T1 一维跳棋

很显然，这是一道模拟题，需要做的就是分类讨论。

- 向左：移动或跳跃
- 向右：移动或跳跃

【参考代码】

```c
#include <stdio.h>

#define SIZE 100010

int chessboard[SIZE];
int arrive[SIZE];

int main() {
    int n;
    int pos;  // 游戏棋子的位置
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &chessboard[i]);
        if (chessboard[i] == 2) pos = i;
    }
    arrive[pos] = 1;

    int p = pos;
    // 向左跳
    if (pos > 1) {
        if (chessboard[pos - 1] == 0) {
            arrive[pos - 1] = 1;
        } else {
            while (p - 2 >= 1 && chessboard[p - 1] == 1 &&
                   chessboard[p - 2] == 0) {
                arrive[p - 2] = 1;
                p -= 2;
            }
        }
    }
    // 向右跳
    p = pos;
    if (pos < n) {
        if (chessboard[pos + 1] == 0) {
            arrive[pos + 1] = 1;
        } else {
            while (p + 2 <= n && chessboard[p + 1] == 1 &&
                   chessboard[p + 2] == 0) {
                arrive[p + 2] = 1;
                p += 2;
            }
        }
    }
    for (int i = 1; i <= n; i++) {
        printf("%d ", arrive[i]);
    }
```

```
    return 0;
}
```

T2 传纸条

这道题是让我们求“同学”间形成了多少个环，也就是多少个联通块。

方法一：模拟法

访问到一个同学时，将该同学的vis标记设置为1，然后沿着该同学传纸条的方向遍历。

```c
#include <stdio.h>
#define SIZE 100010

int n;
int passto[SIZE];
int vis[SIZE];
int cnt = 0;
int size[SIZE];

int main() {
    char type[10];
    scanf("%d", &n);
    scanf("%s", type);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &passto[i]);
    }
    for (int i = 1; i <= n; i++) {
        if (vis[i]) continue;
        vis[i] = 1;
        size[++cnt] = 1;
        int pos = i;
        while (passto[pos] != i) {
            size[cnt]++;
            pos = passto[pos];
            vis[pos] = 1;
        }
    }
    if (type[0] == 'n') {
        printf("%d\n", cnt);
    } else { // 冒泡排序
        for (int i = 1; i < cnt; i++) {
            for (int j = 1; j <= cnt - i; j++) {
                if (size[j] < size[j + 1]) {
                    int temp = size[j];
                    size[j] = size[j + 1];
                    size[j + 1] = temp;
                }
            }
        }
        for (int i = 1; i <= cnt; i++) {
            printf("%d ", size[i]);
        }
    }
    return 0;
}
```

> 因为要排序的数很小，其实用桶排序会更好，当然可以用归并排序、快速排序等速度较快的排序。考虑作为第二题，本题数据规模并不大。

方法二：可以用递归的方法解决

```
void dfs(int x) {
    if(vis[x]) return;
    vis[x] = 1;
    size[cnt]++;
    dfs(passto[x]);
}
// 主程序就留给大家自己思考了
```

方法三：求连通块，可以用并查集来做。

（超纲了~）有兴趣的同学可以去了解一下，会在大二《数据结构与算法》中学习

T3 预备队

由于本题n较小，注意到$kC_n^5 \leq 6C_{70}^5 \approx 7 \times 10^7$,且时限给了$2s$,因此可以$C_n^5$暴力枚举$n$个同学中取出5个的组合，然后用$kC_5^3$复杂度检验五位同学是否能凑出两个协调的小队，若能则统计入答案即可。

假如我们已经枚举出五位同学并且存储在num数组中，则可在五位同学中任意取出3位检验是否能够协调，检验代码如下：

```
bool check(int A, int B, int C) {   //检验ABC三位同学是否协调
    for (int i=1; i<=k; i++) {
        int rem=c[A][i]+c[B][i]+c[C][i];
        if (c[A][i]==c[B][i] && c【==c[C][i] || (c[A][i]!=c[B][i] && c[A][i]!=c[C][i] && c[B][i]!=c[C][i])) continue; //第i个属性值不冲突
        else return false;  //存在冲突
    }
    return true;    //不存在冲突
}

bool Check() {      //检验五位同学能否凑成一个预备队
    int rem=0;
//任意枚举出三位同学
    for (int i=1; i<=5; i++)
        for (int j=i+1; j<=5; j++)
            for (int k=j+1; k<=5; k++) {
                if (check(num[i], num[j], num[k])) ++rem;
            }
    return rem>=2;
}
```

