```
        scanf("%d",&t);
        b+=t;                                    //bi 累加到 b
    }
    printf("%I64d\n",(long long)a*b);            //输出答案，注意用 64 位整数
    return 0;
}
```

5.1.4 部分测试数据和输出结果

测试数据

```
5 4
1
2
3
4
5
1
2
3
4
```

输出结果

```
150
```

5.2 彩球游戏（难度：★★★★☆）

5.2.1 试题

题目描述

Sarah 最近迷上了一个彩球游戏，游戏在一个 $N \times M$ 方格上分布有红、蓝、绿三种颜色的彩球，游戏者每次可以选中一个彩球，进行两种操作，如图 5.2.1 所示。

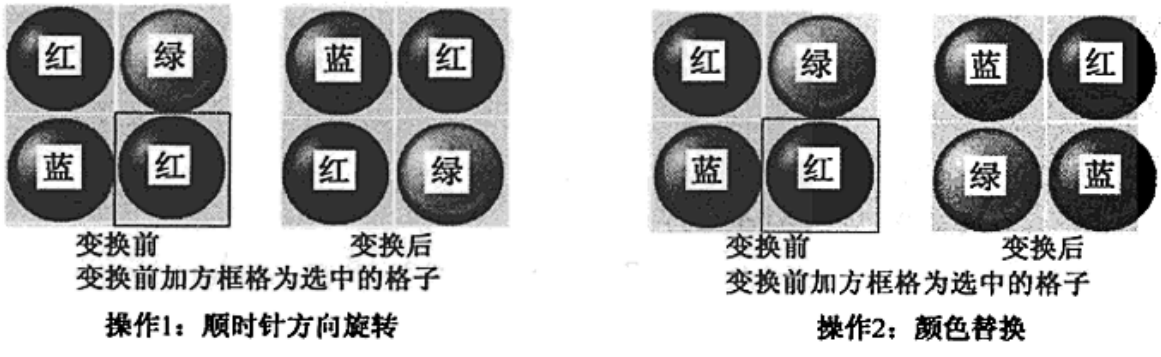

图 5.2.1 彩球游戏的两种操作

操作 1：把以选中的球为右下角的四个相邻小球进行顺时针旋转。

操作 2：把以选中的球为右下角的四个相邻小球进行颜色替换，替换规则为红色变蓝色，

蓝色变绿色，绿色变红色。

注意：每次操作都必须且仅能使用四个小球。

这个游戏的目标是从给出的初始状态出发，用最少的操作达到目标状态。

输入格式

输入不超过 5 组数据。

每组数据的第 1 行为 N, M 两个整数，输入 N=0 表示输入结束。

然后 N 行，每行是长度为 M 的字符串，表示初始状态，用 RBG 代表红蓝绿三种颜色。

然后 N 行，每行是长度为 M 的字符串，表示目标状态。

输出格式

对于每组数据，输出一个整数单独占一行，表示由初始状态到目标状态的最少操作次数，如果无解则输出–1。

输入样例

```
2 2
RG
BR
BR
RG
2 2
RG
BR
BR
BR
GB
0
```

输出样例

```
1
1
```

数据范围

对于 30%的数据，有 $2 \leqslant N, M \leqslant 4, N \times M \leqslant 9$。

对于 50%的数据，有 $2 \leqslant N, M \leqslant 4, N \times M \leqslant 12$。

对于全部的数据，有 $2 \leqslant N, M \leqslant 4, N \times M \leqslant 16$。

5.2.2 题目分析和算法实现

这道题的题意是，给定一个最大为 4×4，只有三种颜色的小球棋盘的两个状态和两种操作，求从初始状态到目标状态的最少操作步数。每次操作对象是相邻 2×2 的 4 个小球，操作具有可逆性。

由于问题具有后效性，简单的动态规划算法难以处理。

状态总数有 3^{16}，大于 4 千万！使用简单的单向广度优先搜索（BFS）也难以在短时间

内得出解。可以用双向 BFS、A*等方法快速求解。

使用双向广度优先搜索，从开始状态和结束状态分别扩展，直到两个扩展队列有交集，状态数减少为 2×3^8，可有效减少搜索的状态总数。对本题而言最坏情况数据也能在数秒内出解。可以对状态进行编码以便判断两个扩展队列是否有交集。

