Chapter 2 Mathematics



Keshuai

2015-11-16

Copyright ：2.0

目录

一．计数问题3

（一）基本计数方法 3

（二）常见计数问题 3

（三）组合数性质 4

（四）典型问题 4

例题：Uva 11529（极角排序计数问题）4

例题：Codeforces 451E（容斥原理+隔板法）6

二．递推关系7

三．数论8

（一）素数8

1、素数筛选法8

2、米勒-拉宾判素数8

3、大区间筛选素数9

例题：uva 1040（大区间筛选素数）9

4、梅森素数10

5、反素数10

应用：分解质因子10

（二）欧几里得算法14

（三）欧拉函数14

（四）模算术15

1、取模快速幂15

2、逆元15

（五）线性模方程16

1、求线性模方程解16

2、中国剩余定理（孙子定理）16

3、离散对数（大步小步算法）17

（六）其他定理19

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)19

2、勒让德记号19

3、凯莱定理(cayley)19

4、牛顿插值法19

5、完美洗牌19

四．博弈20

（一）取石子游戏20

例题：uva 1567（k倍动态减法）21

（二）删边游戏23

（三）图上移动石子24

例题：uva 12163（有向图移动石子）24

五．其他20

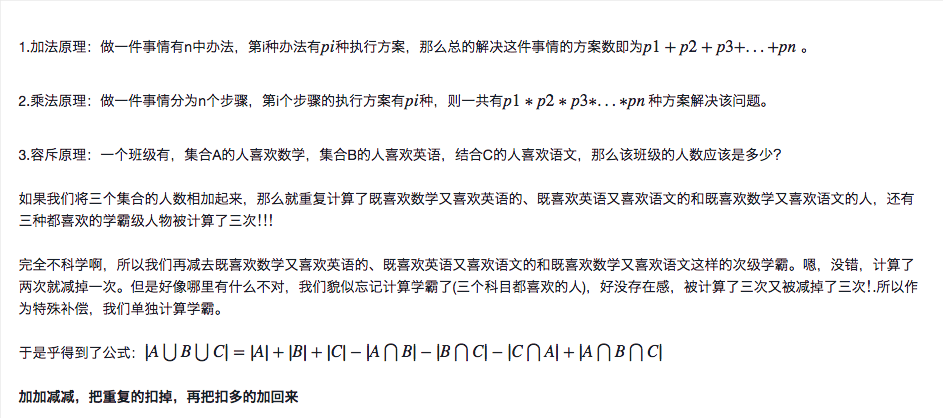
（一）矩阵快速幂20

（二）高斯消元20

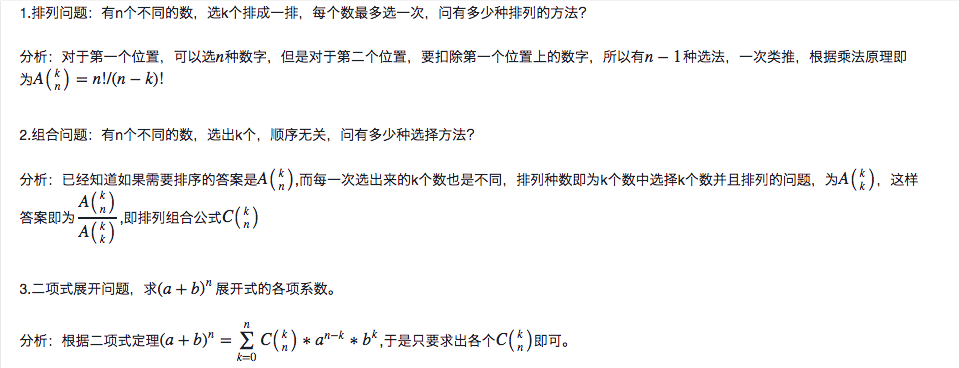
（三）数值方法20

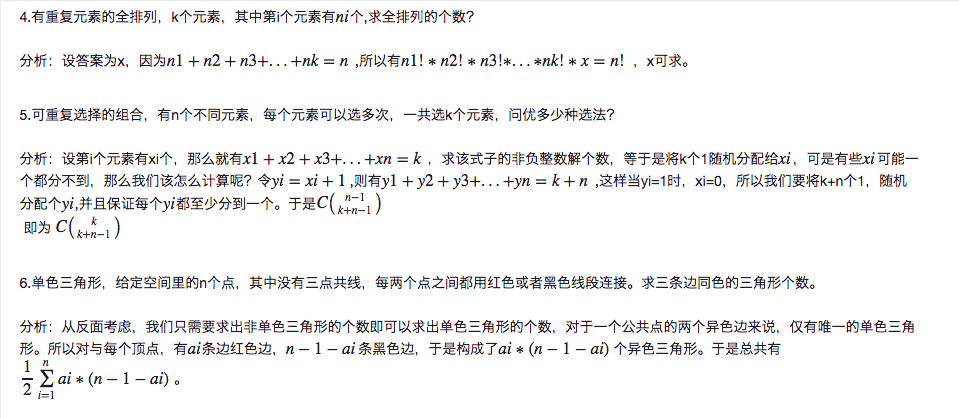
一、计数问题

（一）基本计数方法



（二）常见计数问题





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



lucas定理：用来求解大数的组合数取模问题。



/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*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;

}

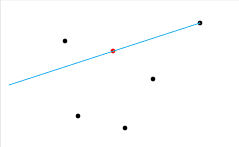
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

（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]);

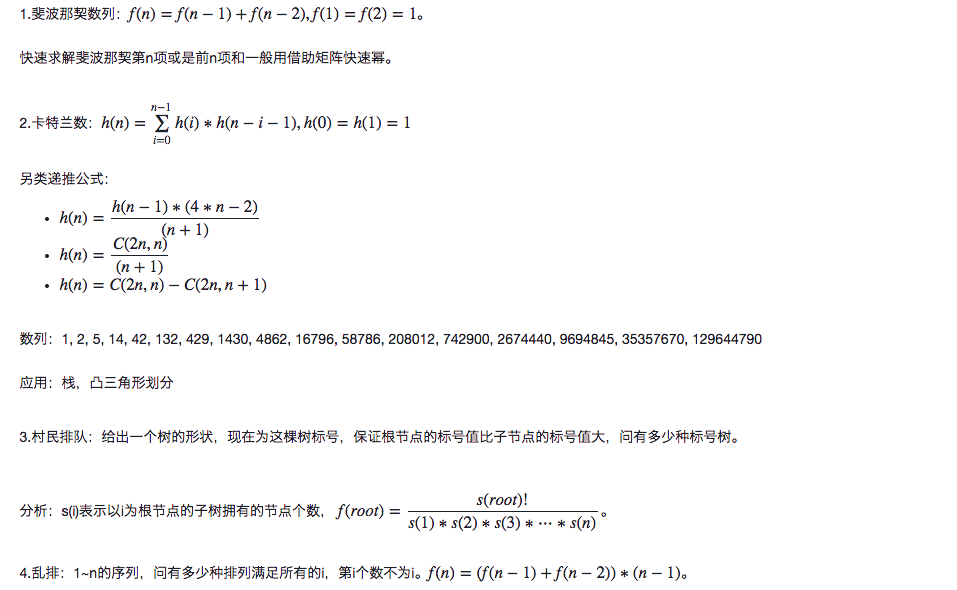
printf("%lld\n", solve());

