Problem A. 最喜欢的课！

题目要求找到最小时间 t ，考虑二分。

对于二分出的时间 mid，对于所有准备好的调料，可以预处理出后缀和 suf。从前往后遍历所有披萨，当前披萨 i 的灵力值可以 $O(1)$ 计算为 $c_i \times (suf_{i+1} + 1)$。如果存在一个披萨灵力值 $\geq k$，那么返回 $true$；反之返回 $false$。

以下是本题的标准程序：

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 5;
int n, k, a[N], b[N], c[N], d[N];
long long suf[N];

bool check(int t) {
    suf[n + 1] = 0;
    for(int i = n; i >= 1; i --) {
        if(b[i] <= t)
            suf[i] = suf[i + 1] + a[i];
        else
            suf[i] = suf[i + 1];
    }
    long long res = 0;
    for(int i = 1; i <= n; i ++) {
        if(d[i] <= t) {
            res = max(res, 1ll * c[i] * (suf[i + 1] + 1));
        }
    }
    return res >= k;
}

void solve() {
    cin >> n >> k;
    for(int i = 1; i <= n; i ++) cin >> a[i];
    for(int i = 1; i <= n; i ++) cin >> b[i];
    for(int i = 1; i <= n; i ++) cin >> c[i];
    for(int i = 1; i <= n; i ++) cin >> d[i];

    int lo = 1, hi = 1e9;
    while(lo <= hi) {
        int mid = (lo + hi) >> 1;
        if(check(mid))
            hi = mid - 1;
        else
            lo = mid + 1;
    }
    if(lo > 1e9) {
        cout << "-1\n";
    } else {
        cout << lo << "\n";
    }
}

int main() {
    int TestCases;  cin >> TestCases;
    while(TestCases --) {
        solve();
    }

    return 0;
}
```

Problem B. 如果能穿越回高中

本题是用于防 AK 的大模拟。题目背景取自三国杀。其中的题目数据也有改自游戏某关（BV1uN411g7RT P143）。

标准程序使用了 BFS 对每一步可能到达的移动结果进行广搜并记录状态，但由于限制了步数，也可以采用 DFS 剪枝的方式进行深搜。

标准程序所计算出来的最短解法如下（不包含样例，按 case 顺序输出）。

```
9
dwwadsdwa
20
aawdsdsdsaasawaawdds
25
wadsdwwasadwawssdswdwdssa
30
sawawawsdwasdasawdwawdsdwdsasd
```

以下是本题的标准程序:

```
#include<bits/stdc++.h>
#define enm(x,y) ((s[x][y]=='w'||s[x][y]=='a'||s[x][y]=='s'||s[x][y]=='d')&&!u[x][y
    ])
#define li(x,y) (x>=0&&x<n&&y>=0&&y<m)
using namespace std;
int n,m,o=50,en,lx,ly;
string s[100];

struct st{
  string s;
  int x,y;
}p,p1;
queue<st>q;
const string f="wasd";
const int dx[4]={-1,0,1,0},dy[4]={0,-1,0,1};

bool u[100][100];
int main(){
  cin>>n>>m>>o;
  for(int i=0;i<n;i++){
    cin>>s[i];
    for(int j=0;j<m;j++){
      if(enm(i,j))en++;
      if(s[i][j]=='+')lx=i,ly=j;
    }
  }
  p.s="";p.x=0;p.y=-1;
  q.push(p);
  while(!q.empty()){
    p=q.front();q.pop();
    int xx=lx,yy=ly,cnt=0;
    memset(u,0,sizeof(u));
    for(int i=0;i<p.x;i++){
      int j=0;
      while(p.s[i]!=f[j])j++;
      while(1){
        xx+=dx[j];yy+=dy[j];
        if(enm(xx,yy)){
          u[xx][yy]=1;
```

