SaoZhu Team

Code Library

East China Normal University

Chen WeiWen - Software Engineering

Cao ZhiJie - Computer Science

Zhu XuLiang – Mathematics

目录

http://blog.csdn.net/cww97/article/details/76615457

常用 STL

map 的 Upperbound

```
map<int,int>::iterator se = mp.upper_bound(mid);
返回迭代器
```

优先队列

```
priority_queue<int>Q;//采用默认优先级构造队列
priority_queue<int,vector<int>,cmp1>que1;//最小值优先
priority_queue<int,vector<int>,cmp2>que2;//最大值优先

Q.push(x);
int x = Q.top(); Q.pop();
```

multiset

```
begin()         返回指向第一个元素的迭代器
clear()         清除所有元素
count()         返回某个值元素的个数
empty()         如果集合为空，返回 true
end()           返回指向最后一个元素的迭代器
erase()         删除集合中的元素 ( 参数是一个元素值，或者迭代器)
find()          返回一个指向被查找到元素的迭代器
insert()        在集合中插入元素
size()          集合中元素的数目
lower_bound()   返回指向大于（或等于）某值的第一个元素的迭代器
upper_bound()   返回大于某个值元素的迭代器
equal_range()   返回集合中与给定值相等的上下限的两个迭代器

multiset <point> po;
multiset <point>::iterator L, R, it;
```

离散化 lower_lound

```
sort(a + 1, a + A+1);
A = unique(a + 1, a + A+1) - (a + 1);
for (int i = 1; i <= n; i++){ // segtree
    int L = lower_bound(a+1, a+A+1, l[i]) - a;
    int R = lower_bound(a+1, a+A+1, r[i]) - a;
    T.update(L, R, i, 1, T.M+1,1);
}
------------use in ChairTree-------------------
for (int i = 1; i <= n; i++) {
    scanf("%d", &arr[i]);
    Rank[i] = arr[i];
}
sort(Rank + 1, Rank + n+1);//Rank 存储原值
int m = unique(Rank + 1, Rank + n +1) - (Rank + 1);//这个 m 很重要，WA 一天系列
for (int i = 1; i <= n; i++) {//离散化后的数组，仅仅用来更新
    arr[i] = lower_bound(Rank + 1, Rank + m+1, arr[i]) - Rank;
}
==================CZJ 排序去重==================
sort(vecs.begin(), vecs.end());
vecs.resize(unique(vecs.begin(), vecs.end()) - vecs.begin());
```

bitset

```
构造函数

bitset<n> b;

b 有 n 位，每位都为 0.参数 n 可以为一个表达式.
如 bitset<5> b0;则"b0"为"00000";

bitset<n> b(unsigned long u);
b 有 n 位,并用 u 赋值;如果 u 超过 n 位,则顶端被截除
如:bitset<5>b0(5);则"b0"为"00101";

bitset<n> b(string s);
b 是 string 对象 s 中含有的位串的副本
string bitval ( "10011" );
bitset<5> b0 ( bitval4 );
则"b0"为"10011";
```

```
bitset<n> b(s, pos);
b 是 s 中从位置 pos 开始位的副本,前面的多余位自动填充 0;
string bitval ("01011010");
bitset<10> b0 ( bitval5, 3 );
则"b0" 为 "0000011010";

bitset<n> b(s, pos, num);
b 是 s 中从位置 pos 开始的 num 个位的副本,如果 num<n,则前面的空位自动填充 0;
string bitval ("11110011011");
bitset<6> b0 ( bitval5, 3, 6 );
则"b0" 为 "100110";

bool any()       是否存在置为 1 的二进制位? 和 none()相反
bool none()      是否不存在置为 1 的二进制位,即全部为 0? 和 any()相反.
size_t count()   二进制位为 1 的个数.
size_t size()    二进制位的个数
flip()           把所有二进制位逐位取反
flip(size_t pos)把在 pos 处的二进制位取反
bool operator[]( size_type _Pos )获取在 pos 处的二进制位
set()            把所有二进制位都置为 1
set(pos)         把在 pos 处的二进制位置为 1
reset()          把所有二进制位都置为 0
reset(pos)       把在 pos 处的二进制位置为 0
test(size_t pos)在 pos 处的二进制位是否为 1?
unsigned long to_ulong( )用同样的二进制位返回一个 unsigned long 值
string to_string ()返回对应的字符串.
```

线性代数

矩阵快速幂

```
#include<cstdio>
#include<cstring>
#include<iostream>
using namespace std;
typedef long long LL;
const LL MOD = 1000000007LL;
struct matrix{
    static const int MATRIX_N = 11;
    LL a[MATRIX_N][MATRIX_N];
    int row, col;
    matrix():row(MATRIX_N),col(MATRIX_N){memset(a,0,sizeof(a));}
    matrix(int x, int y):row(x),col(y){memset(a,0,sizeof(a));}
    LL* operator [] (int x){return a[x];}
    matrix operator * (matrix x){
        matrix tmp(col, x.row);
        for(int i = 0; i < row; i++)
            for(int j = 0; j < col; j++) if(a[i][j])//稀疏矩阵优化
                for(int k = 0; k < x.col; k++) if (x[j][k]){
                    tmp[i][k] += a[i][j] * x[j][k];
                    //mult(a[i][j], x[j][k], MOD);
                    tmp[i][k] %= MOD;
                }
        return tmp;
    }
    void operator *= (matrix x){*this = *this * x;}
    matrix operator ^ (LL x){
        matrix ret(row, col);
        for (int i = 0; i < col; i++) ret[i][i] = 1;
        matrix tmp = *this;
        for (; x; x >>= 1, tmp *= tmp){if (x&1) ret *= tmp;}
        return ret;
    }
    void print(){
        for (int i = 0; i < row; i++){
            for (int j = 0; j < col; j++) printf("%lld
",a[i][j]);puts("");
        }
    }};
```

就是快速幂(czj)

```
LL fast_multi(LL m, LL n, LL mod){//快速乘法
    LL ans = 0;//注意初始化是 0，不是 1
    for (;n; n >>= 1){
        if (n & 1) ans += m;
        m = (m + m) % mod;//和快速幂一样，只不过这里是加
        m %= mod;//取模，不要超出范围
        ans %= mod;
    }
    return ans;
}
LL fast_pow(LL a, LL n, LL mod){//快速幂
    LL ans = 1;
    for (;n;n >>= 1){
        if (n & 1) ans = fast_multi(ans, a, mod);//不能直接乘
        a = fast_multi(a, a, mod);
        ans %= mod;
        a %= mod;
    }
    return ans;
}
```

线性递推函数杜教模板

```
#include <cstring>
#include <cmath>
#include <algorithm>
#include <vector>
#include <string>
#include <map>
#include <set>
#include <cassert>
using namespace std;
#define rep(i,a,n) for (int i=a;i<n;i++)
#define per(i,a,n) for (int i=n-1;i>=a;i--)
#define pb push_back
#define mp make_pair
#define all(x) (x).begin(),(x).end()
#define fi first
#define se second
```

```
#define SZ(x) ((int)(x).size())
typedef vector<int> VI;
typedef long long ll;
typedef pair<int,int> PII;
const ll mod=1000000007;
ll powmod(ll a,ll b) {ll res=1;a%=mod; assert(b>=0);
for(;b;b>>=1){if(b&1)res=res*a%mod;a=a*a%mod;}return res;}
// head

int _;
ll n;
namespace linear_seq {
    const int N=10010;
    ll res[N],base[N],_c[N],_md[N];

    vector<ll> Md;
    void mul(ll *a,ll *b,ll k) {
        rep(i,0,k+k) _c[i]=0;
        rep(i,0,k) if (a[i]) rep(j,0,k) _c[i+j]=(_c[i+j]+a[i]*b[j])%mod;
        for (int i=k+k-1;i>=k;i--) if (_c[i])
            rep(j,0,SZ(Md)) _c[i-k+Md[j]]=(_c[i-k+Md[j]]-
_c[i]*_md[Md[j]])%mod;
        rep(i,0,k) a[i]=_c[i];
    }
    int solve(ll n,VI a,VI b) {
        ll ans=0,pnt=0;
        ll k=SZ(a);
        assert(SZ(a)==SZ(b));
        rep(i,0,k) _md[k-1-i]=-a[i];_md[k]=1;
        Md.clear();
        rep(i,0,k) if (_md[i]!=0) Md.push_back(i);
        rep(i,0,k) res[i]=base[i]=0;
        res[0]=1;
        while ((1ll<<pnt)<=n) pnt++;
        for (int p=pnt;p>=0;p--) {
            mul(res,res,k);
            if ((n>>p)&1) {
                for (int i=k-1;i>=0;i--) res[i+1]=res[i];res[0]=0;
                rep(j,0,SZ(Md)) res[Md[j]]=(res[Md[j]]-
res[k]*_md[Md[j]])%mod;
            }
        }
        rep(i,0,k) ans=(ans+res[i]*b[i])%mod;
        if (ans<0) ans+=mod;
```

```
        return ans;
    }
    VI BM(VI s) {
        VI C(1,1),B(1,1);
        int L=0,m=1,b=1;
        rep(n,0,SZ(s)) {
            ll d=0;
            rep(i,0,L+1) d=(d+(ll)C[i]*s[n-i])%mod;
            if (d==0) ++m;
            else if (2*L<=n) {
                VI T=C;
                ll c=mod-d*powmod(b,mod-2)%mod;
                while (SZ(C)<SZ(B)+m) C.pb(0);
                rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
                L=n+1-L; B=T; b=d; m=1;
            } else {
                ll c=mod-d*powmod(b,mod-2)%mod;
                while (SZ(C)<SZ(B)+m) C.pb(0);
                rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c*B[i])%mod;
                ++m;
            }
        }
        return C;
    }
    int gao(VI a,ll n) {
        VI c=BM(a);
        c.erase(c.begin());
        rep(i,0,SZ(c)) c[i]=(mod-c[i])%mod;
        return solve(n,c,VI(a.begin(),a.begin()+SZ(c)));
    }
};

int main() {
    for (scanf("%d",&_);_;_--) {
        scanf("%lld",&n);
        printf("%d\n",linear_seq::gao(VI{31, 197, 1255, 7997, 50959,
324725, 2069239, 13185773, 84023455},n-2));
    }
}
```

高斯消元（naive）

判断是否有解

求解见红书

```
int Gauss(matrix a, int m, int n){
    int x_cnt = 0;
    int col, k;         //col 为列号,k 为行号
    for (k=0,col=0;k<m&&col<n; ++k, ++col){
        int r = k;      //r 为第 col 列的一个 1
        for (int i=k;i<m;++i) if (a[i][col])r=i;
        if (!a[r][col]){ k--; continue;}
        if (r!=k)for (int i=col;i<=n;++i)
            swap( a[r][i], a[k][i]);
        for (int i=k+1;i<m; ++i)if (a[i][col])//消元
            for (int j=col;j<=n;++j) a[i][j] ^= a[k][j];
    }
    for (int i=k;i<m;++i) if (a[i][n])return -1;
    if (k<=n)return n-k;     //返回自由元个数
}
```

NTT（对任意数取模）

```
// 多项式乘法 系数对 MOD=1000000007 取模， 常数巨大，慎用
// 只要选的 K 个素数乘积大于 MOD*MOD*N,理论上 MOD 可以任取。
#define MOD 1000000007
#define K 3
const int m[K] = {1004535809, 998244353, 104857601};
#define G 3
int qpow(int x, int k, int P) {
    int ret = 1;
    while(k) {
        if(k & 1) ret = 1LL * ret * x % P;
        k >>= 1;
        x = 1LL * x * x % P;
    }
    return ret;
}
struct _NTT {
    int wn[25], P;
    void init(int _P) {
        P = _P;
```

```
    for(int i = 1; i <= 21; ++i) {
        int t = 1 << i;
        wn[i] = qpow(G, (P - 1) / t, P);
    }
}
void change(int *y, int len) {
    for(int i = 1, j = len / 2; i < len - 1; ++i) {
        if(i < j) swap(y[i], y[j]);
        int k = len / 2;
        while(j >= k) {j -= k;k /= 2;}
        j += k;
    }
}
void NTT(int *y, int len, int on) {
    change(y, len);
    int id = 0;
    for(int h = 2; h <= len; h <<= 1) {
        ++id;
        for(int j = 0; j < len; j += h) {
            int w = 1;
            for(int k = j; k < j + h / 2; ++k) {
                int u = y[k];
                int t = 1LL * y[k+h/2] * w % P;
                y[k] = u + t;
                if(y[k] >= P) y[k] -= P;
                y[k+h/2] = u - t + P;
                if(y[k+h/2] >= P) y[k+h/2] -= P;
                w = 1LL * w * wn[id] % P;
            }
        }
    }
    if(on == -1) {
        for(int i = 1; i < len / 2; ++i) swap(y[i], y[len-i]);
        int inv = qpow(len, P - 2, P);
        for(int i = 0; i < len; ++i)
            y[i] = 1LL * y[i] * inv % P;
    }
}
void mul(int A[], int B[], int len) {
    NTT(A, len, 1);
    NTT(B, len, 1);
    for(int i = 0; i < len; ++i) A[i] = 1LL * A[i] * B[i] % P;
    NTT(A, len, -1);
}
```

```
}ntt[K];
int tmp[N][K], t1[N], t2[N];
int r[K][K];
int CRT(int a[]) {
    int x[K];
    for(int i = 0; i < K; ++i) {
        x[i] = a[i];
        for(int j = 0; j < i; ++j) {
            int t = (x[i] - x[j]) % m[i];
            if(t < 0) t += m[i];
            x[i] = 1LL * t * r[j][i] % m[i];
        }
    }
    int mul = 1, ret = x[0] % MOD;
    for(int i = 1; i < K; ++i) {
        mul = 1LL * mul * m[i-1] % MOD;
        ret += 1LL * x[i] * mul % MOD;
        if(ret >= MOD) ret -= MOD;
    }
    return ret;
}
void mul(int A[], int B[], int len) {
    for(int id = 0; id < K; ++id) {
        for(int i = 0; i < len; ++i) {
            t1[i] = A[i];
            t2[i] = B[i];
        }
        ntt[id].mul(t1, t2, len);
        for(int i = 0; i < len; ++i) tmp[i][id] = t1[i];
    }
    for(int i = 0; i < len; ++i) A[i] = CRT(tmp[i]);
}
void init() {
    for(int i = 0; i < K; ++i) {
        for(int j = 0; j < i; ++j) {
            r[j][i] = qpow(m[j], m[i] - 2, m[i]);
        }
    }
    for(int i = 0; i < K; ++i) ntt[i].init(m[i]);
}
```

数论

常用公式和找规律

斯特林公式

n 约等于 sqrt(2*pi*n)*pow(1.0*n/e,n)

带标号连通图计数

1 1 1 4 38 728 26704 1866256 251548592

h(n)=2^(n(n-1)/2)

f(n) = h(n)-sum{C(n-1,k-1)*f(k)*h(n-k)}(k=1...n-1)

不带标号 n 个节点的有根树计数

1, 1, 2, 4, 9, 20, 48, 115, 286, 719, 1842,

不带标号 n 个节点的树的计数

1,2,3,6,11,23,47,106,235

OEIS

A(x) = 1 + T(x) - T^2(x)/2 + T(x^2)/2,

where T(x) = x + x^2 + 2*x^3 + ... is the g.f. for A000081

错排公式

```
D[1] = 0; D[2] = 1;
for(int i = 3; i < 25; i++) {
    D[i] = (i - 1) * (D[i - 1] + D[i - 2]);
}
```

卡特兰数

1 2 5 14 42 132 429 1430 4862 16796

binomial(2*n, n)-binomial(2*n, n-1)

Sum_{k=0..n-1} a(k)a(n-1-k)

Stirling 数，又称为斯特灵数。

在组合数学，Stirling 数可指两类数，都是由 18 世纪数学家 James Stirling 提出的。

第一类 Stirling 数是有正负的，其绝对值是包含 n 个元素的集合分作 k 个环排列的方法数目。

第二类 Stirling 数是把包含 n 个元素的集合划分为正好 k 个非空子集的方法的数目。

递推公式

第一类 Stirling 数是有正负的，其绝对值是包含 n 个元素的集合分作 k 个环排列的方法数目。

递推公式为，

S(n,0) = 0, S(1,1) = 1.

S(n 1,k) = S(n,k-1) nS(n,k)。

第二类 Stirling 数是把包含 n 个元素的集合划分为正好 k 个非空子集的方法的数目。

递推公式为：

S(n,k)=0; (n<k||k=0)

S(n,n) = S(n,1) = 1,

S(n,k) = S(n-1,k-1) kS(n-1,k).

