# 洪水

根据时间进行模拟过程，可以同时模拟更新出洪水当前淹没的状态和果老师当前的移动情况。

用两个$BFS$,分别维护这两个部分即可。

# 别墅晚餐

有个比较容易想到的办法，预处理出二维前缀和（表示一个矩形的`x`的个数），再$n^4$枚举矩形的两个顶点，然后判断矩形内的`x`个数是否为0，更新最优解即可。

但是$O(n^4)$通过不了。

上述思路可以优化，我们预处理每行的前缀和$sum[i][j]$表示$i$行到了$j$列的`x`的个数，更改枚举方法，考虑枚举$l,r$表示矩形长的左右边界，要使周长最大，就得在此基础上最大化宽度，贪心的从第1行开始判断每行的`x`的个数，记录能连续的最大合法长度就是当前的宽，然后更新周长就好了。

时间复杂度：$O(n^3)$

# 写BUG

本题就是要找满足最大中心对称的正方形，为了求边长的最大值，我们二分边长的一半（注意奇偶性讨论），在`check`函数中，如果正反hash相等就返回1。

# 直角三角形

如果我们设符合要求的三角形直角顶点坐标为$A(x_1,y_1)$，那么剩下两点坐标一定为 $B(x_1,yi)$、$C(x_j,y_1)$。所以，如果我们确定一个点为直角顶点，那么可以与之配合成为目标三角形的只有是横、纵坐标都与之相同的点。

由此，我们可以考虑用两个数组 $sumx$、 $sumy$来记录一个横坐标或纵坐标上的点的个数，明显的，当直角顶点为 $A(x_1,y_1)$时，可以组成的目标三角形个数为$(sumx-1)*(sumy-1)$

由此，我们可以枚举每个点，取出它所在的横、纵坐标，用上面的公式计算出在这个点可以组成的目标三角形个数，累加起来，就是答案。

# 取数游戏

考虑破环成链，$dp[i][j]$ 表示 $i-j$这条链可以从两边开始选，先手能比后手多拿到最多的奇数个数，显然$dp[i][j]=max(dp[i][i]-dp[i+1][j],dp[j][j]-dp[i][j-1])$，即枚举取左边还是右边。

然后当 $dp[i][i]-dp[i+1][i+n-1]>1$时这个点可行，直接判断即可。

```
#include <bits/stdc++.h>
using namespace std;
int n, a[110], dp[210][210];
int main() {
    cin >> n;
```

```
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
        a[i] %= 2;
        dp[i][i] = dp[i + n][i + n] = a[i];
    }
    for (int l = 2; l <= n; l++) {
        for (int i = 1; i <= (n << 1); i++) {
            int j = i + l - 1;
            if (j > (n << 1)) {
                break;
            }
            dp[i][j] = max(dp[i][i] - dp[i + 1][j], dp[j][j] - dp[i][j - 1]);
        }
    }
    int ans = 0;
    for (int i = 1; i <= n; i++) {
        if (dp[i][i] - dp[i + 1][i + n - 1] > 0) {
            ans++;
        }
    }
    cout << ans;
}
```

# 数据还原

输入数据，可以使用 $getline$ 解决，一次读入一行，在字符串中处理数据。

然后暴力循环，不要修改原来没有被污损的数据。

我们就可以判断（当然可以用表达式），如果不是 ?，就按照他的，是的话, 0 到 100 循环。

# 传奇厨师

## 二分答案

对于每个需要的食材，我们先二分能做多少菜，

然后可以使用**贪心**算出它去除已有部分还需用钱买的量

只要枚举大包食材的袋数，从而计算出小包用量，选取最小值

# 黑白棋子

首先先按题意模拟 $K$ 次变为最终的状态。

再由最终状态往回推，但是每一次变换不唯一，有两种情况，即一旦确定一个位置是 $B$ 还是 $W$，那么整个环就确定了。

于是总共有 $2^K$ 种情况，最后去除一下重复即可。

# 祖玛游戏

用$dp[i][j][sum]$表示消除 $i-j$ 区间在左边添加了 $sum$ 个珠子的总添加珠子数

答案：$dp[1][n][0]$

转移方程如下：

1.当 $sum < k-1$(不能消)时再加一个，即$dp[i][j][sum] = min(f[i][j][sum], f[i][j][sum+1]+1)$

2.当 $sum = k-1$ 时直接消掉，即$dp[i][j][sum] = dp[i+1][j][0]$

3.当 $i$ 和 $i+1$ 的颜色相同时，可以把 $i$ 加到 $i+1$ 的整体中，即 $dp[i][j][sum] = dp[i+1][j][sum+1]$ 其实满足消的条件后也可不消，但这已经包含在第三种里了。