```
            cnt++;
            break;
          }
          if(s[xx][yy]=='x'||s[xx][yy]=='='){
            xx-=dx[j];yy-=dy[j];
            break;
          }
        }
      }
      if(en-cnt>=o-p.x)continue;
      if(cnt==en){
        for(int i=0;i<4;i++){
          int tx=xx,ty=yy;
          while(1){
            tx+=dx[i];ty+=dy[i];
            if(s[tx][ty]=='x')break;
            if(s[tx][ty]=='='){
              cout<<p.x+1<<"\n";
              cout<<p.s+f[i]<<"\n";
              return 0;
            }
          }
        }
      }

      for(int i=0;i<4;i++){
        if(i==p.y)continue;
        int tx=xx,ty=yy;
        while(1){
          tx+=dx[i];ty+=dy[i];
          if(s[tx][ty]=='x'||s[tx][ty]=='='){
            tx-=dx[i];ty-=dy[i];
            if(tx==xx&&ty==yy)break;
            p1.s=p.s+f[i];p1.x=p.x+1;p1.y=(i+2)%4;
            q.push(p1);
            break;
          }
          if(enm(tx,ty)){
            p1.s=p.s+f[i];p1.x=p.x+1;p1.y=-1;
            q.push(p1);
            break;
          }

          bool dan=0;
          for(int j=0;j<4;j++){
            if(s[tx+dx[j]][ty+dy[j]]==f[(j+2)%4]&&!u[tx+dx[j]][ty+dy[j]]){
              dan=1;break;
            }
          }
          if(dan==1)break;

        }
      }
    }
    return 0;
}
```

Problem C. 被涂色的作文纸

对于本题，可行的解法是线性时间复杂度的动态规划。

设 $f_{i,0/1}$ 表示第 i 个单元格涂上红色/蓝色能获得的最大分数，我们可以得到以下方程：

$$\begin{cases} f_{i,0} = a_i + max(f_{i-1,0} + c_{i-1} + c_i, f_{i-1,1}) \\ f_{i,1} = b_i + max(f_{i-1,1} + c_{i-1} + c_i, f_{i-1,0}) \end{cases}$$

最终的答案为 $max(f_{n,0}, f_{n,1})$ 。

以下是本题的标准程序:

```
#include <iostream>
#include <algorithm>

const int N = 1010;

int n, a[N], b[N], c[N];
int f[N][2];

int main() {
  std::cin >> n;
  for (int i = 1; i <= n; i ++) std::cin >> a[i];
  for (int i = 1; i <= n; i ++) std::cin >> b[i];
  for (int i = 1; i <= n; i ++) std::cin >> c[i];

  f[1][0] = a[1], f[1][1] = b[1];
  for (int i = 2; i <= n; i ++) {
    f[i][0] = a[i] + std::max(f[i - 1][0] + c[i - 1] + c[i], f[i - 1][1]);
    f[i][1] = b[i] + std::max(f[i - 1][1] + c[i - 1] + c[i], f[i - 1][0]);
  }

  std::cout << std::max(f[n][0], f[n][1]) << std::endl;

  return 0;
}
```

Problem D. 数学课上的囚徒问题（1）

暴力枚举给定排列，使用一个数组记录当前数字的使用情况。从头开始执行遍历流程，如果当前数字没有被使用，则它在遍历过程中第一次出现并且属于当前的循环节，将其标记后记录当前循环节的长度增加。当遍历到被使用的数字时，证明该环已经全部遍历完成，计算总长度以后与最大值和最小值分别比较。再寻找下一个没有被使用的数字开始新的枚举流程，以此类推。

以下是本题的标准程序:

```
#include<bits/stdc++.h>
using namespace std;
int a[5010],ss[5010];
signed main()
{
  int n;
  cin>>n;
  for(int i=1;i<=n;i++)
  {
    cin>>a[i];
  }
  int min1=5010,max1=0;
  for(int i=1;i<=n;i++)
  {
    if(ss[i]==0)
    {
```