第一类斯特林数

有符号 Stirling 数（无符号 Stirling 数直接取绝对值）

```
n=0     1
n=1     0 1
n=2     0 -1 1
n=3     0 2 -3 1
n=4     0 -6 11 -6 1
n=5     0 24 -50 35 -10 1
n=6     0 -120 274 -225 85 -15 1
n=7     0 720 -1764 1624 -735 175 -21 1
```

第二类

```
n=0     1
n=1     0 1
n=2     0 1 1
n=3     0 1 3 1
n=4     0 1 7 6 1
n=5     0 1 15 25 10 1
n=6     0 1 31 90 65 15 1
n=7     0 1 63 301 350 140 21 1
n=8     0 1 127 966 1701 1050 266 28 1
n=9     0 1 255 3025 7770 6951 2646 462 36 1
```

欧拉筛

```
#include <cstring>
using namespace std;
int prime[1100000],primesize,phi[11000000];
bool isprime[11000000];

void getlist(int listsize){
    memset(isprime, 1, sizeof(isprime));
    isprime[1] = false;
    for(int i=2;i<=listsize;i++){
        if(isprime[i]) prime[++primesize]=i;
         for(int j = 1; j <= primesize && i*prime[j] <= listsize;j++){
            isprime[i*prime[j]] = false;
            if(i%prime[j] == 0) break;
         }
    }
}
```

莫比乌斯函数模板

```
void Init(){
    memset(vis,0,sizeof(vis));
    mu[1] = 1;
    cnt = 0;
    for(int i=2; i<N; i++){
        if(!vis[i]){
            prime[cnt++] = i;
            mu[i] = -1;
        }
        for(int j=0; j<cnt&&i*prime[j]<N; j++){
            vis[i*prime[j]] = 1;
            if(i%prime[j]) mu[i*prime[j]] = -mu[i];
            else{
                mu[i*prime[j]] = 0;
                break;
            }
        }
    }
}
```

乘法逆元

```
//扩展欧几里得（扩展 gcd）
LL ex_gcd(LL a,LL b,LL &x,LL &y){
    if (a == 0 && b == 0) return -1;
    if (b == 0){x = 1; y = 0; return a;}
    LL d=ex_gcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}

LL mod_inverse(LL a,LL n){//乘法逆元
    LL x,y;
    LL d = ex_gcd(a,n,x,y);
    return (x % n + n) % n;
}

//p 是质数可以 快速幂 p-2
LL quick_mod(LL a, LL b){
    LL ans = 1;
    a %= MOD;
    for(;b; b >>= 1, a = a*a % MOD){
        if(b & 1) ans = ans * a % MOD;
    }
    return ans;
}

//逆元筛：求 1-MAXN 的所有关于 MOD 的逆元
inv[0] = 0; inv[1] = 1;
for(i = 2; i < MAXN; i++){
    inv[i] = inv[MOD % i] * (MOD - MOD / i) % MOD;
    f[i] = (f[i-1] * i) % MOD;
}
```

详解 ACM 组合数处理，

O(n2)算法——杨辉三角

O(n)算法——阶乘取模 + 乘法逆元
C(m,n) = n! / m! / (n - m)!
如果 p 是质数，直接 quick_mod(b, p-2) % p 费马小定理求逆元

```
LL C(LL n, LL m){
    if(m > n) return 0;
    LL ans = 1;
    for(int i = 1; i <= m; i++){
        LL a = (n + i - m) % MOD;
        LL b = i % MOD;
        ans = ans * (a * quick_mod(b, p-2) % MOD) % MOD;
    }
    return ans;
}
```

如果 n,m 很大 达到 1e18，但是 p 很小 <= 1e5 ，我们可以利用这个 p
Lucas 定理：C(n, m) % p = C(n / p, m / p) * C(n%p, m%p) % p

```
LL Lucas(LL n, LL m){
    if(m == 0) return 1;
    return C(n % p, m % p) * Lucas(n / p, m / p) % p;
}
void InitFac(){//阶乘预处理
    fac[0]  = 1;
    for(int i=1; i<=n; i++)
        fac[i] = (fac[i-1] * i) % MOD;
}
LL C(LL n,LL m,LL p,LL fac[]){
    if(n < m) return 0;
    return fac[n] * quick_mod(fac[m] * fac[n-m], p - 2, p) % p;
}
```

组合数奇偶性结论：
如果（n&m) == m 那么 c(m,n)是奇数，否则是偶数

预处理所有数的因数

```
//预处理所有数的因数表
//SPEED: ECNUOJ 1E6 5000MS O(nlogn)
const int N = 100000 + 5;
vector<int > factor[N];
void init(){
    for(int i = 2; i < N; i ++){
        for(int j = i; j < N; j += i){
            factor[j].push_back(i);
        }
    }
}

//预处理质因数表
vector<int> x[N];
bool is[N];
void prime() {
    memset(is, false, sizeof(is));
    for (int i=0; i<N; i++) x[i].clear();
    for (int j=2; j<N; j+=2) x[j].push_back(2);
    for (int i=3; i<N; i+=2)
        if (!is[i]) {
            for (int j=i; j<N; j+=i) {
                is[j] = true;
                x[j].push_back(i);
            }
        }
}
```

随机素数测试和大数分解(POJ 1811)

```
/*************************************************
* Miller_Rabin 算法进行素数测试
* 速度快，可以判断一个 < 2^63 的数是不是素数
*************************************************/
const int S = 8; //随机算法判定次数，一般 8~10 就够了
// 快速乘法，计算 ret = (a*b)%c a,b,c < 2^63
long long mult_mod(long long a,long long b,long long c)
// 快速幂，计算 ret = (a^n)%mod
long long pow_mod(long long a,long long n,long long mod)
// 通过 a^(n-1)=1(mod n)来判断 n 是不是素数
```

```
// n-1 = x*2^t 中间使用二次判断
// 是合数返回 true, 不一定是合数返回 false
bool check(long long a,long long n,long long x,long long t){
    long long ret = pow_mod(a,x,n);
    long long last = ret;
    for(int i = 1; i <= t; i++){
        ret = mult_mod(ret,ret,n);
        if(ret == 1 && last != 1 && last != n-1)return true;//合数
        last = ret;
    }
    if(ret != 1)return true;
    else return false;
}
//************************************************
// Miller_Rabin 算法
// 是素数返回 true,(可能是伪素数)
// 不是素数返回 false
//************************************************
bool Miller_Rabin(long long n){
    if (n < 2) return false;
    if (n == 2) return true;
    if ((n&1) == 0) return false;//偶数
    long long x = n - 1, t = 0;
    for(; (x&1)==0;){x >>= 1; t++;}
    srand(time(NULL));/* *************** */
    for(int i = 0; i < S; i++){
        long long a = rand()%(n-1) + 1;
        if( check(a,n,x,t) ) return false;
    }
    return true;
}
//***********************************************
// pollard_rho 算法进行质因素分解
//***********************************************
long long factor[100];//质因素分解结果（刚返回时时无序的）
int tol;//质因素的个数，编号 0~tol-1
long long gcd(long long a,long long b)
//找出一个因子
long long pollard_rho(long long x,long long c){
    long long i = 1, k = 2;
    srand(time(NULL));
    long long x0 = rand()%(x-1) + 1;
    long long y = x0;
    while(1){
```

```
        i ++;
        x0 = (mult_mod(x0,x0,x) + c)%x;
        long long d = gcd(y - x0,x);
        if( d != 1 && d != x)return d;
        if(y == x0) return x;
        if(i == k){y = x0; k += k;}
    }
}
//对 n 进行素因子分解，存入 factor. k 设置为 107 左右即可
void findfac(long long n,int k){
    if(n == 1)return;
    if(Miller_Rabin(n)){
        factor[tol++] = n;
        return;
    }
    long long p = n;
    int c = k;
    while( p >= n) p = pollard_rho(p,c--);//值变化，防止死循环 k
    findfac(p,k);
    findfac(n/p,k);
}
//POJ 1811
//给出一个 N(2 <= N < 2^54),如果是素数，输出"Prime",否则输出最小的素因子
int main(){
    int T; scanf("%d",&T); long long n;
    while(T--){
        scanf("%I64d",&n);
        if(Miller_Rabin(n)) printf("Prime\n");
        else{
            tol = 0;
            findfac(n,107);
            long long ans = factor[0];
            for(int i = 1; i < tol; i++)
                ans = min(ans,factor[i]);
            printf("%I64d\n",ans);
        }
    }
    return 0;
}
```

中国剩余定理

```
//可以不满足两两互质
int n;
//扩展 gcd 多了一个变量
void ex_gcd(LL a, LL b, LL &d, LL &x, LL &y){
    if (!b) {d = a, x = 1, y = 0;}
    else{
        ex_gcd(b, a % b, d, y, x);
        y -= x * (a / b);
    }
}
//注意 M 的范围
//如果 M 是 int，都要开 ll，M 是 ll，乘法要用快速乘
LL ex_crt(LL *m, LL *r, int n){
    LL M = m[1], R = r[1], x, y, d;
    for (int i = 2; i <= n; ++i){
        ex_gcd(M, m[i], d, x, y);
        if ((r[i] - R) % d) return -1;
        x = (r[i] - R) / d * x % (m[i] / d);
        R += x * M;
        M = M / d * m[i];
        R %= M;
    }
    return R > 0 ? R : R + M;
}
```

离散对数（关于方程 x^A=B(mod C)的解)

首先判断是否有解，即 a,p 是否互质。不互质即无解。不妨令 x = im − j, 其中 m = [√ q] , 这样问题变为求得一组 i j 使得条件满足。此时原式变为 a im−j ≡ b (Mod p), 移
项化简得 (a m) i ≡ ba j (Mod p)。这个时候我们只需穷举 i,j 使得式子成立即可。先从让 j 从
[0,m] 中穷举，并用 hash 记录下 ba j 对应的 j 值。相同的 ba j 记录较大的 j. 接着让 i 从 [1,m]
中穷举，如果 (a m) i 在 hash 表中有对应的 j 存在，则对应的 im − j 是一组解。其中第一次出
现的为最小的解。

```
map<LL, int> Hash;
LL i, j;
LL bsgs(LL a, LL b, LL p){
    LL xx, yy;
    if (exgcd(a, p, xx, yy) != 1)return -1;
    a %= p;
    LL m = ceil(sqrt(p));
    Hash.clear();
    LL tmp, ans = b % p;
    for (int i = 0; i <= m; ++i){
        Hash[ans] = i;
        ans = ans * a % p;
    }
    tmp = f(a, m, p);
    ans = 1;
    for (int i = 1; i <= m; ++i){
        ans = ans * tmp % p;
        if (Hash[ans] != 0) return i * m - Hash[ans];
    }
    return -1;
}
```

生成函数 五边形数定理 整数拆分模板

hdu4658

问一个数 n 能被拆分成多少种方法，且每一种方法里数字重复个数不能超过 k（等于 k）。

f[n]=∑(-1)^(k-1)*(f[n-k*(3k-1)/2]+f[n-k*(3*k+1)/2])*

```cpp
#include <iostream>
#include <string.h>
#include <stdio.h>
using namespace std;
const int N = 100005;
const int MOD = 1000000007;
int dp[N];
void Init()  {
    dp[0] = 1;
    for(int i=1;i<N;i++){
        dp[i] = 0;
        for(int j=1;;j++){
            int t = (3*j-1)*j / 2;
            if(t > i) break;
            int tt = dp[i-t];
            if(t+j <= i) tt = (tt + dp[i-t-j])%MOD;
            if(j&1) dp[i] = (dp[i] + tt)%MOD;
            else    dp[i] = (dp[i] - tt + MOD)%MOD;
        }
    }
}
int Work(int n,int k) {
    int ans = dp[n];
    for(int i=1;;i++){
        int t = k*i*(3*i-1) / 2;
        if(t > n) break;
        int tt = dp[n-t];
        if(t + i*k <= n) tt = (tt + dp[n-t-i*k])%MOD;
        if(i&1) ans = (ans - tt + MOD)%MOD;
        else    ans = (ans + tt)%MOD;
    }
    return ans;
}
int main(){
    Init();
    int n,k,t;
    scanf("%d",&t);
    while(t--){
        scanf("%d%d",&n,&k);
        printf("%d\n",Work(n,k));
    }
    return 0;
}
```

容斥原理 dfs

题意找与 n,m 互质的第 k 个数
思路：二分
找到最小的 x，使得小于或等于 x 的数中满足条件的数的个数大于或等于 k
预处理 n,m 的质因数表
k 是深度，也就是当前质因数位置
t 是奇偶判断
s 是质数乘积
n 是传进去的 x

```
void dfs(LL k,LL t,LL s,LL n)  {
    if(k==num)  {
        if(t&1) ans-=n/s;
        else    ans+=n/s;
        return;
    }
    dfs(k+1,t,s,n);
    dfs(k+1,t+1,s*fac[k],n);  //fac[k]是质因数表
}
```

//二分调用
dfs(0,0,1,mid);

求（1，b）区间和（1，d）区间里面 gcd(x, y) = k 的数的对数（1<=x<=b , 1<= y <= d）。
b 和 d 分别除以 k 之后的区间里面，只需要求 gcd(x, y) = 1 就可以了，这样子求出的数的对数不变。
这道题目还要求 1-3 和 3-1 这种情况算成一种，因此只需要限制 x<y 就可以了

只需要枚举 x，然后确定另一个区间里面有多少个 y 就可以了。因此问题转化成为区间（1， d）里面与 x 互素的数的个数
先求出 x 的所有质因数，因此（1，d）区间里面是 x 的质因数倍数的数都不会与 x 互素，因此，只需要求出这些数的个数，减掉就可以了。
如果 w 是 x 的素因子，则（1，d）中是 w 倍数的数共有 d/w 个。

容斥原理：
所有不与 x 互素的数的个数= 1 个因子倍数的个数 - 2 个因子乘积的倍数的个数 + 3 个……-……

答案很大，用 long long。
所有数的素因子，预先处理保存一下，不然会超时的。

```
#include <cstdio>
#include <cstring>
#include <vector>
#include <algorithm>
using namespace std;
#define N 100005
typedef long long ll;
vector<int> x[N];
bool is[N];

void prime() {
    memset(is, false, sizeof(is));
    for (int i=0; i<N; i++) x[i].clear();

    for (int j=2; j<N; j+=2) x[j].push_back(2);
    for (int i=3; i<N; i+=2)
        if (!is[i]) {
            for (int j=i; j<N; j+=i) {
                is[j] = true;
                x[j].push_back(i);
            }
        }
}
int work(int u, int s, int w) {
    int cnt = 0, v = 1;
    for (int i=0; i<x[w].size(); i++) {
        if ((1<<i) & s) {
            cnt++;
            v *= x[w][i];
        }
    }
    int all = u/v;
    if (cnt % 2 == 0) return -all;
    else return all;
}

int main() {
    prime();
    int T, a, b, c, d, k;
    scanf("%d", &T);
    for (int cas=1; cas<=T; cas++) {
        scanf("%d%d%d%d%d", &a, &b, &c, &d, &k);
        if (k == 0) {
            printf("Case %d: 0\n", cas);
```

```
            continue;
        }
        b /= k, d /= k;
        if (b > d) { a = b; b = d; d = a; }
        long long ans = 0;
        for (int i=1; i<=d; i++) {
            k = min(i, b);
            ans += k;
            for (int j=1; j<(1<<x[i].size()); j++)
                ans -= work(k, j, i);
        }
        printf("Case %d: %I64d\n", cas, ans);
    }
    return 0;
}
```

村民排队问题模型

一堆数，其中有一些两两关系，（A,B）表示 A 在 B 前面，求排列数

```
// UVa11174 Stand in a Line
// Rujia Liu

int mul_mod(int a, int b, int n) {
  a %= n; b %= n;
  return (int)((long long)a * b % n);
}
void gcd(int a, int b, int& d, int& x, int& y) {
  if(!b){ d = a; x = 1; y = 0; }
  else{ gcd(b, a%b, d, y, x); y -= x*(a/b); }
}

int inv(int a, int n) {
  int d, x, y;
  gcd(a, n, d, x, y);
  return d == 1 ? (x%n+n)%n : -1;
}

#include<cstdio>
#include<cstring>
#include<vector>
using namespace std;
```

```
const int maxn = 40000 + 10;
const int MOD = 1000000007;
vector<int> sons[maxn];
int fa[maxn], fac[maxn], ifac[maxn];

int mul_mod(int a, int b) {
  return mul_mod(a, b, MOD);
}

// fac[i] = (i!)%MOD, ifac[i]为 fac[i]在模 MOD 下的逆
void preprocess() {
  fac[0] = ifac[0] = 1;
  for(int i = 1; i < maxn; i++) {
    fac[i] = mul_mod(fac[i-1], i);
    ifac[i] = inv(fac[i], MOD);
  }
}

// 组合数 C(n,m)除以 MOD 的余数
int C(int n, int m) {
  return mul_mod(mul_mod(fac[n], ifac[m]), ifac[n-m]);
}

// 统计以 u 为根的子树有多少种排列。size 为该子树的结点总数
int count(int u, int& size) {
  int d = sons[u].size();
  vector<int> sonsize; // 各子树的大小
  size = 1;
  int ans = 1;
  for(int i = 0; i < d; i++) {
    int sz;
    ans = mul_mod(ans, count(sons[u][i], sz));
    size += sz;
    sonsize.push_back(sz);
  }
  int sz = size-1; // 非根结点的个数
  for(int i = 0; i < d; i++) {
    ans = mul_mod(ans, C(sz, sonsize[i]));
    sz -= sonsize[i];
  }
  return ans;
}
```

```
int main() {
  int T;
  scanf("%d", &T);
  preprocess();
  while(T--) {
    int n, m;
    scanf("%d%d", &n, &m);
    memset(fa, 0, sizeof(fa));
    for(int i = 0; i <= n; i++) sons[i].clear();
    for(int i = 0; i < m; i++) {
      int a, b;
      scanf("%d%d", &a, &b);
      fa[a] = b;
      sons[b].push_back(a);
    }
    // 没有父亲的结点称为虚拟结点的儿子
    for(int i = 1; i <= n; i++)
      if(!fa[i]) sons[0].push_back(i);
    int size;
    printf("%d\n", count(0, size));
  }
  return 0;
}
```

