构造博弈论题解 ---NEU_提我踢沟沟好使

A - A Funny Game

题面

有一个n个硬币摆成的环，Alice和Bob每次从里面取出一个或两个硬币，Alice先手，谁取到最后一个硬币谁就赢。

题解

$n == 1||n == 2$ 时$Alice$可以通过直接取走所有硬币获胜，考虑$n >= 3$ 时， 可以发现一种后手必赢的策略， 即当$Alice$ 取走 1 或 2 个硬币后， 剩余硬币将形成一条链， 此时 Bob 只需要从中间取走 1 或 2 个硬币将这条链分成两条完全相同的链即可，剩下的 Bob 对称的模仿 $Alice$ 的取法。只要 $Alice$ 可以取则 Bob 一定能取。

AC代码

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

void solve() {
    int n;
    while (cin >> n, n)
        cout << (n < 3 ? "Alice\n" : "Bob\n");
}

int main(void) {
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    int T = 1;
    while (T -- ) solve();
    return 0;
}
```

B - Stone

题面

两个人轮流写数字，每次的数字至少比上一个数字大一，至多大K,先写出大于等于N的人输掉比赛。

题解

将模型抽象成有n - 1个石子，一个人最多取走k个，问最后谁赢。（裸的巴什博弈）

巴什博弈问题

问题存在两种不同的形式：

- A和B两个人玩游戏，假设有n个物品，每人每次必须至少拿1个，最多拿m个，**先拿到最后一个物品的人获胜**。思考一种最佳策略?
- A和B两个人玩游戏，假设有n个物品，每人每次必须至少拿1个，最多拿m个，**先拿到最后一个物品的人失败**。思考一种最佳策略?

首先给出结论，不管n和m的取值如何，问题都存在`唯一解`，即该游戏不公平。

赛制1：先拿完的人获胜

首先对n进行分解，n = k(m+1) + l，将n转化成与m+1相关的形式。假设A先拿，B后拿。

如果l=0，即n能被m+1整除，此时B(后手)必胜。因为无论A拿多少(x)，B只需要拿m+1-x个即可，这样在最后一个轮次时，还剩m+1个，在至少拿1个最多拿m个的限制下，B一定能拿完。
反之，如果l!=0，此时A(先手)必胜，A在只需要先拿l个，就能B陷入上一种情况。
结论：n%(m+1)==0时，B必胜，否则A必胜。

赛制2：先拿完的人失败

分析：相比于赛制1，直接思考赛制2可能更复杂，但换个角度想，其实就相当于变成了先拿完n-1个物品的人获胜，因为每次至少要拿1个，只要谁先拿到n-1个物品，剩下的那个人就必须拿完。因此，把赛制1的中的n换成n-1即可。

结论：n%(m+1)==1时，B必胜，否则A必胜。

原文链接：https://blog.csdn.net/qq_36560894/article/details/120297944

AC代码

```cpp
#include <bits/stdc++.h>
using namespace std;int T = 1;
typedef long long ll;

void solve() {
    int n, k;
    while (cin >> n >> k, n)
        cout << ((n - 1) % (k + 1) ? "Tang\n" : "Jiang\n");
}

int main(void) {
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    int T = 1;
    while (T -- ) solve();
    return 0;
}
```

C - nim 游戏

题面

n堆物品，每堆有a_i个，两个玩家轮流取走任意一堆的任意个物品，但不能不取。取走最后一个物品的人获胜。

题解

定义 Nim 和$a_1 \bigoplus a_2 \bigoplus \cdots \bigoplus a_n$。

当且仅当 Nim 和为0时，该状态为必败状态；否则该状态为必胜状态。

证明

为什么异或值会和状态的胜负有关？下面给出了这个定理的证明过程。

为了证明该定理，只需要证明下面三个定理：

- 定理 1：没有后继状态的状态是必败状态。
- 定理 2：对于$a_1 \bigoplus a_2 \bigoplus \cdots \bigoplus a_n \neq 0$的局面，一定存在某种移动使得$a_1 \bigoplus a_2 \bigoplus \cdots \bigoplus a_n = 0$。
- 定理 3：对于$a_1 \bigoplus a_2 \bigoplus \cdots \bigoplus a_n = 0$的局面，一定不存在某种移动使得$a_1 \bigoplus a_2 \bigoplus \cdots \bigoplus a_n = 0$。

对于定理 1，没有后继状态的状态只有一个，即全0局面。此时$a_1 \bigoplus a_2 \bigoplus \cdots \bigoplus a_n = 0$。

对于定理 2，不妨假设$a_1 \bigoplus a_2 \bigoplus \cdots \bigoplus a_n = k \neq 0$。如果我们要将$a_i$ 改为 a_i',则$a_i' = a_i \bigoplus k$。

假设 的二进制最高位1为d，即$2^d \leq k < 2^{d+1}$。根据异或定义，一定有奇数个a_i 的二进制第 位为 1。满足这个条件的a_i 一定也满足$a_i' = a_i \bigoplus k$，因而这也是个合法的移动。

对于定理 3，如果我们要将a_i 改为 a_i'，则根据异或运算律可以得出$a_i = a_i'$，因而这不是个合法的移动。

原文链接: https://oi-wiki.org/math/game-theory/impartial-game/

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

void solve() {
    int n; cin >> n;
    vector<int> sg(n + 1);
    for (int i = 1; i <= n; i ++ )
        cin >> sg[i], sg[i] ^= sg[i - 1];
    cout << (sg[n] ? "Yes\n" : "No\n");
}

int main(void) {
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    int T; cin >> T;
    while (T -- ) solve();
    return 0;
}
```