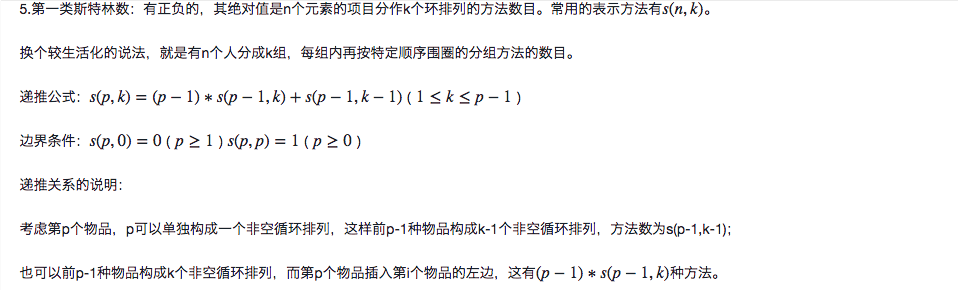
return 0;

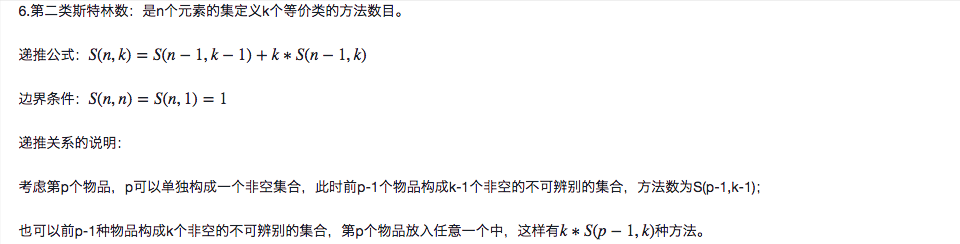
}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

二、递推关系







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](http://baike.baidu.com/view/40117.htm" \t "_blank), [17](http://baike.baidu.com/view/521455.htm" \t "_blank), [31](http://baike.baidu.com/view/521968.htm" \t "_blank), [37](http://baike.baidu.com/view/249918.htm" \t "_blank), [71](http://baike.baidu.com/view/522169.htm" \t "_blank), [73](http://baike.baidu.com/view/522474.htm" \t "_blank), [79](http://baike.baidu.com/view/522182.htm" \t "_blank), [97](http://baike.baidu.com/view/522168.htm" \t "_blank), [107](http://baike.baidu.com/view/521291.htm" \t "_blank), [113](http://baike.baidu.com/view/604199.htm" \t "_blank), [149](http://baike.baidu.com/view/625925.htm" \t "_blank), [157](http://baike.baidu.com/view/1153039.htm" \t "_blank)...

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;

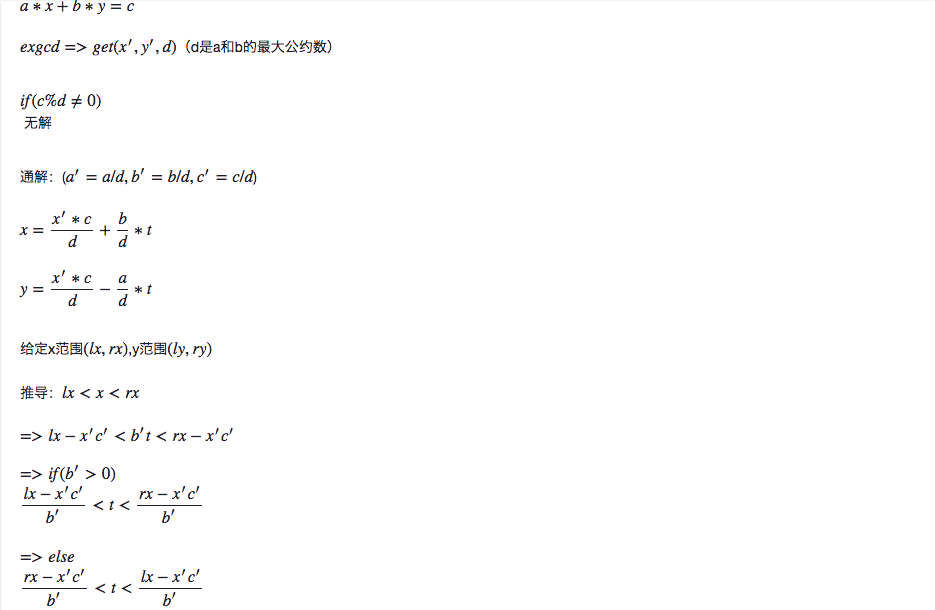
return pow\_mod(a, mod-2, mod);

