Chapter 2　Mathematics

Keshuai

2014-11-20

目录

一、计数问题

（一）基本计数方法

1.加法原理：做一件事情有n中办法，第i种办法有pi种执行方案，那么总的解决这件事情的方案数即为$p1+p2+p3+...+pn$。

2.乘法原理：做一件事情分为n个步骤，第i个步骤的执行方案有pi种，则一共有$p1*p2*p3*...*pn$种方案解决该问题。

3.容斥原理：一个班级有，集合A的人喜欢数学，集合B的人喜欢英语，结合C的人喜欢语文，那么该班级的人数应该是多少？

如果我们将三个集合的人数相加起来，那么就重复计算了既喜欢数学又喜欢英语的、既喜欢英语又喜欢语文的和既喜欢数学又喜欢语文的人，还有三种都喜欢的学霸级人物被计算了三次!!!

完全不科学啊，所以我们再减去既喜欢数学又喜欢英语的、既喜欢英语又喜欢语文的和既喜欢数学又喜欢语文这样的次级学霸。嗯，没错，计算了两次就减掉一次。但是好像哪里有什么不对，我们貌似忘记计算学霸了(三个科目都喜欢的人)，好没存在感，被计算了三次又被减掉了三次!.所以作为特殊补偿，我们单独计算学霸。

于是乎得到了公式：$|A\bigcup B\bigcup C|=|A|+|B|+|C|-|A\bigcap B|-|B\bigcap C|-|C\bigcap A|+|A\bigcap B\bigcap C|$

加加减减，把重复的扣掉，再把扣多的加回来

（二）常见计数问题

1.排列问题：有n个不同的数，选k个排成一排，每个数最多选一次，问有多少种排列的方法？

分析：对于第一个位置，可以选n种数字，但是对于第二个位置，要扣除第一个位置上的数字，所以有$n-1$种选法，一次类推，根据乘法原理即为$A\binom{k}{n}=n!/(n-k)!$

2.组合问题：有n个不同的数，选出k个，顺序无关，问有多少种选择方法？

分析：已经知道如果需要排序的答案是$A\binom{k}{n}$,而每一次选出来的k个数也是不同，排列种数即为k个数中选择k个数并且排列的问题，为$A\binom{k}{k}$，这样答案即为$\frac{A\binom{k}{n}}{A\binom{k}{k}}$,即排列组合公式$C\binom{k}{n}$

3.二项式展开问题，求$(a+b)^n$展开式的各项系数。

分析：根据二项式定理$(a+b)^n=\sum_{k=0}^{n}C\binom{k}{n}*a^{n-k}*b^k$,于是只要求出各个$C\binom{k}{n}$即可。

4.有重复元素的全排列，k个元素，其中第i个元素有ni个,求全排列的个数？

分析：设答案为x，因为$n1+n2+n3+...+nk=n$,所以有$n1!*n2!*n3!*...*nk!*x=n!$，x可求。

5.可重复选择的组合，有n个不同元素，每个元素可以选多次，一共选k个元素，问优多少种选法？

分析：设第i个元素有xi个，那么就有$x1+x2+x3+...+xn=k$，求该式子的非负整数解个数，等于是将k个1随机分配给xi，可是有些xi可能一个都分不到，那么我们该怎么计算呢？令$yi=xi+1$,则有$y1+y2+y3+...+yn=k+n$,这样当yi=1时，xi=0，所以我们要将k+n个1，随机分配个yi,并且保证每个yi都至少分到一个。于是$C\binom{n-1}{k+n-1}$
即为 $C\binom{k}{k+n-1}$

6.单色三角形，给定空间里的n个点，其中没有三点共线，每两个点之间都用红色或者黑色线段连接。求三条边同色的三角形个数。

分析：从反面考虑，我们只需要求出非单色三角形的个数即可以求出单色三角形的个数，对于一个公共点的两个异色边来说，仅有唯一的单色三角形。所以对与每个顶点，有ai条边红色边，$n-1-ai$条黑色边，于是构成了$ai*(n-1-ai)$个异色三角形。于是总共有
$\frac{1}{2}\sum_{i=1}^{n}ai*(n-1-ai)$。

7. 给定 n 维空间的线段，问说线段经过几个格子

分析：将线段平移至原点，等价于(0, 0, 0)->(a, b, c)，ans = a + b + c +.. - gcd(a, b) - gcd(a, c) - .. + gcd(a, b, c) ...

（三）组合数性质

性质1: $C\binom{0}{n} = C\binom{n}{n}$

性质2: $C\binom{k}{n} = C\binom{n-k}{n}$

性质3: $C\binom{k}{n} + C\binom{k+1}{n} = C\binom{k+1}{n+1}$

性质4: $C\binom{k+1}{n} = C\binom{k}{n} * \frac{n-k}{k+1}$

lucas 定理：用来求解大数的组合数取模问题。

$$lucas(n,m,p) = lucas(n/p, m/p, p) * C\binom{m\%p}{n\%p}$$

$$lucas(n,0,p) = 1$$

```
/*****************************************************************************/
/*****************************Conbin-Number.cpp*******************************/
typedef long long type;
type conbin[maxc][maxc];
type pow_mod (type a, type n, type mod) {
     type ans = 1;
     while (n) {
          if (n&1)
               ans = (ans * a) % mod;
          a = (a * a) % mod;
          n >>= 1;
     }
     return ans;
}
type ConbinNum_One(type a, type b, type mod) {
     if (a < b) return 0;
     if (b > a - b)
          b = a - b;
     type up = 1, down = 1;
     for (type i = 0; i < b; i++) {
          up = up * (a-i) % mod;
          down = down * (i+1) % mod;
     }
     return up * pow_mod(down, mod-2, mod) % mod; // 逆元
}
void ConbinNum_Tab (int n, type mod) {
     for (int i = 0; i <= n; i++) {
          conbin[i][0] = conbin[i][i] = 1;
          for (int j = 1; j < i; j++)
               conbin[i][j] = (conbin[i-1][j-1] + conbin[i-1][j]) % mod;
     }
}
type lucas (type a, type b, type mod) {
     if (b == 0) return 1;
     return ConbinNum_One(a%mod, b%mod, mod) * lucas(a/mod, b/mod, mod) % mod;
}
/*****************************************************************************/
```