```
      int sum=1;
      ss[i]=1;
      int x=a[i];
      while(ss[x]==0)
      {
        sum++;
        ss[x]=1;
        x=a[x];
      }
      min1=min(min1,sum);
      max1=max(max1,sum);
    }
  }
  cout<<min1<<' '<<max1<<endl;
  return 0;
}
```

Problem E. 数学课上的囚徒问题（2）

这是一个经典的数学题，现在我们拿 100 个囚徒 100 个盒子来举例，随机排列的 100 个盒子，包含长度为 100 的循环概率是多少？显然，答案是 $\frac{1}{100}$。

这是一个普遍的结果，长度为 99 的概率是 $\frac{1}{99}$，长度为 98 的概率是 $\frac{1}{98}$，因此推广到长度超过 50 循环的概率就是 $\frac{1}{100}+\frac{1}{99}+...+\frac{1}{52}+\frac{1}{51}$，这个数字大概是 0.69，即囚徒们失败的概率，再拿 1 减去这个数即囚徒们成功的概率。以此说明推广到 N 个囚徒和 N 个盒子，发现这就是一个简单的数学公式，答案为 $1-[\frac{1}{N}+\frac{1}{N-1}+\frac{1}{N-2}+...+\frac{1}{\frac{N}{2}+2}+\frac{1}{\frac{N}{2}+1}]$，复杂度为 $O(\frac{N}{2})$。

下面是对于包含长度为 N 的循环出现概率为 $\frac{1}{N}$的补充证明：

先举个简单的例子： 3 个囚徒 3 个盒子，所有组合是：

1 2 3： 3 个 1 阶循环。

1 3 2： 1 个 1 阶循环， 1 个 2 阶循环。

2 1 3： 1 个 1 阶循环， 1 个 2 阶循环。

2 3 1： 1 个 3 阶循环。

3 1 2： 1 个 3 阶循环。

3 2 1： 1 个 1 阶循环， 1 个 2 阶循环。

设 $P(x)=$形成x阶循环的概率，有

$$\begin{cases} P(1)=\frac{4}{6}=\frac{2}{3} \\ P(2)=\frac{3}{6}=\frac{1}{2} \\ P(3)=\frac{2}{6}=\frac{1}{3} \end{cases}$$

那么 100 个囚徒 100 个盒子呢？

对于 $x\in[51,100]$ 来说，出现一个循环后，剩余循环组合里必然不会出现其它 $len>50$ 的循环了，所以这些循环是互斥的。所以，我们先从100个选 $100-x$ 个进行排列组合，然后剩余的 x 个组成循环。得到计算公式$P(x)=\binom{100}{x}\cdot(100-x)!\cdot\frac{x!}{x\cdot 100!}=\frac{1}{x}$。

以上结论经过推广，到 n 个盒子就有 $P(x)=\binom{n}{x}\cdot(n-x)!\cdot\frac{x!}{x\cdot n!}=\frac{1}{x}$。

以下是本题的标准程序：

```
#include<bits/stdc++.h>
using namespace std;
```

```
signed main()
{
  int n;
  cin>>n;
  double sum=0.0;
  for(int i=n;i>=n/2+1;i--)
  {
    double x=1.0/i;
    sum+=x;
  }
  sum=1.0-sum;
  printf("%.20lf",sum);
  return 0;
}
```

Problem F. 讨厌的 English 小作文

字符串长度并不长，唯一的坑点是因为存在空格，所以需要使用 getline 读入。查询的方式就比较多样，不再赘述。

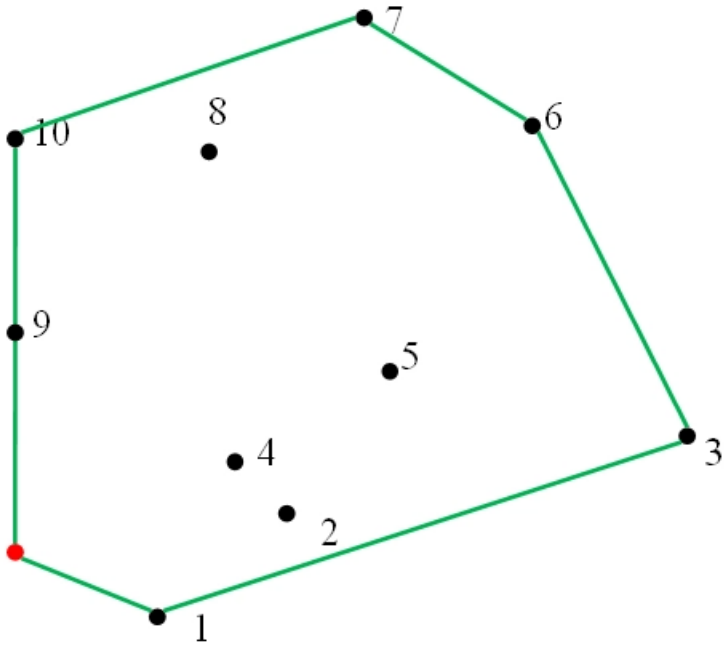
以下是本题的标准程序：