SG 函数，博弈

```
/**********************
每组数据都改变策略
**********************/
#include<iostream>
#include<cstdio>
#include<cstring>
#include<string>
#include<cmath>
#include<algorithm>
using namespace std;
typedef int LL;
const int MAXN = 1e4 + 5;
const int MAXM = 1e4;
int sg[MAXN];
bool Hash[MAXN];
int f[MAXN];
int N;
void getsg(int n){
    memset(sg,0,sizeof sg);
    for (int i=1;i<=MAXM;i++){
        memset(Hash,false,sizeof Hash);
        for(int j = 0; j < N && i >= f[j]; j++) {
            Hash[sg[i-f[j]]] = true;
                        /****************上海大学校赛教训，板不要理解错。
                        这不是一堆拆两堆，这是每个可以转移的状态都标记。
                        ****************/
        }
        for (int j=0;j<=MAXM;j++){
            if (!Hash[j]){
                sg[i]=j; break;
            }
        }
        //cout << i << " " << sg[i] << endl;
    }
}
int main() {
    while(cin >> N, N) {
        for(int i = 0; i < N; i++) {
            scanf("%d", f + i);
        }
        sort(f, f + N);//一定要排序
```

```
        getsg(MAXM);
        int m;
        cin >> m;
        for(int i = 0; i < m; i++) {
            int n;
            cin >> n;
            int ans = 0;
            for(int i = 0; i < n; i++) {
                int x;
                scanf("%d", &x);
                ans ^= sg[x];
            }
            printf("%s", ans ? "W" : "L");
        }
        printf("\n");
    }
    return 0;
}

/***************
独立的棋盘横向移动，看成一个子向另一个子一直在减小，NIM 两子间距
***************/

/***********************
有一个操作可以把一堆拆成两堆，枚举拆分点
***********************/
for(int j = 0; j <= i - x; j++) {
        Hash[sg[j] ^ sg[i - x - j]] = 1;
}
/***********************
拿走最后一个的人输，需要特判全是 1 的情况。
全是 1，分奇偶。不全是 1，同直接 NIM
***********************/
/********************
两维的一样拆分成两个异或。SG[][]两维
0 行，0 列特殊处理。直接设置成离原点的距离。
2 2
.#
..
2 2
.#
.#
0 0
********************/
```

```
/******************
两个操作，合并两堆，或者取掉 1 个
******************/
最后肯定合并成一堆再一个个取
如果全大于 1，先手可以保证 NIM 胜利的情况下先合并
不全为 1，后手可以取完一个一堆的，相当于操作了两次。
此时，DFS+记忆化搜索来解决
dp[i][j]当 1 的个数为 i 时，其他全合并起来一共 j 个
其中的操作包括：
把某堆只有一个的，取走
把两堆只有一个的，合并
把某堆只有一个的，合并给不是一个的
把不是一个的，取走一个
int dfs(int i, int j) {
    if (dp[i][j] != -1) return dp[i][j];
    if (j == 1) return dp[i][j] = dfs(i+1, 0);
    dp[i][j] = 0;
    if (i >= 1 && !dfs(i-1, j)) dp[i][j] = 1;
    else if (j >= 1 && !dfs(i, j-1)) dp[i][j] = 1;
    else if (i >= 1 && j > 0 && !dfs(i-1, j+1)) dp[i][j] = 1;
    else if (i >= 2 && ((j >= 1 && !dfs(i-2, j+3)) || (j == 0 && !dfs(i-
2, 2))))
        dp[i][j] = 1;
    return dp[i][j];
}
/****************
31 游戏
1~6 各 4 张
****************/
直接搜索，据说记忆化也不用
```

反 nim 问题

这题与以往的博弈题的胜负条件不同，谁先走完最后一步谁输，但他也是一类 Nim 游戏，即为 anti-nim 游戏。
首先给出结论：先手胜当且仅当

①所有堆石子数都为 1 且游戏的 SG 值为 0（即有偶数个孤单堆-每堆只有 1 个石子数）；

②存在某堆石子数大于 1 且游戏的 SG 值不为 0.

字符串

Manacher 算法（回文串）

```
#include<cstdio>
#include<string>
#include<cstring>
#include<iostream>
#include<algorithm>
using namespace std;
const int N=233333;//20W

//在 o(n)时间内算出以每个点为中心的最大回文串长度
int Manacher(string st){
    int len = st.size();
    int *p = new int[len+1];
    memset(p,0,sizeof(p));
    int mx = 0,id = 0;
    for (int i = 1;i <= len; i++){
        if (mx > i) p[i] = min(p[2*id-i],mx-i);
        else p[i] = 1;
        while (st[i+p[i]] == st[i-p[i]]) p[i]++;
        if (i + p[i] > mx){mx = i + p[i]; id = i;}
    }
    int ma = 0;
    for (int i = 1; i < len; i++) ma = max(ma, p[i]);
    delete(p);
    return ma - 1;
}

int main(){
    //freopen("fuck.in","r",stdin);
    char st[N];
    while (~scanf("%s",st)){
        string st0="$#";
        for (int i=0; st[i] != '\0'; i++){
            st0 += st[i]; st0 += "#";
        }
        printf("%d\n", Manacher(st0));
    }
    return 0;
}
```

KMP（字符串匹配）

```
#include<cstdio>
#include<cstring>
using namespace std;
typedef long long LL;
const int N=100007;
const int P=1000000007;
char a[N],b[N];
bool mat[N];
int Next[N];//一定要 Next,next 会 CE
LL f[N];

void getNext(int m, char b[]){
    int i = 0,j = -1;
    Next[0] = -1;
    while (i < m){
        if (j == -1 || b[i] == b[j]){
            if (b[++i] != b[++j]) Next[i]=j;
            else Next[i] = Next[j];
        }else j = Next[j];
    }
}
//主程序里每组数据需要 memset a,b 数组!!!
void KMP(int n,char a[], int m, char b[]){
    memset(mat, 0, sizeof(mat));
    int i = 0, j = 0;
    getNext(m, b);//这行视情况可以放在 main 里面
    while (i < n && j < m){
        if (j == -1 || a[i] == b[j]) i++, j++;
        else j = Next[j];
        if (!i && !j)break;
        if (j == m){
            mat[i] = 1;
            //printf("mat[%d]get\n",i);
            j = Next[j];
        }
    }
}
```

Tire 树_刘汝佳

```
// LA3942 Remember the Word
// Rujia Liu
#include<cstring>
#include<vector>
using namespace std;

const int maxnode = 4000 * 100 + 10;
const int sigma_size = 26;

// 字母表为全体小写字母的 Trie
struct Trie {
  int ch[maxnode][sigma_size];
  int val[maxnode];
  int sz; // 结点总数
  void clear() { sz = 1; memset(ch[0], 0, sizeof(ch[0])); } // 初始时只有一个根结点
  int idx(char c) { return c - 'a'; } // 字符 c 的编号

  // 插入字符串 s，附加信息为 v。注意 v 必须非 0，因为 0 代表“本结点不是单词结点”
  void insert(const char *s, int v) {
    int u = 0, n = strlen(s);
    for(int i = 0; i < n; i++) {
      int c = idx(s[i]);
      if(!ch[u][c]) { // 结点不存在
        memset(ch[sz], 0, sizeof(ch[sz]));
        val[sz] = 0;  // 中间结点的附加信息为 0
        ch[u][c] = sz++; // 新建结点
      }
      u = ch[u][c]; // 往下走
    }
    val[u] = v; // 字符串的最后一个字符的附加信息为 v
  }

  // 找字符串 s 的长度不超过 len 的前缀
  void find_prefixes(const char *s, int len, vector<int>& ans) {
    int u = 0;
    for(int i = 0; i < len; i++) {
      if(s[i] == '\0') break;
      int c = idx(s[i]);
      if(!ch[u][c]) break;
      u = ch[u][c];
```

```
      if(val[u] != 0) ans.push_back(val[u]); // 找到一个前缀
    }
  }
};

#include<cstdio>
const int maxl = 300000 + 10; // 文本串最大长度
const int maxw = 4000 + 10;   // 单词最大个数
const int maxwl = 100 + 10;   // 每个单词最大长度
const int MOD = 20071027;

int d[maxl], len[maxw], S;
char text[maxl], word[maxwl];
Trie trie;

int main() {
  int kase = 1;
  while(scanf("%s%d", text, &S) == 2) {
    trie.clear();
    for(int i = 1; i <= S; i++) {
      scanf("%s", word);
      len[i] = strlen(word);
      trie.insert(word, i);
    }
    memset(d, 0, sizeof(d));
    int L = strlen(text);
    d[L] = 1;
    for(int i = L-1; i >= 0; i--) {
      vector<int> p;
      trie.find_prefixes(text+i, L-i, p);
      for(int j = 0; j < p.size(); j++)
        d[i] = (d[i] + d[i+len[p[j]]]) % MOD;
    }
    printf("Case %d: %d\n", kase++, d[0]);
  }
  return 0;
}
```

压缩 Tire 树

```
// UVa11732 strcmp() Anyone?
// Rujia Liu
#include<cstring>
#include<vector>
using namespace std;
const int maxnode = 4000 * 1000 + 10;
const int sigma_size = 26;

// 字母表为全体小写字母的 Trie
struct Trie {
  int head[maxnode]; // head[i]为第 i 个结点的左儿子编号
  int next[maxnode]; // next[i]为第 i 个结点的右兄弟编号
  char ch[maxnode];  // ch[i]为第 i 个结点上的字符
  int tot[maxnode];  // tot[i]为第 i 个结点为根的子树包含的叶结点总数
  int sz; // 结点总数
  long long ans; // 答案
  void clear() { sz = 1; tot[0] = head[0] = next[0] = 0; } // 初始时只有一个根结点

  // 插入字符串 s（包括最后的'\0'），沿途更新 tot
  void insert(const char *s) {
    int u = 0, v, n = strlen(s);
    tot[0]++;
    for(int i = 0; i <= n; i++) {
      // 找字符 a[i]
      bool found = false;
      for(v = head[u]; v != 0; v = next[v])
        if(ch[v] == s[i]) { // 找到了
          found = true;
          break;
        }
      if(!found) {
        v = sz++; // 新建结点
        tot[v] = 0;
        ch[v] = s[i];
        next[v] = head[u];
        head[u] = v; // 插入到链表的首部
        head[v] = 0;
      }
      u = v;
      tot[u]++;
```

```
    }
  }
  // 统计 LCP=u 的所有单词两两的比较次数之和
  void dfs(int depth, int u) {
    if(head[u] == 0) // 叶结点
      ans += tot[u] * (tot[u] - 1) * depth;
    else {
      int sum = 0;
      for(int v = head[u]; v != 0; v = next[v])
        sum += tot[v] * (tot[u] - tot[v]); // 子树 v 中选一个串，其他子树中
再选一个
      ans += sum / 2 * (2 * depth + 1); // 除以 2 是每种选法统计了两次
      for(int v = head[u]; v != 0; v = next[v])
        dfs(depth+1, v);
    }
  }

  // 统计
  long long count() {
    ans = 0;
    dfs(0, 0);
    return ans;
  }
};

#include<cstdio>
const int maxl = 1000 + 10;   // 每个单词最大长度
int n;
char word[maxl];
Trie trie;

int main() {
  int kase = 1;
  while(scanf("%d", &n) == 1 && n) {
    trie.clear();
    for(int i = 0; i < n; i++) {
      scanf("%s", word);
      trie.insert(word);
    }
    printf("Case %d: %lld\n", kase++, trie.count());
  }
  return 0;
}
```

数据结构

树状数组(逆序对)

```
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>
using namespace std;
const int N = 1e5 + 7;
struct binaryIndexTree{
        int val[N], n;
        inline void init(int n){
                this->n = n;
                memset(val, 0, sizeof(val));
        }
        inline void add(int k, int num){
                for (;k <= n; k += k&-k) val[k] += num;
        }
        int sum(int k){
                int sum = 0;
                for (; k; k -= k&-k) sum += val[k];
                return sum;
        }
} T;
int arr[N], n;
int main(){
        T.init(n);
        int sum = 0;
        for (int i = 0; i < n; i++){
                scanf("%d", &arr[i]); arr[i]++;
                sum += T.sum(n) - T.sum(arr[i] - 1);
                T.add(arr[i], 1);
        }
        int ans = sum;
        for (int i = 0; i < n; i++){
                sum += (n - arr[i]) - (arr[i] - 1);
                ans = min(ans, sum);
        }
        printf("%d\n", ans);
}
```

ZKW 线段树（单点修改）

```
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>
using namespace std;
const int N = 2e5 + 7;

struct segTree{
        #define lc (t<<1)
        #define rc (t<<1^1)
        int sum[N], M;

        inline void build(int n){
                M = 1;
                for (;M < n;) M <<= 1;
                if (M!=1) M--;
                memset(sum, sizeof(sum), 0);
                for (int i = 1+M; i <= n+M; i++){
                        scanf("%d", &sum[i]);
                }
                for (int t = M; t >= 1; t--){
                        sum[t] = sum[lc] + sum[rc];
                }
        }

        void add(int t, int x){
                for (sum[t+=M]+=x, t>>=1; t; t>>=1){
                        sum[t] = sum[lc] + sum[rc];
                }
        }

        int query(int l, int r){
                int ans = 0;
                for (l+=M-1,r+=M+1; l^r^1; l>>=1,r>>=1){
                        if (~l&1) ans += sum[l^1];
                        if ( r&1) ans += sum[r^1];
                }
                return ans;
        }
} T;
```

ZKW 线段树（RMQ 区间操作）

```
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>
using namespace std;
typedef long long LL;
const int N = 2e5 + 10;
double EPS = 1e-11;
const LL INF = 0x3f3f3f3f3f3f3f3f;
int n, pos[N], arr[N], pre[N];

struct ZKWsegTree{
    double tree[N];
    int M, n;

    void build(int n, double Mid){
        this->n = n;
        M = 1; while (M < n) M <<= 1; if (M!=1) M--;
        for (int t = 1 ; t <= n; t++) tree[t+M] = 1.0*t*Mid;
        for (int t = n+1; t <= M+1; t++) tree[t+M] = INF;
        for (int t = M; t >= 1; t--) tree[t] = min(tree[t<<1],
tree[t<<1^1]);
        for (int t = 2*M+1; t >= 1; t--) tree[t] = tree[t] - tree[t>>1];
    }

    void update(int l, int r, double val){
        double tmp;
        for (l+=M-1, r+=M+1; l^r^1; l>>=1, r>>=1){
            if (~l&1) tree[l^1] += val;
            if ( r&1) tree[r^1] += val;
            if (l > 1) tmp = min(tree[l], tree[l^1]), tree[l]-=tmp,
tree[l^1]-=tmp, tree[l>>1]+=tmp;
            if (r > 1) tmp = min(tree[r], tree[r^1]), tree[r]-=tmp,
tree[r^1]-=tmp, tree[r>>1]+=tmp;
        }
        for (; l > 1; l >>= 1){
            tmp = min(tree[l], tree[l^1]), tree[l]-=tmp, tree[l^1]-=tmp,
tree[l>>1]+=tmp;
        }
        tree[1] += tree[0], tree[0] = 0;
    }
```

```
    double query(int l, int r){
        double lAns = 0, rAns = 0;
        l += M, r += M;
        if (l != r){
            for (; l^r^1; l>>=1, r>>=1){
                lAns += tree[l], rAns += tree[r];
                if (~l&1) lAns = min(lAns, tree[l^1]);
                if ( r&1) rAns = min(rAns, tree[r^1]);
            }
        }
        double ans = min(lAns + tree[l], rAns + tree[r]);
        for (;l > 1;) ans += tree[l>>=1];
        return ans;
    }
} T;
```

常规线段树（区间操作区间和）

```
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>
using namespace std;
typedef long long LL;
const LL N = 5e5 + 7;

struct segTree{
        #define lc (rt<<1)
        #define rc (rt<<1^1)
        #define lson l, m, rt<<1
        #define rson m+1, r, rt<<1^1
        LL M, sum[N], tag[N];
        inline void build(LL n){
                M = 1; while(M<n) M<<=1; if (M!=1) M--;
                memset(tag, 0, sizeof(tag));
                for (LL leaf = M+1; leaf <= n+M; leaf++) scanf("%lld",
&sum[leaf]);
                for (LL leaf = n+1+M; leaf <= (M<<1^1); leaf++)
sum[leaf] = 0;
                for (LL rt = M; rt >= 1; rt--) sum[rt] = sum[lc] +
sum[rc];
        }
```

```
        inline void pushUp(LL rt){
                sum[rt] = sum[lc] + sum[rc];
        }

        inline void pushDown(LL rt, LL len){
                if (tag[rt] == 0) return;
                tag[lc] += tag[rt];
                tag[rc] += tag[rt];
                sum[lc] += tag[rt] * (len>>1);
                sum[rc] += tag[rt] * (len>>1);
                tag[rt] = 0;
        }

        inline void update(LL L, LL R, LL x, LL l, LL r, LL rt){
                //printf("update(%d, %d, %d, %d, %d, %d)\n", L, R, x, l,
r, rt);
                if (L <= l && r <= R){
                        tag[rt] += x;
                        sum[rt] += (r-l+1) * x;
                        return;
                }
                pushDown(rt, r-l+1);
                LL m = (l + r) >> 1;
                if (L <= m) update(L, R, x, lson);
                if (m <  R) update(L, R, x, rson);
                pushUp(rt);
        }

        LL query(LL L, LL R, LL l, LL r, LL rt){
                if (L <= l && r <= R) return sum[rt];
                pushDown(rt, r-l+1);
                LL m = (l + r) >> 1;
                LL ans = 0;
                if (L <= m) ans += query(L, R, lson);
                if (m <  R) ans += query(L, R, rson);
                return ans;
        }
} T;
```