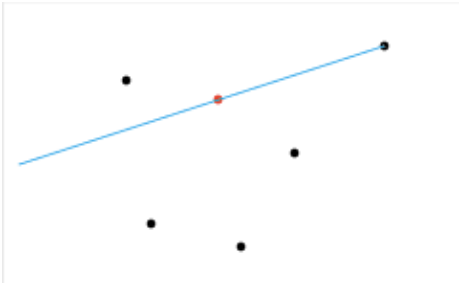
（四）典型问题

极角 r=atan2(y-o.y, x-o.x),有时需要通过将极角加上 2*pi 来保证周期性。

例题：Uva 11529（极角排序计数问题）

题目大意：给出若干个点，保证任意三点不共线，任意选三个点作为三角行，其他点若又在该三角形内，则算是该三角形内部的点，问所有情况的三角形平均每个三角形有多少个内部点。

解题思路：三角形的总数很容易求 C(n,3),现在就是要求各个三角形内部点的总数，同样我们可以反过来，求每个点在多少个三角形的内部。然后我们确定一个点，求该点在多少个三角的内部，剩余 n-1 个点，可以组成 C(n-1,3)个三角形，所以只要求出该点在哪些三角形的外部即可。



红色点为选中的点，将周围点按照与选中点的极角进行排序，每次枚举一点，它的极角为 a，所有极角小于 a+pi 的点，这些点组成的三角形，选中点一定在外部。处理一周的方式是将点的数组扩大两倍，将所有点的极角加上 pi 有保留在延长的数组中。

/**************************uva11529.cpp************************************/

```
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>
using namespace std;
const int N = 1205;
const double pi = 4 * atan(1.0);
const double eps = 1e-9;
int n;
double s, r[2*N];
struct point {
    double x, y;
}p[N];
double Count (int d) {
    int c = 0, mv = 0;
    for (int i = 0; i < n; i++) {
        if (i == d)    continue;
        double a = atan2(p[i].y-p[d].y, p[i].x-p[d].x);
        r[c] = a;
        r[c+n-1] = a + 2*pi;
        c++;
    }
    c = 2 * n - 2;
    sort(r, r + c);
    double ans = 0;
    for (int i = 0; i < n-1; i++) {
        double tmp = r[i] + pi;
        while (tmp > r[mv])
            mv++;
        double cnt = mv - i - 1;
        ans = ans + cnt * (cnt-1) / 2;
    }
    return s - ans;
}
double solve () {
    s = (n-1) * (n-2) * (n-3) / 6.0;
    double c = n * (n-1) * (n-2) / 6.0;
    double ans = 0;
    for (int i = 0; i < n; i++)
        ans += Count(i);
    return ans / c;
}
int main () {
    int cas = 1;
    while (scanf("%d", &n) == 1 && n) {
        for (int i = 0; i < n; i++)
            scanf("%lf%lf", &p[i].x, &p[i].y);
        printf("City %d: %.2lf\n", cas++, solve());
    }
    return 0;
}
```

/***/

例题：Codeforces 451E（容斥原理+隔板法）

题目大意：有 n 个花坛，要选 s 支花，每个花坛有 f[i]支花。同一个花坛的花颜色相同，不同花坛的花颜色不同，问说可以有多少种组合。

解题思路：2^n 的状态，枚举说哪些花坛的花取超过了，剩下的用 C(sum+n-1,n-1)隔板法计算个数，注意奇数的位置要用减的，偶数的位置用加的，容斥原理。

/**************************CF451E.cpp**************************************/

```
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>
using namespace std;
typedef long long ll;
ll pow_mod (ll a, ll k, ll p) {
    ll ans = 1;
    while (k) {
        if (k&1)
            ans = (ans * a) % p;
        a = (a * a) % p;
        k /= 2;
    }
    return ans;
}
ll C (ll a, ll b, ll p) {
    if (a < b)       return 0;
    if (b > a - b)   b = a - b;
    ll up = 1, down = 1;
    for (ll i = 0; i < b; i++) {
        up = up * (a-i) % p;
        down = down * (i+1) % p;
    }
    return up * pow_mod(down, p-2, p) % p; // 逆元
}
ll lucas (ll a, ll b, ll p) {
    if (b == 0)
        return 1;
    return C(a%p, b%p, p) * lucas(a/p, b/p, p) % p;
}
const int maxn = 25;
const ll mod = 1e9+7;
int n;
ll s, f[maxn];
ll solve () {
    ll ans = 0;
    for (int i = 0; i < (1<<n); i++) {
        ll sign = 1, sum = s;
        for (int j = 0; j < n; j++) {
            if (i&(1<<j)) {
                sum -= (f[j]+1);
                sign *= -1;
            }
        }
        if (sum < 0)  continue;
        ans += sign * lucas(sum + n - 1, n - 1, mod);
        ans %= mod;
    }
    return (ans + mod) % mod;
}
int main () {
    scanf("%d%lld", &n, &s);
    for (int i = 0; i < n; i++)
        scanf("%lld", &f[i]);
    printf("%lld\n", solve());
    return 0;
}
```

/***/

二、递推关系

1.斐波那契数列：$f(n)=f(n-1)+f(n-2), f(1)=f(2)=1$。

快速求解斐波那契第n项或是前n项和一般用借助矩阵快速幂。

2.卡特兰数：$h(n)=\sum_{i=0}^{n-1} h(i)*h(n-i-1), h(0)=h(1)=1$

另类递推公式：

- $h(n)=\frac{h(n-1)*(4*n-2)}{(n+1)}$
- $h(n)=\frac{C(2n,n)}{(n+1)}$
- $h(n)=C(2n,n)-C(2n,n+1)$

