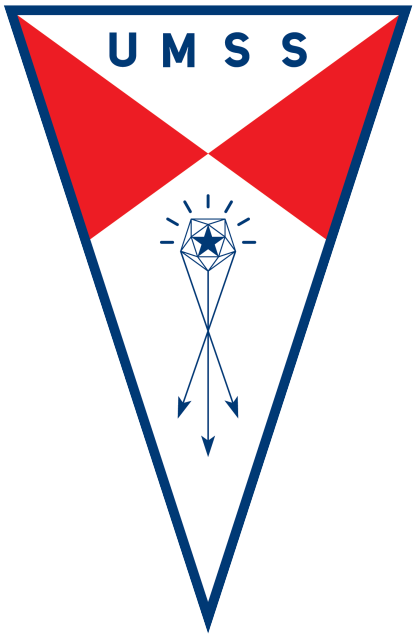
Team Notebook for National ICPC 2024 - Perritos Malvados

Nicolás Alba Murillo

August 30, 2024



UNIVERSIDAD
MAYOR DE SAN SIMÓN
Ciencia y Conocimiento desde 1832

Contents

1 Template

1.1 Template

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

#define pb push_back
#define F first
#define S second
#define all(x) (x).begin(), (x).end()

using ll = long long;
using vl = vector<ll>;

void test_case();
void init();

signed main() {
    ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    init();
    int t = 1;
    cin >> t; // For one test case, comment this
    while (t--) test_case();
```

```
    return 0;
}

void init() {
    // pre-computation before all test cases
}

void test_case() {
    // process one test case
}
```

2 Math

2.1 Catalan Convolution

```
/*
Return Catalan Convolution.

Convolution for k=3
((( A ) B ) C ) D

Where A + B + C + D = N, for N + 1
*/

const int MOD = 1e9 + 7;
ll mul(ll x, ll y) { return (x*y)%MOD; }
ll pot(ll x, ll y) {
    if(y==0) return 1;
    ll ans = pot(x,y/2);
    ans = mul(ans,ans);
    if (y&1)ans=mul(ans,x);
    return ans;
}
ll inv(ll x) { return pot(x, MOD-2); }


// mxN it the double of the max input N, plus max K
const int mxN = 2e6 + 1e6 + 10;
vl fact(mxN,1);
ll cnk(ll n, ll k) {
    if (k < 0 || n < k) return 0;
    ll nOverK = mul(fact[n],inv(fact[k]));
    return mul(nOverK,inv(fact[n-k]));
}

void init() {
    for (int i =1;i<=mxN;i++) {
        fact[i] = mul(fact[i-1],i);
    }
}

// for parethesis example
// number of n+k pairs having k open parethesis at beginning

// (cnk(2n+k,n)*(k+1))/(n+k+1)
ll catalanCov(ll n, ll k) {
    ll up = mul(cnk(2*n+k,n),(k+1)%MOD);
    ll down = (n+k+1)%MOD;
    return mul(up,inv(down));
}

/*
6
(()

ans: 2
*/
// size, and prefix
ll countParenthesisWithPrefix(ll n, string &p) {
    if (n&1) return 0;
    ll k = 0;
    for (auto c : p) {
        if (c=='(') k++;
        else k--;
        if (k<0) return 0;
    }
    n=(n-(ll)p.size()-k)/2;
    return catalanCov(n,k);
}
```

2.2 Catalan

```
/*Catalan, counts the number of ways of:
( A ) B, where |A|+|B| = N, for N+1
*/

const int MOD = 1e9 + 7;
ll mul(ll x, ll y) { return (x*y)%MOD; }
ll pot(ll x, ll y) {
    if(y==0) return 1;
    ll ans = pot(x,y/2);
    ans = mul(ans,ans);
    if (y&1)ans=mul(ans,x);
    return ans;
}
ll inv(ll x) { return pot(x, MOD-2); }

// mxN it the double of the max input 'n'
const int mxN = 2e6 + 10;
vl fact(mxN,1);
void init() {
    for (int i =1;i<=mxN;i++) {
        fact[i] = mul(fact[i-1],i);
    }
}

ll catalan(ll n) {
    if (n<0) return 0;
    ll up = fact[2*n];
    ll down = mul(fact[n],fact[n+1]);
    return mul(up,inv(down));
}
```

2.3 Combinatorics

```
// if k == 0 then 1
// if k negative or no enough choices then 0
// O(min(n, n-k)) lineal
ll nck(ll n, ll k) {
    if (k < 0 || n < k) return 0;
    k = min(k, n-k);
    ll ans = 1;
    for (int i = 1; i <= k; i++) {
        ans = ans * (n-i+1) / i;
    }
    return ans;
}
```

2.4 Count Primes with PI Function

```
// sprime.count_primes(n);
// O(n^(2/3))
// PI(n) = Count prime numbers until n inclusive

struct count_primers_struct {
    vector<int> primes;
    vector<int> mnprimes;
    ll ans;
    ll y;
    vector<pair<pair<ll, int>, char>> queries;

    ll count_primes(ll n) {
        // this y is actually n/y
        // also no logarithms, welcome to reality, this y is the
            best for n=10^12 or n=10^13
        y = pow(n, 0.64);
        if (n < 100) y = n;

        // linear sieve
        primes.clear();
        mnprimes.assign(y + 1, -1);
        ans = 0;
        for (int i = 2; i <= y; ++i) {
            if (mnprimes[i] == -1) {
                mnprimes[i] = primes.size();
                primes.push_back(i);
            }
            for (int k = 0; k < primes.size(); ++k) {
                int j = primes[k];
```

```
        if (i * j > y) break;
        mnprimes[i * j] = k;
        if (i % j == 0) break;
    }
}
if (n < 100) return primes.size();
ll s = n / y;

for (int p : primes) {
    if (p > s) break;
    ans++;
}
// pi(n / y)
int ssz = ans;

// F with two pointers
int ptr = primes.size() - 1;
for (int i = ssz; i < primes.size(); ++i) {
    while (ptr >= i && (ll)primes[i] * primes[ptr] > n)
        --ptr;
    if (ptr < i) break;
    ans -= ptr - i + 1;
}

// phi, store all queries
phi(n, ssz - 1);

sort(queries.begin(), queries.end());
int ind = 2;
int sz = primes.size();

