)在第一行最后一列,则将 K 填在(K-1)的正下方。

4) 若(K-1)既不在第一行,也不在最后一列,如果(K-1)的右上方还未填数,则将 K 填在(K-1)的右上方,否则将 K 填在(K-1)的正下方。

现给定 N,请按上述方法构造 $N\times N$ 的幻方。

分析:使用一个二维数组 g 来存下这个幻方。从 1 开始,依次按照构造规则进行判断是属于哪一种情况,根据不同的规则将数字填入指定的位置(将题意翻译成代码),并且记录下刚才填写的坐标。最后把这个二维数组输出。

```
#include <iostream>
using namespace std;
int n, g[40][40], x, y;
int main() {
    cin >> n;
    g[1][n / 2 + 1] = 1;
    x = 1; y = n / 2 + 1;
    for (int i = 2; i <= n * n; i++) {
        if (x == 1 && y != n) // 第一行但不是最后一列
            g[n][y + 1] = i, x = n, y++;
```

```
            else if (y == n && x != 1) // 最后一列但不是第一行
                g[x - 1][1] = i, x--, y = 1;
            else if (x == 1 && y == n) // 第一行最后一列
                g[2][n] = i, x = 2;
            else if (x != 1 && y != n) { // 不在第一行，也不在最后一列
                if (g[x - 1][y + 1] == 0) // 右上方未填数
                    g[x - 1][y + 1] = i, x--, y++;
                else
                    g[x + 1][y] = i, x++;
                continue;
            }
        }
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++)
                cout << g[i][j] << " ";
            cout << endl;
        }
        return 0;
}
```

例 5-10 (选读)显示屏(洛谷 P5730)。液晶屏上,每个阿拉伯数字都是可以显示成 3×5 的点阵的(其中 X 表示亮点,. 表示暗点)。现在给出数字位数(不超过 100)和一串数字,要求输出这些数字在显示屏上的效果。例如,当输入是:

```
10
0123456789
```

输出应当是(注意每个数字之间都有一列间隔):

```
XXX...X.XXX.XXX.X.X.XXX.XXX.XXX.XXX.XXX
X.X...X...X...X.X.X.X...X.....X.X.X.X.X
X.X...X.XXX.XXX.XXX.XXX.XXX...X.XXX.XXX
X.X...X.X.....X...X...X.X.X...X.X.X...X
XXX...X.XXX.XXX...X.XXX.XXX...X.XXX.XXX
```

分析:显示屏可以分为 7 个显示管,如图 5-2(a)所示,按照 0~6 的对每个显示管进行标号。然后使用一个数组 tubes 来指定每个数字到底有哪些管子是亮的。由于每个数字亮的管子数量还不一样,所以 tubes[i][0]表示数码 i 要显示几个显示管,而 tubes[i][j]表示数码 i 的第 j 个显示管的编号是什么。编写程序时需要人工预处理这些数据,并对各数组进行初始化。

然后还需要知道每个显示管对应哪几个点阵。使用 dot 数组存储这些信息。每个显示管可以点亮 3 个点,因此存储下每个显示管相对于左上角的偏移量——往右是增加 y 坐标,往下是增加 x 坐标,如图 5-2(b)所示。这里使用 dot 变量来存储每个显示管的 3 个点的 x 和 y 的迁移