数列：1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790

应用：栈，凸三角形划分

3.村民排队：给出一个树的形状，现在为这棵树标号，保证根节点的标号值比子节点的标号值大，问有多少种标号树。

分析：s(i)表示以i为根节点的子树拥有的节点个数，$f(root)=\frac{s(root)!}{s(1)*s(2)*s(3)*\cdots*s(n)}$。

4.乱排：1~n的序列，问有多少种排列满足所有的i，第i个数不为i。$f(n)=(f(n-1)+f(n-2))*(n-1)$。

5.第一类斯特林数：有正负的，其绝对值是n个元素的项目分作k个环排列的方法数目。常用的表示方法有$s(n,k)$。

换个较生活化的说法，就是有n个人分成k组，每组内再按特定顺序围圈的分组方法的数目。

递推公式：$s(p,k)=(p-1)*s(p-1,k)+s(p-1,k-1)\ (1\le k\le p-1)$

边界条件：$s(p,0)=0\ (p\ge 1)\ s(p,p)=1\ (p\ge 0)$

递推关系的说明：

考虑第p个物品，p可以单独构成一个非空循环排列，这样前p-1种物品构成k-1个非空循环排列，方法数为s(p-1,k-1);

也可以前p-1种物品构成k个非空循环排列，而第p个物品插入第i个物品的左边，这有$(p-1)*s(p-1,k)$种方法。

6.第二类斯特林数：是n个元素的集定义k个等价类的方法数目。

递推公式：$S(n,k)=S(n-1,k-1)+k*S(n-1,k)$

边界条件：$S(n,n)=S(n,1)=1$

递推关系的说明：

考虑第p个物品，p可以单独构成一个非空集合，此时前p-1个物品构成k-1个非空的不可辨别的集合，方法数为S(p-1,k-1);

也可以前p-1种物品构成k个非空的不可辨别的集合，第p个物品放入任意一个中，这样有$k*S(p-1,k)$种方法。

三、数论

（一）素数

1、素数筛选法

```
/*******************************************************************************/
/*****************************prime-table.cpp*********************************/
/*****************************************
 * 埃拉托斯特尼素数筛选法
 * 筛选一定范围内的所有素数.
 * 复杂度近似于 o(n), 所以数据处理范围为 1e6-1e7 可以接受.
*****************************************/
void sieve(int n) {
     int m = (int)sqrt(n+0.5);
     memset(vis, 0, sizeof(vis));
     for (int i = 2; i <= m; i++) {
          if (!vis[i]) {
               for (int j = i * i; j <= n; j++)
                    vis[j] = 1;
          }
     }
}
int prime_table (int n, int* pri) { // 返回 n 以内素数个数。
     sieve(n);
     int c = 0;
     for (int i = 2; i <= n; i++)
          if (!vis[i])
               pri[c++] = i;
     return c;
}
/*******************************************************************************/
```

2、米勒-拉宾判素数

```
/*******************************************************************************/
/*****************************miller-rabin.cpp*********************************/
/*****************************************
 * 米勒-拉宾算法(miller-rabin)
 * 判断一个数是否为素数.
 *
 * 1. 费马小定理
 *   对于一个素数来说，a^(n-1) % n 恒等于 1 (1 <= a <= n-1)
 * 2. 快速幂取模
 * 3. 大数相乘（long long）取模
 *   两个 long long 型的数相乘有可能大于 long long 型的上限
 *
 * 随机生成一个 a, 判断 a^(n-1) % n 是否为 1, 如果不为 1 可以确定不是素数.
 * 对于合数来说通过测试的概率不足 25%.
 * 判断一定次数，使得该数为素数的概率趋近于 1.
 * 3215031751 需要测试较多次，否则会失准.
*****************************************/
#include <ctime> // possible need;
#include <cstdlib> // possible need;
typedef long long type;
type mul_mod (type a, type b, type mod) {
     type ret = 0;
     while (b) {
          if (b&1)  ret = (ret + a) % mod;
          a = (a + a) % mod;
          b >>= 1;
     }
     return ret;
}
```

```
type pow_mod (type a, type n, type mod) { // mod <= 1e18;
     type ret = 1;
     while (n) {
          if (n&1)
               ret = ret * a % mod;
          a = a * a % mod;
          n >>= 1;
     }
     return ret;
}
bool miller_rabin(type n) {
     if (n < 2)
          return false;

     srand(time(0));
     for (int i = 0; i < 20; i++)
          if (pow_mod(rand() % (n-1) + 1, n-1, n) != 1)
               return false;
     return true;
}
/*****************************************************************************/
```

3、大区间筛选素数

例题：Uva 1404（大区间筛选素数）

题目大意：如果 k 个相邻的素数 p1,p2,…,pk，满足 pk-p1=s，称这些素数组成一个距离为 s 的素数 k 元组，给定区间 a，b，求有多少个距离 s 的 k 元组。

解题思路：筛选素数法，先预处理出[1, sqrt(inf)]的素数表，然后对给定区间[a,b]根据预处理出的素数表筛选出素数即可。

```
/*********************************uva1404.cpp*****************************/
#include <cstdio>
#include <cmath>
#include <vector>
#include <algorithm>
using namespace std;
const int sqrt_inf = 46340;
const int maxn = 2 * 1e9;

int np, pri[sqrt_inf];
bool vis[maxn+5];
vector<int> vec;
void prime_table (int n) {
     np = 0;
     memset(vis, 0, sizeof(vis));
     for (int i = 2; i <= n; i++) {
          if (vis[i])
               continue;
          pri[np++] = i;
          for (int j = i * i; j <= n; j += i)
               vis[j] = 1;
     }
}
int solve () {
     int ret = 0;
     int a, b, s, k;
     vec.clear();
     memset(vis, 0, sizeof(vis));
     scanf("%d%d%d%d", &a, &b, &k, &s);
     for (int i = 0; i < np && pri[i] * pri[i] <= b; i++) {
          int u = pri[i], d = (u - a % u) % u;
          if (u == a + d)
               d += u;
          while (d <= b - a) {
               vis[d] = 1;
               d += u;
```

```
            }
        }
        for (int i = 0; i <= b-a; i++) {
            if (vis[i] == 0 && a + i > 1)
                vec.push_back(a+i);
        }
        for (int i = 0; i + k - 1 < vec.size(); i++) {
            if (vec[i+k-1] - vec[i] == s)
                ret++;
        }
        return ret;
}
int main () {
        prime_table(sqrt_inf);
        int cas;
        scanf("%d", &cas);
        while (cas--) {
            printf("%d\n", solve());
        }
        return 0;
}
/*******************************************************************************/
```

