C4-Solution

A-Valhalla做防疫体温检测

题目分析

本题为签到题，主要考察浮点数读入、比较及if-else语句的使用，根据题意读入、比较并按要求输出即可

示例代码

```
#include <stdio.h>
#include <math.h>
int main()
{
    double t;
    scanf("%lf", &t);//逐项判断，同时注意浮点数精度问题
    if (fabs(t - 37.5) < 1e-6 || t > 37.5)
        printf("Fever");
    else if (fabs(t - 37) < 1e-6|| t > 37)
        printf("Warning");
    else if (t < 37)
        printf("OK");
    return 0;
}
```

B-假日数字游戏

题目分析

本题考察利用 `for` 循环实现的简单递推关系与简单判断。

根据数列各项的数据大小与题目提示可知，在递推计算的过程中，需要用 `long long` 类型的变量来存储计算出的数列每项的值。读入项数 a 后，根据递推式计算对应项数的值，通过模10运算取得个位数输出即可。

示例代码

```c
#include <stdio.h>

int main() {
    int a;
    scanf("%d", &a);
    if (a == 1||a == 2) {
        printf("1");
    } else {
        long long fib1 = 1, fib2 = 1;
        for (int i = 3; i <= a; ++i) {
            long long tmp = fib1 + fib2;
                fib1 = fib2;
                fib2 = tmp;
            /* for循环中也可以替换为如下代码，效果相同
            fib2 = fib1 + fib2;
            fib1 = fib2 - fib1;
            */
        }
        printf("%lld", fib2 % 10); //虽然我们已经知道结果肯定为个位数，不会超出int范围，但fib2是long
    }

    return 0;
}
```

C-灵梦的锐评异变

题目分析

参考例题4-9

依照题目要求，对难度 n 进行分类讨论和统计

注意到分类标准的特殊性，可以用 `switch-case` 函数对 $\left\lfloor \frac{n}{10} \right\rfloor$ 讨论即可

示例代码

```c
#include <stdio.h>
int main(void) {
    int L = 0, Ex = 0, H = 0, N = 0, E = 0;//分别记录五个等级的数量，记得初始化为0
    int score;//记录每一次读取的难度
    while (~scanf("%d", &score)) {
    //不定组数据输入，也可以使用scanf("%d", &score)!=EOF作为循环条件
        switch (score / 10) {//对score/10的值进行讨论
        case 10:
        case 9://score==100的情况和90<=score&&score<100的情况相同
            printf("Lunatic\n");
            L++;
            break;
        case 8://80<=score&&score<90的情况
            printf("Extra\n");
            Ex++;
            break;
        case 7://70<=score&&score<80的情况
            printf("Hard\n");
            H++;
            break;
        case 6://60<=score&&score<70的情况
            printf("Normal\n");
            N++;
            break;
        default://其余情况，也就是score<60的情况
            printf("Easy\n");
            E++;
            break;
        }
    }
    printf("%d %d %d %d %d", L, Ex, H, N, E);
    return 0;
}
```

D-判断三角形！

问题分析

首先筛选出最大的边，将其换到 a 边。然后利用三角形 b^2+c^2 与 a^2 的关系，判断三角形的最大角的形状，接着再用三条边的长度判断等腰性。

参考代码

```c
#include <stdio.h>
#include <math.h>

int main()
{
    int a = 0, b = 0, c = 0, tmp = 0;
    scanf("%d %d %d", &a, &b, &c);
    //首先找出最大边
    if (a < b)
    {
        tmp = a;
        a = b;
        b = tmp;
    }
    if (a < c)
    {
        tmp = a;
        a = c;
        c = tmp;
    }

    if (a >= b + c) //判断能否构成三角形
    {
        printf("no triangle\n");
    }
    else
    {
        if (a * a > b * b + c * c) //判断钝角三角形
        {
            printf("obtuse triangle\n");
        }
        else if (a * a == b * b + c * c) //直角
        {
            printf("right triangle\n");
        }
        else //其余情况为锐角
        {
            printf("acute triangle\n");
        }

        if (a == b && b == c) //判断等边三角形
        {
            printf("equilateral triangle");
        }
        else if (a == b || b == c || a == c) //判断等腰，注意要用else if
        {
            printf("isosceles triangle");
        }
    }
```

```
    return 0;
}
```

