第 13 届电子科技大学趣味赛正式赛第二场题解

A.FIFA World Cup

题解

对于每组询问,输出 $\max(\frac{x}{y},\frac{y}{x})$ 即可。

Note:注意需要保留足够多的小数位数

参考代码（Python）

```
Name, Score = [], []

n = int(input())

for i in range(n):
    s = [_ for _ in input().split(' ')]
    Name.append(s[0])
    Score.append(int(s[1]))

m = int(input())
for i in range(m):
    A, B = map(str, input().split(' '))
    score_a, score_b = 0, 0
    for j in range(n):
        if Name[j] == A:
            score_a = Score[j]
        if Name[j] == B:
            score_b = Score[j]
    print(max(score_a / score_b, score_b / score_a))
```

出题人：Vingying

B.Calculate probability

题解

A与B均为单调不减数组，因此若a[i]>b[j]，那么对于任意k属于[0,j]，a[i]>b[k]。因此我们在遍历数组A时，用一个变量j标记当前满足a[i]>b[j]的最大位置，每次增加j即可。时间复杂度O(n)。

参考代码(C++):

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

```
const int MAXN(1000050);

int n,m;
int a[MAXN],b[MAXN];

void solve()
{
    int i,j;

    scanf("%d%d",&n,&m);
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
    for(i=0;i<m;i++)
        scanf("%d",&b[i]);

    long long ans=0;
    for(i=j=0;i<n;i++)
    {
        while(j<m&&b[j]<a[i])
            j++;
        ans+=j;
    }

    printf("%lld\n",ans);

}

int main()
{
    solve();
    return 0;
}
```

出题人： MrNullll

C. Extended Monty Hall Problem

题解

三门问题的扩展版本。

考虑在剩下的 k 扇门中每一扇门后有车的概率之和为 p，即当前选定门后有车的概率为 $1-p$，那么如果要换肯定取剩下门中概率最大的那扇门。

易得概率的最大值不会低于 $\frac{p}{k}$，当且仅当所有概率相等时取到。那么主持人是否有一种方法可以将这些门有车概率的最大值变为 $\frac{p}{k}$ 呢?

显然是有的，初始选定一扇门后所有剩下的门后有车的概率一定相等。

当换了一扇门以后，剩下的门中有且仅有一个与其它概率不同的值，即为当前换下来的门。

主持人只需要开启换下来的门即可做到最小化拿车概率的最大值。

由于最后剩下来的门数固定，所以只需要保证手上的门有车的概率最小即可，那么策略就是第一轮选定以后一直不换，直到最后一轮主持人开完门之后再换。

设初始门数为 n_0，最后剩余门数为 n，则答案的表达式为 $\frac{1-\frac{1}{n_0}}{n} = \frac{n_0-1}{n\times n_0}$

Code(C++)

```
#include<stdio.h>
#define eps 1e-4
int n, m;
int gcd(int x, int y) {return (x % y == 0) ? y : gcd(y, x % y);}
double p;
int main() {
    scanf("%d%d", &n, &m);
    int res = n - 1;
    double nw = 1.0 / n;
    for(int i = 1; i <= m; i++) {
        int k; scanf("%d", &k);
        res -= k;
    }
    int a = n - 1, b = n * res;
    printf("%d %d\n", a / gcd(a, b), b / gcd(a, b));
}
```

Code(python)

```
n, m = [int(x) for x in input().split()]
A = n - 1
B = n
a = [int(x) for x in input().split()]
i = 0
while i < m :
    n -= a[i]
    i += 1
n -= 1
B *= n
a = A
b = B
while B % A != 0 :
    B = B % A
    t = B
    B = A
    A = t
print(int(a / A), int(b / A))
```

出题人：verjun

D.Schedule Planning

题目大意

有2^n名参赛者，参赛者的能力值构成一个1到2^n的排列，当两个参赛者pk时，能力值大的参赛者晋级。现给定某个参赛者的能力值x_i，询问该参赛者最多可以晋级的次数。

题目分析

对于能力值为x_i的参赛者而言，只要其遇到能力值大于x_i的参赛者，他一定会被淘汰，所以我们应该尽可能让他遇到的参赛者能力值小于x_i。因此最佳方案一定是该参赛者周围的参赛者的能力值均小于x_i，所以一种可行方案是将能力值为x_i的参赛者放在第一位，能力值为1到x_i-1的参赛者排在第2到第x_i个，顺序随意；其他参赛者排在第x_i位之后，顺序随意。