```
#include<bits/stdc++.h>
using namespace std;
signed main()
{
  string s;
  getline(cin,s);
  int sum=0;
  for(int i=0;i<s.length();i++)
  {
    if(s[i]=='E'&&s[i+1]=='n'&&s[i+2]=='g'&&s[i+3]=='l'&&s[i+4]=='i'&&s[i+5]=='s'&&s
    [i+6]=='h')
    sum++;
  }
  cout<<sum<<endl;
  return 0;
}
```

Problem G. 令人头疼的物理课

本题考察计算几何，具体解法为求凸包与求多边形面积。

用通俗（但不严谨）的话来定义凸包如下：给定二维平面上的点集，凸包就是将最外层的点连接起来构成的凸多边形，它能包含点集中所有的点。



首先对原点集求凸包，将该凸包顶点与其边上的点从原点集中剔除，再进行一次求凸包。将两次求得凸包的面积相减即可得到魔戒的面积。

因为数据保证所有点不会均在同一条直线上，第一次切割必然能够得到一个合法的多边形，但第二次切割时就可能出现没有点或点均在一条直线上的情况，注意对于这些情况的判断。

依据Pick定理，我们可以得知最终求得面积的二倍一定是一个正整数，加之本题的数据范围较大，运算应在整数域中进行，以保证不会出现精度问题。

以下是本题的标准程序：

```
#include<iostream>
#include<algorithm>
#include<vector>

using namespace std;

using ll=long long;

const int N = 1e5+9;

struct point{
  ll x,y;
  point(ll x=0,ll y=0):x(x),y(y){}
};
vector<point> ori,in,out;
point operator - (point a,point b){return point(a.x-b.x,a.y-b.y);}
bool cmp(point a,point b){
  if(a.x==b.x)return a.y<b.y;
  return a.x<b.x;
}

ll cross(point a,point b){
  return a.x*b.y-b.x*a.y;
}

vector<point> andrew(vector<point> p){
  if(p.size()<=2)return p;
  vector<point> v;
```

```
  v.push_back(p[0]);
  for(int i=1;i<p.size();i++){
    while(v.size()>=2 && cross(v[v.size()-1] - v[v.size()-2] ,p[i] - v[v.size()-2])
    <0 ){
      v.pop_back();
    }
    v.push_back(p[i]);
  }
  int half=v.size();
  for(int i=p.size()-2;i>=0;i--){
    while(v.size()>half && cross(v[v.size()-1] - v[v.size()-2] ,p[i] - v[v.size()
    -2]) <0 ){
      v.pop_back();
    }
    v.push_back(p[i]);
  }
  v.pop_back();
  return v;
}

ll volume(vector<point> p){
  if(p.size()<=2)return 0;
  ll ans=0;
  for(int i=0;i<p.size()-1;i++){
    ans+=cross(p[i]-p[0],p[i+1]-p[0]);
  }
  return ans;
}