// the order in fenwick will be reversed, because prefix
    sum in a fenwick is just one query
fenwick fw(sz);
for (auto qq : queries) {
    auto na = qq.F;
    auto sign = qq.S;
    auto n = na.F;
    auto a = na.S;
    while (ind <= n)
        fw.add(sz - 1 - mnprimes[ind++], 1);
        ans += (fw.ask(sz - a - 2) + 1) * sign;
    }
    queries.clear();
    return ans - 1;
}

void phi(ll n, int a, int sign = 1) {
    if (n == 0) return;
    if (a == -1) {
        ans += n * sign;
        return;
    }
    if (n <= y) {
        queries.emplace_back(make_pair(n, a), sign);
        return;
    }
    phi(n, a - 1, sign);
    phi(n / primes[a], a - 1, -sign);
}

struct fenwick {
    vector<int> tree;
    int n;

    fenwick(int n = 0) : n(n) {
        tree.assign(n, 0);
    }

    void add(int i, int k) {
        for (; i < n; i = (i | (i + 1)))
            tree[i] += k;
    }

    int ask(int r) {
        int res = 0;
        for (; r >= 0; r = (r & (r + 1)) - 1)
            res += tree[r];
        return res;
    }
};
} ;
```

```
count_primers_struct sprime;
```

2.5 FFT

2.5.1 FFT Shifts trick

```
//FFT Trick, it very useful for shifts in the following:
// Sum j_0_to_n-1 a[j]*a[j+i]
// where i is the number of shifts, and 'a' is some array.

    auto copy = actual;
    reverse(all(copy));
    // be careful with doubles precision, so maybe NTT could be
        useful here.
    // mulitply is the method of FTT or NTT
    auto polinomy = multiply(actual, copy);
    ll m = actual.size();
    answer[0] = polinomy[m-1]; // 0 with m-1, 1 with m-2 =m-1
    for (int i = 1; i <= m-1; i++) { // 1 step no m-1 steps
        // 0 with m-2 is 1 step, 1 with m-3 is one then m-1-1,
            also the last one m-1 is with m-1
        // 0 with m-3 is 2 step, m-1 with m-1-1
        answer[i] = polinomy[m-1-i] + polinomy[2*(m-1)-i+1];
    }
```

2.5.2 FFT

```
// FFT multiplies polinomial 'a' and 'b' in O(n log n)
// you can define double as long double, but maybe TLE
using cd = complex<double>;
void fft(vector<cd> & a, bool invert) {
    ll n = a.size();

    for (ll i = 1, j = 0; i < n; i++) {
        ll bit = n >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;

        if (i < j)
            swap(a[i], a[j]);
    }

    for (ll len = 2; len <= n; len <<= 1) {
        double ang = 2 * PI / len * (invert ? -1 : 1);
        cd wlen(cos(ang), sin(ang));
        for (ll i = 0; i < n; i += len) {
            cd w(1);
            for (ll j = 0; j < len / 2; j++) {
                cd u = a[i+j], v = a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w *= wlen;
            }
        }
    }

    if (invert) {
        for (cd & x : a)
            x /= n;
    }
}

vector<ll> multiply(vector<ll> const& a, vector<ll> const& b) {
    vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    ll n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);

    fft(fa, false);
    fft(fb, false);
    for (ll i = 0; i < n; i++)
        fa[i] *= fb[i];
    fft(fa, true);

    vector<ll> result(n);
```

```
    for (ll i = 0; i < n; i++)
        result[i] = round(fa[i].real()); // fa[i].real() + 0.5 is
            faster
    return result;
}
```

2.5.3 Fast Hadamard Transform

```
// like polynomial multiplication, but XORing exponents
// instead of adding them (also ANDing, ORing)
const int MAXN=1<<18;

#define fore(i,l,r) for(int i=int(l);i<int(r);++i)
#define SZ(x) ((int)(x).size())

ll c1[MAXN+9],c2[MAXN+9];//MAXN must be power of 2!
void fht(ll* p, int n, bool inv){
    for(int l=1;2*l<=n;l*=2)for(int i=0;i<n;i+=2*l)fore(j,0,l){
        ll u=p[i+j],v=p[i+l+j];
        // if(!inv)p[i+j]=u+v,p[i+l+j]=u-v; // XOR
        // else p[i+j]=(u+v)/2,p[i+l+j]=(u-v)/2;
        //if(!inv)p[i+j]=v,p[i+l+j]=u+v; // AND
        //else p[i+j]=-u+v,p[i+l+j]=u;
        if(!inv)p[i+j]=u+v,p[i+l+j]=u; // OR
        else p[i+j]=v,p[i+l+j]=u-v;
    }
}
// like polynomial multiplication, but XORing exponents
// instead of adding them (also ANDing, ORing)
vector<ll> multiply(vector<ll>& p1, vector<ll>& p2){
    int n=1<<(32-__builtin_clz(max(SZ(p1),SZ(p2))-1));
    fore(i,0,n)c1[i]=0,c2[i]=0;
    fore(i,0,SZ(p1))c1[i]=p1[i];
    fore(i,0,SZ(p2))c2[i]=p2[i];
    fht(c1,n,false);fht(c2,n,false);
    fore(i,0,n)c1[i]*=c2[i];
    fht(c1,n,true);
    return vector<ll>(c1,c1+n);
}
```

```
// maxime the OR of a pair of given nums and count
// how many pairs can get that maximum OR
// tested: https://csacademy.com/contest/archive/task/maxor
void test_case() {
    ll n; cin >> n;
    vl a(MAXN),b(MAXN);
    for (int i =0;i<n;i++) {
        ll x;
        cin >> x;
        a[x]++;
        b[x]++;
    }
    vl c = multiply(a,b);
    pair<ll,ll> best = {0, c[0]};
    for (int i = 0;i<MAXN;i++) {
        if (c[i]) best = {i,(c[i]-a[i])/2};
    }
    cout <<best.F << " " << best.S << endl;
}
```

2.5.4 Karatsuba

```
// Multiplication of Polynomials in O(n^1.58)
// with any Module that you want
#define ll long long
const int MOD = 1e9+7;
#define poly vector<ll>
#define fore(i,a,b) for(int i=a,ThxDem=b;i<ThxDem;++i)
typedef int tp;

ll sum(ll x, ll y) {
    ll ans = (x + y) % MOD;
    if (ans < 0) ans += MOD;
    return ans;
}

ll mult(ll x, ll y) {
    ll ans = (x % MOD) * (y % MOD);
    ans %= MOD;
    if (ans < 0) ans += MOD;
```