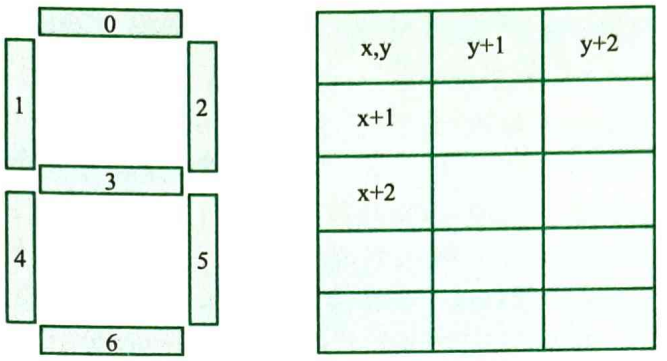

图 5-2　显示管和液晶点阵

量，仍然人工处理数据并初始化。

将数字读入字符数组中，然后开立 out 字符数组用于存储答案。可以知道第 i 个数字中，左上角的 x 坐标是 i*4（使用 basex 变量），y 坐标是 0（使用 basey 变量）。根据上面的信息，枚举这个数字的每一个显示管。对于每一个显示管来说，将它对应的 3 个点的坐标，把 out 数组的指定位置替换成 X。然后输出这个字符数组。代码如下：

```
#include <iostream>
#include <cstdio>
using namespace std;
int main() {
    int tubes[10][8] = { // 数码 i 的第 j 个显示管是什么
        {6, 0, 1, 2, 4, 5, 6}, {2, 2, 5}, {5, 0, 2, 3, 4, 6}, {5, 0, 2, 3, 5, 6}, // 0123
        {4, 1, 2, 3, 5}, {5, 0, 1, 3, 5, 6}, {6, 0, 1, 3, 4, 5, 6}, {3, 0, 2, 5}, // 4567
        {7, 0, 1, 2, 3, 4, 5, 6}, {6, 0, 1, 2, 3, 5, 6} // 89
    };
    int dot [7][3][2] = { // 每个显示管的 3 个点相对于左上角的坐标偏移
        {{0, 0}, {0, 1}, {0, 2}},
        {{0, 0}, {1, 0}, {2, 0}},
        {{0, 2}, {1, 2}, {2, 2}},
        {{2, 0}, {2, 1}, {2, 2}},
        {{2, 0}, {3, 0}, {4, 0}},
        {{2, 2}, {3, 2}, {4, 2}},
        {{4, 0}, {4, 1}, {4, 2}},
    };
    char num[110], out[5][500];
    int n;
    cin >> n;
    for (int i = 0; i < n; i++)
        cin >> num[i];
    for (int i = 0; i < 5; i++)
        for (int j = 0; j < 4 * n - 1; j++)
            out[i][j] = '.';
```

```
    for (int i = 0; i < n; i++) { // 处理每个字符
        int basex = 0, basey = i * 4, digit = num[i] - '0';
        /* basex 和 basey 是每个数字左上角的坐标值，dight 是正在处理哪一个数码（转换成
int)*/
        for (int j = 1; j <= tubes[digit][0]; j++) { // 处理每个要被点亮的显示管
            int tubenum = tubes[digit][j]; //第几个显示管点亮
            out[basex+dot[tubenum][0][0]][basey+dot[tubenum][0][1]] = 'X';
            out[basex+dot[tubenum][1][0]][basey+dot[tubenum][1][1]] = 'X';
            out[basex+dot[tubenum][2][0]][basey+dot[tubenum][2][1]] = 'X';
        }
    }
    for (int i = 0; i < 5; i++, cout << endl)
        for (int j = 0; j < 4 * n - 1; j++)
            cout << out[i][j];
    return 0;
}
```

在程序中使用了 digit 变量和 tubenum 变量作为中间变量，用于记录正在处理哪个数字，以及第几个显示管将被点亮。如果不这么做，这条语句就会写成 out[basex + dot[tubes[num[i]-'0'][j]][0][0]][basey + dot[tubes[num[i]-'0'][j]][0][1]]='X';。虽然它们的意思是一样的，但是这条语句非常冗长难懂，因此如果用到了复杂的数组下标表示，可以利用中间变量来简化下标运算。

本题有点不好理解，且做法不唯一，请读者仔细阅读并尝试实现。

5.4 课后习题与实验

习题 5-1 梦中的统计（洛谷 P1554，USACO 未知年份比赛）。给出两个整数 M 和 N（$1 \leqslant M \leqslant N \leqslant 2 \times 10^9$，$N-M \leqslant 500000$），求每一个数码（从 0 到 9）出现了多少次。

习题 5-2 珠心算测试（洛谷 P2141，NOIP2014 普及组）。给出 n（$n \leqslant 100$）个不超过 10000 互不相同的正整数，求这些数字中，有多少个数恰好等于集合中另外两个不同的数之和。

习题 5-3 爱与愁的心痛（洛谷 P1614）。给出 n（$n \leqslant 3000$）个不超过 100 的非负整数，组成一个序列。求一个长度为 m（$m \leqslant n$）的连续子序列（也就是 m 个连续的数字），使其和最小。例如 8 个数字组成的序列[1,4,7,3,1,2,4,3]求一个长度为 3 的连续子序列，当子序列为[3,1,2]时，总和最小值为 6。

提示：尝试仅使用一重循环完成此题，没有必要重复计算求和的过程。

习题 5-4 牛骨头（洛谷 P2911，USACO2008 October）。贝茜有了 3 个骰子，这 3 个不同的骰子的面数分别为 S_1、S_2、S_3（$1 \leqslant S_1, S_2, S_3 \leqslant 40$）。对于一个有 S 面的骰子每面上的数字是 1,2,…,S。每面上的数字出现的概率均等。现在给出每个骰子的面数，贝茜希望找出在所有“3 面上的数字的和”中，哪个和的值出现的概率最大。如果有很多个和出现的概率相同，那么只需要输出最小的那个。例如骰子的面分别是 3、2、3 时，其和为 5 或 6 出现的可能性最大，答案就是 5。