5.2.3 参考程序及程序分析

```
#include <time.h>
#include <cstdio>
#include <cstring>
#include <queue>
using namespace std;
int n, m;                                    //n 行 m 列的棋盘
int encode(char sm[4][5]) {                  //对棋盘状态进行编码
    int ret = 0;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            ret = ret * 3 + sm[i][j];        //编码方式为三进制，因为只有 3 种颜色
        }
    }
    return ret;
}
void decode(int s, char sm[4][5]) {          //对棋盘状态进行解码
    for (int i = n - 1; i >= 0; --i) {
        for (int j = m - 1; j >= 0; --j) {
            sm[i][j] = s % 3;                //像三进制那样解码
            s /= 3;
        }
    }
}
int pp[4][4];                                //pp 数组用在转换状态时方便计算编码
void init(int n, int m) {                    //初始化 pp 数组
    int cur = 1;
    for (int i = n - 1; i >= 0; --i) {
        for (int j = m - 1; j >= 0; --j) {
            pp[i][j] = cur;
            cur *= 3;
        }
    }
}
#define maxm 43046721                        //最大状态数
queue<int> que[2];                           //两个方向队列
unsigned char table[maxm];                   //哈希表判重，记录某状态走了多少步
char vvv[maxm];                              //标记该状态由正向得到还是逆向得到
```

```
void insert(int v, int a, int b) {      //往哈希表中插入元素
    table[v] = b;                       //状态 v 走了 b 步
    vvv[v] = a;                         //a 为 0 时状态 v 是正向得到的，否则是反向得到的
}
int change[3] = {2, 0, 1};              //改颜色数组
int restore[3] = {1, 2, 0};             //恢复改颜色数组
void modify(char &p) {                  //修改颜色字符
    if (p == 'R') p = 0;
    else if (p == 'G') p = 1;
    else p = 2;
}

FILE * fin ,* fout;                                 //输入输出文件句柄

int main () {
    fin = fopen("colball.in", "r");                 //打开输入文件
    fout = fopen("colball.out", "w");               //打开输出文件

    while (fscanf(fin,"%d", &n), n) {
        while(!que[0].empty()) que[0].pop();        //清空正向队列
        while(!que[1].empty()) que[1].pop();        //清空反向队列
       fscanf(fin,"%d", &m);
       init(n, m);
       char sm[4][5], tm[4][5];                     //初始状态和终止状态
for (int i = 0; i < n; ++i) {                       //读入初始状态到 sm
    fscanf(fin,"%s", sm[i]);
    for (int j = 0; j < m; ++j) {
       modify(sm[i][j]);
    }
}
for (int i = 0; i < n; ++i) {                       //读入终止状态到 tm
    fscanf(fin,"%s", tm[i]);
    for (int j = 0; j < m; ++j) {
       modify(tm[i][j]);
    }
}
memset(vvv, -1, pp[0][0] * 3);
que[0].push(encode(sm));                            //初始状态加入正向队列
que[1].push(encode(tm));                            //终止状态加入反向队列
insert(encode(sm), 0, 0);                           //初始状态加入哈希表
if (vvv[encode(tm)] == 0) {
    fprintf(fout,"0\n");
    continue;
}
```

```
insert(encode(tm), 1, 0);                          //终止状态加入哈希表
int ans = -1;
while (que[0].size() && que[1].size()) {           //当两个队列都有元素时循环
    int p = 0, sz = que[0].size();
    if (que[1].size() < sz) p = 1, sz = que[1].size();
                                                   //扩展元素个数较少的那个方向的队列
    while (sz--) {
        int cur = que[p].front();                  //cur 保存当前扩展队列的队头元素
          que[p].pop();                            //出队列
        int scur = table[cur];                     //scur 保存当前状态走了多少步
        decode(cur, sm);                           //将当前状态解码到 sm
        memcpy(tm, sm, sizeof(sm));
        if (p == 0) {                              //如果是正向方式
          for (int i = 1; i < n; ++i) {            //枚举操作位置
           for (int j = 1; j < m; ++j) {
            int a;
            a = sm[i - 1][j - 1];                  //第一种操作：顺时针转圈
            sm[i - 1][j - 1] = sm[i][j - 1];
            sm[i][j - 1] = sm[i][j];
            sm[i][j] = sm[i - 1][j];
            sm[i - 1][j] = a;

            int nxt = cur;                         //nxt 计算新状态的编码
            nxt += (sm[i - 1][j - 1] - tm[i - 1][j - 1])* pp[i - 1][j - 1];
            nxt += (sm[i - 1][j] - tm[i - 1][j]) * pp[i - 1][j];
            nxt += (sm[i][j - 1] - tm[i][j - 1]) * pp[i][j - 1];
            nxt += (sm[i][j] - tm[i][j]) * pp[i][j];
            int qv = vvv[nxt];                     //查询 nxt 状态有没有访问过
            if (qv == -1) {                        //如果没有访问过
                insert(nxt, p, scur + 1);          //插入哈希表
                que[p].push(nxt);                  //插入队列
            } else if (qv != p) {  //如果另外一种方向访问过，即双向广搜相遇
                ans = scur + table[nxt] + 1;       //找到答案，计算总步数
                goto ex;                           //跳出循环