```
    return ans;
}

#define add(n,s,d,k) fore(i,0,n)(d)[i]=sum((d)[i], mult((s)[i],k))
tp* ini(int n){tp *r=new tp[n];fill(r,r+n,0);return r;}
void karatsura(int n, tp* p, tp* q, tp* r){
        if(n<=0)return;
        if(n<35)fore(i,0,n)fore(j,0,n)r[i+j]=sum(r[i+j],
            mult(p[i],q[j]));
        else {
                int nac=n/2,nbd=n-n/2;
                tp *a=p,*b=p+nac,*c=q,*d=q+nac;
                tp *ab=ini(nbd+1),*cd=ini(nbd+1),
                    *ac=ini(nac*2),*bd=ini(nbd*2);
                add(nac,a,ab,1);add(nbd,b,ab,1);
                add(nac,c,cd,1);add(nbd,d,cd,1);
                karatsura(nac,a,c,ac);karatsura(nbd,b,d,bd);
                add(nac*2,ac,r+nac,-1);add(nbd*2,bd,r+nac,-1);
                add(nac*2,ac,r,1);add(nbd*2,bd,r+nac*2,1);
                karatsura(nbd+1,ab,cd,r+nac);
                free(ab);free(cd);free(ac);free(bd);
        }
}
vector<tp> multiply(vector<tp> p0, vector<tp> p1){
        int n=max(p0.size(),p1.size());
        tp *p=ini(n),*q=ini(n),*r=ini(2*n);
        fore(i,0,p0.size())p[i]=p0[i];
        fore(i,0,p1.size())q[i]=p1[i];
        karatsura(n,p,q,r);
        vector<tp> rr(r,r+p0.size()+p1.size()-1);
        free(p);free(q);free(r);
        return rr;
}
```

2.5.5 NTT

```
// Multiply Poly with special Modules
// MAXN must be power of 2 !!
// MOD-1 needs to be a multiple of MAXN !!
// #define int long long
#define fore(i,a,b) for(ll i=a,ThxDem=b;i<ThxDem;++i)
// const ll MOD=998244353,RT=3,MAXN=1<<18;
const ll MOD=2305843009255636993ll,RT=5,MAXN=1<<18;
typedef vector<ll> poly;
ll mulmod(__int128 a, __int128 b){return ((a%MOD)*(b%MOD)) % MOD;}
ll addmod(ll a, ll b){ll r=a+b;if(r>=MOD)r-=MOD;return r;}
ll submod(ll a, ll b){ll r=a-b;if(r<0)r+=MOD;return r;}
ll pm(ll a, ll e){
        ll r=1;
        while(e){
                if(e&1)r=mulmod(r,a);
                e>>=1;a=mulmod(a,a);
        }
        return r;
}

struct CD {
        ll x;
        CD(ll x):x(x){}
        CD(){}
        ll get()const{return x;}
};
CD operator*(const CD& a, const CD& b){return
    CD(mulmod(a.x,b.x));}
CD operator+(const CD& a, const CD& b){return
    CD(addmod(a.x,b.x));}
CD operator-(const CD& a, const CD& b){return
    CD(submod(a.x,b.x));}
vector<ll> rts(MAXN+9,-1);
CD root(ll n, bool inv){
        ll r=rts[n]<0?rts[n]=pm(RT,(MOD-1)/n):rts[n];
        return CD(inv?pm(r,MOD-2):r);
}
CD cp1[MAXN+9],cp2[MAXN+9];
ll R[MAXN+9];
void dft(CD* a, ll n, bool inv){
        fore(i,0,n)if(R[i]<i)swap(a[R[i]],a[i]);
        for(ll m=2;m<=n;m*=2){
                CD wi=root(m,inv); // NTT
                for(ll j=0;j<n;j+=m){
```

```
                        CD w(1);
                        for(ll k=j,k2=j+m/2;k2<j+m;k++,k2++){
                                CD u=a[k];CD
                                    v=a[k2]*w;a[k]=u+v;a[k2]=u-v;w=w*wi;
                        }
                }
        }
        if(inv){
                CD z(pm(n,MOD-2)); // pm: modular exponentiation
                fore(i,0,n)a[i]=a[i]*z;
        }
}
poly multiply(poly& p1, poly& p2){
        ll n=p1.size()+p2.size()+1;
        ll m=1,cnt=0;
        while(m<=n)m+=m,cnt++;
        fore(i,0,m){R[i]=0;fore(j,0,cnt)
            R[i]=(R[i]<<1)|((i>>j)&1);}
        fore(i,0,m)cp1[i]=0,cp2[i]=0;
        fore(i,0,p1.size())cp1[i]=p1[i];
        fore(i,0,p2.size())cp2[i]=p2[i];
        dft(cp1,m,false);dft(cp2,m,false);
        fore(i,0,m)cp1[i]=cp1[i]*cp2[i];
        dft(cp1,m,true);
        poly res;
        n-=2;
        fore(i,0,n)res.pb(cp1[i].x); // NTT
        return res;
}
```

2.6 Fast Fibonacci

```
// Fast Fibonacci O(log n)
// Use fib(n).F to get the at nth position
pair<ll,ll> fib (ll n) {
    if (n == 0)
        return {0, 1};

    auto p = fib(n >> 1);
    ll c = (p.F * (2*p.S - p.F + MOD)%MOD)%MOD;
    ll d = (p.F * p.F + p.S * p.S)%MOD;
    if (n & 1)
        return {d, (c + d)%MOD};
    else
        return {c, d};
}
/* Fib properties
Addition Rule: F_n+k = F_k * F_n+1 + F_k-1 * F_n
F_2n= Fn * (F_n+1 + F_n-1)

GCD Identity: GCD(F_m,F_n) = F_gcd(m,n)
Cassinis' identity: F_n-1 * F_n+1 - F_n*F_n = (-1)^n
*/
```

2.7 Floor Sums

```
// from atcoder
// floor_sum(n,m,a,b) = sum{0}to{n-1} [(a*i+b)/m]
// O(log m), mod 2^64, n<2^32, m<2^32

constexpr long long safe_mod(long long x, long long m) {
    x %= m;
    if (x < 0) x += m;
    return x;
}

unsigned long long floor_sum_unsigned(unsigned long long n,
                                      unsigned long long m,
                                      unsigned long long a,
                                      unsigned long long b) {
    unsigned long long ans = 0;
    while (true) {
        if (a >= m) {
            ans += n * (n - 1) / 2 * (a / m);
            a %= m;
        }
        if (b >= m) {
            ans += n * (b / m);
            b %= m;
```

```
        }

        unsigned long long y_max = a * n + b;
        if (y_max < m) break;
        // y_max < m * (n + 1)
        // floor(y_max / m) <= n
        n = (unsigned long long)(y_max / m);
        b = (unsigned long long)(y_max % m);
        swap(m, a);
    }
    return ans;
}

long long floor_sum(long long n, long long m, long long a, long
    long b) {
    assert(0 <= n && n < (1LL << 32));
    assert(1 <= m && m < (1LL << 32));
    unsigned long long ans = 0;
    if (a < 0) {
        unsigned long long a2 = safe_mod(a, m);
        ans -= 1ULL * n * (n - 1) / 2 * ((a2 - a) / m);
        a = a2;
    }
    if (b < 0) {
        unsigned long long b2 = safe_mod(b, m);
        ans -= 1ULL * n * ((b2 - b) / m);
        b = b2;
    }
    return ans + floor_sum_unsigned(n, m, a, b);
}
```

2.8 Mobius

2.8.1 Linear Sieve