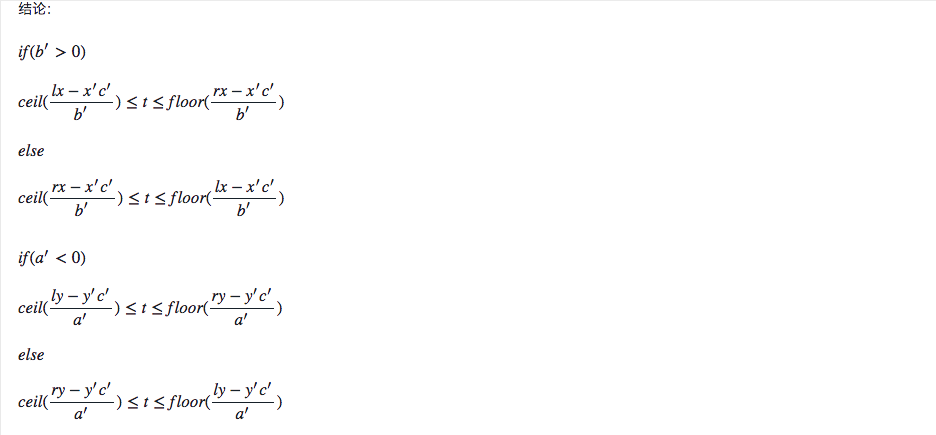
}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

（五）线性模方程

1、求线性模方程解





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：**

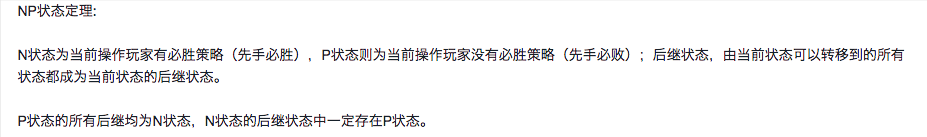
1. 游戏两人轮流操作，并且都以最优方式决策
2. 无法操作时为游戏终止状态
3. 游戏中的同一状态不会重复出现，也不会出现平局
4. 任意状态的决策集合只与当前状态有关，与决策者无关

