```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
constexpr ll mod=1e9+7;
//Binary Exponentiation OK
ll binpow(ll base, ll power, ll mod) {
    base%=mod;
    // If mod is prime, power=power%(mod-1);
    ll result=1;
    while(power>0) {
        if(power&1)result=(result*base)%mod;
        base=(base*base)%mod;
        power>>=1;
    }
    return result;
}
//Fibonacci Number[nth and (n+1)th]
pair<ll, ll> fib (ll n, ll mod) {
    if (n == 0)
        return {0,1};
    auto p = fib(n >> 1, mod);
    ll c = (p.first%mod * (2 * p.second -
p.first + mod)%mod)%mod;
    ll d = ((p.first * p.first)%mod +
(p.second * p.second)%mod)%mod;
    if (n & 1)
        return {d, c + d};
    else
        return {c, d};
}
//Big Multiplicative Mod OK
ll bigMul(ll a, ll b, ll mod) {
    if(a==0)return 0;
    ll ans=(2*bigMul(a/2,b,mod))%mod;
    if(a&1)ans=(ans+(b%mod))%mod;
    return ans;
}
//GCD OK
ll gcd(ll a, ll b) {
    while(b) {
        a%=b;
        swap(a,b);
    }
    return a;
}
//LCM OK
ll lcm(ll a, ll b) {
    return a/gcd(a,b)*b;
}
//Extended GCD OK
ll exGCD(ll a, ll b, ll &x, ll &y) {
    if(b==0) {
        x=1;
        y=0;
        return a;
    }
    ll x1,y1;
    ll d=exGCD(b,a%b,x1,y1);
    x=y1;
    y=x1-(a/b)*y1;
    return d;
}
//Modular Inverse OK
ll modInv(ll a, ll mod) {
    return binpow(a,mod-2,mod);
}
```

```
//Sieve OK
bitset<100000009>bs;
vector<ll>primes;
void sieve(ll ub) {
    bs.set();
    ub++;
    bs[0]=bs[1]=0;
    primes.push_back(2);
    for(ll i=4; i<=ub; i+=2)bs[i]=0;
    for(ll i=3; i<=ub; i+=2) {
        if(bs[i]) {
            for(ll j=i*i; j<=ub;
j+=i)bs[j]=0;
            primes.push_back(i);
        }
    }
}
//Euler Phi
ll phi(ll n) {
    ll PF_idx=0, PF=primes[PF_idx], ans=n;
    while(PF*PF<=n) {
        if(n%PF==0)
            ans-=(ans/PF);
        while(n%PF==0)
            n/=PF;
        PF=primes[++PF_idx];
    }
    if(n!=1)
        ans-=(ans/n);
    return ans;
}
//Euler Phi (1 to n) OK
vector<ll>phi;
void phi_1_to_n(ll n) {
    phi.resize(n + 1);
    phi[0] = 0;
    phi[1] = 1;
    for (ll i = 2; i <= n; i++)phi[i] = i;
    for (ll i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (ll j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}
//Integer Factorization OK
vector<ll>PF(ll n) {
    vector<ll>fact;
    for (auto d:primes) {
        if(d*d>n)break;
        while(n%d==0) {
            fact.push_back(d);
            n/=d;
        }
    }
    if(n>1)fact.push_back(n);
    return fact;
}
//Distinct Prime Factors with count OK
pair<vector<ll>,map<ll,ll> >DPF(ll n) {
    vector<ll>disFact;
    map<ll,ll>factCt;
    for (auto d:primes) {
        if(d*d>n)break;
        if(n%d==0)disFact.push_back(d);
        while(n%d==0) {
            factCt[d]++;
            n/=d;
        }
    }
    if(n>1) {
        if(!factCt[n])disFact.push_back(n);
        factCt[n]++;
    }
    return {disFact,factCt};
}
//Primality Check
bool isPrime(ll x) {
    if((x<=1)||((x!=2)&&(x%2==0)))return
false;
    for (ll d = 3; d * d <= x; d+=2) {
        if (x % d == 0)return false;
    }
    return true;
}
//Number of Divisors
ll numDiv(ll n) {
    ll PF_idx=0, PF=primes[PF_idx], ans=1;
    while(PF*PF<=n) {
        ll power=0;
        while(n%PF==0) {
            n/=PF;
            power++;
        }
        ans*=(power+1);
        PF=primes[++PF_idx];
    }
    if(n>1)
        ans*=2;
    return ans;
}
//Number of Divisors (1 to n)
vector<ll>nod;
ll numDiv_1_to_n(ll n) {
    nod.resize(n+1);
    nod[0]=0;
    nod[1]=1;
    for(ll i=2; i<=n; i++)
        nod[i]=1;
    for(ll i=2; i<=n; i++) {
        if(nod[i]==1) {
            for(ll j=i; j<=n; j+=i) {
                ll power=0;
                ll val=i;
                while(j%val==0) {
                    power++;
                    val*=i;
                }
                nod[j]*=(power+1);
            }
        }
    }
}
//Sum of Divisors
ll sumDiv(ll n) {
    ll ans=1;
    for(ll i=0; primes[i]<=n; i++) {
        if(n%primes[i]==0) {
            ll power=1;
            while(n%primes[i]==0) {
                n/=primes[i];
                power++;
            }
ans*=(powl(primes[i],power)-1)/(primes[i]-1)
;
        }
    }
    return ans;
}
//Factorial Mod OK
vector<ll>fact;
void calFact(ll n, ll mod) {
    fact.resize(n+1);
    fact[0]=1;
    for(ll i=1; i<=n; i++) {
        fact[i]=bigMul(fact[i-1],i,mod);
    }
}
//nCr OK
ll nCr(ll n, ll k, ll mod) {
    if(n<k)return 0;
    ll p_a=fact[n];
    ll p_b=modInv(fact[k],mod);
    ll p_c=modInv(fact[n-k],mod);
    ll ans=bigMul(p_a,p_b,mod);
    ans=bigMul(ans,p_c,mod);
    return ans;
}
//nCr DP OK
ll nCr[35][35]; //initiate with -1
ll nCr_DP(ll n, ll r) {
    if(nCr[n][r]!=-1)return nCr[n][r];
    if(n==0||r==0||n==r)return nCr[n][r]=1;
    return
nCr[n][r]=nCr_DP(n-1,r)+nCr_DP(n-1,r-1);
}
//Catalan Number
vector<ll>catNum(ll n, ll mod) {
    vector<ll>cNum(n+5);
    cNum[0]=cNum[1]=1;
    for(ll i=2; i<=n; i++) {
        cNum[i]=0;
        for(ll j=0; j<i; j++) {

cNum[i]+=(cNum[j]*cNum[i-j-1])%mod;
            if(cNum[i]>=mod)
                cNum[i]-=mod;
        }
    }
    return cNum;
}
```