主席树

poj2104 求区间 k 大值

```
# include <cstdio>
# include <cstring>
# include <iostream>
# include <algorithm>
using namespace std;
const int N = 1e5 + 7;
int arr[N];  //arr[]  原数组的数在 rank[]中的位置;
int Rank[N]; //rank[] 原数组离散化

struct ChairTree{
    #define sum(x) tree[x].w
    #define lson tree[rt].lc, tree[rt1].lc, l, m
    #define rson tree[rt].rc, tree[rt1].rc, m+1, r
    struct node{
        int lc, rc, w;
        node(){}
    } tree[N * 20];
    int root[N], cnt;
    void build(){
        root[0] = cnt = 0;
        memset(tree, 0, sizeof(tree));
    }

    void add(int pos, int val, int &rt, int rt1, int l, int r){
        tree[rt = ++cnt] = tree[rt1];
        tree[rt].w += val;
        if (l == r) return;
        int m = (l + r) >> 1;
        if (pos <= m) add(pos, val, lson);
        else add(pos, val, rson);
    }
         //单点查询
    int query(int k, int rt, int rt1, int l, int r){
        if (l == r) return l;
        int lsize = sum(tree[rt1].lc) - sum(tree[rt].lc);
        int m = (l + r) >> 1;
        if (lsize >= k) return query(k, lson);
        else return query(k - lsize, rson);
    }
         //区间查询
```

```
    LL query(int L, int R, int rt, int rt1, int l, int r){
        if (L <= l && r <= R) return sum(rt1) - sum(rt);
        if (sum(rt1) == sum(rt)) return 0;
        LL ans = 0;
        int m = (l + r) >> 1;
        if (L <= m) ans += query(L, R, lson);
        if (m <  R) ans += query(L, R, rson);
        return ans;
    }
} T;

int main(){
    //freopen("in.txt","r",stdin);
    int _, l, r, k, n, q;
    for (; ~scanf("%d%d", &n, &q);){
        T.build();
        for (int i = 1; i <= n; i++) {
            scanf("%d", &arr[i]);
            Rank[i] = arr[i];
        }
        sort(Rank + 1, Rank + n+1);//Rank 存储原值
        int m = unique(Rank + 1, Rank + n +1) - (Rank + 1);//这个 m 很重
要，WA 一天系列
        for (int i = 1; i <= n; i++) {//离散化后的数组，仅仅用来更新
            arr[i] = lower_bound(Rank + 1, Rank + m+1, arr[i]) - Rank;
        }
        for (int i = 1; i <= n; i++){
            T.add(arr[i], 1, T.root[i], T.root[i-1], 1, m);//填 m 别填 n
        }
        for (; q--;){
            scanf("%d%d%d", &l, &r, &k);
            int pos = T.query(k, T.root[l-1], T.root[r], 1, m);
            printf("%d\n", Rank[pos]);
        }
    }
    return 0;
}
```

归并树（区间第 K 大数）

```
#include <iostream>
#include <algorithm>
#include <cstdio>
#include <vector>
using namespace std;
const int ST_SIZE = (1 << 18) - 1;
const int MAXN = 1E5 + 5;
const int MAXM = 5005;
int N, M;
int A[MAXN]; // 要处理的数组
int I[MAXM], J[MAXM], K[MAXM]; //存查询
int nums[MAXN]; //对 A 排序后的数组
vector<int> dat[ST_SIZE]; //线段树节点数组
//构建线段树
// k 是节点编号， 和区间[l, r)对应
void init(int k, int l, int r) {
    if (r - l == 1) {
        dat[k].push_back(A[l]);
    } else {
        int lch = k * 2 + 1, rch = 2 * k + 2;
        init(lch, l, (l + r) / 2);
        init(rch, (l + r) / 2, r);
        dat[k].resize(r - l); //也不知道有什么用，估计是缩小空间吧
        //利用 STL 的 merge 函数把两个儿子的数列合并
        merge(dat[lch].begin(), dat[lch].end(), dat[rch].begin(),
dat[rch].end(), dat[k].begin());
    }

}
//计算[i, j)中不超过 x 的数的个数
//k 是节点编号， 和区间[l, r)对应 (一开始 k l r 0 0 N)
int query(int i, int j, int x, int k, int l, int r) {
    if (j <= l || r <= i) return 0;//完全不相交
    if (i <= l && r <= j) {
        return upper_bound(dat[k].begin(), dat[k].end(), x) -
dat[k].begin();//完全包含
    int lc = query(i, j, x, k * 2 + 1, l, (l + r) / 2);
    int rc = query(i, j, x, k * 2 + 2, (l + r) / 2, r);
    return lc + rc;
}

void solve() {
    for(int i = 0; i < N; i++) nums[i] = A[i];
    sort(nums, nums + N);
```

```
    init(0, 0, N);
    for (int i = 0; i < M; i++) { //[l, r)
        int l = I[i] - 1, r = J[i], k = K[i];
        int lb = -1, ub = N - 1; //(-1, N-1]
        while(ub - lb > 1) {
            int md = (ub + lb) / 2;
            int ans = query(l, r, nums[md], 0, 0, N);
            if (ans >= k) ub = md;
            else lb = md;
        }
        printf("%d\n", nums[ub]);
    }
}

int main(){
    cin >> N >> M;
    for (int i = 0; i < N; i++) scanf("%d", A + i);
    for (int i = 0; i < M; i++) {
        scanf("%d%d%d", I + i, J + i, K + i);
    }
    solve();
    return 0;
}
```

Treap

```
#include<cstdio>
#include<cstring>
#include<cstdlib>
#include<cassert>
using namespace std;
struct Node{
    Node *ch[2];
    int r, v, s;//s 表示节点数

    Node(int v):v(v){
        ch[0]=ch[1]=NULL;
        r = rand();//在 cstdlib 头声明
        s = 1;
    }

    int cmp(int x){
        if (x == v) return -1;
        return x<v ? 0 : 1;
```

```
    }
    void maintain(){
        s = 1;
        if(ch[0]!=NULL) s+=ch[0]->s;
        if(ch[1]!=NULL) s+=ch[1]->s;
    }
}; //root 全局使用的话可以在这里跟上*root

void rotate(Node* &o,int d){
    Node *k=o->ch[d^1];
    o->ch[d^1]=k->ch[d];
    k->ch[d]=o;
    o->maintain();
    k->maintain();
    o=k;
}
void insert(Node* &o,int x){//o 子树中事先不存在 x
    if(o==NULL) o=new Node(x);
    else{
        //如这里改成 int d=o->cmp(x);
        //就不可以插入相同的值，因为 d 可能为-1
        int d=x<(o->v)?0:1;
        insert(o->ch[d],x);
        if(o->ch[d]->r > o->r)
            rotate(o,d^1);
    }
    o->maintain();
}
void remove(Node* &o,int x){
    if (o==NULL) return ;//空时返回
    int d=o->cmp(x);
    if (d == -1){
        Node *u=o;
        if(o->ch[0] && o->ch[1]){
            int d2=(o->ch[0]->r < o->ch[1]->r)?0:1;
            rotate(o,d2);
            remove(o->ch[d2],x);
        }else{
            if(o->ch[0]==NULL) o=o->ch[1];
            else o=o->ch[0];
            delete u;//这个要放里面
        }
    }
    else remove(o->ch[d],x);
```

```
    if(o) o->maintain();//之前 o 存在,但是删除节点后 o 可能就是空 NULL 了,所以需要先判断 o 是否为空
}
//返回关键字从小到大排序时的第 k 个值
//若返回第 K 大的值，只需要把 ch[0]和 ch[1]全互换就可以了
int kth(Node* o,int k){
    assert(o && k>=1 && k<=o->s);//保证输入合法,根据实际问题返回
    int s=(o->ch[0]==NULL)?0:o->ch[0]->s;
    if(k==s+1) return o->v;
    else if(k<=s) return kth(o->ch[0],k);
    else return kth(o->ch[1],k-s-1);
}
//返回值 x 在树中的排名,就算 x 不在 o 树中也能返回排名
//返回值范围在[1,o->s+1]范围内
int rank(Node* o,int x){
    if(o==NULL) return 1;//未找到 x;
    int num= o->ch[0]==NULL ? 0:o->ch[0]->s;
    if(x==o->v) return num+1;
    else if(x < o->v) return rank(o->ch[0],x);
    else return rank(o->ch[1],x)+num+1;
}

int main(){
    int n=0, v;
    while(scanf("%d",&n)==1 && n){
        Node *root=NULL; //初始化为 NULL
        for(int i=0; i<n; i++){
            int x;
            scanf("%d",&x);
            if(root==NULL) root=new Node(x);
            else insert(root,x);
        }
        while(scanf("%d",&v)==1){
            printf("%d\n",rank(root,v));
        }
    }
    return 0;
}
```

分块

分块入门 1

给出一个长为 n 的数列，以及 n 个操作，操作涉及区间加法，单点查值。

```
int n, blo;
int v[50005],bl[50005],atag[50005];
void add(int a, int b, int c){
    for(int i=a;i<=min(bl[a]*blo,b);i++) v[i]+=c;
    if(bl[a]!=bl[b])
        for(int i=(bl[b]-1)*blo+1;i<=b;i++) v[i]+=c;
    for(int i=bl[a]+1;i<=bl[b]-1;i++) atag[i]+=c;
}
int main(){
    n=read(); blo=sqrt(n);
    for(int i=1;i<=n;i++)v[i]=read();
    for(int i=1;i<=n;i++)bl[i]=(i-1)/blo+1;
    for(int i=1;i<=n;i++){
        int f=read(),a=read(),b=read(),c=read();
        if(f==0)add(a,b,c);
        if(f==1)printf("%d\n",v[b]+atag[bl[b]]);
    }
    return 0;
}
```

分块入门 2

给出一个长为 n 的数列，以及 n 个操作，操作涉及区间加法，询问区间内小于某个值 x 的元素个数。

```
int n,blo;
int v[50005], bl[50005], atag[50005];
vector<int>ve[505];
void reset(int x){
    ve[x].clear();
    for(int i=(x-1)*blo+1;i<=min(x*blo,n);i++)
        ve[x].push_back(v[i]);
    sort(ve[x].begin(),ve[x].end());
}

void add(int a,int b,int c){
```

```
    for(int i=a;i<=min(bl[a]*blo,b);i++)
        v[i]+=c;
    reset(bl[a]);
    if(bl[a]!=bl[b]){
        for(int i=(bl[b]-1)*blo+1;i<=b;i++)v[i]+=c;
        reset(bl[b]);
    }
    for(int i=bl[a]+1;i<=bl[b]-1;i++)
        atag[i]+=c;
}

int query(int a,int b,int c){
    int ans=0;
    for(int i=a;i<=min(bl[a]*blo,b);i++)
        if(v[i]+atag[bl[a]]<c)ans++;
    if(bl[a]!=bl[b])
        for(int i=(bl[b]-1)*blo+1;i<=b;i++)
            if(v[i]+atag[bl[b]]<c)ans++;
    for(int i=bl[a]+1;i<=bl[b]-1;i++){
        int x=c-atag[i];
        ans+=lower_bound(ve[i].begin(),ve[i].end(),x)-ve[i].begin();
    }
    return ans;
}

int main(){
    n=read();blo=sqrt(n);
    for(int i=1;i<=n;i++)v[i]=read();
    for(int i=1;i<=n;i++){
        bl[i]=(i-1)/blo+1;
        ve[bl[i]].push_back(v[i]);
    }
    for(int i=1;i<=bl[n];i++)
        sort(ve[i].begin(),ve[i].end());
    for(int i=1;i<=n;i++){
        int f=read(),a=read(),b=read(),c=read();
        if(f==0)add(a,b,c);
        if(f==1)printf("%d\n",query(a,b,c*c));
    }
    return 0;
}
```

分块入门 3

给出一个长为 n 的数列，以及 n 个操作，操作涉及区间加法，询问区间内小于某个值 x 的前驱（比其小的最大元素）。

```
int n,blo;
int v[100005],bl[100005],atag[100005];
set<int>st[105];
void add(int a,int b,int c){
    for(int i=a;i<=min(bl[a]*blo,b);i++){
        st[bl[a]].erase(v[i]);
        v[i] += c;
        st[bl[a]].insert(v[i]);
    }
    if(bl[a]!=bl[b]){
        for(int i=(bl[b]-1)*blo+1;i<=b;i++){
            st[bl[b]].erase(v[i]);
            v[i] += c;
            st[bl[b]].insert(v[i]);
        }
    }
    for(int i=bl[a]+1;i<=bl[b]-1;i++)
        atag[i]+=c;
}
int query(int a,int b,int c){
    int ans=-1;
    for(int i=a;i<=min(bl[a]*blo,b);i++){
        int val=v[i]+atag[bl[a]];
        if(val<c)ans=max(val,ans);
    }
    if(bl[a]!=bl[b])
        for(int i=(bl[b]-1)*blo+1;i<=b;i++){
            int val=v[i]+atag[bl[b]];
            if(val<c)ans=max(val,ans);
        }
    for(int i=bl[a]+1;i<=bl[b]-1;i++){
        int x=c-atag[i];
        set<int>::iterator it=st[i].lower_bound(x);
        if(it==st[i].begin())continue;
        --it;
        ans=max(ans,*it+atag[i]);
    }
    return ans;
}
int main(){
```

```
    n=read();blo=1000;
    for(int i=1;i<=n;i++)v[i]=read();
    for(int i=1;i<=n;i++){
        bl[i]=(i-1)/blo+1;
        st[bl[i]].insert(v[i]);
    }
    for(int i=1;i<=n;i++){
        int f=read(),a=read(),b=read(),c=read();
        if(f==0)add(a,b,c);
        if(f==1)printf("%d\n",query(a,b,c));
    }
    return 0;
}
```

左偏树

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <iostream>
using namespace std;
const int MAXN = 1e5 + 5;
struct node{
    int l,r,dis,key;
} tree[MAXN];
int far[MAXN];
int Find(int x) {
    if(far[x] == x) return x;
    return far[x] = Find(far[x]);
}

int merge(int a,int b){
    if(!a) return b;
    if(!b) return a;
    if(tree[a].key < tree[b].key) swap(a, b);//大堆
    tree[a].r = merge(tree[a].r,b);
    far[tree[a].r] = a;//并查
    if(tree[tree[a].l].dis < tree[tree[a].r].dis)
swap(tree[a].l,tree[a].r);
    if(tree[a].r)tree[a].dis = tree[tree[a].r].dis + 1;
    else tree[a].dis = 0;
    return a;
}
```

```
int pop(int a){
    int l = tree[a].l;
    int r = tree[a].r;
    far[l] = l;//因为要暂时删掉根，所以左右子树先作为根
    far[r] = r;
    tree[a].l = tree[a].r = tree[a].dis = 0;
    return merge(l,r);
}

int main(){
    int N, M;
    while(cin >> N){
        for(int i = 1; i <= N; i++){
            int x;
            far[i] = i;
            scanf("%d", &x);
            tree[i].key = x;
            tree[i].l = tree[i].r = tree[i].dis = 0;
        }
        cin >> M;
        while(M--) {
            int x, y;
            scanf("%d%d", &x, &y);
            x = Find(x);
            y = Find(y);
            if(x == y) {
                printf("-1\n");
            } else {
                int ra = pop(x);
                tree[x].key /= 2;
                ra = merge(ra, x);
                int rb = pop(y);
                tree[y].key /= 2;
                rb = merge(rb, y);
                x = merge(ra, rb);
                printf("%d\n", tree[x].key);
            }
        }
    }
    return 0;
}
```

Splay

```
#include<cstdio>
#include<algorithm>
using namespace std;
struct Node{
    int key;//size
    Node *l,*r,*f;//left,right,father
};
class SplayTree{
public:
    void Init(){rt=NULL;}
    void Zag(Node *x){//left rotate
        Node *y=x->f;//y is the father of x
        y->r = x->l;
        if (x->l)x->l->f = y;//if x has left child
        x->f =y->f;
        if (y->f){//y is not root
            if (y==y->f->l)y->f->l=x;//y if left child
            else y->f->r=x;//y is right child
        }
        y->f=x; x->l=y;
    }
    void Zig(Node *x){//right rotate
        Node *y=x->f;//y is the father of x
        y->l = x->r;
        if (x->r)x->r->f=y;
        x->f = y->f;
        if (y->f){
            if (y==y->f->l)y->f->l=x;
            else y->f->r=x;
        }
        y->f=x; x->r=y;
    }
    void Splay(Node *x){
        while (x->f){
            Node *p=x->f;
            if (!p->f){
                if (x==p->l)Zig(x);
                else Zag(x);
            }else if (x==p->l){
                if (p==p->f->l){Zig(p);Zig(x);}
                else {Zig(x);Zag(x);}
            }else {//x==p->r
```