D - 取石子

题解

首先明确，只要还有石子，那么一定能取

不妨设有两堆石子 a_1，a_2，假设现在还有石子且无法取

那么有$a_1+a_2\geq 0, a_2<a_1<0$

①两不等式矛盾：两个负数相加必定不是非负数

②石子数不可能为负数

所以只要还有石子，就一定能取

那么问题就转换为两人轮流从一堆石子取一粒，谁先不能取谁输

我们可以假设石子一共有x粒，甲取了k粒

若是甲赢，那么甲一定比乙多取了一颗，得$x=2k+1$，x为奇数

若是乙赢，那么甲一定和乙取的石子数一样，得$x=2k$，x为偶数

就推出了双方赢的充要条件

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

void solve() {
    int n; cin >> n;
    vector<ll> a(n + 1);
    for (int i = 1; i <= n; i ++ )
        cin >> a[i], a[i] += a[i - 1];
    cout << (a[n] & 1 ? "Alice\n" : "Bob\n");
}

int main(void) {
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    int T = 1;
    while (T -- ) solve();
    return 0;
}
```

E - Roy&October之取石子

题解

只有6的倍数是必败态，其余是必胜态。

首先0个石子的状态一定是必败态，因为对面在上一轮已经拿完了。

观察1~5个石子，发现$1=p^0,2=2^1,3=3^1,4=2^2,5=5^1$，都是必胜态，可以一次拿完赢得游戏。

然后6个石子没办法一下拿完（因为$6 \neq p^k$）。可以知道只能拿1~5个石子，这样都会转移到前面的必胜态，只不过这个必胜态已经是对面的了，**所以说6个石子是你的必败态，在你面前出现6个石子又轮到你拿的时候，你必定失败**。

这样一直往后找到12的时候，发现7 ~ 11 都是必胜态（一次把石子总数拿到6个石子然后对面就输了），而12是必败态（可以枚举1~12中的所有p^k，都转移到对面的必胜态）。

于是猜想**所有6^n的状态是必败态，其余所有状态($6n+1, 6n+2, 6n+3, 6n+4, 6n+5$)都是必胜态**。

我们采用数学归纳法证明：

当$n = 0$时，结论成立，因为0~5上面已经说明过了。

现在假设$0 \sim 6n-1$都满足结论。

先证明$6n$为必败态：因为任何p^k，都不是6的倍数，所以$6n$个石子拿完一次**不会还是6的倍数**，故必定转移到对面的必胜态，所以$6n$是必败态。

显然$6n + r(r = 1, 2, 3, 4, 5)$只需要拿掉r便可以转移到$6n$，是对面的必败态，所以$6n + r(r = 1, 2, 3, 4, 5)$是必胜态。

证毕。

AC代码

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

void solve() {
    int n; cin >> n;
    cout << (n % 6 ? "October wins!\n" : "Roy wins!\n");
}

int main(void) {
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    int T; cin >> T;
    while (T -- ) solve();
    return 0;
}
```

F - Climbing the Hill

题面

有n个人爬山，按升序给出每个人的位置，现在，Alice 和 Bob 每次可以选择一个不在山顶的人向上爬若干步，但不能越过其他人，两人轮流操作，使得最后一个人到达山顶的获胜。

题解

阶梯nim游戏的模板。

阶梯nim游戏。

有n个台阶，第i个台阶有ai个石子。每次，我们可以选择一个台阶，将上面至少1个石子放到它下面的台阶上去(第一个台阶的石子可以放到地上，地上的石子不能动)，两个人轮流操作，不能操作的输，问，谁能获胜。

考虑用sg函数从特殊到一般分析

假设只有一个台阶，那么当a1 = 0时，先手必输，sg = 0;

当a1 = 1时，我们可以将当前石子放到地上，到达sg = 1的情况...

总结可得sg = a1

再考虑有两个台阶的情况

当a1 = 0, a2 = 1时，我们只能把第二个台阶的石子放到第一个台阶上，变成a1 = 1, a2 = 0，因为sg(1, 0) = 1, 所以sg(0, 1) = 0

考虑(1, 1)，此时只能变成(0, 1)[sg = 0]或者(2, 0)[sg = 2]，所以sg(1, 1) = 1

...归纳可得sg(x, 1) = x

考虑(0, 2), 它可以变成(1, 1)[sg = 1]和(2, 1)[sg = 2]，所以sg(0, 2) = 0。

考虑(1, 2)，它可以变成(0, 2)[sg = 0], (2, 1)[sg = 2]和(3, 0)[sg = 3]，所以sg(1, 2) = 1

综上归纳，sg(x, y) = x

我们发现，第二个台阶不管有多少个石子都对sg值无影响，这与地板的性质是一样的

那么，我们将第二个台阶看做地板，那么第三个台阶就相当于第一个台阶，贡献就是a3...一直往后推，我们就能发现，只有奇数的台阶可以产生贡献，所以我们把所有奇数台阶的石子求异或，然后判断是否为0即可。