```
/* For getting the primes less than mxN in O(mxN)*/
const int mxN = 1e6 + 10;
vl sv(mxN); // if prime sv[i]==i, it stores the lowest prime of
    'i'
vl primes;
void init() { // O(n)
    for (int i = 2;i<mxN;i++) {
        if (sv[i]==0) {
            sv[i]=i;
            primes.pb(i);
        }
        for (int j = 0;j<primes.size() && primes[j]*i<mxN;j++) {
            sv[primes[j]*i] = primes[j];
            if (primes[j] == sv[i]) break;
        }
    }
}

// factorization using linear sieve, O(prime_count(n))
// Very fast factorization but only for (n < mxN) !!!
void fact(map<ll,int> &f, ll num) {
    while (num>1) {
        ll p = sv[num];
        while (num % p == 0) num/=p, f[p]++;
    }
}
```

2.8.2 Mobius Inclusion-Exclusion Example

How many numbers are there less than or equal to n that are free of squares?. Contrainsts: $1 <= n <= 10^{12}$

Change the statement to count the reverse and then sustract: How many numbers are ... that can be divided by a square of a prime. So the answer will be $n-$ Summatory with Inclusion-Exclusion of $f(prime)$

$$f(prime) = floor(\frac{n}{p*p})$$

So in those cases of summatory with primes, you can use Mobius to adding or substracting. Final answer is

$$n - \Sigma_{i=1}^{\sqrt{n}} \mu(i) \left\lfloor \frac{n}{i^2} \right\rfloor$$

Or in programming terms:

```
long long ans = n;
for (int i =1;i<=sqrt(n);i++) {
    ans -= mo[i] * n/(i*i);
}
```

2.8.3 Mobius

```
/*
Mobius Function.
 Multiplicative function that is useful to inclusion/exclusion
     with
 prime numbers (it gives the coeficient). Also you can reduce some
 sumatories using its equality with unit(n) function (see the key
     below)

 unit(n) = [ n == 1 ]
 unit(1) = 1, unit(2) = 0, unit(0) = 0

 (This is the key!!)
 unit(n) = sum_{ d divides n } mobius(d)

 mobius(1) = 1
 mobius(quantity of primes is odd) = -1
 mobius(quantity of primes is even) = 1
 mobius(n is divisible by a square prime) = 0

 Check https://codeforces.com/blog/entry/53925 for more
     information.
 Check mobius examples
*/

// This is shorter code and O(n log n)
const int mxN = 1e5 + 10;
vl mo(mxN);
void init() { // Call init() first !!!
    mo[1] = 1;
    for (int i = 1;i<mxN;i++)
        for (int j = i+i;j<mxN;j+=i) mo[j]-=mo[i];
}


// This is O(n), but not too much difference of speed with the
    other
const int mxN = 1e6 + 10;
vl sv(mxN); // prime if sv[i]==i, it stores the lowest prime of i
vl primes; // Primes less than mxN
vl mo(mxN); // Mobius
void init() {
    // sv[1] = 1; // Check if needed
    for (int i = 2;i<mxN;i++) { // Linear Sieve
        if (sv[i]==0) { sv[i]=i; primes.pb(i); }
        for (int j = 0;j<primes.size() && primes[j]*i<mxN;j++) {
            sv[primes[j]*i] = primes[j];
            if (primes[j] == sv[i]) break;
        }
    }

    mo[1] = 1; // Mobius
    for (int i=2;i<mxN;i++) {
       if (sv[i/sv[i]] == sv[i]) mo[i] = 0;
       else mo[i] = -1*mo[i/sv[i]];
    }
}
```

2.8.4 Multiplicative Functions

The following functions are all multiplicative functions, where p is a prime number and k is a positive integer:

- The constant function: $I(p^k) = 1$.
- The identity function: $\mathrm{Id}(p^k) = p^k$.
- The power function: $\mathrm{Id}_a(p^k) = p^{ak}$, where a is a constant.
- The unit function: $\chi(p^k) = [p^k = 1]$.

- The divisor function: $\sigma_a(p^k) = \sum_{i=0}^{k} p^{ai}$, denoting the sum of the a-th powers of all the positive divisors of the number.
- The Möbius function: $\mu(p^k) = [k = 0] - [k = 1]$.
- Euler's totient function: $\varphi(p^k) = p^k - p^{k-1}$.

Note: $[P]$ refers to the boolean expression, i.e., $[P] = 1$ when P is true, and 0 otherwise.

2.8.5 [gcd(a_i, a_j) == k] Queries

```
// Given an array and q queries of count pairs gcd(a_i,a_j)==k
// i < j, a_i < 1e5, q < 1e5, n < 1e5
// Complexity O(n * log n + q)
// Tested:
    https://www.hackerrank.com/contests/ab-yeh-kar-ke-dikhao-returns/challenges/gcd-pairs

const int mxN = 1e5 + 10;
vl mo(mxN);
void init() { // Call init() first !!!
    mo[1] = 1;
    for (int i = 1;i<mxN;i++)
        for (int j = i+i;j<mxN;j+=i) mo[j]-=mo[i];
}

vl cnt(mxN), dcnt(mxN), ans(mxN);
void test_case() {
    ll n,q; cin >> n >> q;
    for (int i=0;i<n;i++) {
        ll x;cin >> x;
        cnt[x]++; // cnt[x] = quantity of X's in array
    }
    for (int i = 1;i<mxN;i++)
        for (int j = i;j<mxN;j+=i)
            dcnt[i] += cnt[j];
    // dcnt[x] = quantity of a_i divisible by x
    for (int k = 1;k<mxN;k++) {
        for (int d = 1; d <= mxN/k; d++) {
            ll totalCnt = d*k < mxN ? dcnt[d*k] : 0;
            ans[k] += mo[d] * totalCnt * totalCnt;
        }
        ans[k] += cnt[k]; // substracting j>=i
        ans[k] /= 2;
    }

    while (q--) {
        ll k; cin >> k;
        if (k < mxN) cout << ans[k] << "\n";
        else cout << 0 << "\n";
    }
}
```

2.8.6 [gcd(i, j) == k]

```
// Counts pairs gcd(i,j) == k
// 1 <= i <= a, 1 <= j <= b, where (1,2) equals to (2,1)
// Call Mobius First !!!
// O(min(a,b)/K)
// Tested: https://vjudge.net/problem/HDU-1695
ll solve(ll a, ll b, ll k) {
    if (k==0) return 0;
    a/=k;
    b/=k;
    if (a > b) swap(a,b);
    if (a == 0) return 0;
    ll ans = 0;
    for (ll d = 1; d <= a; d++) {
        ans += (a/d) * (b/d) * mo[d];
    }
    ll sub = 0; // Substracting equals, e.g. (1,2) to (2,1)
    for (ll d = 1;d <= a; d++) {
        sub += (a/d)*(a/d) * mo[d];
    }
    ans-=(sub-1)/2;
    return ans;
}
```

2.8.7 $\Sigma_{i=1}^{n} \frac{n}{gcd(i,n)}$