**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）

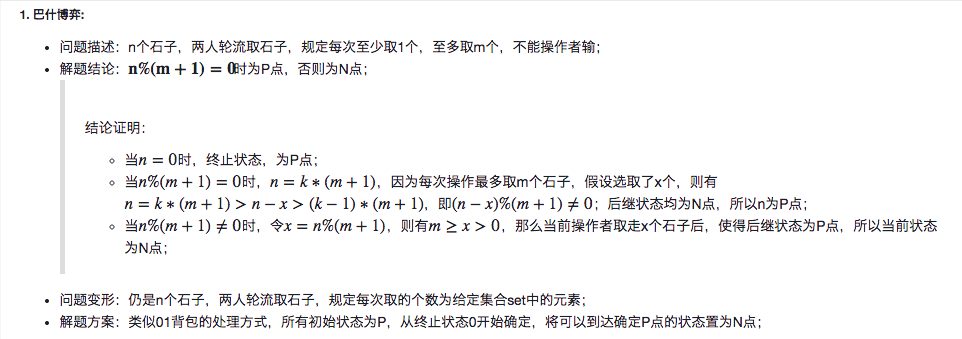
**解决方法：**

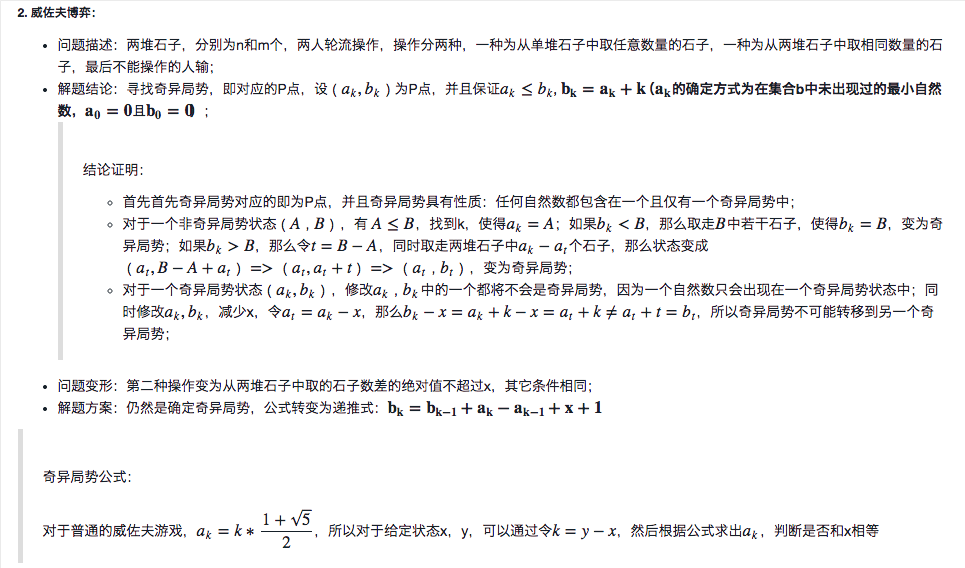
* 拓扑逆序倒推
* 记忆化搜索

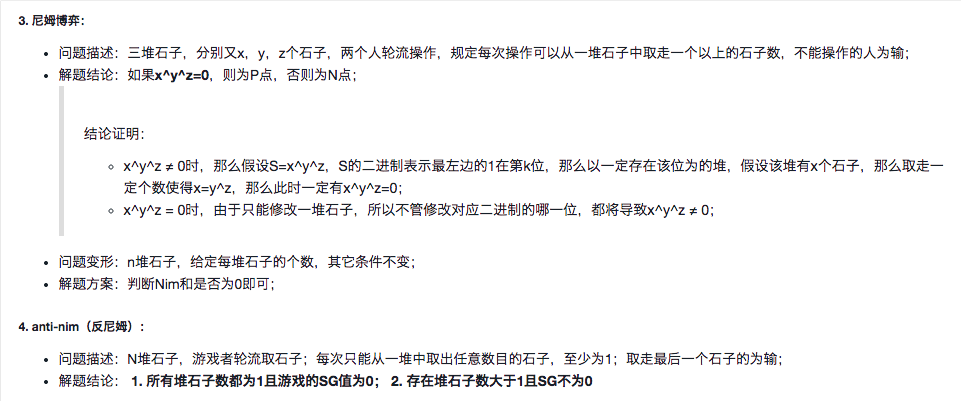


**典型问题：**

1. 取石子游戏







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);