E-航小萱挑战哥德巴赫猜想

题目分析

这道题简单考察了一下大家对于素数的理解，如果你对于素数的理解是“长得看起来像素数的东西”那就悲剧了。

素数的定义是，一个大于1的自然数，除了1和它本身外，不能被其他自然数整除。

所以这就可以导出这个题的思路：从2开始到 $\frac{n}{2}$ ，枚举每一个数 a ，对于 a 做如下判断：2 到 $sqrt(a)$ 之间是否有a的因子，如果没有，那么说明 a 是质数，然后就可以判断 $n-a$ 是否为质数 。

我写的标程要更复杂一些（因为一开始出题的时候 n 的规模达到了 10^6 ），里面有素数筛以及二分分，大家简单看看就好。

参考代码

```
#include<stdio.h>
#include<time.h>
int Not_Prime[10020],Prime[10020],cnt;//Not_Prime表示判定一个数是否为质数，1则不是，0则是；Prime为质
int n[1200];
int main(){
        for(int i=2; i<=10000; i++){
                if(Not_Prime[i]) continue;//如果不是质数，则跳过
                cnt++;//筛选到一个新质数
                Prime[cnt] = i;
                for(int j=2; i*j<=10000; j++) Not_Prime[i*j] = 1;//给前面的合数标记为非质数
        }
        //这个筛选质数的算法不是线性的，但效率也比较高
        int t;scanf("%d",&t);
        for(int i=1; i<=t; i++) scanf("%d",&n[i]);
        for(int i=1; i<=t; i++){
                int l,r,mid,j;
                for(j=1; j<=cnt&&Prime[j]<=n[i]/2; j++){
                        int k = n[i] - Prime[j];
                        if(k==Prime[j]){
                                printf("%d %d\n",Prime[j],Prime[j]);
                                break;
                        }
                        l=j,r=cnt,mid;//二分答案，从第j位到质数表末尾找到自己想要的答案
                        while(l<r-1){
                                mid = (l+r)>>1;
                                if(Prime[mid]<k) l=mid;//如果区间的中心比想要的值小，则舍弃左半边选
                                else r=mid;//反正选择左半边
                        }
                        if(Prime[r]==k){//如果质数表里有我们想要的答案，那么只能在Prime[r]处取得
                                printf("%d %d\n",Prime[j],k);
                                break;
                        }
                }
        }
        return 0;
}
```

F-万能的二分法

难度	考点
2	二分

问题分析

本题考查用二分法对单调函数的方程求解，具体操作方式已在 HINT 中给出，此处不再赘述。

注意到方程左边函数是单调递减的，因此函数值大于 0 时应当用中点替换区间左端点，小于 0 时用中点替换区间右端点。

结束条件设置比题目要求的精度稍高即可。

参考代码

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

int main()
{
        double y, l = 10, r = 30, m;
        scanf("%lf", &y);
        while (r - l > 1e-6)
        {
                m = l + (r - l) / 2;//取区间中点
                if (exp(-sqrt(m / 10)) / (log(m / 10)) - y < 0)//函数值小于0
                        r = m;//中点替换区间右端点
                else
                        l = m;
        }
        printf("%.3lf", m);
        return 0;
}
```

G-哪吒数

题目分析

此题旨在让大家学会对一个数分解质因数的较为快速的做法。

对于任意一个正整数 n，有唯一质因数分解方式 $n=p_1^{a_1}p_2^{a_2}\cdots p_k^{a_k}$，其中 $p_1<p_2<\cdots<p_k$ 为 n 的 k 个不同质因数，其诸指数 a_i 均为正整数。

对于 n 的任意一个哪吒数 d，若 $p_i|d$，则 $p_i^{a_i}|d$（否则 d 与 $\frac{n}{d}$ 有共同因子 p_i，不互质），即 d 只可能为 $p_1^{a_1},p_2^{a_2},\ldots,p_k^{a_k}$ 这 k 个数中部分的乘积。举例而言，对于数 $n=360=2^3\times3^2\times5$，360 的哪吒数 d 只可能为 $2^3,3^2,5$ 三个数中部分的乘积，即 d 可能的取值为 $1,5,8,9,40,45,72,360$.

对于 n 的哪吒数 d， n 的第 i 个质因子 p_i ($i=1,2,\ldots,k$)， $p_i|d$ 要么成立要么不成立，因此有 2^k 个哪吒数，所有哪吒数之和应为 $(1+p_1^{a_1})(1+p_2^{a_2})\cdots(1+p_k^{a_k})$，通过这个算式即可计算答案。上述 $n=360$ 的例子中，答案应为 $(1+2^3)(1+3^2)(1+5)=540$.

下面着重介绍一个叫较快的分解质因数的方法：

从第一个质数 $i=2$ 开始枚举，如果 $i|n$，说明 i 为 n 的质因数 p_j，不断进行 $n=n/i$ 操作，直到 i 不再是 n 的因数为止，记录除以 i 的次数即为 p_j 的指数 a_j，然后 $i=i+1$ 进行下一次循环。每一次循环代码如下：