```
// Multiplicative Function
// Calc f(n), f(n) = sum_{1 to n} n / gcd(n,i)

// f(p) = (p-1)*p + 1
// f(p^k) = f(p^(k-1)) + p^k*p^(k-1)*(p-1)

const int mxN = ll(1e7) + 10;
vl lp(mxN); // least prime factor
vl pw(mxN); // power of least prime factor
vl fn(mxN); // answer of f(n)
vl primes;

void init() { // O(n), Call init first !!!!
    fn[1] = pw[1] = lp[1] = 1;
    for (ll i = 2;i<mxN;i++) {
        if (lp[i] == 0) {
            lp[i] = pw[i] = i;
            fn[i] = (i-1)*i + 1;
            primes.pb(i);
        }

        for (auto p : primes) {
            ll j = i*p;
            if (j >= mxN) break;
            if (lp[i] != p) {
                lp[j] = pw[j] = p;
                fn[j] = fn[i] * fn[p];
            } else {
                lp[j] = p;
                pw[j] = pw[i] * p;
                ll fk = fn[pw[i]] + pw[j] * pw[i] * (p-1);
                fn[j] = fn[i/pw[i]] * fk;
                break;
            }
        }
    }
}
```

2.9 Modular Arithmethics

2.9.1 Big Expontent Modular Exponentiation

```
// Calc a^b^c % MOD
// MOD is prime

ll pou(ll a, ll b, ll m) {
    ll ans = 1;
    while (b) {
        if (b&1) ans *= a, ans%=m;
        a*=a;
        a%=m;
        b/=2;
    }
    return ans;
}

void test_case() {
    ll a, b, c;
    cin >> a >> b >> c;
    // fermat theorem
    // a^(p-1) = 1 (mod p)
    b = pou(b, c, MOD - 1);
    a = pou(a, b, MOD);

    cout << a << "\n";
}
```

2.9.2 Chinease Remainder

```
/*
Finds this system congrence
X = a_1 (mod m_1)
X = a_2 (mod m_2)
...
X = a_k (mod m_k)
 */
```

```
// No sure time complexity, but fast
// I think it is related to lcm or
// Maybe O(mult(M))
ll x, y;
/// O(log(max(a, b)))
ll euclid(ll a, ll b) {
    if(b == 0) { x = 1; y = 0; return a; }
    ll d = euclid(b, a%b);
    ll aux = x;
    x = y;
    y = aux - a/b*y;
    return d;
}

pair<ll, ll> crt(vector<ll> A, vector<ll> M) {
    ll n = A.size(), ans = A[0], lcm = M[0];
    for (int i = 1; i < n; i++) {
        ll d = euclid(lcm, M[i]);
        if ((A[i] - ans) % d) return {-1, -1};
        ll mod = lcm / d * M[i];
        ans = (ans + x * (A[i] - ans) / d % (M[i] / d) * lcm) %
            mod;
        if (ans < 0) ans += mod;
        lcm = mod;
    }
    return {ans, lcm};
}
```

2.9.3 Diophantine Ecuations

```
Encuentra x, y en la ecuacin de la forma ax + by = c
Agregar Extended Euclides.

ll g;
bool diophantine(ll a, ll b, ll c) {
    x = y = 0;
    if (!a && !b) return (!c); // slo hay solucin con c = 0
    g = euclid(abs(a), abs(b));
    if (c % g) return false;
    a /= g; b /= g; c /= g;
    if (a < 0) x *= -1;
    x = (x % b) * (c % b) % b;
    if (x < 0) x += b;
    y = (c - a*x) / b;
    return true;
}
```

2.9.4 Discrete Log

```
Devuelve un entero x tal que a^x = b (mod m) or -1 si no existe
    tal x.

int expmod(int b, int e, int m) { // Always check if change int
    to ll !!!
    int ans = 1;
    while (e) {
        if (e&1) ans = (1ll*ans*b) % m;
        b = (1ll*b*b) % m;
        e /= 2;
    }
    return ans;
}

ll discrete_log(ll a, ll b, ll m) {
    a %= m, b %= m;
    if (b == 1) return 0;
    int cnt = 0;
    ll tmp = 1;
    for (int g = __gcd(a, m); g != 1; g = __gcd(a, m)) {
        if (b%g) return -1;
        m /= g, b /= g;
        tmp = tmp*a / g % m;
        ++cnt;
        if (b == tmp) return cnt;
    }
    map<ll, int> w;
    int s = ceil(sqrt(m));
    ll base = b;
    for (int i = 0; i < s; i++) {
        w[base] = i;
```

```
        base = base*a % m;
    }
    base = expmod(a, s, m);
    ll key = tmp;
    for (int i = 1; i <= s+1; i++) {
        key = key*base % m;
        if (w.count(key)) return i*s - w[key] + cnt;
    }
    return -1;
}
```

2.9.5 Extended Euclides

```
// It finds X and Y in equation:
// a * X + b * Y = gcd(a, b)

int x, y;

int euclid(int a, int b) {
       if (b == 0) {
              x = 1;
              y = 0;
              return a;
       }
       int aux = x;
       x = y;
       y = aux - a/b*y;
       return euclid(b, a % b);
}
```

2.9.6 Modular Combinatorics

```
/* Combinatorics with a Prime Module
nCk(n,k) = How many ways you can choose 'k' items from an array
    of 'n' items with a given prime module.

Use:
 - NCK nck(max N, prime module);
 - nck.nCk(n,k)
*/

struct NCK {
    ll MAX_N;
    ll MOD;
    vl fact;

    explicit NCK(ll maxN, ll mod) : MAX_N(maxN), MOD(mod) {
        fact.resize(MAX_N + 1, 1);
        fact[0] = 1;
        REP(i, 1, MAX_N) {
            fact[i] = fact[i - 1] * (i % MOD);
            fact[i] %= MOD;
        }
    }

    ll inv(ll a){
        return powmod(a, MOD-2); // MOD is prime, otherwise use
            powmod(a, eulerPhi(mod) - 1)
    }

    ll powmod(ll a, ll b){
        if (b == 0) return 1;
        ll mid = powmod(a, b / 2);
        ll ans = (mid * mid) % MOD;
        if (b & 1) {
            ans *= a;
            ans %= MOD;
        }
        return ans;
    }

    ll nCk(ll n, ll k){ // TODO: add if's when case is zero
        ll nOverK = (fact[n] * inv(fact[k])) % MOD;
        return (nOverK * inv(fact[n-k])) % MOD;
    }
};
```

2.10 Optimized Polard Rho