}

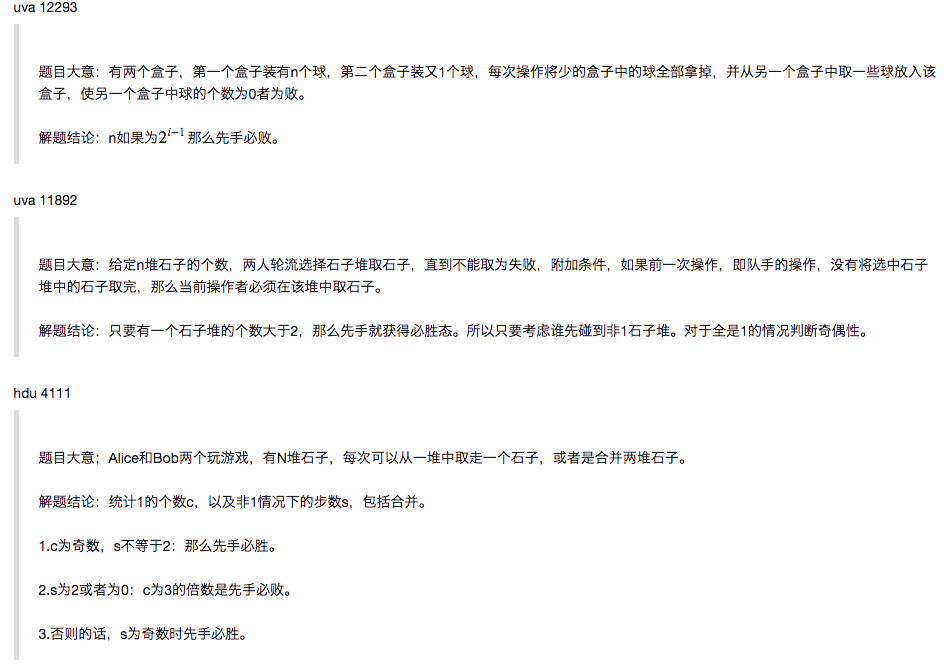
}

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

6. 一些题目结论



1. 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;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

1. 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;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

五、其他

1. 矩阵快速幂

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*pow\_mat.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

const int maxn = 105;

const int mod = 1e9 + 7;

struct Mat {

int r, c, s[maxn][maxn];

void init(int r, int c) {

this->r = r;

this->c = c;

memset(s, 0, sizeof(s));

}

}tmp;

void mul(const Mat& a, const Mat& b, Mat& c) {

tmp.init(a.r, b.c);

for (int i = 0; i < tmp.r; i++) {

for (int j = 0; j < tmp.c; j++)

for (int k = 0; k < a.c; k++)

tmp.s[i][j] = (tmp.s[i][j] + 1LL \* a.s[i][k] \* b.s[k][j] % mod) % mod;

}

c = tmp;

}

Mat pow\_mat(Mat ret, Mat x, int n) {

while (n) {

if (n&1) mul(x, ret, ret);

mul(x, x, x);

n >>= 1;

}

return ret;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

1. 高斯消元

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*guass\_elimination.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <cmath>

#include <algorithm>

using namespace std;

const int maxn = 30;

const double eps = 1e-9;

typedef double Mat[maxn+5][maxn+5];

void gauss\_elimination (Mat a, int n) {

for (int i = 0; i < n; i++) {

int r = i;

for (int j = i + 1; j < n; j++)

if (fabs(a[j][i]) > fabs(a[r][i]))

r = j;

if (r != i) {

for (int j = 0; j <= n; j++)

swap(a[r][j], a[i][j]);

}

if (fabs(a[i][i]) < 1e-9)

continue;

for (int k = i + 1; k < n; k++) {

double f = a[k][i] / a[i][i];

for (int j = 0; j <= n; j++)

a[k][j] -= a[i][j] \* f;

}

}

for (int i = n-1; i >= 0; i--) {

for (int j = i+1; j < n; j++)

a[i][n] -= a[j][j] \* a[i][j];

a[i][i] = a[i][n] / a[i][i];

if (fabs(a[i][i]) < 1e-9)

a[i][i] = 0;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

1. 数值方法

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Simpson.cpp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cstdio>

#include <cstring>

#include <cmath>

#include <algorithm>

using namespace std;

double F(double x) { return x; }

double simpson(double a, double b) {

double c = (b + a) / 2;

return (F(a) + 4 \* F(c) + F(b)) \* (b - a) / 6;

}

double asr(double a, double b, double ep, double A) {

double c = (a + b) / 2;

double L = simpson(a, c), R = simpson(c, b);

if (fabs(L + R - A) <= 15 \* ep) return L + R + (L + R - A) / 15;

return asr(a, c, ep/2, L) + asr(c, b, ep/2, R);

}

double asr(double a, double b, double ep) {

return asr(a, b, ep, simpson(a, b));

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/