```
//数组p[k]为n的第k个质数，a[k]为n的质因数分解中p[k]的指数
if(n % i == 0) {
    p[k] = i;
    a[k] = 0;
        while(n % i == 0) {
                a[k]++;
                n /= i;
        }
    k++;
}
```

由于每一次遍历到 n 的一个质因数 p_j，都把 n 的分解中所有的 p_j 除掉了，此时 $n=p_{j+1}^{a_{j+1}}p_{j+2}^{a_{j+2}}\cdots p_k^{a_k}$，因此保证了 i 下一个遍历到的 n 的因数为 p_{j+1}。每分解出 n 的一个质因数，之后的操作都是对更新后的 n 进行质因数分解。

遍历条件设为 $i*i<=n$，超过范围后结束循环。在遍历到 p_k 之前 n 将会变为 $p_k^{a_k}$，如果 $a_k\geq 2$，$p_k*p_k<=p_k^{a_k}$，则一定会遍历到 p_k，且此次循环后 n 会变为 1；如果 $a_k=1$，则在遍历到 p_k 之前，n 将会变为 p_k，$p_k*p_k>p_k$，可知不会遍历到 p_k，因此循环结束后 $n=p_k$。综上，可以在结束循环后进行判断，若此时 n 为 1，说明 n 已经被完全分解，否则说明此时的 n 是最初 n 的最后一个质因数 p_k，且 $a_k=1$。

综合起来，分解质因数的代码如下：

```
//数组p记录n的质因数，数组a记录n的质因数分解中的指数，k为不同质因数个数
k = 0;
for(i = 2; i * i <= n; i++) {
    if(n % i == 0) {
            p[k] = i;
        a[k] = 0;
                while(n % i == 0) {
                        a[k]++;
                        n /= i;
                }
        k++;
        }
}
if(n != 1) {
    p[k] = n;
    a[k] = 1;
    k++;
}
//此时p[0],p[1],...,p[k-1]即为最初n的k个质数，a[i]为对应的指数。
```

最后计算 $(1+p_1^{a_1})(1+p_2^{a_2})\cdots(1+p_k^{a_k})$ 得到答案。

本题的实际做法中，不需要用数组存储 p_i 和 a_i ，可以在遍历的时候用临时变量 t 记录 $p_i^{a_i}$ ，然后将 $t+1$ 乘到结果上即可。

AC示例代码