下面问题转化为如何枚举出五位同学的组合，容易使用dfs(此题亦即简单递归)解决，详见代码.

```
void solve(int now, int pre) {    //从pre到n枚举第now位同学
    if (now==6) {    //递归边界：如果已经选完五位同学，那么进入检验阶段
        if (Check()) ++res;    //五位同学能组成预备队，计入答案
        return;
    }
    for (int i=pre; i<=n; i++) {
        num[now]=i;    //第now位同学为i
        solve(now+1, i+1);//为避免重复计数，组合中五位同学的编号保持递增，因此从i+1开始递归枚举下一位同学
    }
}
```

本题旨在考查大家对递归的掌握，所以屑出题人Lg在提示中加入了一句极具诱导性的话:"请尽可能利用递归的思维解决此题". 其实递归层数是固定的五层，因此可以使用简单的五重循环代替递归枚举所有C_n^5种组合.代码如下:

```
void solve2() {
    for (num[1]=1; num[1]<=n; num[1]++) {
        for (num[2]=num[1]+1; num[2]<=n; num[2]++) {
            for (num[3]=num[2]+1; num[3]<=n; num[3]++) {
                for (num[4]=num[3]+1; num[4]<=n; num[4]++) {
                    for (num[5]=num[4]+1; num[5]<=n; num[5]++) {
                        if (Check()) ++res;
                    }
                }
            }
        }
    }
}
```

两个$solve$本质上是同一种方法的两个不同写法，最好都能够掌握并灵活运用。本题完整代码如下:

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

int n, k;
int c[102][22];
int num[6], res;

bool check(int A, int B, int C) {   //检验ABC三位同学是否协调
    for (int i=1; i<=k; i++) {
        int rem=c[A][i]+c[B][i]+c[C][i];
        if (c[A][i]==c[B][i]==c[C][i] || (c[A][i]!=c[B][i] && c[A][i]!=c[C][i] && c[B][i]!=c[C][i])) continue; //第i个属性值不冲突
        else return false;  //存在冲突
    }
    return true;    //不存在冲突
}

bool Check() {      //
    int rem=0;
    for (int i=1; i<=5; i++)
        for (int j=i+1; j<=5; j++)
            for (int k=j+1; k<=5; k++) {
                if (check(num[i], num[j], num[k])) ++rem;
            }
```

```
    return rem>=2;
}

void solve(int now, int pre) {
    if (now==6) {
        if (Check()) ++res;
        return;
    }
    for (int i=pre; i<=n; i++) {
        num[now]=i;
        solve(now+1, i+1);
    }
}

void solve2() {
    for (num[1]=1; num[1]<=n; num[1]++) {
        for (num[2]=num[1]+1; num[2]<=n; num[2]++) {
            for (num[3]=num[2]+1; num[3]<=n; num[3]++) {
                for (num[4]=num[3]+1; num[4]<=n; num[4]++) {
                    for (num[5]=num[4]+1; num[5]<=n; num[5]++) {
                        if (Check()) ++res;
                    }
                }
            }
        }
    }
}