```
// Fast factorization with big numbers, use fact method
// Seems to be O(log^3(n)) !!! Need revision
#define fore(i, b, e) for(int i = b; i < e; i++)
ll gcd(ll a, ll b){return a?gcd(b%a,a):b;}
ll mulmod(ll a, ll b, ll m) {
  ll r=a*b-(ll)((long double)a*b/m+.5)*m;
  return r<0?r+m:r;
}
ll expmod(ll b, ll e, ll m){
  if(!e)return 1;
  ll q=expmod(b,e/2,m);q=mulmod(q,q,m);
  return e&1?mulmod(b,q,m):q;
}
bool is_prime_prob(ll n, int a){
  if(n==a)return true;
  ll s=0,d=n-1;
  while(d%2==0)s++,d/=2;
  ll x=expmod(a,d,n);
  if((x==1)||(x+1==n))return true;
  fore(_,0,s-1){
    x=mulmod(x,x,n);
    if(x==1)return false;
    if(x+1==n)return true;
  }
  return false;
}
bool rabin(ll n){ // true iff n is prime
  if(n==1)return false;
  int ar[]={2,3,5,7,11,13,17,19,23};
  fore(i,0,9)if(!is_prime_prob(n,ar[i]))return false;
  return true;
}
// optimized version: replace rho and fact with the following:
const int MAXP=1e6+1; // sieve size
int sv[MAXP]; // sieve
ll add(ll a, ll b, ll m){return (a+=b)<m?a:a-m;}
ll rho(ll n){
  static ll s[MAXP];
  while(1){
    ll x=rand()%n,y=x,c=rand()%n;
    ll *px=s,*py=s,v=0,p=1;
    while(1){
      *py++=y=add(mulmod(y,y,n),c,n);
      *py++=y=add(mulmod(y,y,n),c,n);
      if((x=*px++)==y)break;
      ll t=p;
      p=mulmod(p,abs(y-x),n);
      if(!p)return gcd(t,n);
      if(++v==26){
        if((p=gcd(p,n))>1&&p<n)return p;
        v=0;
      }
    }
    if(v&&(p=gcd(p,n))>1&&p<n)return p;
  }
}
void init_sv(){
  fore(i,2,MAXP)if(!sv[i])for(ll j=i;j<MAXP;j+=i)sv[j]=i;
}
void fact(ll n, map<ll,int>& f){ // call init_sv first!!!
  for(auto&& p:f){
    while(n%p.F==0){
      p.S++; n/=p.F;
    }
  }
  if(n<MAXP)while(n>1)f[sv[n]]++,n/=sv[n];
  else if(rabin(n))f[n]++;
  else {ll q=rho(n);fact(q,f);fact(n/q,f);}
}
```

2.11 SubFactorial

```
/* Denote as !n or derangement numbers
  Count the number of permutations where no element is in
  the originial position, formally p[i] != i

  it can be seen as f(n) = n!-sum_i=1_to_n { cnk(n,i)*f(n-i) }
  f(0)=1, f(1) = 1
```

```
  1 0 1 2 9 44 265 1,854 14,833 133,496

  n! = sumi=0,i<=n,{cnk(n,i)!i}
  d[i] = (d[i-1]+d[i-2])*(i-1)
*/
const int mxN = 2e6 + 10; // max number
ll add(ll x, ll y) { return (x+y)%MOD; }
ll mul(ll x, ll y) { return (x*y)%MOD; }
vl subFact(mxN);
void init() {
    subFact[0] = 1;
    subFact[1] = 0;
    for (int i = 2;i<mxN;i++) {
        subFact[i] = mul(add(subFact[i-1],subFact[i-2]),i-1);
    }
}
```

2.12 Ternary Search

```
// this is for find minimum point in a parabolic
// O(log3(n))
// TODO: Improve to generic!!!
ll left = 0;
ll right = n - 1;
while (left + 3 < right) {
    ll mid1 = left + (right - left) / 3;
    ll mid2 = right - (right - left) / 3;
    if (f(b, lines[mid1]) <= f(b, lines[mid2])) {
        right = mid2;
    } else {
        left = mid1;
    }
}
ll target = -4 * a * c;
ll ans = -1; // find the answer, in this case any works.
for (ll mid = left; mid <= right; mid++) {
    if (f(b, lines[mid]) + target < 0) {
        ans = mid;
    }
}
```

2.13 Theorems

Erdős–Szekeres Theorem

This theorem is related to increasing and decreasing sequences. Suppose $a, b \in N$, $n = ab + 1$, and $x_1, x_2, \ldots, x_n$ is a sequence of n real numbers. Then this sequence contains a monotonic increasing (decreasing) subsequence of $a + 1$ terms or a monotonic decreasing (increasing) subsequence of $b + 1$ terms. Dilworth's lemma is a generalization of this theorem.

Grundy Numbers in Game Theory

Grundy numbers are used in game theory to analyze games that can be represented as directed state graphs. In these graphs, if a player loses in a state, its Grundy number is zero; otherwise, it is a positive number. The Grundy number for each vertex is defined as:

$$\text{Grundy}(\text{losing state with no moves}) = 0$$

$$\text{Grundy}(\text{vertex}) = \text{MEX}(\text{adjacent\_nodes[vertex]})$$

where MEX stands for the "minimum excludant," which is the smallest non-negative integer not present in the set of Grundy numbers of adjacent nodes.
If you have multiple independent games, the final Grundy number is calculated as:

$$\text{Grundy}(\text{game}_1) \oplus \text{Grundy}(\text{game}_2) \oplus \text{Grundy}(\text{game}_3) \oplus \cdots \oplus \text{Grundy}(\text{game}_n)$$

where $\oplus$ denotes the bitwise XOR operation.

3 DP

3.1 Convex Hull Trick

```
/**
 * Author: Simon Lindholm
 * Description: Container where you can add lines of the form
    kx+m, and query maximum values at points x.
 *  Useful for dynamic programming (``convex hull trick'').
 * Time: O(\log N)
 * Status: stress-tested
 */