实现的时候用记忆化搜索即可。

# 出租

$f[i][j]$表示第一套房子出租前 $i$ 个月，第二套房子出租前 $j$ 个月的最大收益

后面转移略。

# 最短路上的边

在dijkstra算法基础上做点小修改就可以对一个点找到属于最短路径的那部分。

对于一个点$C$,边组成了一个$DAG$，可以用动态规划求解:

$to(v)$：表示点$C$通往点$v$的路径数量

$from(v)$：表示从点$v$到其他任意点的数量

计算$to(v)$：

$to(C) = 1$

$to(v) = \sum to(u)$，$(u, v)$是$DAG$上的一条边。

计算$from(v)$:

$from(v) = 1 + \sum from(w)$，$(v, w)$是$DAG$上的一条边。

$to(v)$的值是按dijkstra算法处理顶点的顺序计算，而$from(v)$则按相反顺序计算。

知道了这些值，我们就可以计算出从点$C$到其他点有多少条路径包含一条路径$R = (A, B)$

如果$R$不是$DAG$上的边，则数量为$0$，否则数量则为$to(A) \times from(B)$。包含$R$的最短路径总数为每个点$C$的这些成绩之和。

# 构环

我需要发现一个环$A$能生成环$B$，并且所有数都是正整数。

首先，注意到环$A$的总和一定为环$B$总和的三分之一。

然后对于输入的环，我们可以确定对于每个位置$k$满足，$A_{k+3} - A_k = B_{k+2} - B_{k+1}$

这个性质加上环$B$的和就足够解决问题。

当环的长度$N$不能被3整除，解是唯一的。

假设第一个元素$(A_1)$等于1。从差值我们可以确定$A_4, A_7 \ldots A_{N-2}$。因为$N$和3互质，我们可以确定所有数字$A_2$到$A_N$。然后最后我们使得整个的和值也满足要求，即在每个元素上加一个数即可。

如果$N$能被3整除，答案是不唯一的，现在就能得到三个链组成整个环：

- $A_1, A_4 \ldots, A_{N-2}$
- $A_2, A_5, \ldots, A_{N-1}$
- $A_3, A_6, \ldots, A_N$.

我们需要确定$A_1, A_2, A_3$的值(并使得他们在各自链上的差值满足要求)，保证所有数字都是正的，并且和值是满足要求的。

对于每一个链，都可以确定第一个数字的可能最小值，这样链中的所有数字都是正的。例如，如果差值产生链1、$-4$、5，那么我们需要对所有数至少加5，使所有数都为正数。

这样我们可以设置$A_1$为最小值并且保证第一个链都是正的。同理$A_2$也是如此，然后$A_3$由$B_2 - A_1 - A_2$计算得到。

# 不是子序列

定义$f(s)$为最短长度不是$s$的子序列的字符串。假设字符串$t$不是$s$的子序列，假设$t$的首字母为$c$：

- 如果$s$不包含字符$c$，字符串$t$的长度为1，一个单字符$c$
- 否则假设$i$是$c$在$s$中最左边出现的位置$(s[i] = c)$. $t$如果不是$s$的子序列当前仅当$t.substr(1)$不是$s.substr(i+1)$的子序列。