            }

            a = sm[i - 1][j - 1];                  //还原为原来的状态
            sm[i - 1][j - 1] = sm[i - 1][j];
            sm[i - 1][j] = sm[i][j];
            sm[i][j] = sm[i][j - 1];
            sm[i][j - 1] = a;

            sm[i - 1][j - 1] = change[sm[i - 1][j - 1]]; //第二种操作，变颜色
            sm[i - 1][j] = change[sm[i - 1][j]];
```

```
            sm[i][j - 1] = change[sm[i][j - 1]];
            sm[i][j] = change[sm[i][j]];

            nxt = cur;                                  //nxt 计算新状态的编码
            nxt += (sm[i - 1][j - 1]- tm[i - 1][j - 1])*pp[i - 1][j - 1];
            nxt += (sm[i - 1][j] - tm[i - 1][j]) * pp[i - 1][j];
            nxt += (sm[i][j - 1] - tm[i][j - 1]) * pp[i][j - 1];
            nxt += (sm[i][j] - tm[i][j]) * pp[i][j];
            qv = vvv[nxt];                              //查询 nxt 状态有没有访问过
            if (qv == -1) {                             //如果没有访问过
               insert(nxt, p, scur + 1);                //插入哈希表
               que[p].push(nxt);                        //插入队列
             } else if (qv != p) {   //如果另外一种方向访问过，即双向广搜相遇
                ans = scur + table[nxt] + 1; //找到答案，计算总步数
                goto ex;                                //跳出循环
             }
             //还原为原来的状态
             sm[i - 1][j - 1] = restore[sm[i - 1][j - 1]];
             sm[i - 1][j] = restore[sm[i - 1][j]];
             sm[i][j - 1] = restore[sm[i][j - 1]];
             sm[i][j] = restore[sm[i][j]];
          }
         }
      } else {                                          //如果是反向方式
      for (int i = 1; i < n; ++i) {                     //枚举操作位置
       for (int j = 1; j < m; ++j) {
          int a;
          a = sm[i - 1][j - 1];                         //第一种操作：顺时针转圈
          sm[i - 1][j - 1] = sm[i - 1][j];
          sm[i - 1][j] = sm[i][j];
          sm[i][j] = sm[i][j - 1];
          sm[i][j - 1] = a;
          int nxt = cur;                                //nxt 计算新状态的编码
          nxt += (sm[i - 1][j- 1]-tm[i - 1][j - 1])*pp[i - 1][j - 1];
          nxt += (sm[i - 1][j] - tm[i - 1][j]) * pp[i - 1][j];
          nxt += (sm[i][j - 1] - tm[i][j - 1]) * pp[i][j - 1];
          nxt += (sm[i][j] - tm[i][j]) * pp[i][j];
          int qv = vvv[nxt];                            //查询 nxt 状态有没有访问过
          if (qv == -1) {                               //如果没有访问过
             insert(nxt, p, scur + 1);                  //插入哈希表
             que[p].push(nxt);                          //插入队列
           } else if (qv != p) { //如果另外一种方向访问过，即双向广搜相遇
              ans = scur + table[nxt] + 1;              //找到答案，计算总步数
              goto ex;                                  //跳出循环
           }
```

```
a = sm[i - 1][j - 1];                    //还原为原来的状态
sm[i - 1][j - 1] = sm[i][j - 1];
sm[i][j - 1] = sm[i][j];
sm[i][j] = sm[i - 1][j];
sm[i - 1][j] = a;

sm[i - 1][j - 1] = restore[sm[i - 1][j - 1]];
                                         //第二种操作，变颜色
sm[i - 1][j] = restore[sm[i - 1][j]];
sm[i][j - 1] = restore[sm[i][j - 1]];
sm[i][j] = restore[sm[i][j]];

nxt = cur;                               //nxt 计算新状态的编码
nxt += (sm[i-1][j-1] - tm[i-1][j-1]) * pp[i-1][j-1];
nxt += (sm[i - 1][j] - tm[i - 1][j]) * pp[i - 1][j];
nxt += (sm[i][j - 1] - tm[i][j - 1]) * pp[i][j - 1];
nxt += (sm[i][j] - tm[i][j]) * pp[i][j];
qv = vvv[nxt];                           //查询 nxt 状态有没有访问过
if (qv == -1) {                          //如果没有访问过
   insert(nxt, p, scur + 1);             //插入哈希表
   que[p].push(nxt);                     //插入队列
} else if (qv != p) {  //如果另外一种方向访问过，即双向广搜相遇
    ans = scur + table[nxt] + 1;         //找到答案，计算总步数
    goto ex;                             //跳出循环
                    }

                    //还原为原来的状态
                    sm[i - 1][j - 1] = change[sm[i - 1][j - 1]];
                    sm[i - 1][j] = change[sm[i - 1][j]];
                    sm[i][j - 1] = change[sm[i][j - 1]];
                    sm[i][j] = change[sm[i][j]];
                   }
                }
            }
        }
    }
    ex:;
    fprintf(fout,"%d\n", ans);                    //输出结果
  }
  fclose(fin);                                    //关闭输入文件
  fclose(fout);                                   //关闭输出文件
}
```

5.2.4 部分测试数据和输出结果

测试数据

```
3 3
BGR
RBB
GGG
RRR
RGB
RRR
3 2
RR
GB
GB
BB
GR
RR
2 3
BGG
GBR
BGG
GGR
3 3
BRG
BRR
BGG
RRR
BBR
RRG
3 3
BBB
BBB
BBG
GBB
BBB
BBB
0 0
```

输出结果

```
7
7
5
6
4
```