现在，我们来看这道题。

我们新建一个b数组，其中bi表示第i个人和第i-1个人距离的差距，特别的，b1等于第1个人离山顶的距离。我们考虑让第i个人向上走x的距离，那么，bi -> bi – x，bi+1 -> bi+1 + x，相当于，我们从第i个台阶拿了x个石子到第i+1个台阶上去。这与阶梯nim游戏的规则是一样的(只是石子向下移动变成了向上移动)，直接套阶梯nim游戏的结论即可。

AC代码

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

void solve() {
    int n, k;
    while (cin >> n >> k) {
        int ans = 0;
        vector<int> a(n + 1);
        for (int i = 1; i <= n; i ++ ) cin >> a[i];
        a[0] = 0;
        if (k == 1) {
            cout << "Alice\n";
            continue;
        }
```

```
        else if (k != 2) a[0] = -1;
        for (int i = n; i >= 1; i -= 2) ans ^= a[i] - a[i - 1] - 1;
        cout << (ans ? "Alice\n" : "Bob\n");
    }
}

int main(void) {
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    int T = 1;
    while (T -- ) solve();
    return 0;
}
```

G - Fox and Card Game

题面

有n排卡，第i排卡一共有si张卡，每张卡有个点数。第一个人可以选择一排卡，并取走第一张卡。第二个人选择一排卡，并取走最后一张卡。两人轮流操作。两人都想让自己获得的卡的点数和最大，问最后两人点数各是多少。

题解

对于有偶数张卡的那排，两人一定是各取上下两半，因为这样做至少会有一个人受益。而如果有一个人取了其中一张，另一个人可以马上取另一张，使得仍满足各取一半。那么，如果不受益的那方取了其中一张，收益一方一定会取另一张使得自己受益。对于有奇数张卡的那排，第一个人取了后，变成偶数排，各取一半，相当于第一个取奇数卡排的人多获得了中间张的收益，剩下的各取上下两半。那么，两人一定每次选能拿的最大的中间张。将所有奇数排中间的卡的点数按从大到小排序后，两人依次拿走，剩下卡的各取上下两半即可。

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

void solve() {
    int m; cin >> m;
    int sum1 = 0, sum2 = 0;
    vector<int> mid;
    while (m -- ) {
        int n; cin >> n;
        int x;
        for (int i = 1; i <= n >> 1; i ++ )
            cin >> x, sum1 += x;
        if (n & 1)
            cin >> x, mid.push_back(x);
        for (int i = 1; i <= n >> 1; i ++ )
            cin >> x, sum2 += x;
    }
    sort(mid.begin(), mid.end(), greater<int>());
    for (int i = 0; i < mid.size(); i ++ )
        if (i & 1)
            sum2 += mid[i];
        else
```

```
            sum1 += mid[i];
    cout << sum1 << " " << sum2 << "\n";
}

int main(void) {
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    int T = 1;
    while (T -- ) solve();
    return 0;
}
```

H - Tree Game

题面

给你一颗n个节点的树，每个节点有若干石子，先手可以选择在一个节点放一个棋子，每个回合，每个人可以从棋子所在节点取一个石子，并将棋子移动到与当前节点相邻的一个节点。问有多少个棋子所在的初始节点使得先手必胜。

题解

我们首先考虑，如果当前节点石子数为0，那么肯定就是先手必败状态。

然后，如果我们把棋子从一个点移动到一个石子个数不比它小的节点，那么后手可以将这个棋子再移动回来，这样的话，相当于当前节点和我们移动的节点的石子数各减少1，那样的话，因为目标节点不比当前节点石子个数少，每次移动对方都能移动回来，这样，我们就会面临当前节点石子个数为0的情况，导致失败。

所以，我们每次移动都是向权值比当前权值小的节点移动。

这样就简单了我们按权值从小到大求每个点的情况所有石子数最少的点一定是先手必败状态。因为一个点只能向比它石子数少的点转移，那么它能转移到的点的情况我们已经算完了。如果一个点能转移到一个先手必败状态，那么它就是先手必胜状态，否则它就是先手必败状态

AC代码

```
#include<bits/stdc++.h>
using namespace std;
const int N = 2e5 + 10;
vector<int> e[N];
int a[N];
int f[N];
bool dp[N];
void solve()
{
    int n;
    vector<pair<int, int> > v;
    cin >> n;
    for(int i = 1 ; i <= n ; i++)
    {
        int x;
        cin >> x;
        a[i] = x;
        v.push_back({x , i});
```

```
    }
    for(int i = 1 ; i < n ; i++)
    {
        int x ,y ;
        cin >> x >> y;
        e[x].push_back(y);
        e[y].push_back(x);
    }
    sort(v.begin(), v.end());
    for(auto i : v)
    {
        int x = i.first;
        int y = i.second;
        bool f = 0;
        for(int i : e[y])
        {
            if((!dp[i]) && a[i] < x)
                f = 1;
        }
        dp[y] = f;
    }
    for(int i = 1 ; i <= n ; i++)
        if(dp[i]) cout << i << ' ';
    cout << endl;
}
int main()
{
    solve();
}
```