```
            if (p==p->f->r){Zag(p);Zag(x);}
            else {Zag(x);Zig(x);}
        }
    }
    rt=x;
}
Node *Find(int x){
    Node *T=rt;
    while (T){
        if (T->key==x){Splay(T);return T;}
        else if (x<T->key)T=T->l;
        else T=T->r;
    }
    return T;
}
void Insert(int x){
    Node *T=rt,*fa=NULL;
    while (T){
        fa=T;
        if (x<T->key)T=T->l;
        else if(x>T->key)T=T->r;
        else return ;//two the same keys
    }
    T=(Node*)malloc(sizeof(Node));
    T->key=x;
    T->l=T->r=NULL;
    T->f=fa;
    if (fa){
        if (fa->key>x)fa->l=T;
        else fa->r=T;
    }
    Splay(T);
}
void Delete(int x){
    Node *T=Find(x);
    if (NULL==T)return ;//error
    rt=Join(T->l,T->r);
}
Node *Maxnum(Node *t){
    Node *T=t;
    while (T->r)T=T->r;
    Splay(T);
    return T;
}
```

```
    Node *Minnum(Node *t){
        Node *T=t;
        while (T->l)T=T->l;
        Splay(T);
        return T;
    }
    Node *Last(int x){
        Node *T=Find(x);
        T=T->l;
        return (Maxnum(T));
    }
    Node *Next(int x){
        Node *T=Find(x);
        T=T->r;
        return (Minnum(T));
    }
    Node *Join(Node *t1,Node *t2){
        if (NULL==t1)return t2;
        if (NULL==t2)return t1;
        Node *T=Maxnum(t1);
        T->l=t2;
        return T;
    }
    void Split(int x,Node *&t1,Node *&t2){
        Node *T=Find(x);
        t1=T->l; t2=T->r;
    }
    void Inorder(Node *T){
        if (NULL==T)return ;
        Inorder(T->l);
        printf("%d->",T->key);
        Inorder(T->r);
    }
    void _Delete(){Delete(rt);}
    void Delete(Node *T){
        if (NULL==T)return ;
        Delete(T->l);
        Delete(T->r);
        free(T);
    }
private: Node *rt;//root
};
```

AVL 树

```
//codevs1285 平衡树
//by cww97
#include<cstdio>
#include<iostream>
#include<algorithm>
#define INF 0xfffffff
#define BASE 1000000
using namespace std;
int ans=0;
struct Node{
    int x,bf,h;//bf=balance factor,h=height
    Node *l,*r;
};

class AVLTree{
public:
    void Init() { rt = NULL; }
    int H(Node *T){return (T==NULL)?0:T->h;}
    int BF(Node *l,Node *r){//get balance factor
        if (NULL==l && NULL==r) return 0;
        else if (NULL == l) return -r->h;
        else if (NULL == r) return  l->h;
        return l->h - r->h;
    }

    Node *Lrorate(Node *a){//left rorate
        Node *b;
        b=a->r;
        a->r=b->l;
        b->l=a;
        a->h=max(H(a->l),H(a->r)) + 1;
        b->h=max(H(b->l),H(b->r)) + 1;
        a->bf=BF(a->l,a->r);
        b->bf=BF(b->l,b->r);
        return b;
    }
    Node *Rrorate(Node *a){//right rorate
        Node *b;
        b=a->l;
        a->l=b->r;
        b->r=a;
```

```
    a->h=max(H(a->l),H(a->r)) + 1;
    b->h=max(H(b->l),H(b->r)) + 1;
    a->bf=BF(a->l,a->r);
    b->bf=BF(b->l,b->r);
    return b;
}
Node *LRrorate(Node *a){//left then right
    a->l = Lrorate(a->l);
    Node *c;
    c=Rrorate(a);
    return c;
}
Node *RLrorate(Node *a){//right then left
    a->r=Rrorate(a->r);
    Node *c;
    c=Lrorate(a);
    return c;
}

void Insert(int x){_Insert(rt,x);}
void _Insert (Node *&T,int x){
    if (NULL==T){
        T=(Node*)malloc(sizeof(Node));
        T->x=x;
        T->bf=0;T->h=1;
        T->l=T->r=NULL;
        return ;
    }
    if      (x < T->x) _Insert(T->l,x);
    else if (x > T->x) _Insert(T->r,x);
    else return ; //error :the same y

    T->h=max(H(T->l),H(T->r))+1;//maintain
    T->bf=BF(T->l,T->r);

    if (T->bf > 1 || T->bf < -1){//not balanced
        if      (T->bf > 0 && T->l->bf > 0)T=Rrorate(T);
        else if (T->bf < 0 && T->r->bf < 0)T=Lrorate(T);
        else if (T->bf > 0 && T->l->bf < 0)T=LRrorate(T);
        else if (T->bf < 0 && T->r->bf > 0)T=RLrorate(T);
    }
}

void GetPet(int x){//get pet or person
```

```
        if (NULL==rt){return ;}
        int small=0,large=INF;
        //printf("x=%d\n",x);
        int flag;
        if (Find(rt,x,small,large)){
            printf("find %d\n",x);
            _Delete(rt,x);
        }else if (small==0)flag=1;
        else if (large==INF)flag=0;
        else if (large-x<x-small)flag=1;
        else flag=0;

        if (!flag){//choose large
            _Delete(rt,small);
            ans=(ans+x-small)%BASE;
        }else {
            _Delete(rt,large);
            ans=(ans+large-x)%BASE;
        }
    }
    bool Find(Node *T,int x,int &small,int &large){
        if (NULL==T)return 0;
        if (x==T->x)return 1;
        if (x<T->x){
            large=min(large,T->x);
            return Find(T->l,x,small,large);
        }else{
            small=max(small,T->x);
            return Find(T->r,x,small,large);
        }
    }
    void _Delete(Node *&T,int x){
        if (NULL==T)return ;
        if      (x < T->x){//y at left
            _Delete(T->l,x);
            T->bf=BF(T->l,T->r);
            if (T->bf<-1){
                if (1==T->r->bf)T=RLrorate(T);
                else T=Lrorate(T);//bf==0 or -1
            }
        }else if (x > T->x){//y at right
            _Delete(T->r,x);
            T->bf=BF(T->l,T->r);
            if (T->bf>1){
```

```
                if (-1==T->l->bf)T=LRrorate(T);
                else T=Rrorate(T);//bf==0 or 1
            }
        }else {//here is x
            if (T->l&&T->r){//left &&right
                Node *t=T->l;
                while (t->r)t=t->r;
                T->x=t->x;
                _Delete(T->l,t->x);
                T->bf=BF(T->l,T->r);
                if (T->bf<-1){
                    if (1==T->r->bf)T=RLrorate(T);
                    else T=Lrorate(T);//bf==0 or -1
                }
            }else {//left || right
                Node *t=T;
                if (T->l)T=T->l;
                else if(T->r)T=T->r;
                else {free(T);T=NULL;}
                if (T)free(t);
            }
        }
    }

    //Debug,you will not need it at this problem
    void show(){InOrder(rt);puts("EndShow");}
    void InOrder(Node *T){//print l rt r
        if (NULL==T)return ;
        InOrder(T->l);
        printf("%d ",T->x);
        InOrder(T->r);
    }
    void Free(){FreeTree(rt);}
    void FreeTree(Node *T){
        if (NULL==T)return ;
        FreeTree(T->l);
        FreeTree(T->r);
        free(T);
    }

private:
    Node *rt;//root
};
```

```
int main(){
    freopen("fuck.in","r",stdin);
    int n,x,op,a=0,b=0;
    scanf("%d",&n);
    AVLTree T; T.Init();
    for (;n--;){
        scanf("%d%d",&op,&x);
        //if pets>people put pets into the tree
        //else put people into the tree
        if (op==0){//come a pet
            a++;
            if (a>b)T.Insert(x);//more pet
            else T.GetPet(x);//more people
        }else{//come a person
            b++;
            if (a<b)T.Insert(x);//more people
            else T.GetPet(x);//more pet
        }
    }
    printf("%d\n",ans%BASE);
    T.Free();
    return 0;
}
```

图论

最小生成树（prim）

hdu1102

```
#include<cstdio>
#include<cstring>
#include<iostream>
using namespace std;
const int N=107;
int n,g[N][N];
int prim(){
    int minw[N];//MinWeight
    bool used[N];
    memset(used,0,sizeof(used));
    memset(minw,0x7f,sizeof(minw));
    minw[1]=0;
    int sum=0;
    while (1){
        int v=-1;
        for (int i=1;i<=n;i++){
            if (!used[i]&&(v==-1||minw[i]<minw[v]))v=i;
        }
        if (v==-1)break;
        used[v]=1;
        sum+=minw[v];
        for (int i=0;i<=n;i++){
            minw[i]=min(minw[i],g[v][i]);
        }
    }
    return sum;
}
```

次小生成树

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<climits>
#include<algorithm>
using namespace std;
#define N 510
int map[N][N], lowcost[N], pre[N], max1[N][N], stack[N];
bool visit[N];
int n, m, sum;

void prim(){ //默认 1 在 MST 中
    int temp, k;
    int top; //保存最小生成树的结点

    memset(visit, false, sizeof(visit)); //初始化
    visit[1] = true;
    sum = top = 0;
    for(int i = 1; i <= n; ++i){
        pre[i] = 1;
        lowcost[i] = map[1][i];
    }
    lowcost[1] = 0;
    stack[top++] = 1; //保存 MST 的结点

    for(int i = 1; i <= n; ++i){
        temp = INT_MAX;
        for(int j = 1; j <= n; ++j)
            if(!visit[j] && temp > lowcost[j])
                temp = lowcost[k = j];
        if(temp == INT_MAX) break;
        visit[k] = true;
        sum += temp;
        for(int j = 0; j < top; ++j) //新加入点到 MST 各点路径最大值
            max1[stack[j]][k] = max1[k][stack[j]] =
max(max1[stack[j]][pre[k]], temp);
        stack[top++] = k; //保存 MST 的结点

        for(int j = 1; j <= n; ++j) //更新
            if(!visit[j] && lowcost[j] > map[k][j]){
                lowcost[j] = map[k][j];
```

```
                pre[j] = k; //记录直接前驱
            }
    }
}

int main(){
    int ncase, start, end, cost, minn;
    scanf("%d", &ncase);
    while(ncase--)  {
        for(int i = 1; i < N; ++i) //初始化不为 0,1 必须用循环。。。。
            for(int j = 1; j < N; ++j){
                map[i][j] = INT_MAX;
                max1[i][j] = 0;
            }
        scanf("%d%d", &n, &m);
        for(int i = 1; i <= m; ++i){
            scanf("%d%d%d", &start, &end, &cost);
            //if(cost < map[start][end])(POJ 竟然出现重边的时候不选择最小的
~~~)
            map[start][end] = map[end][start] = cost;
        }
        prim();
        minn = INT_MAX;
        for(int i = 1; i <= n; ++i)
            for(int j = 1; j <= n; ++j)
                if(i != j && i != pre[j] && j != pre[i])
//枚举 MST 以外的边
                    minn = min(minn, map[i][j] - max1[i][j]);
//求出{MST 外加入边-MST 环上权值最大边}最小值
        if(minn != 0) printf("No\n");
        else printf("Yes\n");
    }
    return 0;
}
```

最短路（SPFA）

```
#include <queue>
#include <cstdio>
#include <vector>
#include <cstring>
#include <iostream>
#include <algorithm>
using namespace std;
const int N = 100007;
const int INF=0x3f3f3f3f;
struct SPFA{
    struct Edge{
        int from,to,cost,a,b;
        Edge(){}
        Edge(int x,int y,int z,int a,int b):
            from(x),to(y),cost(z),a(a),b(b){}
        bool canPass(){return a>=cost;}//题目里的
        void read(){scanf("%d%d%d%d%d",&from,&to,&a,&b,&cost);}
    };
    int n, m;
    vector <Edge> edges;
    vector <int > G[N];
    inline void AddEdge(){
        Edge e; e.read();
        if (!e.canPass())return ;
        edges.push_back(e);
        int top = edges.size();
        G[e.from].push_back(top-1);
    }
    inline void Init(int n,int m){
        this -> n = n; this -> m = m;
        edges.clear();
        for (int i=0;i<=n;i++)G[i].clear();
        for (int i=1;i<=m;i++) AddEdge();
    }
    bool inq[N];
    int d[N],cnt[N];
    inline int spfa(int s,int t){
        queue<int> Q;
        memset(inq, 0, sizeof(inq));
        memset(cnt, 0, sizeof(cnt));
        memset( d ,INF,sizeof( d ));
```

```
        d[s] = 0;inq[s]=1;Q.push(s);
        for (;!Q.empty();){
            int u = Q.front();Q.pop();
            inq[u]= 0;
            for (int i=0;i<G[u].size();i++){
                Edge &e = edges[G[u][i]];
                int val = d[u],s = d[u] ;//
                val %= (e.a + e.b);//特殊题目
                if (val>e.a) s+=e.b-(val-e.a);//
                else if (e.a<val+e.cost)s+=e.a+e.b-val;//
                if (d[u]<INF&&s+e.cost<d[e.to]){
                    d[e.to] = s + e.cost;
                    if (!inq[e.to]){
                        Q.push(e.to);inq[e.to] = 1;
                        if (++cnt[e.to]>n)return INF;
                    }
                }
            }
        }
        return d[t];
    }
}g;
int main(){
    int n,m,s,t;
    for (int T=0;~scanf("%d%d%d%d",&n,&m,&s,&t);){
        g.Init(n,m,s,t);
        int ans = g.spfa(s,t);
        printf("Case %d: %d\n",++T,ans);
    }
    return 0;
}
bool find_negative_loop() {//找负环
        memset(d, 0, sizeof(d));
        for(int i = 0; i < V; i++) {
                for(int j = 0; j < E; j++) {
                        edge e = es[j];
                        if(d[e.to] > d[e.form] + e.cost) {
                                d[e.to] = d[e.from] + e.cost;
                                if(i == V - 1) return true;
                        }
                }
        }
        return false;
}
```

最短路 Dijkstra 同时求解次短路

```
#include <cstdio>
#include <cstring>
#include <queue>
#include <algorithm>
using namespace std;
typedef long long LL;
typedef pair<LL, int> P;
const int INF = 0x3f3f3f3f3f3f3f3f;
const int N = 2e5 + 7;

struct Edge{
    int to;
    LL cost;
    Edge(int tv = 0, LL tc = 0):to(tv), cost(tc){}
};
vector<Edge> G[N];
int n, m;
LL dist[N];     //最短距离
LL dist2[N];    //次短距离

LL Dijkstra(){
    memset(dist, INF, sizeof(dist));
    memset(dist2, INF, sizeof(dist2));
    //从小到大的优先队列
    //使用 pair 而不用 edge 结构体
    //是因为这样我们不需要重载运算符
    //pair 是以 first 为主关键字进行排序
    priority_queue<P, vector<P>, greater<P> > Q;
    //初始化源点信息
    dist[1] = 0;
    Q.push(P(0, 1));
    //同时求解最短路和次短路
    for(; !Q.empty();){
        P p = Q.top(); Q.pop();
        //first 为 s->to 的距离，second 为 edge 结构体的 to
        int v = p.second;
        LL d = p.first;
        //当取出的值不是当前最短距离或次短距离，就舍弃他
        if (dist2[v] < d) continue;
        for (int i = 0; i < G[v].size(); i++){
            Edge &e = G[v][i];
```

```
            LL d2 = d + e.cost;
            if (dist[e.to] > d2){
                swap(dist[e.to], d2);
                Q.push(P(dist[e.to], e.to));
            }
            //printf("dist2[%d] = %d, d2 = %d\n", e.to, dist2[e.to], d2);
            if (dist2[e.to] > d2 && dist[v] < d2){
                dist2[e.to] = d2;
                Q.push(P(dist2[e.to], e.to));
            }
        }
    }
    return dist2[n];
}
```

多源最短路(Floyed)

```
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>
using namespace std;
const int INF = 0x3f3f3f3f;
const int N = 200;
int n, G[N][N], d[N][N];
inline void Init(int n){
    this -> n = n;
    memset(G,INF,sizeof(G));
    memset(d,INF,sizeof(d));
}
inline void AddEdge(int f,int t){
    G[f][t] = d[f][t] = 1;
}
inline void floyed(){
    for (int k=1;k<=n;k++)
      for (int i=1;i<=n;i++)if (i!=k)
        for (int j=1;j<=n;j++)if (j!=i&&j!=k)
          d[i][j]=min(d[i][j], d[i][k]+d[k][j]);
}
```

欧拉回路 dfs