4、梅森素数

* 梅森素数，(2^k) - 1 形式的素数

* 性质：一个数若能差分成若干个不同的梅森素数的积，那么该数的所有因子和可以写成 2^n 形式。

* 梅森素数前 10 项（幂）：2，3，5，7，13，17，19，31，61，89…

5、反素数

反素数是一种素数，当它的数字反过来后，仍然是一个素数。
13, 17, 31, 37, 71, 73, 79, 97, 107, 113, 149, 157...

应用：

（1）分解质因子

方法 1：试除法

```
/*******************************************************************************/
/**************************normal-divfactor.cpp*********************************/
/*****************************************
 * 试除法分解因子
 * 将一个数分解成质因子.
 *
 * 1. 米勒-拉宾(特殊情况下的优化)
 *   判断一个数为素数
 *
 * 枚举 2~sqrt(n)的数，判断是否为 n 的因子.
 * 复杂度为 o(sqrt(n)).
*****************************************/
typedef long long type;
void div_factor (int& cnt, type* factor, type n) {
        cnt = 0;
        type m = (type)sqrt(n+0.5);
        for (type i = 2; i <= n && i <= m; i++) {
            if (n % i == 0) {
                while (n % i == 0) {
                    factor[cnt++] = i;
                    n /= i;
                }
            }
            /* 可以通过用 miller-rabin 算法判断进行优化 */
```

```
    }
    if (n != 1)
        factor[cnt++] = n;
}
/*****************************************************************************/
```

方法 2：费马方法

```
/*****************************************************************************/
/***************************feramat-divfactior.cpp****************************/
/*****************************************
 * 费马方法（feramat）
 * 将整数拆分成质因子.
 *
 * 1. 米勒-拉宾
 *   判断一个数是否为素数
 *
 * 一个整数 N, 可以分解成一个奇数上 2^k 次方,即 N = (2 * n + 1) * 2^k;
 * 令 M = 2 * n + 1, 如果 M 不为素数, 那么 M 肯定可以拆分成两个奇数相乘 M = c * d;
 * 假设 c >= d, 令 a = (c + d) / 2, b = (c - d) / 2;
 * 那么 M = a * a - b * b = 4 * (c + d) / 4;
 * 于是这要枚举 a, 保证 a * a - M 为完全平数即可.
 *
 * 对于因子为比较大的素数, 并且个数比较少的情况来说,复杂度仍比较高.
*****************************************/
#include <ctime> // possible need;
#include <cstdlib> // possible need;
typedef long long type;

type pow_mod (type a, type n, type mod) {
    type ans = 1;
    while (n) {
        if (n&1)
            ans = ans * a % mod;
        a = a * a % mod;
        n >>= 1;
    }
    return ans;
}
bool miller_rabin(type n) {
    if (n < 2)
        return false;
    srand(time(0));
    for (int i = 0; i < 20; i++)
        if (pow_mod(rand() % (n-1) + 1, n-1, n) != 1)
            return false;
    return true;
}
void feramat_factor (int& cnt, type* factor, type n) {
    if (miller_rabin(n)) {
        factor[cnt++] = n;
        return;
    }
    type x = (type) sqrt(n + 0.5);
    while (x < n) {
        type w = x * x - n;
        type y = (type) sqrt(w + 0.5);
        if (w == y * y) {
            feramat_factor(cnt, factor, x+y);
            feramat_factor(cnt, factor, x-y);
            break;
        }
        x++;
    }
}
```

```
void feramat_divfactor (int& cnt, type* factor, type n) { // N = (2*n+1) * 2^k;
     cnt = 0;
     if (n == 0)  return;
     while ((n&1) == 0) {
          factor[cnt++] = 2;
          n >>= 1;
     }
     if (n == 1)
          factor[cnt++] = n;
     else
          feramat_factor(cnt, factor, n);
}
/*****************************************************************************/
```

方法 3：pollard_rho 算法

```
/*****************************************************************************/
/*************************pollard-rho-divfactor.cpp*************************/
/*****************************************
 * pollard_rho
 * 将整数拆分成质因子.
 *
 * 1. 米勒-拉宾
 *   判断一个束是否为素数
 * 2. 欧几里得
 *   求两个数的最大公约数
 *
 * 根据一定规则生成 x, y, 求出 y-x 和 n 的最大公约数, 若不为 1 则为整数的一个因子.
*****************************************/
#include <ctime> // possible need;
#include <cstdlib> // possible need;
typedef long long type;

type gcd (type a, type b) {
     return b == 0 ? a : gcd(b, a%b);
}
type pow_mod (type a, type n, type mod) {
     type ans = 1;
     while (n) {
          if (n&1)
               ans = ans * a % mod;
          a = a * a % mod;
          n >>= 1;
     }
     return ans;
}
bool miller_rabin(type n) {
     if (n < 2)
          return false;
     srand(time(0));
     for (int i = 0; i < 20; i++)
          if (pow_mod(rand() % (n-1) + 1, n-1, n) != 1)
               return false;
     return true;
}
type pollard_rho(type n, type tmp) {
     int i = 1, k = 2;
     type x = rand() % n;
     type y = x;
     while(true) {
          i++;
          x = ( mul_mod(x, x, n) + tmp) % n;
          type d = gcd( (y > x ? y - x : x - y), n);
          if (d != 1 && d != n)
               return d;
```

```
            if (y == x)     return n;
            if (i == k) {
                y = x;
                k <<= 1;
            }
        }
    }
    void findfactor (int& cnt, type* factor, type n) {
        if(miller_rabin(n)) {
            factor[cnt++] = n;
            return;
        }
        type p = n;
        while(p >= n)
            p = pollard_rho(p, rand() % (n-1) + 1);
        findfactor (cnt, factor, p);
        findfactor (cnt, factor, n / p);
    }
    void pollard_divfactor (int& cnt, type* factor, type n) {
        cnt = 0;
        srand(time(0));
        findfactor(cnt, factor, n);
    }
    /*************************************************************************/
```