I - 取石子游戏

题面

一堆石子有n个，两人轮流取，先取者第1次可以取任意多个，但不能全部取完，以后每次取的石子数不能超过上次取子数的2倍。取完者胜.先取者负输出"Second win".先取者胜输出"First win"。

题解

这个跟威佐夫博弈和取石子游戏有一个很大的不同点，就是游戏规则的动态化。后两者的规则中，每次可以取的石子的策略集合是基本固定的，但是这次有规则2：一方每次可以取的石子数依赖于对手刚才取的石子数。

这个游戏叫做Fibonacci Nim，肯定和Fibonacci数列f[n]：1,2,3,5,8,13,21,34,55,89,... 有密切的关系。如果试验一番之后，可以猜测：先手胜当且仅当n不是Fibonacci数，换句话说，必败态构成Fibonacci数列。

下面简单谈谈“先手败当且仅当n为Fibonacci数列”这一结论是怎么得来的。

这里要用到一个很有用的定理：任何正整数可以表示为若干个不连续的 Fibonacci 数之和。这里定理涉及到数论，这里不做证明。下面只谈如何把一个正整数表示为若干个不连续Fibonacci 数之和。

比如，我们要分解83，注意到83被夹在55和89之间，于是把83可以写成83=55+28；然后再想办法分解28，28被夹在21和34之间，于是28=21+7；依此类推 7=5+2，故83=55+21+5+2。

如果 n 是 Fibonacci 数，比如 n = 89。89前面的两个Fibonacci 数是34和55。如果先手第一次取的石子不小于34颗，那么一定后手赢，因为 89 - 34 = 55 = 34 + 21 < 2*34，注意55是Fibonacci数。此时后手只要将剩下的全部取光即可，此时先手必败。故只需要考虑先手第一次取得石子数 < 34 即可，于是剩下的石子数 x 介于 55 到 89 之间，它一定不是一个 Fibonacci 数。于是我们把 x 分解成 Fibonacci 数：x = 55 + f[i] + ... + f[j]，其中55 > f[i] > ... > f[j]，如果 f[j] ≤ 先手一开始所取石子数 y 的两倍，那么对后手就是面临 x 局面的先手，所以根据之前的分析，后手只要先取 f[j] 个即可，以后再按之前的分析就可保证必胜。

下证：f[j] ≤ 2y

反证法：假设f[j]>2y，则 y < f[j]/2 = (f[j-1] + f[j-2])/2 < f[j-1]。而最初的石子数是个斐波那契数，即

n = f[k] = x + y < f[k-1] + f[i] + ... + f[j] + f[j-1] ≤ f[k-1]+f[i]+f[i-1] ≤ f[k-1]+f[k-2] ≤

f[k] （注意第一个不等号是严格的），矛盾！f[j] ≤ 2y得证。

如果 n 不是 Fibonacci 数，比如n=83，我们看看这个分解有什么指导意义：假如先手取2颗，那么后手无法取5颗或更多，而5是一个Fibonacci数，如果猜测正确的话，（面临这5颗的先手实际上是整个游戏的后手）那么一定是整个游戏的先手取走这5颗石子中的最后一颗，而这个我们可以通过第二类归纳法来绕过，同样的道理，根据"先手败当且仅当n为Fibonacci数列"，接下去先手取走接下来的后21颗中的最后一颗，再取走后55颗中的最后一颗，那么先手赢。

原文链接：https://blog.csdn.net/ACdreamers/article/details/8586135

```cpp
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <map>
#include<cmath>

using namespace std;
const int N = 95;
typedef long long LL;

int n, k;
LL sum;
int a[N], b[N];
LL f[N];

int main (void) {

    f[0] = f[1] = 1;
    for (int i = 2; i < N; i ++ ) f[i] = f[i - 1] + f[i - 2];
    while (scanf("%d", &n), n) {
        int c = 0;
        for (int i = 2; i < N; i ++ )
            if (f[i] == n) c = 1, puts("Second win");
        if (!c) puts("First win");
    }
    return 0;
}
```

J - Gluttony

题面

给定一个长为n的序列A,其中保证序列中各个元素互不相同，现要求将序列A打乱并重新组合成一个序列B,使得∀ S⊂{1,2,...,n}且s!= ϕ，以及全集有 $\sum_{x\in S} A_x \neq \sum_{x\in S} B_x$。如果无解输出-1，有解输出任意一组。

1 ≤ n ≤ 22，0 ≤ ai ≤ 10^9

题解

实际上，我们把a排序后，让a整体往前移一位，然后把a[1]放到a[n]，从而得到b，可以保证这样的b一定是满足条件的。

证明：对于集合$S=\{x_1,x_2,\dots,x_k\}$，若n不在S中，则必然有$\sum_{i=1}^{k} b_{x_i} > \sum_{i=1}^{k} a_{x_i}$

反之，则设S′ 为S的补集，有$S'=\{y_1,y_2,\dots,y_{n-k}\}$，不难发现有$\sum_{i=1}^{n-k} b_{y_i} > \sum_{i=1}^{n-k} a_{y_i}$，从而有$\sum_{i=1}^{k} b_{x_i} < \sum_{i=1}^{k} a_{x_i}$。