```
#include<stdio.h>

int main() {
        long long n, i, s=1, t;
        scanf("%lld", &n);
        for(i = 2; i * i <= n; ++i) {
                if(n % i != 0) continue; //i不是n的因数，跳过本次循环
                t = 1; //用t临时记录p的a次幂
                while(n % i == 0) {
                        t *= i;
                        n /= i;
                }
                s *= t + 1;
        }
        if(n != 1) s *= n + 1;
        printf("%lld", s);
}
```

深入分析

关于分解质因数

分解质因数之前见过了几次，如C3-J，E3-I，这次特意又出了一道题。

补充两个分解质因数的优化做法：

优化一

n 范围不太大时，可以事先打表打出 $\sqrt{n}$ 以内的所有质数（或者用欧式筛），存到数组 $prime$ 中，然后遍历时不再遍历 i，而是遍历 $prime[i]$，可以加快算法。

优化二

Pollard-Rho算法，不做要求，感兴趣的同学可以自行搜索查阅，或者看看这篇文章 算法学习笔记(55): Pollard-Rho算法 - 知乎 (zhihu.com) 。该算法时间复杂度可以估计为 $O(n^{\frac{1}{4}})$ 。

本题时间复杂度分析

标准做法中，时间复杂度决定于分解质因数，根据循环条件可知理论上时间复杂度为 $O(\sqrt{n})$，理论上 10^{16} 范围内的 n 可能超时。但实际上只有 n 为质数或者 n 为两个非常接近的质数乘积或者 n 为质数的平方时，才会真正遍历将近 $\sqrt{n}$ 次，否则在之前遍历的过程中将 n 不断除以 p，使得 n 会缩小，实际运算次数可能远小于 $\sqrt{n}$ 次。举个极端的例子，如果 n 为 32 倍的前 13 个质数之积，将近 10^{16}，却只需要不到 40 次循环便可分解完毕。因此由于设定了“保证 n 至少能够分解为 3 个质数相乘”的条件，最坏的情况为 $n = 2 * p_1 * p_2$， p_1, p_2 为两个很接近 $\sqrt{5 \times 10^{15}}$ 的质数，经测试不会超时。

Unaccepted的做法

最直接的做法（会TLE）：根据题意，如果 d 是哪吒数，即 $d|n$ 且 d 与 $\frac{n}{d}$ 互质，则 n/d 也是哪吒数。找到 n 的所有约数 d，通过辗转相除法判断 $\gcd(d, \frac{n}{d})$ 是否为 1 确定是否为哪吒数。从 1 开始枚举 d，只需枚举到 $sqrt(n)$ 即可，即循环条件为 $d * d \leq n$，若是哪吒数则结果加上 $d + n/d$，最后输出结果。核心代码如下：

```
s = 0;
for(i = 1; i * i < n; ++i) {
        if(n % i !=0) continue; //不是n的约数，跳过本次循环
        long long a = n/i, b = i; //用临时变量a和b记录n/i和i
        while(b) { //辗转相除法
                long long c = a%b;
                a = b;
                b = c;
        } //此时a为i和n/i的最大公约数
        if(a==1LL) s += i + n/i; //如果i与n/i互质，将结果加上i+n/i
}
```

但由于数据范围达到了 10^{16}，这个做法部分测试点会超时。

辗转相除法求最大公约数的时间复杂度是 $O(logn)$（证明较复杂，感兴趣可以上网查询），因此直接枚举因数验证是否互质的做法时间复杂度是 $O(\sqrt{n} \cdot logn)$，但由于辗转相除法速度非常快，实际上大部分情况可以看作常数处理。因此为了卡掉这个做法，让大家都用分解质因数来做，设定数据范围达到了 10^{16}，对 n 设定了“至少能够分解为 3 个质数相乘”的条件，使得分解质因数的做法不会超时；而由于枚举因数必须枚举至 $\sqrt{n}$，每次还要用辗转相除法，因此该做法会超时。

Author: 哪吒

H 式神们夜里不睡觉

难度	考点
2	进制转换

简要题意

实现任意进制转换。

问题分析

关于任意进制的转换

对于一个任意进制转换的问题，我们首先应当把待转换的数码用我们熟悉的进制来表示，即十进制，否则对于一个字符串我们几乎无法获得任何有效信息。k 进制转十进制的方法较为容易，可以按如下实现：

```
long long n, len, k1, k2, i;
char s1[100];//转换前的字符串
scanf("%lld%lld%s", &k1, &k2, s1);
len = strlen(s1); //获取字符串s1的长度
for (i = 0; i < len; i++)
{
        if (isupper(s1[i]))//如果s1[i]是大写字母
                n = n * k1 + s1[i] - 'A' + 10;
        else
                n = n * k1 + s1[i] - '0';
}
```

注意到上面使用了两个函数，一个是 `strlen()`，用来获取一个字符串的长度，这个函数包含在 `string.h` 头文件中；一个是 `isupper()`，用来判断一个字符是否是大写字母，这个函数包含在 `ctype.h` 头文件中。当然，判断大写字母也可以写成 `'A'<=s1[i]&&s1[i]<='Z'`。

而对于十进制转 k 进制，对于一般正进制的题目来说，基本思路是将待转换的数不停地对 k 取余，然后将余数倒序输出。然而，在负进制的背景下，这个操作有些许不同之处。

关于负进制

首先我们应该知道的是，C 语言中 `%` 运算的含义是对某个数**取余**，你可以试试运行下面这个代码：