int main(){
  int n;
  ll ans=0;
  cin>>n;
  for(int i=0;i<n;i++){
    ll x,y;
    cin>>x>>y;
    ori.push_back(point(x,y));
  }
  sort(ori.begin(),ori.end(),cmp);
  out=andrew(ori);
  ans+=volume(out);
  sort(out.begin(),out.end(),cmp);
  for(int i=0,j=0;i<ori.size();i++){
    while(ori[i].x>out[j].x || (ori[i].x==out[j].x&&ori[i].y>out[j].y)){
      j++;
    }
    if(ori[i].x!=out[j].x||ori[i].y!=out[j].y){
      in.push_back(ori[i]);
    }
  }
  in=andrew(in);
  ll res=volume(in);
  if(!res){
    cout<<"-1";
  }
  else{
    ans-=res;
    cout<<ans;
  }
  return 0;
}
```

Problem H. 竟然是高中生物学竞赛

本题数据范围较小，做法较多。其中一种做法是枚举所有选取节点的方案，判断是否联通和连接未选取的节点，记录花费最小的方案。判断联通可以用并查集，判断连接节点可以用 BFS 。

以下是本题的标准程序：

```
#include<bits/stdc++.h>
using namespace std;
int to[205],ne[205],he[205],tot;
void add(int x,int y){
  tot++;to[tot]=y;
  ne[tot]=he[x];he[x]=tot;
}
const int M=1e9;
int n,m,v[25],ans;
bool u[25],f[25];
int dad[25];
int find(int x){
  if(dad[x]==x)return x;
  return dad[x]=find(dad[x]);
}
bool check(){
  int tar=n;
  for(int i=1;i<=n;i++){
    f[i]=0;dad[i]=i;
  }
  for(int i=1;i<=n;i++){
    if(!u[i])continue;
    if(!f[i]){
      f[i]=1;tar--;
    }
    for(int j=he[i];j;j=ne[j]){
      int y=to[j];
      if(!f[y]){
        f[y]=1;tar--;
      }
      if(u[y]){
        int xx=find(i),yy=find(y);
        if(xx!=yy)dad[xx]=yy;
      }
    }
  }
  if(tar>0)return 0;
  for(int i=1;i<=n;i++){
    if(u[i]&&find(i)!=find(1))return 0;
  }
  return 1;
}
void dfs(int x,int y){
  if(y>=ans)return;
  if(x==n+1){
    if(check()){
      if(y<ans)ans=y;
    }
  }
  else{
    dfs(x+1,y);u[x]=1;
    dfs(x+1,y+v[x]);u[x]=0;
  }
}
void solve(){
  scanf("%d%d",&n,&m);
  tot=0;ans=M;
```

```
  memset(he,0,sizeof(he));
  memset(u,0,sizeof(u));
  for(int i=1;i<=n;i++)scanf("%d",&v[i]);
  for(int i=0;i<m;i++){
    int x,y;scanf("%d%d",&x,&y);
    add(x,y);add(y,x);
  }
  u[1]=1;dfs(2,v[1]);
  if(ans==M)printf("-1\n");
  else printf("%d\n",ans);
  return;
}
int main(){
  int tes=1;
  scanf("%d",&tes);
  while(tes--)solve();
  return 0;
}
```

Problem I. 技术老师与 CCPC 现象

根据题意可知，在任意两位大学生 A B 之间，其中位置相对靠后的大学生必然因靠前的学生产生 $h_{self} \cdot t_{other}$ 的不满度，而靠前的大学生不会对靠后的学生产生不满度。也就是说，任意两位大学生之间必然有一次不满度产出，且不满度产出只和二人之间的顺序有关。因此，我们只要贪心的保证每两个人之间的不满度产出均最小，即可保证总不满度最小。

同时，可证明上述排序一定能够实现，流程如下：

我们假设使得相互之间不满度均最小的排列方式不存在，那么必然存在 a, b, c 使得以下不等式同时成立：

$$\begin{cases} t_a \cdot h_b < t_b \cdot h_a \\ t_c \cdot h_a < t_a \cdot h_c \\ t_b \cdot h_c < t_c \cdot h_b \end{cases}$$

计算可得，以上不等式无法同时成立，即证明一定存在一种排列方式使得任意两人之间的不满度均最小。

所以，我们只需要按照 $h_i \cdot t_j$ 的规则进行排序即可保证不满度最小，同时注意最终要求 **同不满度方案按照编号最小优先输出** ，编写排序规则时应包含这一点。

以下是本题的标准程序：