int main() {
    scanf("%d%d", &n, &k);
    for (int i=1; i<=n; i++) {
        for (int j=1; j<=k; j++) {
            scanf("%d", &c[i][j]);
        }
    }
    solve(1, 1);
    // solve2();
    printf("%d\n", res);
}
```

(分割线以下感性理解即可，不必完全掌握，供学有余力的读者思考.)

更加优雅的做法

数据范围扩大至$k \leq 20, n \leq 1000$

算法：二进制状态压缩，枚举，排序，二分查找

每位同学有k个属性值，因此可以将第p位同学抽象为一个k维$vector$(向量) p，第i个维度的坐标值代表第i个属性值，记作$c_{p,i}$.

如何简单表示一个$vector$?

一个$vector$需要判断k个坐标，但由于存在限定条件$0 \leq c_{p,k} \leq 2, k \leq 20$.且一个$longlong$类型的变量有64位，因此可以将一个$vector$映射到一个$longlong$类型的变量.具体过程如下：

定义$longlong$类型变量num.对于$vector$的第i个坐标，记作$c_i(0 \le c_i \le 2)$,用num的第$[3(i-1):3i-1]$这三位来记录，其中$0,1,2$分别记录为$000,001,010$即可. 例如若$vector=(0,1,2,2,1)$,则用num的低15位记录,其余高位全部补0，有$num=(000,001,010,010,001)_2=(657)_{10}$

（事实上记录单个$vector$单个坐标用两位即可，但这里开了三位，是为后面相加做铺垫）

采用这种方法，可以把任意的$vector$映射到$longlong$变量num中.

如果我们已经得到一个确定的$vector$，即k个属性值，如何迅速判断给定的n个$vector$中是否存在这个$vector$？如果存在，又如何找出它的编号？

排序+二分查找。

首先将给定的n个$vector$映射到num中，将$num[]$数组按升序排序，由于数据规模较小，因此可以使用选择排序、冒泡排序、归并排序、快速排序等正常的排序方法。

若目标$vector$二进制压缩后值为$goal$,则我们可以使用二分查找寻找其在num中的位置，详见代码.

```
//其实就是类似于猜数字游戏啦
//Lower_bound函数求出num数组中第一个不小于goal的位置
//然后递归完得到结果pos后再比较一下pos下的值是否等于goal，等于就找到了
int Lower_bound(int l, int r, long long goal) {
    if (l==r) return l;
    int mid=l+r>>1;
    if (num[mid]<goal) return Lower_bound(mid+1, r, goal);
    else return Lower_bound(l, mid, goal);
}
```

三位协调同学组成的小队记为一个set。一个预备队，即有至少两种方案凑出一个set的五个$vector$记作一个$meta$. 本题目标即求出可能的$meta$种数.

前提：所有k元向量互不相同

事实上，我们并不容易看出如下性质：

引理1:若五个k元向量能够构成一个$meta$,那其中能且仅能构出两个set

引理2:若一set中两元素确定，则第三元素唯一确定，且不与确定的两个向量中任意一个相等.

先证明引理2.由于构成set的条件是三个k元向量每一位都相同或互不相同，故对于每一位而言，仅有$\{0,0,0\},\{1,1,1\},\{2,2,2\},\{0,1,2\}$四种，对应的和为$0,3,6$.下面讨论选定2个确定第三个的情况：

选定2个向量，第$i(i \le k)$位有以下情况：$\{0,0\},\{0,1\}\{0,2\}\{1,1\}\{1,2\}\{2,2\}$，对应的和依次为$0,1,2,2,3,4$,所需第三向量第$i$位依次为$0,2,1,1,0,2$,（注意到$\{0,2\}$和$\{1,1\}$的和以及所需数字均相同，但不会影响引理2的合理性.） 由于选定两个向量后，每一位的和确定，因此所需第三向量的每一位也确定，又由于所有向量不重复，故引理2前半部分得证.