这里的$s.substr(i)$表示字符串$s$第$i$个字符的后缀字符串。

因此我们可以通过$dp$计算结果：

定义$dp[i]$为$f(s.substr(i))$的长度。并且预处理$next(i, c)$：最小的下标$j$满足$i \leq j$并且$s[j] = c$,于是：

$$dp[i] = min_c dp[next(i, c) + 1] + 1$$

然后怎么计算字典序最小的答案呢：

答案的第一个字符应该是满足以下条件最小的$c$：

$$dp[i] = dp[next(i, c) + 1] + 1$$

所以这个过程就是求解字典序最小最短的不是字符串$s.substr(next(0, c))$,所以就是跟上面相同的处理过程。

时间复杂度为$O(|A|m)$,其中|A|为字符集大小。

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

typedef long long ll;

char s[200005];
int dp[200005], nt[200005][26];
int main(){
```

```
    scanf("%s", s+1);
    int n = strlen(s+1);
    dp[n+1] = 1;
    for(int i=n;i>=1;i--){
        for(int j=0;j<26;j++) nt[i][j] = nt[i+1][j];
        nt[i][s[i]-'a'] = i;
    }
    for(int i=n;i>=1;i--){
        dp[i] = 1e9;
        for(int j=0;j<26;j++){
            if(nt[i][j] == 0) dp[i] = 1;
            else dp[i] = min(dp[i], dp[nt[i][j]+1]+1);
        }
    }
    int cur = 0;
    for(int i=0;i<dp[1];i++){
        for(int j=0;j<26;j++){
            if(i == dp[1]-1){
                if(nt[cur+1][j] == 0){
                    printf("%c", j+'a');
                    break;
                }
            }
            else{
                if(dp[nt[cur+1][j]+1]+1 == dp[1]-i){
                    printf("%c", j+'a');
                    cur = nt[cur+1][j];
                    break;
                }
            }
        }
    }
}
```

# 打字员

定义$f[i][j]$为按了$i$下键盘之后产生了长度为$j$的串的方案数，初始$f[0][0]$

对于某个$f[i][j]$，要么按一次数字键（0或者1）变成$f[i+1][j+1]$，要么按一次退格键变成$f[i+1][j-1]$，也就是说，每一个$f[i][j]$只会对两个状态产生影响，但是一个$f[i][j]$会被很多状态影响，显然从$i$推$i+1$会简单很多，则有：

- $f[i+1][j+1]+=2*f[i][j]$
- $f[i+1][max(j-1,0)]+=f[i][j]$

注意$j=0$的时候相当于屏幕上此时没有字符，此时按退格键虽然不会改变字符，但仍然是一个有效状态，所以取$max(j-1,0)$。这样仅仅是得到了按$i$下键盘时得到了长度为$j$的串的方案数，但是题目里所给的串是一个特定的串，是所有长度为$j$的串的其中之一，又因为长度为$j$的串中每个字符要么是0要么是1，总共有 $2^j$ 种串，所以按$n$下得到某个长度为$len$的串的方案数应为$f[n][len]/2^j$。

# 队列与纸牌

显然我们发现这个生成的序列一定是一个 $V$ 字形，$1$ 是最底端。

然后这个删除序列一定有这样一个性质：构造删除序列的时候，新加入的数字要么是未出现过的数字的最大值，要么严格小于出现过的数字的最小值。

证明：我们假设当前最小值是从左边取出来的，那么从左边取一定是严格小于它的，从右边取的话，你不可能取出小于未出现过的数字的最大值的数，因为这个未出现的最大值还没有被取出来，比他小的自然也取不出来。而如果从最小值到最大值都取了的话，你下一步取出的一定是新的最小值。

所以我们设$(f[i][j])$表示删除序列大小为 $i$，当前最小值为 $j$ 的方案数，那么$(f[i][j]=\sum_{k=j}^{n} f[i-1][k])$，也就是要么原来的最小值就是 $j$ ，这次取出来了一个未出现的最大值，要么这次取的数是 $j$。

用前缀和优化可以做到$(O(n^2))$。

但是有个问题，$1$ 可能在 $k$ 位之前就出现了，所以$(f[k][1])$不是答案，答案是$(f[k][1]-f[k-1][1])$，这样就减去了 $1$ 提前出现的情况，之后后面的序列方案数就是$(2^{n-k-1})$，即：枚举它是从左边还是右边出来的。