习题 5-5 开灯（洛谷 P1161）。路上有无数盏关着的灯，编号为 1,2,3,…。“反向操作”是指将关着的灯打开，或者把打开的灯关上。一共有 n（$n \leqslant 5000$）次操作，第 i 次操作给出两

个数 a_i 和 t_i，其中 a_i 是 1~1000 的实数，$t_i \leqslant T$。每次操作需要把所有编号为 $\lfloor a_i \rfloor$，$\lfloor 2 \times a_i \rfloor$，$\lfloor 3 \times a_i \rfloor$，…，$\lfloor t_i \times a_i \rfloor$ 的灯全部“反向操作”一次。所有操作结束后，编号最小的开着的灯是哪一盏？

本题中 $\lfloor x \rfloor$ 是将实数向下取整，例如 $\lfloor 2 \rfloor = \lfloor 2.1 \rfloor = \lfloor 2.9 \rfloor = 2$。测试数据保证对于所有的 i 来说，$t_i \times a_i$ 的最大值不超过 2000000，且所有 t_i 的总和不超过 2000000。

习题 5-6　蛇形方阵（洛谷 P5731）。输出蛇形方阵。例如输入 4 时，输出：

```
 1  2  3  4
12 13 14  5
11 16 15  6
10  9  8  7
```

习题 5-7　杨辉三角（洛谷 P5732）。给出 n（$n \leqslant 20$），输出杨辉三角的前 n 行。例如当 n=6 时，输出：

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
```

习题 5-8　Mc 生存——插火把（洛谷 P1789）。有一天，linyorson 在“我的世界”中新建了一个 $n \times n$（$n \leqslant 100$）的方阵，现在他有 m 个火把和 k 个萤石，分别放在 (x_1, y_1)，…，(x_m, y_m) 和 (o_1, p_1)，…，(o_k, p_k) 的位置，没有光或没放东西的地方会生成怪物。请问：在这个方阵中有几个点会生成怪物？

火把的照亮范围是：

```
| 暗 | 暗 |  光  | 暗 | 暗 |
| 暗 | 光 |  光  | 光 | 暗 |
| 光 | 光 | 火把 | 光 | 光 |
| 暗 | 光 |  光  | 光 | 暗 |
| 暗 | 暗 |  光  | 暗 | 暗 |
```

萤石：

```
| 光 | 光 |  光  | 光 | 光 |
| 光 | 光 |  光  | 光 | 光 |
| 光 | 光 | 萤石 | 光 | 光 |
| 光 | 光 |  光  | 光 | 光 |
| 光 | 光 |  光  | 光 | 光 |
```

习题 5-9　压缩技术（洛谷 P1319）。设某汉字由 $N \times N$ 的 0 和 1 的点阵图案组成。依照以

下规则生成压缩码的连续一组数值:从汉字点阵图案的第一行第一个符号开始计算,从左到右,由上至下。第一个数表示连续有几个0,第二个数表示接下来连续有几个1,第三个数表示再接下来连续有几个0,第四个数接着连续几个1,以此类推。例如以下汉字点阵图案:

```
0001000
0001000
0001111
0001000
0001000
0001000
1111111
```

对应的压缩码是:7 3 1 6 1 6 4 3 1 6 1 6 1 3 7(第一个数是N,其余各位交替表示0和1的个数,压缩码保证$N\times N=$交替的各位数之和)

现在给出压缩码,请求出汉字点阵图(不留空格)。

习题5-10 压缩技术——续集版(洛谷P1320)。压缩方法和上例一样,但是给出汉字的点阵图,需要求出对应的压缩码。

习题5-11 方块变换(洛谷P1205,USACO Training)。一块$N\times N(1\leqslant N\leqslant 10)$正方形的黑白瓦片的图案(输入中由@和-组成)要被转换成新的正方形图案。写一个程序来找出将原始图案按照以下转换方法转换成新图案的最小方式。

1) 转90°:图案按顺时针转90°。

2) 转180°:图案按顺时针转180°。

3) 转270°:图案按顺时针转270°。

4) 反射:图案在水平方向翻转(以中央铅垂线为中心形成原图案的镜像)。

5) 组合:图案在水平方向翻转,然后再按照1~3之间的一种再次转换。

6) 不改变:原图案不改变。

7) 无效转换:无法用以上方法得到新图案。

如果有多种可用的转换方法,请选择序号最小的那个。需要注意的是,转换操作只有1次。例如输入样例是

```
3
@-@
---
@@-
@-@
@--
--@
```

可以发现,原图(输入数据的2~4行)按顺时针旋转90°可以变为所给图形转换后的图形(输入数据的后3行),所以输出1。