后半部分比较显然.故引理2完整得证。

在引理2的基础上，我们接着证明引理1.假设一个$meta$中五个向量分别为a,b,c,d,e,由于两个set至少涉及到6个向量，由于鸽巢原理，至少有一个向量被共享.因此引理1转化为有且仅有一个向量被共享.只需排除有两个向量被共享的情况即可.

若a,b被共享，那假设$\{a,b,c\}$，$\{a,b,d\}$分别构成set,由引理2可知，$c=d$,与引理前提中每个向量互不相同相矛盾，因此不存在两个向量被共享的情况.

只有一个元素共享，因此引理1得证。

推论1：$meta$中共享元素唯一.

到问题求解环节，根据引理1以及推论1,我们只需枚举每个$meta$中被共享的向量p_i，然后寻找有多少对向量二元组能够与p构成set,统计对数即为cnt_i,那么p_i对应的$meta$数即为$C_{cnt_i}^2 = \frac{cnt[i]*cnt[i-1]}{2}$ 整体时间复杂度$\mathrm{O}(n^2(logn+k))$

code:

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
#include<cstring>
#include<cmath>
#include<algorithm>
#include<map>
#include<queue>
#include<vector>
#include<set>
#include<bitset>
#include<cassert>
#include<ctime>
#define next MabLcdGsaddsa
#define R register
#define debug puts("lg")
#define mod 998244353
#define chkmax(x, y) (x=max(x, y))
#define chkmin(x, y) (x=min(x, y))
using namespace std;
typedef long long ll;
// typedef int ll;
typedef long double ld;
typedef unsigned long long ull;
typedef double dl;

const ll N=1e3+3, M=22;

ll cnt[N];
ll sum[N*N], tot;    //sum数组表示
ll num[N];
ll n, k;
const ll p[5]={0, 2, 1, 0, 2};
//p表示和为0，1，2，3，4时第三个vector的坐标值
int main() {
    // read(n); read(k);
    cin>>n>>k;
    for (R ll i=1; i<=n; i++) {
        for (R ll j=0, x; j<k; j++) {
            read(x);
            num[i]+=x*(1ll<<(j*3));
        }
    }
    sort(num+1, num+n+1);
    for (R ll i=1; i<=n; i++) {
        for (R ll j=i+1; j<=n; j++) {
            sum[++tot]=num[i]+num[j];
//将第i和j个vector的映射值相加记录入sum，后面结合p数组可以求出与i和j构成set的第三个
vector
```

```
        }
    }
    for (R ll i=1, c; i<=tot; i++) {
        c=0;
        for (R ll j=0; j<k; j++) {
            c+=p[((sum[i]>>(j*3))&7)]<<(j*3);
        }
//查找vector c是否存在
        ll x=lower_bound(num+1, num+n+1, c)-num;
        if (num[x]==c) ++cnt[x];
    }
    ll res=0;
    for (R ll i=1; i<=n; i++) {
        res+=cnt[i]*(cnt[i]-1)/2;
    }
    cout<<res<<endl;
}
//屑出题人Lg写的C++，用的是stl自带lower_bound和sort
```