```
#include<iostream>
#include<algorithm>

using namespace std;

const int N=2e5+9;

struct peo{
    int num;
    long long time,hate;
}p[N];

bool cmp(peo a,peo b){
    if(a.hate*b.time==b.hate*a.time)
```

```
    return a.num<b.num;
    return a.hate*b.time>b.hate*a.time;
}

int main(){
    int n;
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>p[i].time>>p[i].hate;
        p[i].num=i;
    }
    sort(p+1,p+1+n,cmp);
    for(int i=1;i<=n;i++){
        cout<<p[i].num<<' ';
    }
}
```

Problem J. 逃离历史车轮的碾压

本题有BFS与最短路两种解法。

因为本题并没有特别大的数据范围，所以常见单源最短路算法均能正常通过，只要注意添加对于已坍塌的点的判断就可以当成板子题看待，该做法不做展开细讲。

对于BFS做法，关键在于如何处理零号路径，我们考虑在原BFS队列的基础上额外建立一个队列。我们在BFS的过程中，将经过一号路径搜索得到的点打入原队里，零号路径则打入额外队列。当额外队列不为空时，优先从额外队列取出点进行BFS直至额外队列为空，再处理原队列。注意不要忘记对点是否崩塌进行判断。

以下是本题的标准程序:

```
#include<iostream>
#include<algorithm>
#include<queue>

using namespace std;

const int N=1e5+9;
int n,m;
int t[N],st[N],ans=114514;
queue<pair<int,int>> q,inm;

vector< pair<int,int> > rd[N];

void bfs(pair<int,int> p){
    if(ans!=114514)return;
    if(p.first==n)ans=p.second;
    st[p.first]=1;
    for(auto x:rd[p.first]){
        if(!st[x.first]){
            if(x.second&&t[x.first]>p.second+1){
                q.push({x.first,p.second+1});
            }
            else if(!x.second&&t[x.first]>p.second){
                inm.push({x.first,p.second});
            }
        }
    }
}
```

```
int main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++)cin>>t[i];
    for(int i=1;i<=m;i++){
        int u,v,x;
        cin>>u>>v>>x;
        rd[u].push_back({v,x});
        rd[v].push_back({u,x});
    }
    st[1]=1;
    q.push({1,0});
    while(!q.empty()||!inm.empty()){
        while(!inm.empty()){
            pair<int,int> now=inm.front();
            inm.pop();
            bfs(now);
        }
        if(!q.empty()){
            pair<int,int> now=q.front();
            q.pop();
            bfs(now);
        }
    }
    if(ans!=114514){
        cout<<"YES\n"<<ans;
    }
    else{
        cout<<"NO";
    }
}
```

Problem K. 水晶书架上的各种地理书

首先考虑一种简单情况：从左到右只增不减最少要用多少个木板?

我们遍历每一个元素 a_i，若 $a_{i-1} > a_i$，说明出现了递减区间，必须要对 a_i 进行增高处理。

但是如果只增加 a_i 的值，就可能导致原本不用增高的 a_{i+1} 也要使用木板增高。

因为最终结果与木板的长度无关，所以为了不影响后面的元素，我们可以直接用长度为无限长的木板把从第 i 个位置到最后一个位置的元素垫起来。

再考虑题目的这种情况，可以用类似方法求使右区间递减的最小代价。

枚举左右区间的分界点 i 和 $i+1$，将左区间所有用到的木板从延伸到无限远改为延伸到位置 i，同理右区间的木板向左延伸到位置 $i+1$。

观察到左右两边的部分木板会挨在一起，那么就可以将两个木板拼为一个，在此分界点的最小代价就是左右区间代价的最大值。

代价最小的分界点就是题目的答案。

以下是本题的标准程序：