```
const int N = 9999,INF=0x3f3f3f3f;
struct EulerCircle{
    struct Edge{
        int to,nxt;
        Edge(){}
        Edge(int x,int y):to(x),nxt(y){}
    }edges[N];
    int head[N],n,E;
    inline void Init(const int &n){
        this->n = n; E = 0;
        for (int i=0;i<=n;i++)head[i]=-1;
    }
    inline void AddEdge(int f,int t){
        edges[++E] = Edge(t,head[f]);
        head[f] = E ;
    }
    stack<int >S;
    bool vis[N];//use when dfs
    inline void dfs(int x){//get EulerCircle
        Edge e;
        for (int i=head[x];i!=-1;i=e.nxt){
            e = edges[i];
            if (vis[i])continue;
            vis[i] = 1;
            dfs(e.to);
            S.push(x);
        }
    }
    inline void getEulerCircle(){
        while (!S.empty())S.pop();
        memset(vis,0,sizeof(vis));
        dfs(1);
        for (;!S.empty();S.pop())
            printf("%d ",S.top());
        puts("1");
    }
} g ;
```

混合图欧拉回路

解析见紫书 376

ATTENTION:需要注意的是，网络流里是有反向边的，dinic 跑完之后反向边不要添加到新图里面了
加到 Dinic 里面

```
    inline void buildEuler(int n){
        for (int i=1;i<=n;i++){
            for (int nxt,j=head[i];j!=-1;j=nxt){
                Edge &e = edges[j]; nxt = e.nxt;
                if (e.to==s||e.to==t) continue ;
                if (!e.cap)continue;
                if (e.flow==e.cap)gg.AddEdge(e.to,e.from);
                else gg.AddEdge(e.from, e.to);
            }
        }
    }
```

main 哇哦

```
int d[N];//degree = out - in
bool work(int n){
    int flow = 0;
    for (int i=1;i<=n;i++){
        if (d[1]&1)return 0;
        if (d[i]>0){
            g.AddEdge(g.s,i,d[i]>>1);
            flow += d[i]>>1;
        }else if (d[i]<0)
            g.AddEdge(i,g.t,-(d[i]>>1));
    }
    if (flow != g.maxFlow()) return 0;
    return 1;
}

int main(){
    //freopen("in.txt","r",stdin);
    int T,x,y,n,m;
    scanf("%d",&T);
    for (char ch;T--;){
        scanf("%d%d",&n,&m);
        g.Init(n,0,n+1);
```

```
        gg.Init(n);
        memset(d,0,sizeof(d));
        for (int i=1;i<=m;i++){
            scanf("%d%d %c\n",&x,&y,&ch);
            if (ch=='D') gg.AddEdge(x,y);
            else g.AddEdge(x,y,1);
            d[x]++;d[y]--;//Degree
        }
        if (!work(n))puts("No euler circuit exist");
        else {
            g.buildEuler(n);
            gg.getEulerCircle();
        }
        if (T)puts("");
    }
    return 0;
}
```

拓扑排序 topsort

```
#include <iostream>
#include <cstring>
#include <queue>
#include <cstdio>
using namespace std;
const int maxn=107;
int x,y,m,n;
bool f[maxn][maxn];
int in[maxn];

int main(){
    //freopen("fuck.in" ,"r",stdin);
    while(scanf("%d%d",&n,&m)==2&&(m||n)){
        memset(f ,0,sizeof(f ));
        memset(in,0,sizeof(in));
        for(;m--;){
            scanf("%d%d",&x,&y);
            f[x][y] = 1;
            in[y]++;
        }

        int A=0,ans[maxn];
        queue<int>Q;
```

```
        for (int i=1;i<=n;i++)if (in[i]==0)Q.push(i);
        while(!Q.empty()){
            int x=Q.front(); Q.pop();
            ans[++A]=x;
            for(int j=1;j<=n;j++)if (f[x][j]){
                if (--in[j]==0)Q.push(j);
            }
        }
        for (int i=1;i<A;i++)printf("%d ",ans[i]);
        printf("%d\n",ans[A]);
    }
    return 0;
}
```

最大流（Dinic）

```
#include<queue>
#include<stack>
#include<cstdio>
#include<vector>
#include<cstring>
#include<iostream>
using namespace std;
typedef long long LL;
const int INF=0x3f3f3f3f;
const int N = 9999;
struct Dinic{
    struct Edge{
        int from,to,cap,flow,nxt;
        Edge(){}
        Edge(int u,int v,int c,int f,int n):
            from(u),to(v),cap(c),flow(f),nxt(n){}
    }edges[N];
    int n, s, t, E, head[N];
    bool vis[N]; //use when bfs
    int d[N],cur[N];//dist,now edge,use in dfs
    inline void AddEdge(int f,int t,int c){
        edges[++E] = Edge(f,t,c,0,head[f]);
        head[f] = E;
        edges[++E] = Edge(t,f,0,0,head[t]);
        head[t] = E;
    }
    inline void Init(int n,int s,int t){
```

```
        this -> n = n ; E = -1;
        this -> s = s ; head[s] = -1;
        this -> t = t ; head[t] = -1;
        for (int i=0;i<=n;i++) head[i] = -1;
    }
    inline bool BFS(){
        memset(vis,0,sizeof(vis));
        queue<int >Q;
        d[s] = 0; vis[s] = 1;
        for (Q.push(s);!Q.empty();){
            int x = Q.front(); Q.pop();
            for (int nxt,i = head[x];i!=-1;i = nxt){
                Edge &e = edges[i]; nxt = e.nxt;
                if (vis[e.to]||e.cap<=e.flow)continue;
                vis[e.to]=1;
                d[e.to]=d[x]+1;
                Q.push(e.to);
            }
        }
        return vis[t];
    }
    inline int DFS(const int& x,int a){
        if (x==t||a==0){return a;}
        int flow = 0, f, nxt;
        for (int& i=cur[x];i!=-1;i=nxt){
            Edge& e = edges[i]; nxt = e.nxt;
            if (d[x]+1!=d[e.to])continue;
            if ((f=DFS(e.to,min(a,e.cap-e.flow)))<=0)continue;
            e.flow += f;
            edges[i^1].flow-=f;//c^
            flow+=f; a-=f;
            if (!a) break;
        }
        return flow;
    }
    inline int maxFlow(){return maxFlow(s,t);}
    inline int maxFlow(int s, int t){
        int flow = 0;
        for (;BFS();){
            for (int i=0;i<=n;i++)cur[i]=head[i];
            flow += DFS(s,INF) ;
        }
        return flow;
    }
```

```
    inline void buildEuler(int n){
        for (int i=1;i<=n;i++){
            for (int nxt,j=head[i];j!=-1;j=nxt){
                Edge &e = edges[j]; nxt = e.nxt;
                if (e.to==s||e.to==t) continue ;
                if (!e.cap)continue;
                if (e.flow==e.cap)gg.AddEdge(e.to,e.from);
                else gg.AddEdge(e.from, e.to);
            }
        }
    }
} g ;
```

费用流（SPFA）

```
#include<queue>
#include<cmath>
#include<cstdio>
#include<vector>
#include<cstring>
#include<iostream>
#include<algorithm>
using namespace std;
typedef long long LL;
const int N = 2007;
const int INF=0x3f3f3f3f;
const double EPS = 1e-6;
struct MCMF{
    struct Edge{
        int from,to,cap,flow,nxt;
        double cost;
        Edge(){}
        Edge(int x,int y,int z,int u,double v,int n){
            from=x;to=y;cap=z;flow=u;cost=v;nxt=n;
        }
    }edges[N];
    int E,head[N],n,s,t,inq[N],p[N],a[N];
    double d[N];
    inline void Init(int n,int s,int t){
        this->n = n; E = -1;
        this->s = s; this->t = t;
        memset(head,-1,sizeof(head));
    }
```

```
    inline void AddEdge(int f,int t,int c,double w){
        edges[++E] = Edge(f,t,c,0, w,head[f]);
        head[f] = E;
        edges[++E] = Edge(t,f,0,0,-w,head[t]);
        head[t] = E;
    }
    bool spfa(int s,int t,int flow,double &cost){
        for (int i=0;i<=n;i++)d[i]=INF;
        memset(inq,0,sizeof(inq));
        d[s]=0;inq[s]=1;p[s]=0;a[s]=INF;
        queue<int>Q;Q.push(s);
        for (;!Q.empty();){
            int nxt, u =Q.front();Q.pop();inq[u]=0;
            for (int i=head[u];i!=-1;i=nxt){
                Edge &e = edges[i]; nxt = e.nxt;
                if (e.cap<=e.flow||d[e.to]<=d[u]+e.cost)continue;
                d[e.to] = d[u] + e.cost;
                p[e.to] = i;
                a[e.to] = min(a[u],e.cap-e.flow);
                if (!inq[e.to]){Q.push(e.to);inq[e.to]=1;}
            }
        }
        if (d[t]==INF)return 0;//false
        flow += a[t];
        cost += (double)d[t]*(double)a[t];
        for (int u=t;u!=s;u=edges[p[u]].from){
            edges[p[u]  ].flow += a[t];
            edges[p[u]^1].flow -= a[t];
        }
        return 1;//true
    }
    double mcmf(){//需要保证初始网络中没有负权
        int flow =0;
        double cost = 0;
        for (;spfa(s,t,flow,cost););
        return cost;
    }//MinCostMaxFlow
} g ;
```

强连通分量 Tarjan

hdu5934

```
#include <cmath>
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>
using namespace std;
typedef long long LL;
const LL N = 1e3 + 7;
const LL INF = 1e8 + 7;
struct bomb{
        LL x, y, r, c;
        void read(){
                scanf("%lld%lld%lld%lld", &x, &y, &r, &c);
        }
} bombs[N];
LL sqr(LL x){return x*x;}
double dist(LL i, LL j){
        return sqrt(sqr(bombs[i].x - bombs[j].x)
                +sqr(bombs[i].y - bombs[j].y));
}
LL n;
struct tarjan {//复杂度 O(N+M)
        const static LL MAXN = N;  //点数
        const static LL MAXM = N*N;//边数
        struct Edge{
                LL to,next;
        } edge[MAXM];
        LL head[MAXN], tot;
        LL Low[MAXN],DFN[MAXN],Stack[MAXN],Belong[MAXN];//Belong 数组的值
是 1~scc
        LL Index,top;
        LL scc;//强连通分量的个数
        LL num[MAXN];//各个强连通分量包含点的个数，数组编号 1~scc //num 数组
不一定需要，结合实际情况
        bool Instack[MAXN];
        void addedge(LL u,LL v){
                edge[tot].to = v;
                edge[tot].next = head[u];
                head[u] = tot++;
        }
```

```
        void Tarjan(LL u){
                LL v;
                Low[u] = DFN[u] = ++Index;
                Stack[top++] = u;
                Instack[u] = true;
                for(LL i = head[u]; ~i; i = edge[i].next){
                        v = edge[i].to;
                        if( !DFN[v] ){
                                Tarjan(v);
                                Low[u] = min(Low[u], Low[v]);
                        }
                        else if(Instack[v] && Low[u] > DFN[v])
                                Low[u] = DFN[v];
                }
                if(Low[u] == DFN[u]){
                        scc++;
                        do{
                                v = Stack[--top];
                                Instack[v] = false;
                                Belong[v] = scc;
                                num[scc]++;
                        }while( v != u);
                }
        }

        LL in[MAXN];
        LL solve(LL N){
                memset(DFN, 0, sizeof(DFN));
                memset(num, 0, sizeof(num));
                memset(Instack, 0, sizeof(Instack));
                Index = scc = top = 0;
                for(LL i = 1; i <= N; i++) if(!DFN[i]) Tarjan(i);

                //for this problem
                memset(in, 0, sizeof(in));
                for (LL u = 1; u <= n; u++){
                        for (LL e = head[u]; ~e; e = edge[e].next){
                                LL v = edge[e].to;
                                if (Belong[u] != Belong[v])
in[Belong[v]]++;
                        }
                }
                LL ans = 0;
```

```
                for (LL i = 1; i <= scc; i++) if (in[i] == 0){
                        LL cost = INF;
                        for (LL j = 1; j <= n; j++) if (Belong[j] == i){
                                cost = min(cost, bombs[j].c);
                        }
                        ans += cost;
                }
                return ans;
        }

        void init(){
                tot = 0;
                memset(head, -1, sizeof(head));
        }

} g;

int main(){
        //freopen("in.txt", "r", stdin);
        LL T;
        scanf("%lld", &T);
        for (LL cas = 1; cas <= T; cas++){
                printf("Case #%lld: ", cas);
                scanf("%lld", &n);
                for (LL i = 1; i <= n; i++){
                        bombs[i].read();
                }
                g.init();
                for (LL i = 1; i < n; i++){
                        for (LL j = i+1; j <= n; j++){
                                double d = dist(i, j);
                                if (bombs[i].r >= d) g.addedge(i, j);
                                if (bombs[j].r >= d) g.addedge(j, i);
                        }
                }
                LL ans = g.solve(n);
                printf("%lld\n", ans);
        }
        return 0;
}
```

倍增 LCA + 最大生成树(truck)

```
#include<cstdio>
#include<vector>
#include<cstring>
#include<iostream>
#include<algorithm>
using namespace std;
const int INF=0x3f3f3f3f;
const int N = 1e5 + 5;
int n,m;

struct gragh{
    struct Edge{
        int from,to,w;
        Edge(){}
        Edge(int x,int y,int z):from(x),to(y),w(z){}
        bool operator < (const Edge& a)const{
            return w < a.w;
        }
    }edges[N],be[N];
    int E,f[N],fa[N][20],di[N][20],dep[N];
    bool vis[N];
    vector<int >G[N];
    int F(int x){//鼗
        return f[x]==x?x:(f[x]=F(f[x]));
    }

    inline void link(int x,int y,int z){
        edges[++E]=Edge(x,y,z);
        G[x].push_back(E);
    }

    void build(){
        E=0;
        for (int i=1;i<=n;i++)G[i].clear();
        int x,y,z;
        for (int i=1;i<=n;i++)f[i]=i;
        for (int i=1;i<=m;i++){
            scanf("%d%d%d",&x,&y,&z);
            be[i]=Edge(x,y,z);
            f[F(x)]=F(y);
        }
```

```
}

void kruskal(){
    int treenum = 0;//forests
    memset(vis,0,sizeof(vis));
    for (int i=1;i<=n;i++)if (!vis[F(i)]){
        treenum++;vis[F(i)]=1;
    }
    for (int i=1;i<=n;i++)f[i]=i;
    sort(be+1,be+m+1);
    int cnt = 0;
    for (int i=m;i>=1;i--){
        int x = be[i].from;
        int y = be[i].to  ;
        if (F(x)==F(y))continue;
        f[F(x)]=F(y);
        cnt++;
        link(x,y,be[i].w);
        link(y,x,be[i].w);
        if (cnt==n-treenum)break;
    }
}

void dfs(int x){
    vis[x] = 1;
    for (int i=1;i<=17;i++){
        if(dep[x]<(1<<i))break;
        fa[x][i]=fa[fa[x][i-1]][i-1];
        di[x][i]=min(di[x][i-1],di[fa[x][i-1]][i-1]);
    }
    for (int i=0;i<G[x].size();i++){
        Edge e = edges[G[x][i]];
        if (vis[e.to])continue;
        fa[e.to][0] = x;
        di[e.to][0] = e.w;
        dep[e.to] = dep[x]+1;
        dfs(e.to);
    }
}

int lca(int x,int y){
    if (dep[x]<dep[y])swap(x,y);
    int t = dep[x] - dep[y];
    for (int i=0;i<=17;i++)
```

```
            if ((1<<i)&t) x = fa[x][i];
        for (int i=17;i>=0;i--)
            if (fa[x][i]!=fa[y][i]){
            x=fa[x][i];y=fa[y][i];
        }
        if (x==y)return x;
        return fa[x][0];
    }

    int ask(int x,int f){//f:father
        int ans = INF;
        int t = dep[x]-dep[f];
        for (int i=0;i<=17;i++)if(t&(1<<i)){
            ans=min(ans,di[x][i]);
            x = fa[x][i];
        }
        return ans;
    }

    void work(){
        build();
        kruskal();
        memset(vis,0,sizeof(vis));
        for (int i=1;i<=n;i++)if(!vis[i])dfs(i);
        int q,x,y;
        scanf("%d",&q);
        while (q--){
            scanf("%d%d",&x,&y);
            if (F(x)!=F(y))puts("-1");
            else {
                int t = lca(x,y);
                x = ask(x,t);
                y = ask(y,t);
                printf("%d\n",min(x,y));
            }
        }
    }
}g;

int main(){
    for (;~scanf("%d%d",&n,&m);)g.work();
    return 0;
}
```

树链剖分

```
struct TreeChain{
    struct Edge{
        int from, to, nxt;
        Edge(){}
        Edge(int u, int v, int n):
            from(u), to(v), nxt(n){}
    }edges[N];
    int n, E, head[N];

    int tim;
    int siz[N]; //用来保存以 x 为根的子树节点个数
    int top[N]; //用来保存当前节点的所在链的顶端节点
    int son[N]; //用来保存重儿子
    int dep[N]; //用来保存当前节点的深度
    int fa[N];  //用来保存当前节点的父亲
    int tid[N]; //用来保存树中每个节点剖分后的新编号，线段树
    int Rank[N];//tid 反向数组，不一定需要

    inline void AddEdge(int f, int t){
        edges[++E] = Edge(f, t, head[f]);
        head[f] = E;
    }
    inline void Init(int n){
        tim = 0;
        this -> n = n ; E = -1;
        for (int i = 0; i <= n; i++) head[i] = -1;
        for (int i = 0; i <= n; i++) son[i] = -1;
    }

    void dfs1(int u, int father, int d){
        dep[u] = d;
        fa[u] = father;
        siz[u] = 1;
        int nxt;
        for(int i = head[u]; i != -1; i = nxt){
            Edge &e = edges[i]; nxt = e.nxt;
            if (e.to == father) continue;
            dfs1(e.to, u, d + 1);
            siz[u] += siz[e.to];
            if(son[u]==-1 || siz[e.to] > siz[son[u]]) son[u] = e.to;
        }
```