T4 蚂蚁国手

这道题目是上学期CPL期末考试第三题五子棋的魔改版，仍然是判断三步之内有无胜者，对于母题而言有更简单直观的做法，在这里先不作讲解，对于子题而言，由于棋子的行走方式和棋局结束的多变性，最直观的方式就是对于每一颗蚂蚁的棋子都向所有可能的方向走出一步，如果出现将军的情况就让每一颗Corax的棋子向所有的方向尝试，如果Corax每个棋子的尝试均失败，则蚂蚁获胜；如果蚂蚁的所有棋子都没有制胜，则Corax获胜。

题目本身相对复杂，囿于出题人时间有限标程也没有经过很好的优化，大家可以看个乐，你也可以尝试对它进行优化。但其复杂性的目的是培养大家在有限时间内获取部分分的能力和思想，你可以思考题目最后数据范围对你获得部分分有什么助益，希望大家经此一役后能在期末考试时遇到没有思路的题目时稳扎稳打，收获令自己满意的成绩

【参考代码】

```
#include <math.h>
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#define enemy(m) ((m) == 'h' || (m) == 'c' || (m) == 'g' || (m) == 's')
#define ally(m) ((m) == 'k' || (m) == 'b' || (m) == 'e')
char map[10][10], tmp[10][10];
int dir[150];
int dx[150][30], dy[150][30];
bool flag;
void copy() {
    memset(tmp, '#', sizeof(tmp));
    for (int i = 1; i <= 5; i++) {
        for (int j = 1; j <= 9; j++) {
            tmp[i][j] = map[i][j];
        }
    }
}
bool cornerCheck(char c, int x, int y) {
    if (x <= 0 || x >= 6 || y <= 0 || y >= 10)
        return false;
    if (enemy(c)) {
        if (enemy(tmp[x][y]))
```

```
            return false;
    } else {
        if (ally(tmp[x][y]))
            return false;
        if (c != 'e' && (x >= 4 || y <= 3 || y >= 7))
            return false;
    }
    return true;
}
int checkObs(int x, int y, int nx, int ny) {
    int cnt = 0;
    if (y == ny) {
        for (int i = (x < nx ? x : nx) + 1; i < (x > nx ? x : nx); i++) {
            if (tmp[i][y] != '#')
                cnt++;
        }
    } else {
        for (int i = (y < ny ? y : ny) + 1; i < (y > ny ? y : ny); i++) {
            if (tmp[x][i] != '#')
                cnt++;
        }
    }
    return cnt;
}
bool attack(char c, int x, int y) {
    if (c == 'c') {
        for (int i = 1; i <= 9; i++) {
            if (tmp[x][i] == 'k' && checkObs(x, y, x, i) == 0)
                return true;
        }
        for (int i = 1; i <= 5; i++) {
            if (tmp[i][y] == 'k' && checkObs(x, y, i, y) == 0)
                return true;
        }
    } else if (c == 'g') {
        for (int i = 1; i <= 9; i++) {
            if (tmp[x][i] == 'k' && checkObs(x, y, x, i) == 1)
                return true;
        }
        for (int i = 1; i <= 5; i++) {
            if (tmp[i][y] == 'k' && checkObs(x, y, i, y) == 1)
                return true;
        }
    } else if (enemy(c)) {
        for (int i = 0; i < dir[c]; i++) {
            if (c == 'h' && tmp[x + (abs(dx[c][i]) > abs(dy[c][i])) *
                                    (dx[c][i] < 0 ? -1 : 1)]
                                [y + (abs(dx[c][i]) < abs(dy[c][i])) *
                                    (dy[c][i] < 0 ? -1 : 1)] != '#')
                continue;
            int nx = x + dx[c][i];
            int ny = y + dy[c][i];
            if (cornerCheck(c, nx, ny) && tmp[nx][ny] == 'k')
                return true;
        }
    }
    return false;
}
```

```
void move(int x, int y) {
    char c = map[x][y];
    copy();
    tmp[x][y] = '#';
    for (int i = 0; i < dir[c]; i++) {
        if (c == 'h' &&
            tmp[x + (abs(dx[c][i]) > abs(dy[c][i])) * (dx[c][i] < 0 ? -1 :
1)]
                [y + (abs(dx[c][i]) < abs(dy[c][i])) *
                        (dy[c][i] < 0 ? -1 : 1)] != '#')
            continue;
        int nx = x + dx[c][i];
        int ny = y + dy[c][i];
        if (!cornerCheck(c, nx, ny))
            continue;
        if (c == 'c') {
            if (checkObs(x, y, nx, ny))
                continue;
            if (enemy(tmp[nx][ny]))
                continue;
        }
        if (c == 'g') {
            if (checkObs(x, y, nx, ny) > 1)
                continue;
            else if (checkObs(x, y, nx, ny) == 1 && !ally(tmp[nx][ny]))
                continue;
            else if (checkObs(x, y, nx, ny) == 0 && tmp[nx][ny] != '#')
                continue;
        }
        char storec = tmp[nx][ny];
        tmp[nx][ny] = c;
        for (int bx = 1; bx <= 5; bx++) {
            for (int by = 1; by <= 9; by++) {
                if (attack(tmp[bx][by], bx, by)) {
                    bool solutable = false;
                    for (int j = 1; j <= 5; j++) {
                        for (int k = 1; k <= 9; k++) {
                            if (ally(tmp[j][k])) {
                                char chess = tmp[j][k];
                                tmp[j][k] = '#';