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

void solve() {
    int n; cin >> n;
    vector<int> w(n), b(n);
    iota(b.begin(), b.end(), 0);
    for (int i = 0; i < n; i ++ )
        cin >> w[i];
    sort(b.begin(), b.end(), [&] (int &i, int &j) {
        return w[i] < w[j];
    });
    vector<int> ans(n);
    for (int i = 0; i < n; i ++ )
        ans[b[i]] = w[b[(i + 1) % n]];
    for (int i = 0; i < n; i ++ )
        cout << ans[i] << " \n"[i == n - 1];
}

int main(void) {
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    int T = 1;
    while (T -- ) solve();
    return 0;
}
```

K - Resources Distribution

题面

一个正方体每个面分为n*n*个小方块，每个小方块填上*1~6*n*n的数字，要求所有方块数字不一样且绕一圈数字和相同。

题解

容易知道所有的绕圈方式一共有3*n*种，而这些绕圈方式每个方块会被遍历两遍，同时每圈的和都是相同的，因此每圈的和*sum* = (*1* + *6*n*n) * 6 * n * n / （3n）= 2*n*(1+6 * n * n) , 同时，每一圈恰好有2*n*个点对（即每个方格和与他相对的方格），因此只需构造每个方格以及与他相对的方格的和为(*1*+*6*n*n)即可。

```cpp
#include <bits/stdc++.h>
using namespace std;
void solve()
{
    int n;
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
            cout << i * n + j + 1 << " ";
        cout << endl;
    }
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < 2 * n; j++)
            cout << n * n + 2 * n * i + j + 1 << " ";
        for (int j = 0; j < n; j++)
            cout << 5 * n * n - (2 * i + 1) * n + j + 1 << " ";
        for (int j = 0; j < n; j++)
            cout << 5 * n * n - (2 * i + 2) * n + j + 1 << " ";
        cout << endl;
    }
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
            cout << 5 * n * n + (i + 1) * n - j << " ";
        cout << endl;
    }
}
int main()
{
    solve();
}
```

L - No Prime Differences

题解

如果n为奇数，从1到n*m按顺序先填奇数行，再填偶数行。

如果n为偶数，从1到n*m按顺序先填偶数行，再填奇数行。

这样一定保证了相邻两列间元素的差值为1，相邻两行间元素的差值为(n/2 + 1) * m 或者 (n/2) * m，由于n,m均大于等于4，因此差值一定不是质数。

AC代码

```cpp
#include <bits/stdc++.h>
using namespace std;
#define endl "\n"
const int N = 1e3 + 10;
int a[N][N];
void solve()
{
    int n, m;
    cin >> n >> m;
    int cnt = 1;
    if (n & 1)
    {
        for (int i = 1; i <= n; i += 2)
            for (int j = 1; j <= m; j++)
                a[i][j] = cnt++;
        for (int i = 2; i <= n; i += 2)
            for (int j = 1; j <= m; j++)
                a[i][j] = cnt++;
    }
    else
    {
        for (int i = 2; i <= n; i += 2)
            for (int j = 1; j <= m; j++)
                a[i][j] = cnt++;
        for (int i = 1; i <= n; i += 2)
            for (int j = 1; j <= m; j++)
                a[i][j] = cnt++;
    }
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= m; j++)
            cout << a[i][j] << ' ';
        cout << endl;
    }
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    int t;
    cin >> t;
```

```
    while (t--)
        solve();
}
```

M - Hidden Word

题面

给你一个字符串（长度固定为27），然后让你将这个字符串变成两行2*13的字符串，使得其有一条路能够走出原字符串的路径来(向相邻的格子走，斜着也可以)。题目保证每个字母都至少出现一次。

题解

首先考虑无解情况，显然当给定串中存在两个相同的字母相邻时一定无解。

在有解时，可以取两个相同字符的中点截断，前一半放第一行，后一半反转放第二行，如果有任意一行长度超过13，那么就将该行从前面截到多余的部分，并将多余部分反转放另一行开头。

AC代码

```cpp
#include<bits/stdc++.h>
using namespace std;
#define endl "\n"
void solve()
{
    string s;
    cin >> s;
    for(int i = 0 ; i < 27 ; i++)
        if(s[i] == s[i + 1])
        {
            cout << "Impossible" << endl;
            return ;
        }
    unordered_map<char, int> mp;
    string s1 = "";
    string s2 = "";
    for(int i = 0 ; i < 27; i++)
    {
        if(mp.count(s[i]))
        {
            int dis = i - mp[s[i]] - 1;
            s1 += s.substr(0 , dis / 2 + mp[s[i]] + 1);
            s2 += s.substr(dis / 2 + mp[s[i]] + 1, (dis + 1) / 2);
            if(i != 26) s2 += s.substr(i + 1);
            break;
        }
        mp[s[i]] = i;
    }
    cerr << s1 << endl << s2 << endl;
    if(s1.length() == 13)
    {
        reverse(s2.begin() , s2.end());
        cout << s1 << endl << s2 << endl;
    }
    else if(s1.length() < 13)
```

```
    {
        string tmp = s2.substr(13);
        s2 = s2.substr(0 , 13);
        s1 = tmp + s1;
        reverse(s2.begin(), s2.end());
        cout << s1 << endl << s2 << endl;
    }
    else
    {
        string tmp = s1.substr(0 , s1.length() - 13);
        s1 = s1.substr(s1.length() - 13);
        reverse(tmp.begin(), tmp.end());
        reverse(s2.begin(), s2.end());
        s2 = tmp + s2;
        cout << s1 << endl << s2 << endl;
    }