对于x_i的晋级次数。假设某一次还有y个人能力值小于等于x_i，若y为偶数，则这y个人可以正好两两配对，一定有$\frac{y}{2}$个人晋级下一轮，能力值为x_i的参赛者还是排在第一位；若y为奇数，这这y个人中最后一位遇到的对手一定是能力值大于x_i的，前y个人晋级人数为$\frac{y-1}{2}$。因此无论y是奇数还是偶数，下一轮前y个人晋级的人数为$y'=\lfloor\frac{y}{2}\rfloor$，且若$y'$大于0，则能力值为$x_i$的参赛者一定会晋级。所以晋级次数为$\lfloor log_2 x_i\rfloor$。

参考代码1

```
#include<cstdio>
#define ll long long
using namespace std;

ll n,m;

int main()
{
    scanf("%lld%lld",&n,&m);
    for(int i=1;i<=m;++i)
    {
        ll x,ans=0;
        scanf("%lld",&x);
        x>>=1;
        while(x)
        {
            ans++;
            x>>=1;
        }
        printf("%lld\n",ans);
    }
    return 0;
}
```

参考代码2

跑得要慢一点。

```
#include<cstdio>
#include<cmath>
using namespace std;

int n,m;

int main()
{
    scanf("%d%d",&n,&m);
    for(int i=1;i<=m;++i)
    {
```

```
        long long x;
        scanf("%lld",&x);
        long long ans=0;
        while(pow(2,ans)<x&&pow(2,ans+1)<=x)
        ans++;
        printf("%lld\n",ans);
    }
    return 0;
}
```

出题人：Bob_Wang

E.Exchange Gifts

题解

交换礼物的规则是每个人依次把礼物传给下一个人，实际上礼物之间的相对顺序不变，可以想象成一个有 n 个固定标号的圆环，顺时针转动。不难发现，经 n 次交换后，礼物就会回到原本的位置上，继续交换只会重复先前的状态，因此最少交换次数 k 要么不存在，要么在 $[0,n-1]$ 范围内。我们对每份礼物单独考虑，对于编号为 i 的礼物，只有当它在同样编号为 i 的小朋友手上时才会破坏题目要求，而这样的情况在 n 次交换中只会出现一次。即，若礼物 i 的初始位置为 pos_i ，那么 $|i-pos_i|$ 次交换就是不合法的，因为在交换结束后会存在至少一份礼物 i 不满足条件。那么只要像这样依次找出所有不合法的交换次数，再从 0 开始枚举，找出最小的合法交换次数即可。复杂度为 $O(n)$ 。

参考代码（C++）：

```
#include<bits/stdc++.h>
using namespace std;
const int maxn=2e5+10;
int a[maxn],b[maxn];
int main()
{
    int T;
    scanf("%d",&T);
    for(int i=1;i<=T;i++){
        int n;
        scanf("%d",&n);
        for(int i=1;i<=n;i++){
            scanf("%d",&a[i]);
            b[(a[i]-i+n)%n]=1;
        }
        int k=0;
        while(b[k]&&k<n)k++;
        if(k==n)printf("-1\n");
        else printf("%d\n",k);
        for(int i=0;i<n;i++)
        b[i]=0;
    }
    return 0;
}
```

出题人：dReese

F.The Circle Game

题意

将$1,2,3,\ldots,n$(n为偶数)排成一个环，使任意连续半个环的和的最小值最大，只需输出最大的最小值。

题解

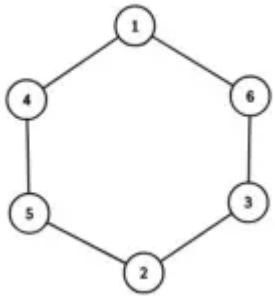
首先有一个结论：设问题的答案为ans，$sum=1+2+\cdots+n=\frac{n(n+1)}{2}$,则一定有$ans\times 2<sum$

这个结论不难证明:

> 我们任选连续$\frac{n}{2}$个数（以下称为“半环”），假设它们的和为s_1,剩下$\frac{n}{2}$个数的和为s_2,由定义有$s_1\geq ans$,$s_2\geq ans$,因此$sum=s_1+s_2\geq ans\times 2$。
>
> 另外，$sum=ans\times 2$也是不可能的。我们考虑相邻的两个“半环”，例如$1-4-5-2-3-6-1$中的$1-4-5$和$4-5-2$，两个半环的差值为$(4+5+2)-(1+4+5)=2-1=1$。即**两个相邻半环的差值等于环上两个正对面的数的差值**，由于环上不存在相同的数，这个差值一定不为0。如果$sum=ans\times 2$，则所有的“半环”的和都将等于ans,显然是不可能的。