 // For minimum you can multiply by -1 'k' and 'm' when adding,
    and the answer when querying.
 // Tested in https://atcoder.jp/contests/dp/submissions/55836691
#pragma once

struct Line {
	mutable ll k, m, p;
	bool operator<(const Line& o) const { return k < o.k; }
	bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
	// (for doubles, use inf = 1/.0, div(a,b) = a/b)
	static const ll inf = LLONG_MAX;
	ll div(ll a, ll b) { // floored division
		return a / b - ((a ^ b) < 0 && a % b); }
	bool isect(iterator x, iterator y) {
		if (y == end()) return x->p = inf, 0;
		if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
		else x->p = div(y->m - x->m, x->k - y->k);
		return x->p >= y->p;
	}
	void add(ll k, ll m) {
		auto z = insert({k, m, 0}), y = z++, x = y;
		while (isect(y, z)) z = erase(z);
		if (x != begin() && isect(--x, y)) isect(x, y =
		    erase(y));
		while ((y = x) != begin() && (--x)->p >= y->p)
			isect(x, erase(y));
	}
	ll query(ll x) {
		assert(!empty());
		auto l = *lower_bound(x);
		return l.k * x + l.m;
	}
};
```

3.2 DP Mask Over Submasks

```
// DP of submask over submasks
// O(3^n)

// j&(-j); get a '1' bit of j
// for (int j=i;j;j = (j-1)&i){...} j is submask of 'i' the mask

ll dp[1<<18]; // answer of mask
ll cst[1<<18]; // cost of use a submask
ll a[18][18]; // elements
ll pos[1<<18]; // trick to get fast the pos
void test_case() {
    ll n;
    cin >> n;
    for (int i =0;i<n;i++) {
        for (int j =0;j<n;j++) {
            cin >> a[i][j];
        }
    }
    for (int i =0;i<n;i++) {
        pos[1<<i] = i;
    }
    for (int i = 0;i<(1<<n);i++) {
        ll j = i;
        vl idxs;
        while (j) {
            ll k = j&(-j);
            idxs.pb(pos[k]);
            j^=k;
        }
```

```
        for (int j = 0;j<idxs.size();j++) {
            for (int k = j+1;k<idxs.size();k++) {
                cst[i] += a[idxs[j]][idxs[k]];
            }
        }
        dp[i] = cst[i];
        for (int j=i;j;j = (j-1)&i) {
            dp[i] = max(dp[i],cst[j]+dp[i^j]);
        }
    }
    cout << dp[(1<<n)-1] << "\n";
}
```

3.3 Divide and Conquer Optimization

```
// TODO:
//
    https://cp-algorithms.com/dynamic_programming/divide-and-conquer-dp.html
// https://usaco.guide/plat/DC-DP?lang=cpp
```

3.4 Knuths Optimization

```
/*
Knuth's Optimization

For dp[i][j] = min_{i<=k<j} dp[i][k] + dp[k+1][j] + cost[i][j]

Optimizing that from O(n^3) to O(n^2), it's required to hold the
    following:
opt[i][j] = optimal splitting point of dp[i][j], the 'k' the
    minimizes the above definition

opt[i][j-1] <= opt[i][j] <= opt[i+1][j]

You can demostrate that by the following:

a<=b<=c<=d
cost(b,c) <= cost(a,d), // an contained interval is <= of the
    interval
cost(a,c)+cost(b,d) <= cost(a,d)+cost(b,c) // partial
    intersection <= total intersection

Complexity: O(n^2)
*/
void test_case() {
    cin >> n;nums.assign(n,0);pf.assign(n+1,0); // READ input!!!
    for (int i =0;i<n;i++) cin>>nums[i], pf[i+1] = pf[i] + nums[i];

    for (int i = 0;i<n;i++) { // base case
        dp[i][i] = 0; // depends of the dp!!!!
        opt[i][i] = i;
    }

    for (int i =n-2;i>=0;i--) {
        for (int j = i+1;j<n;j++) {
            dp[i][j] = inf; // set to inf, any option is better, or
                use -1 or a flag!!
            ll cost = sum(i,j); // depends of problem
            for (int k = opt[i][j-1]; k<=
                min(j-1ll,opt[i+1][j]);k++) {
                ll actual = dp[i][k] + dp[k+1][j] + cost;
                if (actual < dp[i][j]) { // if flag '-1' used,
                    change here!!
                    dp[i][j] = actual;
                    opt[i][j] = k;
                }
            }
        }
    }

    cout << dp[0][n-1] << "\n";
}
```

3.5 Linear Recurrence Cayley Halmiton

```
// O(N^2 log K)
```

```
const int mod = 1e9 + 7;

template <int32_t MOD>
struct modint {
  int32_t value;
  modint() = default;
  modint(int32_t value_) : value(value_) {}
  inline modint<MOD> operator + (modint<MOD> other) const {
      int32_t c = this->value + other.value; return modint<MOD>(c
      >= MOD ? c - MOD : c); }
  inline modint<MOD> operator * (modint<MOD> other) const {
      int32_t c = (int64_t)this->value * other.value % MOD; return
      modint<MOD>(c < 0 ? c + MOD : c); }
  inline modint<MOD> & operator += (modint<MOD> other) {
      this->value += other.value; if (this->value >= MOD)
      this->value -= MOD; return *this; }
  modint<MOD> pow(uint64_t k) const {
    modint<MOD> x = *this, y = 1;
    for (; k; k >>= 1) {
      if (k & 1) y *= x;
      x *= x;
    }
    return y;
  }
  modint<MOD> inv() const { return pow(MOD - 2); } // MOD must be
      a prime
};

using mint = modint<mod>;

vector<mint> combine (int n, vector<mint> &a, vector<mint> &b,
    vector<mint> &tr) {
  vector<mint> res(n * 2 + 1, 0);
  for (int i = 0; i < n + 1; i++) {
    for (int j = 0; j < n + 1; j++) res[i + j] += a[i] * b[j];
  }
  for (int i = 2 * n; i > n; --i) {
    for (int j = 0; j < n; j++) res[i - 1 - j] += res[i] * tr[j];
  }
  res.resize(n + 1);
  return res;
};
// transition -> for(i = 0; i < x; i++) f[n] += tr[i] * f[n-i-1]
// S contains initial values, k is 0 indexed
mint LinearRecurrence(vector<mint> &S, vector<mint> &tr, long
    long k) {
  int n = S.size(); assert(n == (int)tr.size());
  if (n == 0) return 0;
  if (k < n) return S[k];
  vector<mint> pol(n + 1), e(pol);
  pol[0] = e[1] = 1;
  for (++k; k; k /= 2) {
    if (k % 2) pol = combine(n, pol, e, tr);
    e = combine(n, e, e, tr);
  }
  mint res = 0;
  for (int i = 0; i < n; i++) res += pol[i + 1] * S[i];
  return res;
}

void test_case() {
    ll n;
    cin >> n; // Fibonacci
    vector<mint> initial = {0, 1}; // F0, F1
    vector<mint> tr = {1, 1};
    cout << LinearRecurrence(initial, tr, n).value << "\n";
}
```

3.6 Linear Recurrence Matrix Exponciation