```
#include <iostream>
#include <cstring>
#include <algorithm>

const int N = 1e5 + 10;
```

```
int n, a[N];
long long left[N], right[N];

int main() {
    int T;
    std::cin >> T;
    while (T --) {
        memset(left, 0, sizeof(long long) * (n + 1));
        memset(right, 0, sizeof(long long) * (n + 1));

        std::cin >> n;
        for (int i = 1; i <= n; i ++) std::cin >> a[i];

        if (n == 1) {
            std::cout << 0 << std::endl;
            continue;
        }

        for (int i = 2; i <= n; i ++) left[i] = left[i - 1] + std::max(0, a[i - 1] -
    a[i]);
        for (int i = n - 1; i >= 1; i --) right[i] = right[i + 1] + std::max(0, a[i
   + 1] - a[i]);

        long long result = 1e18;
        for (int i = 1; i < n; i ++)
            result = std::min(result, std::max(left[i], right[i + 1]));
        std::cout << result << std::endl;
    }

    return 0;
}
```

Problem L. 心理课喝桶装水

3 个人 3 天 3 桶水，根据已知条件可以判断出来， 1 个人一天喝 $\frac{1}{3}$ 桶水。

至于判断小数，我们可以将人数与天数相乘，再对 3 取模，这样就能判断是否是小数了。如果取模结果为 0 ，则证明可以购买整数桶水正好喝完。

以下是本题的标准程序:

```
#include<bits/stdc++.h>
using namespace std;
signed main()
{
  int x,y;
  cin>>x>>y;
  if(x==0||y==0)
  cout<<0;
  else
  {
    int sum=x*y;
    if(sum%3!=0)cout<<"xiaoshu";
    else
    cout<<sum/3;
  }
  return 0;
}
```

Problem M. 又找不到体育老师了

能够注意到 n 的范围非常小，所以可以考虑暴力，但是思考过程还是稍微有点复杂的。

首先要注意到，如果图上无障碍物，那么只需要再放置两个障碍物在 $(2,1)$ $(1,2)$ 的位置就肯定可以阻止，所以最小值肯定是小于等于 2 。

方法一：暴力方法，多重循环判断，把所有可能出现的情况都考虑进去。这种做法会导致代码很长，最后可以实现但是会非常麻烦。

方法二：比 2 小的数就是 1 和 0 ，判断是否为 0 的方法就是直接查询能否从起点走到终点，判断 1 的方法就是遍历所有点（除了起点和终点），在该点上放置障碍物，然后暴力查询能否从起点走到终点。输出其中最小值即可。

以下是本题的标准程序：

```
#include<bits/stdc++.h>
using namespace std;
int n;
bool f[1005][1005],c[1005][1005];
int xi[4]={-1,0,1,0},yi[4]={0,-1,0,1};
bool check(){
  queue<int>q;q.push(0);
  memset(c,0,sizeof(c));
  c[0][0]=1;
  while(!q.empty()){
    int p=q.front(),x=p/n,y=p%n;q.pop();
    if(x==n-1&&y==n-1)return 1;
    for(int i=0;i<4;i++){
      int xx=x+xi[i],yy=y+yi[i];
      if(xx>=0&&xx<n&&yy>=0&&yy<n&&!f[xx][yy]&&!c[xx][yy]){
        c[xx][yy]=1;
        q.push(xx*n+yy);
      }
    }
  }
  return 0;
}
signed main(){
  cin>>n;
  for(int i=0;i<n;i++){
    string s;cin>>s;
    for(int j=0;j<n;j++){
      if(s[j]=='x')f[i][j]=1;
    }
  }
  if(!check()){
    cout<<"0\n";
    return 0;
  }
  for(int i=0;i<n;i++){
    for(int j=0;j<n;j++){
      if(!(i==0&&j==0)&&!(i==n-1&&j==n-1)&&!f[i][j]){
        f[i][j]=1;
        if(!check()){
          cout<<"1\n";
          return 0;
        }
        f[i][j]=0;
      }
    }
  }
  cout<<"2\n";
  return 0;
```

```
}
```