（二）欧几里得算法

```
/*************************************************************************************/
/*********************************gcd.cpp*****************************************/
/**********************
 * 欧几里得算法：求出 a 和 b 的最大公约数 d；
 *
 * 拓展欧几里得算法：求解线性方程的解
 * a * x + b * y = d;
 * 保证求出的解|x|+|y|最小。
 * d 为 a 和 b 的最大公约数，即当有线性方程 a * x + b * y = c，d = gcd(a,b)，c % d != 0 时，方程无解。
**********************/
type gcd(type a, type b) {
     return b == 0 ? a : gcd(b, a%b);
}

void exgcd(type a, type b, type& d, type& x, type& y) {
     if (!b)
          d = a, x = 1, y = 0;
     else {
          exgcd(b, a%b, d, y, x);
          y-= x * (a/b);
     }
}
/*************************************************************************************/
```

（三）欧拉函数

```
/*************************************************************************************/
/*************************************************************************************/
/**********************
 * 欧拉函数：phi(x)等于不超过 x 且和 x 互质的整数个数
 *
 * phi(n)= n * (1-1/p1) * (1-1/p2) * ... * (1-1/pn); (pi 为 n 的质因子)
**********************/
int euler_phi(int n) {
     int m = (int)sqrt(n+0.5);
     int ans = n;
     for (int i = 2; i <= m; i++) {
          if (n % i == 0) {
               ans = ans / i * (i-1);
               while (n%i==0) n /= i;
          }
     }
     if (n > 1)
          ans = ans / n * (n - 1);
     return ans;
}
void phi_table(int n, int* phi) {
     for (int i = 2; i <= n; i++)
          phi[i] = 0;
     phi[1] = 1;
     for (int i = 2; i <= n; i++) {
          if (!phi[i]) {
               for (int j = i*2; j <= n; j += i) {
                    if (!phi[j])
                         phi[j] = j;
                    phi[j] = phi[j] / i * (i - 1);
               }
          }
     }
}
/*************************************************************************************/
```

（四）模算术

1、取模快速幂

```
/*****************************************************************************/
/*****************************modular-equation.cpp*****************************/
/*****************************************
 * 模算术
 * (a * b) % mod = ((a % mod) * (b % mod)) % mod
*****************************************/
type mul_mod (type a, type b, type mod) { // 当 mod^2 > inf 时
     type ret = 0;
     while (b) {
          if (b&1)
               ret = (ret + a) % mod;
          a = (a + a) % mod;
          b >>= 1;
     }
     return ret;
}
type pow_mod (type a, type n, type mod) { // mod <= 1e18;
     type ret = 1;
     while (n) {
          if (n&1)
               ret = ret * a % mod;
          a = a * a % mod;
          n >>= 1;
     }
     return ret;
}
/*****************************************************************************/
```

2、逆元

```
/*****************************************************************************/
/*****************************inv.cpp*****************************************/
/*********************
 * 逆元：a * inv(a,mod) % mod = 1;
 * mod 为素数是才有逆元
*********************/
typedef long long type;
type inv (type a, type mod) {
     type d, x, y;
     exgcd(a, mod, d, x, y);
     return d == 1 ? (x+mod)%mod : -1;
}

type inv (type a, type mod) { // 费马小定理，a^(mod-1) % mod = 1;
     return pow_mod(a, mod-2, mod);
}
/*****************************************************************************/
```

（五）线性模方程

1、求线性模方程解

$a*x+b*y=c$

$exgcd => get(x',y',d)$ (d是a和b的最大公约数)

$if(c\%d \neq 0)$
无解

通解: $(a'=a/d, b'=b/d, c'=c/d)$

$$x=\frac{x'*c}{d}+\frac{b}{d}*t$$

$$y=\frac{x'*c}{d}-\frac{a}{d}*t$$

给定x范围(lx, rx),y范围(ly, ry)

推导: $lx<x<rx$

$=> lx-x'c'<b't<rx-x'c'$

$=> if(b'>0)$
$$\frac{lx-x'c'}{b'}<t<\frac{rx-x'c'}{b'}$$

$=> else$
$$\frac{rx-x'c'}{b'}<t<\frac{lx-x'c'}{b'}$$

结论:

$if(b'>0)$

$$ceil(\frac{lx-x'c'}{b'})\le t\le floor(\frac{rx-x'c'}{b'})$$

$else$

$$ceil(\frac{rx-x'c'}{b'})\le t\le floor(\frac{lx-x'c'}{b'})$$

$if(a'<0)$

$$ceil(\frac{ly-y'c'}{a'})\le t\le floor(\frac{ry-y'c'}{a'})$$

$else$

$$ceil(\frac{ry-y'c'}{a'})\le t\le floor(\frac{ly-y'c'}{a'})$$

2、中国剩余定理（孙子定理）

```
/*************************************************************************************/
/*********************************china.cpp*******************************************/
/**********************
 * 中国剩余定理（孙子定理）
 *
 * 在多个线性模方程下，x = ai % mi(保证 mi 和 mj 在 i≠的情况下互质)
 * 令 M = 所有 mi 的积，wi = M / mi，且 gcd(wi,mi)=1
 * 用拓展欧几里得求出 pi 和 qi，使得 wi*pi + mi*qi = 1
 * 令 ei = wi * pi，则方程组的解为 x = (e1*a1 + e2*a2 + ... + en*an) % M
 * 即在 M 的剩余系中 x 有唯一解
**********************/
typedef long long type;
```

```
void exgcd (type a, type b, type& d, type& x, type& y) {
     if (b == 0)
          d = a, x = 1, y = 0;
     else {
          exgcd(b, a%b, d, y, x);
          y -= (a/b) * x;
     }
}
type china (type* a, type* m, int n) {
     type Mi = 1;
     for (int i = 0; i < n; i++)
          Mi *= m[i];

     type ret = 0;
     for (int i = 0; i < n; i++) {
          type w = Mi/m[i], d, x, y;
          exgcd(m[i], w, d, x, y);

          ret = (ret + y*w*a[i]) % Mi;
     }
     return (ret % Mi + Mi) % Mi;
}
/*****************************************************************************/
```

3、离散对数（大步小步算法）

```
/*****************************************************************************/
/*****************************************************************************/
/**********************
 * 离散对数（大步小步算法）
 * 求解 a^x = b % mod（mod 为素数）
 *
 * 根据欧拉定理，只需要检查 x=0,1..mod-1 是不是解即可（a^(mod-1) % mod = 1）
 * 算法：先检查前 m 项，m 取 sqrt(mod)，并且计算出 a^m 的逆 a^-m
 * 然后考虑 m+1 ~ 2m 项，假设存在 ei * a^m = b % mod，两边同乘 a^-m 得 ei = b' % mod
 * 所以每次将 b 乘以 a^-m 之后在 1~m 中查找即可（map 优化）
 * 之所以 m 取 sqrt(mod)是因为在此时复杂度最低
**********************/
#include <map>
using namespace std;
typedef long long type;

type mul_mod (type a, type b, type mod) {
     return (type)a * b % mod;
}
type pow_mod (type a, type n, type mod) {
     type ans = 1;
     while (n) {
          if (n&1)
               ans = mul_mod(ans, a, mod);
          a = mul_mod(a, a, mod);
          n /= 2;
     }
     return ans;
}

void exgcd (type a, type b, type& d, type& x, type& y) {
     if (!b)
          d = a, x = 1, y = 0;
     else {
          exgcd (b, a%b, d, y, x);
          y -= x * (a/b);
     }
}
```

```
type inv (type a, type mod) {
     type d, x, y;
     exgcd(a, mod, d, x, y);
     return d == 1 ? (x+mod)% mod : -1;
}

int log_mod (type a, type b, type mod) {
     type m = (type)sqrt(mod+0.5), v, e = 1;
     v = inv(pow_mod(a, m, mod), mod); //  计算 a^(-m)
     map<type, type> g;

     g[1] = 0;
     for (int i = 1; i < m; i++) {
          e = mul_mod(e, a, mod);
          if (!g.count(e)) // 记录 e^i
               g[e] = i;
     }

     for (int i = 0; i < m; i++) {
          if (g.count(b))
               return i*m+g[b];
          b = mul_mod(b, v, mod);
     }
     return -1;
}
/*****************************************************************************/
```

（六）其他定理

1、数论四大定理（威尔逊定理，欧拉定理，孙子定理，费马小定理）

威尔逊定理：

若 p 是质数，则 p 可以整除（p-1）!+1

欧拉定理：

也称费马-欧拉定理，若有 n，a 为正整数，且 n，a 互质

a^phi(n) = 1 (mod n)

孙子定理：

也称中国剩余定理，见文档 16 页

费马小定理：

假如 p 是质数，且(a,p)=1，那么 a^(p-1) ≡1（mod p） 。

2、勒让德记号

对于整数 a 和**奇素数** p，tag = a^((p-1)/2) (mod p)

- tag = 0 时，a = 0 (mod p)
- tag = 1 时，存在某个整数 x，x^2 = a (mod p)
- tag = -1 时，不存在某个整数 x，x^2 = a (mod p)

3、凯莱定理（cayley）

有 n 个标志节点的树的数目等于 n^(n-2)（仅是 cayley 在组合数学中的应用）

4、牛顿插值法

给出一个序列的 n 项，求满足序列规律的后 c 项，要求尽量小

每次求两项之间的差，形成一个新的序列，直到序列公差为 0 时，回带回原来的序列

5、完美洗牌

给定 n 张牌，从 1~n，每次将 1~n/2 分别放到 2x 的位置，n/2+1~n 的牌放到(x-n/2) * 2 - 1 的位置

等价于每次将第 x 张牌放到 2x %(2p+1)的位置上去

现在要判断洗 n-1 次牌是否使得牌变成原先序列，只要判断 2^(n-1) % (2p+1) = 1

四、博弈

问题特征：

1. 游戏两人轮流操作，并且都以最优方式决策
2. 无法操作时为游戏终止状态
3. 游戏中的同一状态不会重复出现，也不会出现平局
4. 任意状态的决策集合只与当前状态有关，与决策者无关

问题层次：

- 对于给定的初始状态判断胜负
- 寻找单回合的必胜选择
- 寻找多回合的必胜选择（交互题）

解决方法：

- 拓扑逆序倒推
- 记忆化搜索

NP状态定理:

N状态为当前操作玩家有必胜策略（先手必胜），P状态则为当前操作玩家没有必胜策略（先手必败）；后继状态，由当前状态可以转移到的所有状态都成为当前状态的后继状态。

P状态的所有后继均为N状态，N状态的后继状态中一定存在P状态。

典型问题：

（一）取石子游戏

1. 巴什博弈:

- 问题描述：n个石子，两人轮流取石子，规定每次至少取1个，至多取m个，不能操作者输；
- 解题结论：$\mathbf{n\%(m+1)=0}$时为P点，否则为N点；

> 结论证明:
>
> - 当$n=0$时，终止状态，为P点；
> - 当$n\%(m+1)=0$时，$n=k*(m+1)$，因为每次操作最多取m个石子，假设选取了x个，则有$n=k*(m+1)>n-x>(k-1)*(m+1)$，即$(n-x)\%(m+1)\neq 0$；后继状态均为N点，所以n为P点；
> - 当$n\%(m+1)\neq 0$时，令$x=n\%(m+1)$，则有$m\geq x>0$，那么当前操作者取走x个石子后，使得后继状态为P点，所以当前状态为N点；

- 问题变形：仍是n个石子，两人轮流取石子，规定每次取的个数为给定集合set中的元素；
- 解题方案：类似01背包的处理方式，所有初始状态为P，从终止状态0开始确定，将可以到达确定P点的状态置为N点；

2. 威佐夫博弈:

- 问题描述：两堆石子，分别为n和m个，两人轮流操作，操作分两种，一种为从单堆石子中取任意数量的石子，一种为从两堆石子中取相同数量的石子，最后不能操作的人输；
- 解题结论：寻找奇异局势，即对应的P点，设（a_k,b_k）为P点，并且保证$a_k\leq b_k$，$\mathbf{b_k=a_k+k}$（$\mathbf{a_k}$的确定方式为在集合b中未出现过的最小自然数，$\mathbf{a_0=0}$且$\mathbf{b_0=0}$）；

> 结论证明:
>
> - 首先首先奇异局势对应的即为P点，并且奇异局势具有性质：任何自然数都包含在一个且仅有一个奇异局势中；
> - 对于一个非奇异局势状态（A，B），有$A\leq B$，找到k，使得$a_k=A$；如果$b_k<B$，那么取走B中若干石子，使得$b_k=B$，变为奇异局势；如果$b_k>B$，那么令$t=B-A$，同时取走两堆石子中a_k-a_t个石子，那么状态变成（$a_t,B-A+a_t$）=>（a_t,a_t+t）=>（a_t，b_t），变为奇异局势；
> - 对于一个奇异局势状态（a_k,b_k），修改a_k，b_k中的一个都将不会是奇异局势，因为一个自然数只会出现在一个奇异局势状态中；同时修改a_k,b_k，减少x，令$a_t=a_k-x$，那么$b_k-x=a_k+k-x=a_t+k\neq a_t+t=b_t$，所以奇异局势不可能转移到另一个奇异局势；

- 问题变形：第二种操作变为从两堆石子中取的石子数差的绝对值不超过x，其它条件相同；
- 解题方案：仍然是确定奇异局势，公式转变为递推式：$\mathbf{b_k=b_{k-1}+a_k-a_{k-1}+x+1}$

> 奇异局势公式:
>
> 对于普通的威佐夫游戏，$a_k=k*\frac{1+\sqrt{5}}{2}$，所以对于给定状态x，y，可以通过令$k=y-x$，然后根据公式求出$a_k$，判断是否和x相等

3. 尼姆博弈：

- 问题描述：三堆石子，分别又x，y，z个石子，两个人轮流操作，规定每次操作可以从一堆石子中取走一个以上的石子数，不能操作的人为输；
- 解题结论：如果**x^y^z=0**，则为P点，否则为N点；

> 结论证明：
>
> - x^y^z ≠ 0时，那么假设S=x^y^z，S的二进制表示最左边的1在第k位，那么以一定存在该位为的堆，假设该堆有x个石子，那么取走一定个数使得x=y^z，那么此时一定有x^y^z=0；
> - x^y^z = 0时，由于只能修改一堆石子，所以不管修改对应二进制的哪一位，都将导致x^y^z ≠ 0；

- 问题变形：n堆石子，给定每堆石子的个数，其它条件不变；
- 解题方案：判断Nim和是否为0即可；

4. anti-nim（反尼姆）：

- 问题描述：N堆石子，游戏者轮流取石子；每次只能从一堆中取出任意数目的石子，至少为1；取走最后一个石子的为输；
- 解题结论：**1. 所有堆石子数都为1且游戏的SG值为0； 2. 存在堆石子数大于1且SG不为0**

5. K 倍动态减法

问题描述：N 个石子，，两人轮流操作取石子，取的石子数不能超过对手上一次取的石子数 m 的 K 倍，取到最后一个石子的人胜利，第一次取只能取 1~N-1 个石子。先手必胜时输出最小的首次操作。

解题结论：构造数列，将 N 写成数列中一些项的和，使得这些被取到的项的相邻两个倍数差距>K

例题：Uva 1567（K 倍动态减法）

/*****************************uva1567.cpp*************************************/

```
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int maxn = 1e6+5;

int N, K, a[maxn], b[maxn];
int main () {
     int cas;
     scanf("%d", &cas);
     for (int i = 1; i <= cas; i++) {
          scanf("%d%d", &N, &K);
          int p = 0, q = 0;
          a[0] = b[0] = 0;
          while (a[p] < N) {
               a[p+1] = b[p] + 1;
               p++;
               while (a[q + 1] * K < a[p])
                    q++;
               b[p] = b[q] + a[p];
          }
          printf("Case %d: ", i);
          if (N == a[p])
               printf("lose\n");
          else {
               int ans;
               while (N) {
                    if (N >= a[p]) {
                         N -= a[p];
                         ans = a[p];
                    }
                    p--;
               }
               printf("%d\n", ans);
          }
     }
     return 0;
}
```