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

N - Vladik and fractions

题解

可以发现：2/n = 1/n + 1 / (n *(n + 1)) + 1 / (n + 1)

同时，当n == 1时，无解

AC代码

```
#include<bits/stdc++.h>
using namespace std;
void solve()
{
    int n;
    cin >> n;
    if(n == 1)
    {
        cout << -1 << endl;
        return;
    }
    cout << n << " " << n + 1 << " " << n * (n + 1) << endl;
}
int main()
{
    solve();
}
```

O - GRA-Tower Defense Game

题解

给一个n个点m条边的简单无向图,已知用k个辐射范围为1的防御塔可以覆盖全图.构造一种方案,使用r个辐射范围为2的防御塔来覆盖全图,r需要小于等于k,无需最小化r.

题面

假设x1,x2.....xk是一种使用辐射范围为1的防御塔的构造方案,那么xi覆盖的点集可以表示为dis(xi,x)<=1.这个点集中任意两个点必然满足dis(x,x')<=2.也就是说在点集中任意选择一个点放置辐射范围为2的防御塔,都必然可以覆盖这个点集.因此bfs模拟即可.

AC代码

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    ios::sync_with_stdio(false);cin.tie(0),cout.tie(0);
    //freopen("std.in","r",stdin);freopen("std.out","w",stdout);
    auto solve=[&]()->void
    {
        int n,m,k;cin>>n>>m>>k;
        vector<vector<int>> e(n+1);
        vector<bool> v(n+1,0);
        for(int i=0,x,y;i<m;++i)
cin>>x>>y,e[x].push_back(y),e[y].push_back(x);
        queue<int> ans;
        for(int i=1;i<=n;++i)
        {
            if(v[i]) continue;
            v[i]=true;ans.push(i);
            queue<array<int,2>> q;q.push({i,2});
            while(!q.empty())
            {
                auto [x,y]=q.front();q.pop();
                if(!y) continue;
                for(auto z:e[x]) v[z]=1,q.push({z,y-1});
            }
        }
        cout<<ans.size()<<'\n';
        while(!ans.empty()) cout<<ans.front()<<' ',ans.pop();
    };
    int T=1;
    //cin>>T;
    while(T--) solve();
    return 0;
}
/*
**Tashkent won't resist.
**Please be gentle...
*/
```

P - Hashing Trees

题面

给定一颗树的高度h和序列a,ai表示与根节点距离为i的节点数,判断这棵树是否存在非同构树同样满足高度h和序列a,如果存在打印任意两颗非同构树的父节点序列。

题解

考虑ai和a[i+1],其实就是树的第i层有ai个节点,i+1层有a[i+1]个节点.如果ai=1或者a[i+1]=1,此时我们无论如何连接这两层,都是同构的.其余情况我们可以选择把a[i+1]个节点全接在上一层同一个节点上,也可以选择平均分配连接上一层,这样必然可以得到两颗非同构树,构造完毕.

AC代码

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    ios::sync_with_stdio(false);cin.tie(0),cout.tie(0);
    //freopen("std.in","r",stdin);freopen("std.out","w",stdout);
    auto solve=[&]()->void
    {
        int h,t=1,s=0;cin>>h;
        vector<int> a(h+1);
        for(int i=0;i<=h;++i) cin>>a[i],s+=a[i];
        vector<array<int,2>> fa(s+1);
        for(int i=0,l=0,r=0;i<=h;++i)
        {
            if(i&&a[i]>1&&a[i-1]>1) t=0;
            for(int j=r+1,k=l;j<=r+a[i];++j)
            {
                fa[j][0]=l,fa[j][1]=k++;
                if(k>r) k-=r-l+1;
            }
            l=r+1;r=l+a[i]-1;
        }
        if(t) return cout<<"perfect\n",void();
        cout<<"ambiguous\n";
        for(int i=0;i<2;++i) for(int j=1;j<=s;++j) cout<<fa[j][i]<<" \n"
[j==s];
    };
    int T=1;
    //cin>>T;
    while(T--) solve();
    return 0;
}
/*
**Tashkent won't resist.
**Please be gentle...
*/
```

Q - Epidemic in Monstropolis

题面

给定n个怪物,每个怪物都有一个初始权值ai,在任何时刻都可以操控一个怪物吞噬与之相邻且权值严格小于它的怪物,然后这个怪物的权值增加被吞噬怪物的权值,被吞噬的怪物从队列中删除.构造出一个操作序列,在执行操作后所有剩下的怪物的权值序列与长度为k的序列b完全相同,或者报告不存在这样的操作序列.

题解

考虑序列b的第一个元素,它必然是序列a的前缀和,因为每个怪物被吞噬必然产生贡献不可能凭空消失.b的后续元素同理.也就是说,a的划分方式是唯一的,双指针即可进行划分.接下来考虑a的一个划分块内如何构造.用块内最大元素不断吞噬其他元素即可,可以证明无法构造的情况只有块内所有元素相等,否则必然可以一直用最大元素来吞噬构造.