                                for (int z = 0; z < dir[chess]; z++) {
                                    if (chess == 'e' &&
                                        tmp[j + dx[chess][z] / 2]
                                           [k + dy[chess][z] / 2] != '#')
                                        continue;
                                    int newx = j + dx[chess][z];
                                    int newy = k + dy[chess][z];
                                    if (cornerCheck(chess, newx, newy)) {
                                        char sc = tmp[newx][newy];
                                        tmp[newx][newy] = chess;
                                        bool solved = true;
                                        for (int o = 1; o <= 5; o++) {
                                            for (int p = 1; p <= 9; p++) {
                                                if (enemy(tmp[o][p])) {
                                                    if (attack(tmp[o][p],
o,
                                                               p)) {
                                                        solved = false;
```

```
                                                    break;
                                                }
                                            }
                                        }
                                        if (!solved)
                                            break;
                                    }
                                    if (solved)
                                        solutable = true;
                                    tmp[newx][newy] = sc;
                                }
                                if (solutable)
                                    break;
                            }
                            tmp[j][k] = chess;
                            if (solutable)
                                break;
                        }
                    }
                    if (solutable)
                        break;
                }
                if (!solutable) {
                    flag = 0;
                    return;
                }
            }
        }
        tmp[nx][ny] = storec;
    }
}
void init() {
    dir['k'] = 4;
    dx['k'][0] = 1, dy['k'][0] = 0;
    dx['k'][1] = -1, dy['k'][1] = 0;
    dx['k'][2] = 0, dy['k'][2] = 1;
    dx['k'][3] = 0, dy['k'][3] = -1;
    dir['b'] = 4;
    dx['b'][0] = 1, dy['b'][0] = 1;
    dx['b'][1] = 1, dy['b'][1] = -1;
    dx['b'][2] = -1, dy['b'][2] = 1;
    dx['b'][3] = -1, dy['b'][3] = -1;
    dir['e'] = 4;
    dx['e'][0] = 2, dy['e'][0] = 2;
    dx['e'][1] = 2, dy['e'][1] = -2;
    dx['e'][2] = -2, dy['e'][2] = 2;
    dx['e'][3] = -2, dy['e'][3] = -2;
    dir['h'] = 8;
    dx['h'][0] = 1, dy['h'][0] = 2;
    dx['h'][1] = 1, dy['h'][1] = -2;
    dx['h'][2] = -1, dy['h'][2] = 2;
    dx['h'][3] = -1, dy['h'][3] = -2;
    dx['h'][4] = 2, dy['h'][4] = 1;
    dx['h'][5] = 2, dy['h'][5] = -1;
    dx['h'][6] = -2, dy['h'][6] = 1;
    dx['h'][7] = -2, dy['h'][7] = -1;
    dir['s'] = 3;
```

```
    dx['s'][0] = -1, dy['s'][0] = 0;
    dx['s'][1] = 0, dy['s'][1] = 1;
    dx['s'][2] = 0, dy['s'][2] = -1;
    dir['c'] = dir['g'] = 24;
    for (int i = 0; i < 4; i++) {
        dx['c'][i] = dx['g'][i] = i + 1;
        dy['c'][i] = dy['g'][i] = 0;
    }
    for (int i = 4; i < 8; i++) {
        dx['c'][i] = dx['g'][i] = 3 - i;
        dy['c'][i] = dy['g'][i] = 0;
    }
    for (int i = 8; i < 16; i++) {
        dx['c'][i] = dx['g'][i] = 0;
        dy['c'][i] = dy['g'][i] = i - 7;
    }
    for (int i = 16; i < 24; i++) {
        dx['c'][i] = dx['g'][i] = 0;
        dy['c'][i] = dy['g'][i] = 15 - i;
    }
}
int main() {
    int t;
    init();
    scanf("%d", &t);
    while (t--) {
        flag = 1;
        for (int i = 1; i <= 5; i++) {
            scanf("%s", map[i] + 1);
        }
        getchar();
        for (int i = 1; i <= 5; i++) {
            for (int j = 1; j <= 9; j++) {
                if (enemy(map[i][j])) {
                    move(i, j);
                }
                if (!flag)
                    break;
            }
            if (!flag)
                break;
        }
        if (flag)
            printf("YES\n");
        else
            printf("NO\n");
    }
    return 0;
}
```