```
#include <stdio.h>

int main()
{
        printf("%d\n", -10 % -3); //输出-1
        printf("%d\n", 10 % -3); //输出1
        return 0;
}
```

因此，如果我们待转换的整数是一个负数，在取余的过程中余数是可能为负的，而我们不能直接将一个负余数作为转换后的结果直接输出。为了得到一个正余数，我们可以从商“借”一位到余数。

对于一个除数是负数的除法：

$$(\text{商}+1)\times\text{除数}+(\text{余数}-\text{除数})=\text{被除数}$$

由于余数的绝对值一定小于除数的绝对值，所以我们新得到的余数一定是一个正数。在负进制的转换中，我们只要每次判断余数的正负，然后根据相应的方法进行处理即可：

```
        char s0[37] = {"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"};//需要用到的进制字符
        char s2[100];//转换后的字符串
        int i=0;
        while (n)
        {
                long long reNum = n % k2;//余数
                if (reNum < 0)//余数为负数
                {
                        reNum -= k2;//余数变为正数
                        s2[i] = s0[reNum];
                        n = n / k2 + 1;//商加1
                }
                else
                {
                        s2[i] = s0[reNum];
                        n /= k2;
                }
                i++;
        }
        while (i--)//逆序输出
                printf("%c", s2[i]);
```

当然，我们不能忘记 `0` 这个特殊数字，它在任意进制下的表示都为 `0`，特判一下就可以了。

参考代码

```c
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#define LL long long

char s0[37] = {"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"};

int main()
{
	char s1[100], s2[100];
	LL n, k1, k2, T, len, i;
	scanf("%lld", &T);
	while (T--)
	{
		n = 0;
		scanf("%lld%lld%s", &k1, &k2, s1);
		if (s1[0] == '0')//0的特判
			printf("0\n");
		else
		{
			len = strlen(s1);
			for (i = 0; i < len; i++)
			{
				if (isupper(s1[i]))
					n = n * k1 + s1[i] - 'A' + 10;
				else
					n = n * k1 + s1[i] - '0';
			}
			int i = 0;
			while (n)
			{
				LL reNum = n % k2;
				if (reNum < 0)
				{
					reNum -= k2;
					s2[i] = s0[reNum];
					n = n / k2 + 1;
				}
				else
				{
					s2[i] = s0[reNum];
					n /= k2;
				}
				i++;
			}
			while (i--)
				printf("%c", s2[i]);
			printf("\n");
		}
	}
```

```
        return 0;
}
```

I 破损的三角铁

难度	考点
2	暴力，组合

简要题意

求从 n 根铁棒中选 4 根组成正三角形的方案数。

问题分析

用 4 根铁棒组成正三角形，则必有 2 根长度相等，且等于另外 2 根长度之和。

注意到铁棒长度 $a_i \leq 5000$，时间复杂度 $O(n^2)$ 可以通过。直接枚举上述两种铁棒的长度即可。

记 num_i 为长度为 i 的铁棒的个数。

1. 选取长度相等的 2 根木棒，方法数为 $C\_{num_i}^{2}$。
2. 从剩下的木棒中取出 2 根长度之和为 i 的铁棒，令其中一根长度为 j，则另一根长度为 $i-j$。若 $j=i-j$，则方法数为 $C\_num\_i^{2}$；若 $j \neq i-j$，则方法数为 $C\_{num_j}^{1} \times C\_num\_i-j^{1}$。

由乘法原理将方案数相乘，再将所有情况相加即可得到答案。

注意随时取模。

参考代码

```
#include <stdio.h>
#define LL long long

const LL N = 1000000007;