接下来我们考虑满足$ans\times 2<sum$的最大的ans是否成立。

通过样例可以发现，这个答案对于$n=4$和6都成立。事实上，对于所有偶数n都成立。



> 例如$n=6$时$sum=21$,观察题中给出的$1-4-5-2-3-6-1$，可以发现环上任何两个正对面的数都相差1,从1开始，每个数减去正对面的数分别等于$-1,+1,-1,+1,-1,+1$，这样我们任选一个半环，容易发现它和剩下那个半环的差值一定为1(例如$(1+4+5)-(2+3+6)=-1+1-1=-1$)，因此任意两个互补半环的和一定分别等于10和11。 因此答案为10。
>
> $n=8$时，$sum=36$,考虑$1-4-5-8-2-3-6-7-1$,
>
> 每个数减去正对面的数分别等于$-1,+1,-1,+1,+1,-1,+1,-1$。任意两个互补半环的差值为0或2，则任意两个互补半环的和分别为18、18或者17、19,因此答案为17。
>
> 我们可以根据以上方法构造出所有n的最优方案，具体的构造方法是：
>
> 首先任选一个位置填1,然后在1的正对面填2,
>
> 然后在2的顺时针下一个位置填3,然后在3的正对面填4,
>
> 然后在4的顺时针下一个位置填5,然后在5的正对面填6...
>
> 直到把环填满。
>
> 当$n=4k+2$时，任意两个互补半环的差值为1，答案为$\frac{sum-1}{2}$

> 当$n=4k$时，任意两个互补半环的差值为0或2，答案为$\frac{sum}{2}-1$
>
> 可以将两种情况合并，得到$ans=\lfloor\frac{sum+1}{2}\rfloor-1$。($\lfloor\rfloor$表示"向下取整")

时间复杂度：$O(1)$

参考代码(C)

```c
#include <stdio.h>
int main() {
    int n;
    scanf("%d", &n);
    printf("%d", (n * (n + 1) / 2 + 1) / 2 - 1);
    return 0;
}
```

出题人：Blackbird

G.Arrange the desktop

题解

观察后可发现，摆放整齐后对应的位置的纵坐标一定是不大于待摆放图标的纵坐标的，所以计算移动距离时关于 y 轴的绝对值是可以去掉的，原问题等价于 x 轴上求 k 个点移动到另外 k 个位置的最小移动总距离。

不难想到，将点和位置均按 x 坐标大小排序后，第 i $(1 \leq i \leq k)$ 个点移动到第 i 个位置，按照这种移动方式移动的总距离是最小的（即每次把剩下的最左边的点移动到剩下的最左边的位置）。

假设 x 轴上两个点，x_1, x_2，x_2 位于 x_1 右边，即 $x_2 \geq x_1$。则 x_2 匹配的位置 y_2 一定也要大于等于 x_1 匹配的位置 y_1。否则，若 $x_1 \leq y_2 \leq x_2$，则会多移动 $x_2 - x_1$ 的距离。

参考代码（C++）：

```cpp
#include <bits/stdc++.h>
#define ll long long
#define int ll
using namespace std;
const int N = 2e3 + 5;

int n, m, k;

void init () {}

void charming () {
    init ();
    cin >> n >> m >> k;
    vector <pair <int, int> > p1, p2;
    vector <string> mp (n + 1);
    for (int i = 1; i <= n; ++i) {
        cin >> mp[i], mp[i] = '#' + mp[i];
    }
    for (int j = 1; j <= m; ++j) {
        for (int i = 1; i <= n; ++i) {
            if ((i - 1) * m + j <= k && mp[i][j] == '0')
```

```
            p1.emplace_back (make_pair (i, j));
            if ((i - 1) * m + j > k && mp[i][j] == '1')
            p2.emplace_back (make_pair (i, j));
        }
    }
    int res = 0, siz = p1.size ();
    for (int i = 0; i < siz; ++i) {
        auto [x1, y1] = p1[i];
        auto [x2, y2] = p2[i];
        res += abs (x1 - x2) + abs (y1 - y2);
    }
    cout << res << endl;
}

signed main () {
    charming ();
    return 0;
}
```

出题人：Charming