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    ios::sync_with_stdio(false);cin.tie(0),cout.tie(0);
    //freopen("std.in","r",stdin);freopen("std.out","w",stdout);
    auto solve=[&]()->void
    {
        int n;cin>>n;
        vector<int> a(n);
        for(int i=0;i<n;++i) cin>>a[i];
        queue<pair<int,char>> ans;
        int k,t=1;cin>>k;
        for(int i=0,l=0,r,x,mx,s=0,p;i<k;++i)
        {
            cin>>x;
            if(!t) continue;
            s=0;r=l;mx=0;p=-1;
            while(r<n&&s<x) s+=a[r],mx=max(mx,a[r++]);
            if(i==k-1&&r!=n) {t=0;continue;}
            if(s!=x) {t=0;continue;}
            if(r==l+1) {l=r;continue;}
            for(int j=l;j+1<r&&p<0;++j) if(a[j]==mx&&a[j+1]!=mx) p=j;
            for(int j=l+1;j<r&&p<0;++j) if(a[j]==mx&&a[j-1]!=mx) p=j;
            if(p<0) {t=0;continue;}
            if(p+1<r&&a[p+1]!=mx)
            {
                for(int j=p+1;j<r;++j) ans.push({i+p-l+1,'R'});
                for(int j=p-1;j>=l;--j) ans.push({i+j-l+2,'L'});
            }
            else
            {
                for(int j=p-1;j>=l;--j) ans.push({i+j-l+2,'L'});
                for(int j=p+1;j<r;++j) ans.push({i+1,'R'});
            }
            l=r;
        }
        if(!t) return cout<<"NO\n",void();
```

```
        cout<<"YES\n";
        while(!ans.empty()) cout<<ans.front().first<<'
'<<ans.front().second<<'\n',ans.pop();
    };
    int T=1;
    //cin>>T;
    while(T--) solve();
    return 0;
}
/*
**Tashkent won't resist.
**Please be gentle...
*/
```

R - IMP-Party

题面

给定3k个点和m条边的无向图,保证图中存在一个大小为2k的团,要求构造一个大小为k的团,有多个答案输出任意一个即可。

题解

首先需要知道什么是团.团的定义是任意两点之间都有连边的点集.考虑枚举点对(i,j),如果i,j之间没有连边,说明i和j之间至少有一个是团外的点,我们不妨认为他们全是团外的点,尽管有可能有些团内的点被误认为是团外的点,但是由于团外的点只有k个,最多有k个团内的点被删除,剩余k个点仍可满足要求.

AC代码

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    ios::sync_with_stdio(false);cin.tie(0),cout.tie(0);
    //freopen("std.in","r",stdin);freopen("std.out","w",stdout);
    auto solve=[&]()->void
    {
        int n,m;cin>>n>>m;
        vector<vector<bool>> e(n+1,vector<bool>(n+1,0));
        vector<bool> v(n+1,0);
        for(int i=0,x,y;i<m;++i) cin>>x>>y,e[x][y]=e[y][x]=1;
        for(int i=1;i<=n;++i)
        {
            if(v[i]) continue;
            for(int j=1;j<=i;++j)
            {
                if(j==i||v[j]) continue;
                if(!e[i][j]) {v[i]=v[j]=1;break;}
            }
        }
        for(int i=1,t=0;t!=n/3;++i) if(!v[i]) cout<<i<<' ',++t;
    };
    int T=1;
    //cin>>T;
```

```
    while(T--) solve();
    return 0;
}
/*
**Tashkent won't resist.
**Please be gentle...
*/
```

Bonus

根据定义大小为k的团内的点度数至少为k-1,我们把所有点按度数排好序,在前2k个点中随机输出k个点,正确概率是很高的.不放心的话也可以写个check函数多随机几次.这里随机数生成器推荐使用mt19964,不管是生成数据对拍,随机数异或哈希都很好用.具体mt19964的实现原理感兴趣的话自行搜索学习,这里只给出使用方法.