int main()
{
        int n, i, j, x;
        LL a[5005] = {0}, ans = 0, ans1;
        scanf("%d", &n);
        for (i = 0; i < n; i++)
        {
                scanf("%d", &x);
                a[x]++;//记录每种长度铁棒的数量
        }
        for (i = 2; i <= 5000; i++)
        {
                if (a[i] >= 2LL)
                {
                        ans1 = a[i] * (a[i] - 1) / 2 % N;//取两根长度相等的铁棒的方法数
                        for (j = 1; j <= i / 2; j++)
                        {
                                if (2 * j != i)
                                        ans += ans1 * a[j] % N * a[i - j] % N;
                                else if (2 * j == i)
                                        ans += ans1 * a[j] % N * (a[j] - 1) / 2 % N;
                                ans %= N;
                        }
                }
        }
        printf("%lld", ans);
        return 0;
}
```

J-海星木刻

题目分析

首先，解析一下题目：

- 有 M 个人可能参加婚礼，如果要参加需要满足两个条件：
- 其一，这个人必须要收到木刻
- 其二，这个人要么自己的好朋友确定参加婚礼，要么不需要好朋友参加就会参加婚礼。

现在需要计算出有多少人参加婚礼，需要在上述人数的基础上再加上新郎新娘，如果所有收到木刻的人都参加了婚礼，那么还需要算上风子。

思路分析：

首先读入数据，我们首先标记上收到木刻的人（标记为vis[i] = 1），只有他们才可能参加，其他人就被记为不可能参加（vis[i] = 0)。

然后读入每个人的“好朋友”（记录在invite数组中），首先，如果他收到木刻也肯定不会来，那么他和没收到木刻没有区别，也要标记为不可能参加（vis[i] = 0），其次，如果他收到木刻就一定参加，那么标记为一定参加（vis[i] = 2），需要注意，由于只有收到木刻才能满足条件，这里需要判断他是否收到木刻。

接下来就是重点——判断每个人是否能参加婚礼。对于之前已经标记过一定参加或一定不参加的，前面已经统计过，就不需要考虑，直接continue即可。这里主要考虑某个人的朋友是否到场——我们通过一个while循环，不断寻找“朋友的朋友的朋友的朋友”，这里通过flag数组记录某个人是否已经被找过，如果这里找到了一个已经被找过的人，说明形成了“环”，那就要跳出。或者找到了一个一定能来/一定不能来的人，那么就可以按照这个人来不来的情况标记前面我们走过这一串人的参与情况。这里我们使用pre数组记录一个人的上一个人是谁，用于在找到一定能来/不能来的人之后进行回溯。

在回溯过程中，我们将路径上的每个人标记为能去或不能去，因为这一次搜索结束后一定能确定路径上的人是否会去，所以他们的vis[i]也能被标记为0或2其中之一，这样就保证了每个人最多只会被搜索一次，保证了效率。

这里需要注意，因为每个人至多只有一次vis[i]置为2（被记录为一定能去）的情况，且每个人一定会被最终记录为0或2，那么只需要在每次将一个人vis[i]置为2时进行ans++即可。

最后判断ans是否等于N，等于则+=3，否则+=2，分别对应风子和新郎新娘的到场情况。

标准程序：

```c
#include <stdio.h>
#define MAXN 110000

int main(){
    int vis[MAXN], invite[MAXN], pre[MAXN], flag[MAXN];
    for (int i = 0; i < MAXN; i++){
                vis[i] = 0;
                invite[i] = 0;
                pre[i] = 0;
                flag[i] = 0;
            }
            int ans = 0;
            int N,M,tmp;
            scanf("%d%d", &N, &M);
            for (int i = 0; i < N; i++){
                scanf("%d", &tmp);
                tmp--; // 1~N to 0~N-1
                vis[tmp] = 1;
            }
            for (int i = 0; i < M; i++){
                scanf("%d", &invite[i]);
                invite[i]--; // 1~N to 0~N-1
                if (invite[i] == -2 && vis[i] == 1){
                    vis[i] = 2;
                    ans++;
                }
                else if (invite[i] == -1){
                    vis[i] = 0;
                }
            }
            for (int i = 0; i < M; i++){
                if (vis[i] == 2 || vis[i] == 0)continue;
                int pos = i;
                while (vis[pos] == 1){
                    flag[pos] = 1;
                    pre[invite[pos]] = pos;
                    pos = invite[pos];
                    if (flag[pos] && vis[pos] == 1){
                        pos = pre[pos];
                        vis[pos] = 0;
                        break;
                    }
                }
                if (vis[pos] == 2){
                    while (pos != i){
                        vis[pos] = 2;
                        pos = pre[pos];
                        ans++;
                    }
                    vis[i] = 2;
```

```
                }
                else {
                    while (pos != i && vis[pos] != 0){
                        vis[pos] = 0;
                        pos = pre[pos];
                    }
                    vis[i] = 0;
                }
            }
            if (ans == N)ans++;
            ans += 2;
            printf("%d", ans);
    return 0;
}
```