```
    }
    void dfs2(int u, int tp){
        top[u] = tp;
        tid[u] = ++tim;
        Rank[tid[u]] = u;
        if (son[u] == -1) return;
        dfs2(son[u], tp);
        int nxt;
        for(int i = head[u]; i != -1; i = nxt){
            Edge &e = edges[i]; nxt = e.nxt;
            if(e.to == son[u] || e.to == fa[u]) continue;
            dfs2(e.to, e.to);
        }
    }
    LL query(int u, int v){
        int f1 = top[u], f2 = top[v];
        LL tmp = 0;
        for (; f1 != f2;){
            if (dep[f1] < dep[f2]){
                swap(f1, f2);
                swap(u, v);
            }
            tmp += T.query(tid[f1], tid[u]);
            u = fa[f1]; f1 = top[u];
        }
        if (dep[u] > dep[v]) swap(u, v);
        return tmp + T.query(tid[u], tid[v]);
    }
} g ;
```

树分治

poj1741 模板题

```
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>
using namespace std;
const int N = 1e4 + 7;
const int INF = 0x3f3f3f3f;
int n, k, ans;
```

```
struct Edge{
    int from, to, w, nxt;
    Edge(){}
    Edge (int f, int t, int _w, int n):from(f),to(t),w(_w),nxt(n){}
} edges[N * 2];
bool vis[N];
int head[N], E, siz[N], dep[N];
void Init(){
    E = 0;
    memset(head, -1, sizeof(head));
    memset(vis, false, sizeof(vis));
}
void AddEdge(int u,int v,int w){
    edges[E] = Edge(u, v, w, head[u]);
    head[u] = E++;
}
int dfssize(int u, int pre){
    siz[u] = 1;
    for(int i = head[u]; i != -1; i = edges[i].nxt){
        Edge &e = edges[i];
        if(e.to == pre || vis[e.to])continue;
        siz[u] += dfssize(e.to, u);
    }
    return siz[u];
}
//找重心
void dfsroot(int u, int pre, int totnum, int &minn, int &root){
    int maxx = totnum - siz[u];
    for (int i = head[u]; i != -1;i = edges[i].nxt){
        Edge &e = edges[i];
        if(e.to == pre || vis[e.to]) continue;
        dfsroot(e.to, u, totnum, minn, root);
        maxx = max(maxx, siz[e.to]);
    }
    if(maxx < minn){minn = maxx; root = u;}
}
//求每个点离重心的距离
void dfsdep(int u,int pre,int dist, int &num){
    dep[num++] = dist;
    for(int i = head[u]; i != -1; i = edges[i].nxt){
        Edge &e = edges[i];
        if(e.to == pre || vis[e.to])continue;
        dfsdep(e.to, u, dist + e.w, num);
    }
```

```
}
//计算以 u 为根的子树中有多少点对的距离小于等于 K
int calc(int u, int d){
    //printf("calc(%d, %d)\n", u, d);
    int ans = 0, num = 0;
    dfsdep(u, -1, d, num);
    sort(dep, dep + num);
    int i = 0, j = num - 1;
    for (; i < j; i++){
        while (dep[i]+dep[j]>k && i<j) j--;
        ans += j - i;
    }
    return ans;
}
void solve(int u){
    int Max = N, root, minn = INF;
    int totnum = dfssize(u, -1);
    dfsroot(u, -1, totnum, minn, root);
    ans += calc(root, 0);
    vis[root] = 1;
    for(int i = head[root]; i != -1; i = edges[i].nxt){
        Edge &e = edges[i];
        if (vis[e.to]) continue;
        ans -= calc(e.to, e.w);
        solve(e.to);
    }
}
int main(){
    int u, v, w;
    for (; ~scanf("%d%d", &n, &k) && (n|k);){
        Init();
        for(int i = 1; i < n; i++){
            scanf("%d%d%d", &u, &v, &w);
            AddEdge(u, v, w); AddEdge(v, u, w);
        }
        ans = 0;
        solve(1);
        printf("%d\n", ans);
    }
    return 0;
}
```

图的割点、桥和双连通分支的基本概念

[点连通度与边连通度]

在一个无向连通图中，如果有一个顶点集合，删除这个顶点集合，以及这个集合中所有顶点相关联的边以
后，原图变成多个连通块，就称这个点集为 割点集合。一个图的 点连通度的定义为，最小割点集合中的顶
点数。

类似的，如果有一个边集合，删除这个边集合以后，原图变成多个连通块，就称这个点集为 割边集合。一
个图的 边连通度的定义为，最小割边集合中的边数。

[双连通图、割点与桥]

如果一个无向连通图的点连通度大于 1，则称该图是 点双连通的(point biconnected)，简称 双连通或 重连通。

一个图有割点，当且仅当这个图的点连通度为 1，则割点集合的唯一元素被称为 割点(cut point)，又叫 关节
点(articulation point)。

如果一个无向连通图的边连通度大于 1，则称该图是 边双连通的(edge biconnected)，简称双连通或重连通。

一个图有桥，当且仅当这个图的边连通度为 1，则割边集合的唯一元素被称为 桥(bridge)，又叫 关节边
(articulation edge) 。

可以看出，点双连通与边双连通都可以简称为双连通，它们之间是有着某种联系的，下文中提到的双连通，
均既可指点双连通，又可指边双连通。

[双连通分支]

在图 G 的所有子图 G'中，如果 G'是双连通的，则称 G'为 双连通子图。如果一个双连通子图 G'它不是任何一
个双连通子图的真子集，则 G'为 极大双连 通子图。 双连通分支(biconnected component)，或 重连通分支，
就是图的极大双连通子图。特殊的，点双连通分支又叫做 块。

[求割点与桥]

该算法是 R.Tarjan 发明的。对图深度优先搜索，定义 DFS(u)为 u 在搜索树（以下简称为树）中被遍历到的次
序号。定义 Low(u)为 u 或 u 的子树中能通过非父子边追溯到的最早的节点，即 DFS 序号最小的节点。根据
定义，则有：

Low(u)=Min { DFS(u) DFS(v) (u,v)为后向边(返祖边) 等价于 DFS(v)<DFS(u)且 v 不为 u 的父亲节点 Low(v) (u,v)
为树枝边(父子边) }

一个顶点 u 是割点，当且仅当满足(1)或(2) (1) u 为树根，且 u 有多于一个子树。 (2) u 不为树根，且满足存
在(u,v)为树枝边(或称父子边，即 u 为 v 在搜索树中的父亲)，使得 DFS(u)<=Low(v)。

一条无向边(u,v)是桥，当且仅当(u,v)为树枝边，且满足 DFS(u)<Low(v)。

[求双连通分支]

下面要分开讨论点双连通分支与边双连通分支的求法。

对于点双连通分支，实际上在求割点的过程中就能顺便把每个点双连通分支求出。建立一个栈，存储当前
双连通分支，在搜索图时，每找到一条树枝边或后向边(非横叉边)，就把这条边加入栈中。如果遇到某时满
足 DFS(u)<=Low(v)，说明 u 是一个割点，同时把边从栈顶一个个取出，直到遇到了边(u,v)，取出的这些边与
其关联的点，组成一个点双连通分支。割点可以属于多个点双连通分支，其余点和每条边只属于且属于一
个点双连通分支。
对于边双连通分支，求法更为简单。只需在求出所有的桥以后，把桥边删除，原图变成了多个连通块，则
每个连通块就是一个边双连通分支。桥不属于任何一个边双连通分支，其余的边和每个顶点都属于且只属
于一个边双连通分支。

[构造双连通图]

一个有桥的连通图，如何把它通过加边变成边双连通图？方法为首先求出所有的桥，然后删除这些桥边，
剩下的每个连通块都是一个双连通子图。把每个双连通子图收缩为一个顶点，再把桥边加回来，最后的这
个图一定是一棵树，边连通度为 1。
统计出树中度为 1 的节点的个数，即为叶节点的个数，记为 leaf。则至少在树上添加(leaf+1)/2 条边，就能
使树达到边二连通，所以至少添加的边数就是(leaf+1)/2。具体方法为，首先把两个最近公共祖先最远的两
个叶节点之间连接一条边，这样可以把这两个点到祖先的路径上所有点收缩到一起，因为一个形成的环一
定是双连通的。然后再找两个最近公共祖先最远的两个叶节点，这样一对一对找完，恰好是(leaf+1)/2 次，
把所有点收缩到了一起。

动态规划

各种背包

背包

```
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int N = 100007;
struct node {
    int v,w,n;
    node(){}
    node(int x,int y,int z){ v=x,w=y,n=z;}
}a[N];
int f[N];
int main(){
    //freopen("fuck.in","r",stdin);
    int cash,n,x,y;
    for (;~scanf("%d%d",&cash,&n);){
        int A = 0;
        for(int i=1;i<=n;i++){
            scanf("%d%d",&x,&y);
            for (int t=0;(1<<t)<x;t++){
                int tt=1<<t;
                a[++A]=node(y*tt,y*tt,1);
                x -= tt;
            }
            if (x)a[++A]=node(y*x,y*x,1);
        }
        memset(f,0,sizeof(f));//01 背包
        for (int i=1;i<=A;i++)
            for (int j=cash;j>=a[i].v;j--)
                f[j]=max(f[j],f[j-a[i].v]+a[i].w);
        int ans = 0;//get ans
        for (int i=0;i<=cash;i++) ans=max(ans,f[i]);
        printf("%d\n",ans);
    }
    return 0;
}
```

多重背包通用模板（单调队列）

```
int f[N];
int va[N], vb[N];//MAX_V
void pack(int V,int v,int w,int n){
    if (n==0||v==0) return;
    if (n==1){//01 背包
        for (int i=V;i>=v;--i)
            f[i]=max(f[i],f[i-v]+w);
        return;
    }
    if (n*v>=V-v+1){//多重背包(n >= V / v)
        for (int i=v;i<=V;++i)
            f[i]=max(f[i],f[i-v]+w);
        return;
    }
    for (int j = 0 ; j < v ; ++j ){
        int *pb = va, *pe = va - 1;
        int *qb = vb, *qe = vb - 1;
        for (int k=j,i=0;k<=V;k+=v,++i){
            if (pe==pb+n){
                if(*pb == *qb) ++qb;
                ++pb;
            }
            int tt = f[k] - i * w;
            *++pe = tt;
            while (qe>=qb&& *qe<tt)--qe;
            *++qe = tt;
            f[k] = *qb + i * w;
        }
    }
}

//主程序调用
        memset(f,0,sizeof(f));    //pack
        for (int i=1;i<=n;i++)
            pack(cash,a[i].v,a[i].w,a[i].n);
        int ans = 0;              //getAns
        for (int i=0;i<=cash;i++) ans=max(ans,f[i]);
        printf("%d\n",ans);
```

小价值大重量背包问题

```
#include <stdio.h>
#include <iostream>
#include <algorithm>
#include <cmath>
#include <ctime>
using namespace std;
const int MAXN = 1005, MAXV = 1000005;
int f[MAXV];
int v[MAXN];
int w[MAXN];
const int INF = 0x3f3f3f3f;
int main(){
    int T;
    cin >> T;
    while(T--) {
        int n, V;
        cin >> n >> V;
        fill(f, f + MAXV, INF);
        f[0] = 0;
        for(int i = 0; i < n; i++) scanf("%d", v + i);
        for(int i = 0; i < n; i++) scanf("%d", w + i);
        for(int i = 0; i < n; i++) {
            for(int j = MAXV - 1; j >= v[i] ; j--) {
                f[j] = min(f[j], f[j - v[i]] + w[i]);
            }
        }
        int ans = -1;
        for(int i = MAXV - 1; i >= 0; i--) {
            if (f[i] <= V) {
                ans = i;
                break;
            }
        }
        if (ans != -1) printf("%d\n", ans);
        else printf("No solution\n");
    }
     return 0;
}
```

输出 LCS 序列

```
#include <string.h>
#include <stdio.h>
#include <iostream>
#include <algorithm>
#include <stdlib.h>
#include <string>
#include <vector>
using namespace std;
string s1,s2;
string lcs2(string s1,string s2){
    if(s1==""||s2=="")return "";
    int m=s1.size() + 1;
    int n=s2.size() + 1;
    printf("%d %d\n",m,n);
    int lcs[m][n];
    memset(lcs,0,sizeof(lcs));
    for(int i=1;i<m;i++)
        for(int j=1;j<n;j++){
            if(s1[i-1]==s2[j-1])
                lcs[i][j]=lcs[i-1][j-1]+1;
            else
                lcs[i][j]=lcs[i-1][j]>=lcs[i][j-1]?lcs[i-1][j]:lcs[i][j-
1];//取上侧或左侧的最大值
        }
        int i=m-2;
        int j=n-2;
        string ss="";
        while(i!=-1&&j!=-1) {
            if(s1[i]==s2[j]) {
                //printf("%c\n",s1[i]);
                ss+=s1[i];
                i--; j--;
            } else {
                if(lcs[i+1][j+1]==lcs[i][j]) {
                    i--; j--;
                } else {
                    if(lcs[i][j+1]>=lcs[i+1][j]) i--;
                    else j--;
                }
            }
        }
```

```
        reverse(ss.begin(),ss.end());//将字符串倒置
        return ss;
}
int main(){
    while(cin>>s1>>s2){
        string s=lcs2(s1,s2);
        cout << s<<endl;
    }
    return 0;
}
```

TSP 旅行商问题.(状压 DP)

```
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;

const int MAXN = 16;
const int INF = 0x3f3f3f3f;
int n;
int dp[1 << MAXN][MAXN];
int d[MAXN][MAXN];

int rec(int s, int v) {
    //记忆化
    if (dp[s][v] >= 0) {return dp[s][v];}
    //已经访问过所有节点并返回零号节点
    if (s == (1 << n) - 1 && v == 0) {
        return dp[s][v] = 0;
    }
    int res = INF;
    for (int u = 0; u < n; u++) {
        if (!(s >> u & 1)) {
            //下一步移动到顶点 u
            res = min(res, rec(s | 1 << u, u) + d[v][u]);
        }
    }
    return dp[s][v] = res;
}

int solve() {
    memset(dp, -1, sizeof(dp));
```

```
    printf("%d\n", rec(0, 0));
}

void floyd() {
    for(int k = 0; k < n; k++) {
        for(int i = 0; i < n; i++) {
            for(int j = 0; j < n; j++) {
                if (i == j) d[i][j] = 0;
                else d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
            }
        }
    }
}

int main(){
    int T;
    cin >> T;
    while(T--) {
        memset(d, INF, sizeof(d));
        int m;
        cin >> n >> m;
        for(int i = 0; i < m; i++) {
            int x, y, v;
            scanf("%d%d%d", &x, &y, &v);
            if (d[x - 1][y - 1] > v) d[x - 1][y - 1] = d[y - 1][x - 1] =
v;
        }
        //每个点只能走一次就是原本的 TSP 问题，可以走多次先求各点最短路转化为
TSP
        //道理比如 1 - 3, 1 - 2 那么 1 要走 3 次， TSP 求不出 但是求一波 Floyd
以后， 3 - 2 有条路了，相当于改变了图
        floyd();
        solve();
    }
    return 0;
}
```

DAG 序列反向 DP

```
#include<bits/stdc++.h>
using namespace std;
#define LL long long
const int maxn = 1e5+7;
const int mod = 1e9+7;
vector<int>e[maxn];
LL a[maxn],b[maxn],d[maxn];
LL ans[maxn];
int n,m;
int main(){
        while(scanf("%d%d",&n,&m)!=EOF)  {
                for(int i = 0;i<=n;i++)
                        e[i].clear();
                memset(d,0,sizeof(d));
                memset(ans,0,sizeof(ans));
                for(int i = 1;i<=n;i++)
                        scanf("%lld%lld",&a[i],&b[i]);
                for(int i = 1;i<=m;i++){
                        int u,v;
                        scanf("%d%d",&u,&v);
                        e[v].push_back(u);
                        d[u]++;
                }
                queue<int>q;
                for(int i = 1;i<=n;i++)
                        if(d[i]==0)q.push(i);
                while(!q.empty()){
                        int u = q.front();q.pop();
                        for(int i = 0;i<e[u].size();i++){
                                int v = e[u][i];
                                ans[v]=(ans[v]+(ans[u]+b[u])%mod)%mod;
                                if(--d[v]==0)
                                        q.push(v);
                        }
                }
                LL res = 0;
                for(int i = 1;i<=n;i++)
                        res = (res + 1LL*ans[i]*a[i]%mod)%mod;
                printf("%lld\n",res);
        }
}
```

数位 dp