/***/

6. 一些题目结论

uva 12293

> 题目大意：有两个盒子，第一个盒子装有n个球，第二个盒子装又1个球，每次操作将少的盒子中的球全部拿掉，并从另一个盒子中取一些球放入该盒子，使另一个盒子中球的个数为0者为败。
>
> 解题结论：n如果为2^{i-1}那么先手必败。

uva 11892

> 题目大意：给定n堆石子的个数，两人轮流选择石子堆取石子，直到不能取为失败，附加条件，如果前一次操作，即队手的操作，没有将选中石子堆中的石子取完，那么当前操作者必须在该堆中取石子。
>
> 解题结论：只要有一个石子堆的个数大于2，那么先手就获得必胜态。所以只要考虑谁先碰到非1石子堆。对于全是1的情况判断奇偶性。

hdu 4111

> 题目大意；Alice和Bob两个玩游戏，有N堆石子，每次可以从一堆中取走一个石子，或者是合并两堆石子。
>
> 解题结论：统计1的个数c，以及非1情况下的步数s，包括合并。
>
> 1.c为奇数，s不等于2：那么先手必胜。
>
> 2.s为2或者为0：c为3的倍数是先手必败。
>
> 3.否则的话，s为奇数时先手必胜。

（二）删边游戏

例题： Uva 12033（树形删边游戏）

题目大意：给定图，以 0 为根节点，每条边有一个长度，两个人轮流操作，每次为一条边上色，上一个单位长度，当一条边的颜色被涂满，则算作是减掉整段子树。判断先手是否必胜。

解题思路：SG 定理，对于当前节点 u，每次考虑字节点 v，u-v 边的长度为 l

当 l 为 1 时：sg(u) ^= (sg(v) + 1)

当 l 为奇数时： 需要判断 sg(v)奇偶性，奇数-1，偶数+1；

当 l 为偶数时：sg(u) ^= sg(v)

```
/*******************************uva12033.cpp*********************************/
#include <cstdio>
#include <cstring>
#include <vector>
#include <algorithm>

using namespace std;
const int maxn = 1005;
int N, W[maxn][maxn];
vector<int> g[maxn];

int dfs (int u, int p) {
     int ret = 0;

     for (int i = 0; i < g[u].size(); i++) {
          int& v = g[u][i];
          if (v != p) {
               int sg = dfs(v, u);
               if (W[u][v] == 1)
                    ret ^= (sg+1);
               else if (W[u][v]&1)
                    ret ^= (sg + (sg&1 ? -1 : 1));
               else
                    ret ^= sg;
          }
     }
     return ret;
}

int main () {
     int cas, u, v, w;
     scanf("%d", &cas);
     for (int k = 1; k <= cas; k++) {
          scanf("%d", &N);
          for (int i = 0; i < N; i++)
               g[i].clear();
          for (int i = 1; i < N; i++) {
               scanf("%d%d%d", &u, &v, &w);
               g[u].push_back(v);
               g[v].push_back(u);
               W[u][v] = W[v][u] = w;
          }
          printf("Case %d: %s\n", k, dfs(0, -1) ? "Emily" : "Jolly");
     }
     return 0;
}
/******************************************************************************/
```

（三）图上移动石子

例题： Uva 12163（有向图移动石子）

题目大意:两个人进行游戏，对于每一局有一个无向图，给出无向图，每个节点有个 K 值，两人轮流操作，每次可以选中国一个含有石子的节点,将该节点的一个石子拿掉,然后选择 K 个有边连接的节点加上一个石子(节点可以重复选择)，每个节点的子节点不会超过 15 个。不能操作的人视为失败。每局有 n 轮，给定每轮中每个节点上石子的初始值，问先手胜利还是失败。

解题思路：有向图上移动石子的组合游戏，对于没有子节点的节点 SG 值为 0，然后对于每个节点，用记忆化的方式处理出 SG 值，注意因为要选中 K 个节点，但是子节点的个数最多为 15，然后对于选中偶数次的节点可视没选，所以枚举 215 状态即可，并且要保证选中的个数奇偶性和 K 值相同。

对于每一轮给定的初始状态，计算各个子游戏的 Nim 和即可。

/**********uva12163.cpp**********/

```
#include <cstdio>
#include <cstring>
#include <vector>
#include <map>
#include <algorithm>
using namespace std;
const int maxn = 105;
vector<int> g[maxn];
int N, M, s[maxn], val[maxn];

inline int bitcount (int x) {
     return x == 0 ? x : bitcount(x>>1) + (x&1);
}
int SG (int x) {
     if (s[x] != -1)
          return s[x];
     map<int, int> vis;
     int n = g[x].size();
     if (n == 0)
          return s[x] == 0;
     for (int i = 0; i < (1<<n); i++) {
          int bit = bitcount(i);
          if (bit > val[x] || (bit&1) != (val[x]&1))
               continue;

          int tmp = 0;
          for (int j = 0; j < n; j++) {
               if (i&(1<<j))
                    tmp ^= SG(g[x][j]);
          }
          vis[tmp] = 1;
     }
     int ret = -1;
     while (vis.count(++ret));
     return s[x] = ret;
}

void init () {
     scanf("%d%d", &N, &M);
     for (int i = 0; i < maxn; i++)
          g[i].clear();
     int u, v;
     for (int i = 0; i < M; i++) {
          scanf("%d%d", &u, &v);
          g[u].push_back(v);
     }
     for (int i = 0; i < N; i++)
          scanf("%d", &val[i]);

     memset(s, -1, sizeof(s));
     for (int i = 0; i < N; i++)
          s[i] = SG(i);
}
```

```
void solve () {
     int Q;
     scanf("%d", &Q);

     for (int i = 1; i <= Q; i++) {
          int ret = 0, x;
          for (int j = 0; j < N; j++) {
               scanf("%d", &x);
               if (x&1)
                    ret ^= s[j];
          }
          printf("Round#%d: %s\n", i, ret ? "WINNING" : "LOSING");
     }
     printf("\n");
}

int main () {
     int cas;
     scanf("%d", &cas);
     for (int k = 1; k <= cas; k++) {
          init();

          printf("Game#%d:\n", k);
          solve();
     }
     return 0;
}
/*********************************************************************************/
```