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    ios::sync_with_stdio(false);cin.tie(0),cout.tie(0);
    //freopen("std.in","r",stdin);freopen("std.out","w",stdout);
    auto solve=[&]()->void
    {
        int n,m;cin>>n>>m;
        vector<array<int,2>> in(n+1);
        for(int i=0;i<=n;++i) in[i][0]=0,in[i][1]=i;
        for(int i=0,x,y;i<m;++i) cin>>x>>y,++in[x][0],++in[y][0];
        sort(in.begin(),in.end(),greater<array<int,2>>());
        mt19937_64 rnd(time(NULL));
        shuffle(in.begin(),in.begin()+n/3*2,rnd);
        sort(in.begin(),in.begin()+n/3,[&](array<int,2> x,array<int,2> y)
{return x[1]<y[1];});
        for(int i=0;i<n/3;++i) cout<<in[i][1]<<' ';
    };
    int T=1;
    //cin>>T;
    while(T--) solve();
    return 0;
}
/*
**Tashkent won't resist.
**Please be gentle...
*/
```

S - Weighting a Tree

题解

首先，考虑这个图如果是一棵树的话，那么答案显然应该可以从叶子节点逐渐向根节点靠拢的同时为边赋值，最后再检查一下根节点是否合法即可。

因此我们考虑在原图不是树的时候，可以理解为在其生成树上去添加了一些新的边（当然，此时我们可以构造出来在这棵生成树上除了根节点以外，所有节点均满足条件的方案），那么我们考虑加上这些边后对于节点权值的影响，是否可以将根节点也变得合法，可以发现，

1.若所添加的边连接的两个节点u,v的深度的奇偶性不同的话，该边在添加进去时，能够保证根节点不会被扰动的同时（即与根节点直接相连的边权值无需发生变化），其余的节点依然合法。

2.若所添加的边连接的两个节点u,v的深度的奇偶性相同的话，该边在添加进去时，根节点一定会被扰动，并且这样的扰动，一定是以添加边权值的二倍为幅度的。

所以，我们可以先dfs跑出生成树的答案，再枚举不在树上的边（即所要加入的边），由于添加新边时，如果想做到其他节点依然合法，根节点的变化幅度一定是偶数，因此在dfs跑完后，如果此时根节点的所有邻边的权值和与根节点的权值相差为奇数，一定是无解的。

在枚举边时，可以发现只有当边的两个节点u,v深度的奇偶性相同时，才能做到只修改根节点，因此若没有这样的边存在，且此时根节点不合法，那么一定无解。

其余情况只需要在遇到这样的边是，若此时根节点权值为x，根节点所有的邻边的权值和为y，那么如果u，v两个点的深度为奇数就给边赋值为(x – y) / 2,否则为 -(x – y) / 2， 具体可以手动模拟一下。

其他剩余的边全都赋值为0即可。

AC代码

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define endl "\n"
const int N = 2e5 + 10;
ll dep[N];
ll a[N];
vector<int> e[N];
set<pair<int, int>> se;
map<pair<int, int>, ll> ans;
ll dfs(int u, int fa)
{
    dep[u] = dep[fa] + 1;
    ll sum = 0;
    for (int i : e[u])
    {
        if (dep[i] != 0 || i == fa)
            continue;
        if (i < u)
            se.insert({i, u});
        else
            se.insert({u, i});
        ll tmp = dfs(i, u);
        sum += tmp;
    }
    return a[u] - sum;
}
ll dfs2(int u , int fa)
{
    ll sum = 0;
    dep[u] = dep[fa] + 1;
    for (int i : e[u])
    {
        if (dep[i] != 0 || i ==  fa)
            continue;
        ll tmp = dfs2(i, u);
```

```
        sum += tmp;
        if(i < u) ans[{i , u}] = tmp;
        else ans[{u, i}] = tmp;
    }
    return a[u] - sum;
}
void solve()
{
    ll n, m;
    cin >> n >> m;
    vector<pair<int, int>> ed;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    for (int i = 0; i < m; i++)
    {
        int x, y;
        cin >> x >> y;
        if (x > y)
            swap(x, y);
        ed.push_back({x, y});
        e[x].push_back(y);
        e[y].push_back(x);
    }
    ll tmp = dfs(1, 0);
    if (tmp % 2 != 0)
    {
        cout << "NO" << endl;
        return;
    }
    int ct = 0;
    for (int i = 0; i < m; i++)
    {
        if (se.count(ed[i]))
            continue;
        int u = ed[i].first;
        int v = ed[i].second;
        if ((dep[u] & 1) == (dep[v] & 1))
        {
            if (dep[u] & 1) tmp = tmp;
            else tmp = -tmp;
            a[u] -= tmp / 2;
            a[v] -= tmp / 2;
            ans[{u, v}] = tmp / 2;
            tmp = 0;
            break;
        }
    }
    for(int i = 1 ; i <= n ; i++)
        dep[i] = 0;
    ll tt = dfs2(1 , 0);
    if(tt != 0 )
        cout  << "NO" << endl;
    else
    {
        cout << "YES" << endl;
        for(int i = 0 ; i < m ; i++)
```

```
            cout << ans[ed[i]] << endl;
    }

}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    solve();
}
```

T - Dreamoon and Sets

题面

给定整数n,k.构造n个大小为4的不相交集合,每个集合中元素两两gcd均为k,所有集合中出现的最大元素为m,构造要求最小化m.

题解

首先对于一个集合中的元素必然是形如k*p1*,kp2...其中p1,p2...互质.问题转化为找到4*n*个*p*,*每4个互质的p分成一组.由于每组中至多出现一个偶数,因此无论如何p都要取到*6n-1,这是一个下界.考虑能否只用小于等于这个下界的p来构造.首先看奇数部分,相邻的3个奇数一定是互质的,接下来只需要放入一个偶数即可.按照上述构造方案一组内三个奇数是6*x*+*1*,6*x*+3,6*x*+*5*,*此时放入*6*x*+4这个偶数必然满足互质的要求.

AC代码

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    ios::sync_with_stdio(false);cin.tie(0),cout.tie(0);
    //freopen("std.in","r",stdin);freopen("std.out","w",stdout);
    auto solve=[&]()->void
    {
        int n,k;cin>>n>>k;
        cout<<k*(6*n-1)<<'\n';
        for(int i=0,j=0;i<n;++i,j+=6) cout<<k*(j+1)<<' '<<k*(j+3)<<' '<<k*
(j+5)<<' '<<k*(j+4)<<'\n';
    };
    int T=1;
    //cin>>T;
    while(T--) solve();
    return 0;
}
/*
**Tashkent won't resist.
**Please be gentle...
*/
```