```
/*****************
        不要 62
******************/
#include<iostream>
#include<cstdio>
#include<cstring>
#include<string>
using namespace std;
typedef int LL;
LL a[30];
LL dp[30][2];
LL z[30] = {1};
LL n;
LL dfs(LL pos, LL stat, bool limit) {
    if(pos == -1) return 1;
    if(!limit && dp[pos][stat] != -1) return dp[pos][stat];
    LL up = limit ? a[pos] : 9;
    LL ans = 0;
    for(LL i = 0; i <= up; i++) {
        if(i == 2 && stat) continue;
        if(i == 4) continue;
        ans += dfs(pos - 1, i == 6, limit && i == a[pos]);
    }
    if(!limit) dp[pos][stat] = ans;
    return ans;
}
LL solve(LL x) {
    LL pos = 0;
    while(x) {
        a[pos++] = x % 10;
        x /= 10;
    }
    return dfs(pos - 1, 0, true);
}
int main()
{
    memset(dp, -1, sizeof(dp));
    LL n1, n2;
    while(scanf("%d%d", &n1, &n2), n1 + n2) {
        printf("%d\n", solve(n2) - solve(n1 - 1));
    }
```

```
    return 0;
}

/*****************
        只要 49
******************/
#include<iostream>
#include<cstdio>
#include<cstring>
#include<string>
using namespace std;
typedef long long  LL;
LL a[100];
LL dp[100][2];
LL z[100] = {1};
LL n;
LL dfs(LL pos, LL stat, bool limit) {
    if(pos == -1) return 0;
    if(!limit && dp[pos][stat] != -1) return dp[pos][stat];
    LL up = limit ? a[pos] : 9;
    LL ans = 0;
    for(LL i = 0; i <= up; i++) {
        if(stat && i == 9) {
            ans += limit ? (n % z[pos] + 1) : z[pos];
        } else {
            ans += dfs(pos - 1, i == 4, limit && i == a[pos]);
        }
    }
    if(!limit) dp[pos][stat] = ans;
    return ans;
}
LL solve(LL x) {
    LL pos = 0;
    while(x) {
        a[pos++] = x % 10;
        x /= 10;
    }
    return dfs(pos - 1, 0, true);
}
int main()
{
    for(int i = 1;i < 30; i++) {
        z[i] = z[i - 1] * 10;
    }
```

```
    memset(dp, -1, sizeof(dp));
    int t;
    cin >> t;
    while(t--) {
        cin >> n;
        cout << solve(n) << endl;
    }
    return 0;
}
```

第二种方法

```
LL n, dp[25][3];
//dp[i][j]:长度为 i，状态为 j
int digit[25];
//nstatus: 0：不含 49，1：不含 49 但末尾是 4，2 :含 49
LL DFS(int pos, int status, int limit)
{
    if(pos <= 0) // 如果到了已经枚举了最后一位，并且在枚举的过程中有 49 序列出现
        return status==2;//注意是 ==
    if(!limit && dp[pos][status]!=-1)   //对于有限制的询问我们是不能够记忆化的
        return dp[pos][status];
    LL ans = 0;
    int End = limit?digit[pos]:9;   // 确定这一位的上限是多少
    for(int i = 0; i <= End; i++)   // 每一位有这么多的选择
    {
        int nstatus = status;       // 有点 else s = statu 的意思

        if(status==0 && i==4)//高位不含 49，并且末尾不是 4 ，现在末尾添 4 返回 1 状态
            nstatus = 1;
        else if(status==1 && i!=4 && i!=9)//高位不含 49，且末尾是 4，现在末尾添加的不是 4 返回 0 状态
            nstatus = 0;
        else if(status==1 && i==9)//高位不含 49，且末尾是 4，现在末尾添加 9 返回 2 状态
            nstatus = 2;
        ans+=DFS(pos-1, nstatus, limit && i==End);
    }
    if(!limit)
        dp[pos][status]=ans;
    return ans;
```

```
}

第三种方法·没有 DFS 的纯 DP

LL dp[27][3];
int c[27];
//dp[i][j]:长度为 i 的数的第 j 种状态
//dp[i][0]:长度为 i 但是不包含 49 的方案数
//dp[i][1]:长度为 i 且不含 49 但是以 9 开头的数字的方案数
//dp[i][2]:长度为 i 且包含 49 的方案数
void init()
{
    memset(dp,0,sizeof(dp));
    dp[0][0] = 1;
    for(int i = 1; i <= 20; i++)
    {
        dp[i][0] = dp[i-1][0]*10-dp[i-1][1];
        dp[i][1] = dp[i-1][0]*1;
        dp[i][2] = dp[i-1][2]*10+dp[i-1][1];
    }
}

/*****************
被 13 整除且包含“13”
******************/
#include<iostream>
#include<cstdio>
#include<cstring>
#include<string>
using namespace std;
typedef int LL;
LL a[30];
LL dp[30][15][3];
LL z[30] = {1};
LL n;

LL dfs(LL pos, LL mod, LL stat, bool limit) {
    if(pos == -1) return mod == 0 && stat == 2;
    if(!limit && dp[pos][mod][stat] != -1) return dp[pos][mod][stat];
    LL up = limit ? a[pos] : 9;
    LL ans = 0;
    for(LL i = 0; i <= up; i++) {
        LL ns = stat;
        if(stat == 0 && i == 1) ns = 1;
```

```
        if(stat == 1 && i != 1) ns = 0;
        if(stat == 1 && i == 3) ns = 2;
        ans += dfs(pos - 1, (mod * 10 + i) % 13, ns, limit && i ==
a[pos]);
    }
    if(!limit) dp[pos][mod][stat] = ans;
    return ans;
}

LL solve(LL x) {
    LL pos = 0;
    while(x) {
        a[pos++] = x % 10;
        x /= 10;
    }
    return dfs(pos - 1, 0, 0, true);
}

int main()
{
    memset(dp, -1, sizeof(dp));
    for(LL i = 1; i < 30; i++) {
        z[i] = z[i - 1] * 10;
    }
    while(cin >> n) {
        cout << solve(n)<< endl;
    }
    return 0;
}
```

计算几何(见红书)

没有快滚

其他

C++高精度

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <string>
#include <algorithm>
using namespace std;

const int MAXN = 600;
char s1[MAXN], s2[MAXN];
struct bign
{
        int len, s[MAXN];
        bign (){
                memset(s, 0, sizeof(s));
                len = 1;
        }
        bign (int num) { *this = num; }
        bign (const char *num) { *this = num; }
        bign operator = (const int num){
                char s[MAXN];
                sprintf(s, "%d", num);
                *this = s;
                return *this;
        }
        bign operator = (const char *num){
                for(int i = 0; num[i] == '0'; num++) ;  //È¥Ç°µ¼0
                len = strlen(num);
                for(int i = 0; i < len; i++) s[i] = num[len-i-1] - '0';
                return *this;
        }
        bign operator + (const bign &b) const //+
        {
                bign c;
                c.len = 0;
                for(int i = 0, g = 0; g || i < max(len, b.len); i++){
                        int x = g;
                        if(i < len) x += s[i];
```

```
                if(i < b.len) x += b.s[i];
                c.s[c.len++] = x % 10;
                g = x / 10;
        }
        return c;
}
bign operator += (const bign &b){
        *this = *this + b;
        return *this;
}
void clean(){
        while(len > 1 && !s[len-1]) len--;
}
bign operator * (const bign &b) //*
{
        bign c;
        c.len = len + b.len;
        for(int i = 0; i < len; i++){
                for(int j = 0; j < b.len; j++){
                        c.s[i+j] += s[i] * b.s[j];
                }
        }
        for(int i = 0; i < c.len; i++){
                c.s[i+1] += c.s[i]/10;
                c.s[i] %= 10;
        }
        c.clean();
        return c;
}
bign operator *= (const bign &b){
        *this = *this * b;
        return *this;
}
bign operator - (const bign &b){
        bign c;
        c.len = 0;
        for(int i = 0, g = 0; i < len; i++){
                int x = s[i] - g;
                if(i < b.len) x -= b.s[i];
                if(x >= 0) g = 0;
                else{
                        g = 1;
                        x += 10;
                }
```

```
                c.s[c.len++] = x;
        }
        c.clean();
        return c;
}
bign operator -= (const bign &b){
        *this = *this - b;
        return *this;
}
bign operator / (const bign &b){
        bign c, f = 0;
        for(int i = len-1; i >= 0; i--){
                f = f*10;
                f.s[0] = s[i];
                while(f > b || f == b)
                {
                        f -= b;
                        c.s[i]++;
                }
        }
        c.len = len;
        c.clean();
        return c;
}
bign operator /= (const bign &b){
        *this  = *this / b;
        return *this;
}
bign operator % (const bign &b){
        bign r = *this / b;
        r = *this - r*b;
        return r;
}
bign operator %= (const bign &b){
        *this = *this % b;
        return *this;
}
bool operator < (const bign &b){
        if(len != b.len) return len < b.len;
        for(int i = len-1; i >= 0; i--){
                if(s[i] != b.s[i]) return s[i] < b.s[i];
        }
        return false;
}
```

```
        bool operator > (const bign &b){
                if(len != b.len) return len > b.len;
                for(int i = len-1; i >= 0; i--)
                {
                        if(s[i] != b.s[i]) return s[i] > b.s[i];
                }
                return false;
        }
        bool operator == (const bign &b){
                return !(*this > b) && !(*this < b);
        }
        string str() const{
       string res = "";
       for(int i = 0; i < len; i++) res = char(s[i]+'0') + res;
       return res;
    }
};


int main(){
        bign a, b, c;
        while(scanf("%s %s", s1, s2) != EOF)
        {
                a = bign(s1);
                b = bign(s2);
                c = a / b;
                cout << c.str() << endl;
        }
        return 0;
}
```

Java 大整数

一：在 java 中的基本头文件（java 中叫包）

```
import java.io.*

importjava.util.*        我们所用的输入 scanner 在这个包中

importjava.math.*          我们下面要用到的 BigInteger 就这这个包中
```

二：输入与输出

```
读入 Scanner cin=new Scanner（System.in）
While(cin.hasNext())   //相当于 C 语言中的！=EOF
n = cin.nextInt();       //输入一个 int 型整数
n = cin.nextBigInteger();   //输入一个大整数
System.out.print(n);      //输出 n 但不换行
System.out.println();     //换行
System.out.println(n);    //输出 n 并换行
System.out.printf(“%d\n”,n);     //类似 C 语言中的输出
```

三：定义变量

```
定义单个变量：
int a,b,c;      //和 C++ 中无区别
BigInteger  a;   //定义大数变量 a
BigIntegerb= new BigInteger("2");   //定义大数变量 b 赋值为 2；
BigDecimaln;      //定义大浮点数类 n；

定于数组：
int a[]= new int[10]   //定义长度为 10 的数组 a
BigInteger  b[] =new BigInteger[100]   //定义长度为 100 的数组 a
```

四：表示范围

```
布尔型 boolean 1 true,false false
字节型 byte 8 -128-127 0
字符型 char 16 ‘\u000’-\uffff ‘\u0000’
短整型 short 16 -32768-32767 0
整型 int 32 -2147483648,2147483647 0
长整型 long 64 -9.22E18,9.22E18 0
浮点型 float 32 1.4E-45-3.4028E+38 0.0
双精度型 double 64 4.9E-324,1.7977E+308 0.0
```

```
BigInteger 任意大的数，原则上只要你的计算机内存足够大，可以有无限位
五：常用的一些操作
A=BigInteger.ONE;   //把 0 赋给 A
B=BigInteger. valueOf (3);    //把 3 赋给 B;
A[i]=BigInteger. valueOf (10);    //把 10 赋给 A[i]
c=a.add(b)        //把 a 与 b 相加并赋给 c
c=a.subtract(b)   //把 a 与 b 相减并赋给 c
c=a.multiply(b)   //把 a 与 b 相乘并赋给 c
c=a.divide(b)     //把 a 与 b 相除并赋给 c
c=a.mod(b)        // 相当于 a%b
a.pow(b)          //相当于 a^b
a.compareTo(b):       //根据该数值是小于等于或大于 a 返回 -1、0 或 1;
a.equals(b):     //判断两数是否相等，也可以用 compareTo 来代替;
a.min(b), a.max(b):   //取两个数的较小、大者;
```

例题：hdu5920
题意：求一个数用 50 个以内的回文数加起来的方案

```java
import java.math.*;
//import java.io.*;
import java.util.*;
public class Main {
    public static BigInteger fanzhuan(BigInteger n){
        int k = String.valueOf(n).length();
        BigInteger ret = BigInteger.ZERO;
        for (int i=1;i<=k;i++){
            ret = ret.add(n.mod(BigInteger.TEN));
            ret = ret.multiply(BigInteger.TEN);
            n = n.divide(BigInteger.TEN);
        }
        ret = ret.divide(BigInteger.TEN);
        return ret;
    }
    public static void main(String[] argv){
        Scanner cin = new Scanner(System.in);
        int T =cin.nextInt();
        for (int cas=1;cas<=T;cas++){
            BigInteger n = cin.nextBigInteger();
            int N = String.valueOf(n).length();
            int A = 0;
            BigInteger ans[] = new BigInteger[55];
            for (;N>1;){
                //System.out.println("n=" + n);
                BigInteger one0 = BigInteger.TEN.pow(N>>1);
```

```
                BigInteger half = n.divide(one0);
                if (N<=2){
                    if (N==1){
                        ans[++A] = n;
                        n = BigInteger.ZERO;
                        break;
                    }else {//2 wei
                        if (n.compareTo(BigInteger.valueOf(19))==0){
                            ans[++A] = BigInteger.valueOf(11);
                            ans[++A] = BigInteger.valueOf( 8);
                            n = BigInteger.ZERO;
                            break;
                        }else if (n.compareTo(BigInteger.valueOf(19))==-
1){
                            ans[++A] = BigInteger.valueOf(9);
                            ans[++A] = n.subtract(BigInteger.valueOf(9));
                            n = BigInteger.ZERO;
                            break;
                        }//else continue;
                    }
                }
                half = half.subtract(BigInteger.ONE);
                //System.out.println("half=" + half);
                BigInteger fan = fanzhuan(half);
                if (N%2>0) fan = fan.mod(one0);
                //System.out.println("fan=" + fan);
                BigInteger jan = half.multiply(one0).add(fan);
                //System.out.println("jan=" + jan);
                ans[++A] = jan ;
                n = n.subtract(jan);
                N = String.valueOf(n).length();
            }
            if (n.compareTo(BigInteger.ZERO)==1)ans[++A] = n;
            System.out.printf("Case #%d:\n%d\n",cas,A);
            for (int i=1;i<=A;i++){
                System.out.println(ans[i]);
            }
        }
        cin.close();
    }
}
```

头文件

```
#include <iostream>
#pragma comment(linker, "/STACK:1024000000,1024000000")
#include <stdio.h>
#include <fstream>
#include <iomanip>
#include <cmath>
#include <string>
#include <string.h>
#include <sstream>
#include <cctype>
#include <climits>
#include <set>
#include <map>
#include <deque>
#include <queue>
#include <vector>
#include <iterator>
#include <algorithm>
#include <stack>
#include <functional>
/*int 类型最大值 INT_MAX，short 最大值为 SHORT_MAX
long long 最大值为 LONG_LONG_MAX*/
//cout << "OK" << endl;
#define _clr(x,y) memset(x,y,sizeof(x))
#define _inf(x) memset(x,0x3f,sizeof(x))
#define pb push_back
#define mp make_pair
#define FORD(i,a,b) for (int i=(a); i<=(b); i++)
#define FORP(i,a,b) for (int i=(a); i>=(b); i--)
#define REP(i,n) for (int i=0; i<(n); i++)
using namespace std;
const int INF = 0x3f3f3f3f;
const double eps = 1e-8;
const double EULER = 0.577215664901532860;
const double PI = 3.14159265358979323846;
const double E = 2.71828182845904523536028;

typedef long long LL;
```

输入挂

```
const int BUFSIZE = 100 * 1024 * 1024;
char Buf[BUFSIZE + 1], *buf = Buf;
template<class T>
void read(T &a){
    for(a=0; *buf<'0' || *buf>'9'; buf++);
    while(*buf>='0' && *buf<='9'){
        a = a*10+(*buf-'0'); buf++;
    }
}
```

对拍模板,freopen

FOR ACM OI
在 linux 的 shell 脚本对拍命令
执行方法：在终端下，进入当前目录，输入"sh ./nick.sh",（其中 nick.sh 为当前 shell 脚本名）ubuntu14.04 下实测成功

```
while true; do
./make>tmp.in #出数据
./tmp<tmp.in>tmp.out #被测程序
./tmp2<tmp.in>tmp2.out #正确（暴力）程序
if diff tmp.out tmp2.out; then #比较两个输出文件
printf AC #结果相同显示 AC
else
echo WA #结果不同显示 WA，并退出
#cat tmp.out tmp2.out
exit 0
fi #if 的结束标志,与 C 语言相反，0 为真
done # while 的结束标志
#BY NICK WONG 2014-08-29
#在终端下，进入当前目录，输入"sh ./nick.sh",（其中 nick.sh 为当前 shell 脚本名） '#'表示单行注释
#diff 在两文件相同时返回空串
```

freopen 的关闭

```
    freopen("in.txt", "r", stdin);
    fclose(stdin);
    freopen("CON", "r", stdin);
```

windows 下对拍

```
:again

D:\cb-work4\gen\bin\Debug\gen.exe
D:\cb-work4\duiA\bin\Debug\duiA.exe
D:\cb-work4\duiB\bin\Debug\duiB.exe
fc C:\Users\admin\Desktop\duipai\out1.txt
C:\Users\admin\Desktop\duipai\out2.txt
if not errorlevel 1 goto again
pause
```

linux 下对拍

```
if diff test.out test.ans;then
echo AC
else
echo WA
exit 0
fi
done
*/
```