```
// Solves F_n = C_n-1 * F_n-1 + ... + C_0*F_0 + p + q*n + r*n^2
// O((n+3)^3*log(k))
// Also solves (k steps)-min path of a matrix in same complexity
// Tested and for more details see:
    https://codeforces.com/blog/entry/80195
const int MOD = 1e9 + 7;
const int N = 10 + 3; // 10 is MAX N, 3 is for p,q,r
inline ll add(ll x, ll y) { return (x+y)%MOD; }
inline ll mul(ll x, ll y) { return (x*y)%MOD; }

// const ll inf = ll(1e18) + 5; // for k-min path
```

```
struct Mat {
    array<array<ll,N>,N> mt;
    Mat(bool id=false) {
        for (auto &x : mt) fill(all(x),0);
        if (id) for (int i=0;i<N;i++) mt[i][i]=1;
        //for (auto &x : mt) fill(all(x),inf); // For k-min path
        //if (id) for (int i =0;i<N;i++) mt[i][i]=0;
    }
    inline Mat operator * (const Mat &b) {
        Mat ans;
        for (int k=0;k<N;k++)for(int i=0;i<N;i++)for(int
            j=0;j<N;j++)
            ans.mt[i][j]=add(ans.mt[i][j],mul(mt[i][k],b.mt[k][j]));
            //ans.mt[i][j] = min(ans.mt[i][j],mt[i][k]+b.mt[k][j]);
                // For K-min Path

        return ans;
    }
    inline Mat pow(ll k) {
        Mat ans(true),p=*this; // Note '*'!!
        while (k) {
            if (k&1) ans = ans*p;
            p=p*p;
            k>>=1;
        }
        return ans;
    }
    string db() { // Optional for debugging
        string ans;
        for (int i =0;i<mt.size();i++)for (int j=0;j<mt.size();j++)
                ans +=to_string(mt[i][j]),ans+=" \n"[j==N-1];
        return "\n"+ans;
    }
};

// Important!!! Remember to set N = MAX_N + 3
// Solves F_n = C_n-1 * F_n-1 + ... + C_0*F_0 + p + q*n + r*n^2
// f = {f_0,f_1,f_2,f_3,...,f_n}
// c = {c_0,c_1,c_2,c_3,...,c_n}
ll fun(vl f, vl c, ll p, ll q, ll r, ll k) {
    ll n = c.size();
    if (k < n) return f[k];
    reverse(all(c)),reverse(all(f));
    Mat mt,st;
    for (int i = 0;i<n;i++) mt.mt[0][i]=c[i];
    for (int i = 1;i<n;i++) mt.mt[i][i-1]=1;
    for (int i = 0;i<n;i++) st.mt[i][0]=f[i];

    vl extra = {p,q,r}; // To extend here with 1*p,i*q,i*i*r,etc
    for (int i=0;i<extra.size();i++) {
        st.mt[n+i][0]=1; //1,i,i*i,i*i*i
        mt.mt[0][n+i]=extra[i];//p,q,r
        mt.mt[n+i][n]=1; //pascal
        for (int j=1;j<=i;j++) { //pascal
            st.mt[n+i][0]*=n;//1,i*i,i*i*i
            mt.mt[n+i][n+j]=mt.mt[n+i-1][n+j]+mt.mt[n+i-1][n+j-1];
        }
    }
    return (mt.pow(k-(n-1))*st).mt[0][0];
}
```

3.7 Longest Increasing Subsequence

```
// Find the Longest Increasing Subsequence of an array in O(n log
    n)
// 1 2 3 5 10 2 -1 100 500 -> input
// 1 2 3 5 10 100 500 -> Lis
int lis(vl &nums) {
    vl best;
    int n = nums.size();
    for (int i = 0; i < n; i++) {
        // For non-decreasing
        // int idx = upper_bound(all(best), nums[i]) -
            best.begin();

        // For increasing
        int idx = lower_bound(all(best), nums[i]) - best.begin();
        if (idx == best.size()) {
            best.pb(nums[i]);
        } else {
            best[idx] = min(best[idx], nums[i]);
```

```
        }
    }

    return best.size();
}

// Also you can do this with Segment Tree in O(n log n)
```

3.8 Multiple Knacksack Optimizacion

```
/*
Multiple Knacksack

You have a knacksack of a capacity, and 'n' objects with
value, weight, and a number of copies that you can buy of that
    object.

Maximize the value without exceding the capacity of the knacksack.

Time complexity is O(W*N*sum)
W = capacity, N = number of objects,
sum is: for (int i =0, sum = 0;i<n;i++) sum += log2(copies[i])

Tested in https://cses.fi/problemset/task/1159/
n<=100, capacity<=10^5, copies[i]<=1000
*/
ll multipleKnacksack(vl &value, vl& weight, vl&copies, ll
    capacity) {
    vl vs,ws;
    ll n = value.size();
    for (int i = 0;i<n;i++) {
        ll h=value[i],s=weight[i],k=copies[i];
        ll p = 1;
        while (k>p) {
            k-=p; // Binary Grouping Optimization
            vs.pb(s*p);
            ws.pb(h*p);
            p*=2;
        }
        if (k) {
            vs.pb(s*k);
            ws.pb(h*k);
        }
    }
    vl dp(capacity+1);
    // 0-1 knacksack
    for (int i =0;i<ws.size();i++) {
        for (int j = capacity;j>=ws[i];j--) {
            dp[j] = max(dp[j],dp[j-ws[i]] + vs[i]);
        }
    }
    return dp[capacity];
}
```

3.9 Optimizing Pragmas For BitSet

```
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")

// The difference to TLE and AC in
    https://cses.fi/problemset/task/2137/
```

3.10 Separation Optimization

```
No se si tiene un nombre, pero en varios problemas vi este truco:

dp[x] = f(x,y) donde se necesita un optimo 'y'.

Truco: si puedes separarlo en f(x,y) en f(x) (+) f(y), puedes
    guardar en una estructa optima f(y) como un segment tree.
```

3.11 Typical Problems

3.11.1 Coin Change

```
// infinite number of coins
```

```cpp
// Get the minimum number of coins that sum a value.
void test_case() {
    ll n, x;
    cin >> n >> x;
    vl dp(x + 1, inf - 1);
    vl coin(n);
    rep(i, 0, n) cin >> coin[i];
    dp[0] = 0;
    rep(i, 0, x) {
        each(c, coin) {
            if (c + i > x) continue;
            dp[i + c] = min(dp[i + c], dp[i] + 1);
        }
    }
    if (dp[x] + 1== inf) {
        cout << "-1\n";
    } else {
        cout << dp[x] << "\n";
    }
}
```

3.11.2 Edit Distance

```cpp
// editDistance(a, b, a.size(), b.size());

// Cuantas operaciones, (insert, remove, remplazar) necesito
// para que string a y b sean iguales.
int editDistance(string a, string b, int m, int n)
{
    if (m == 0) return n;
    if (n == 0) return m;
    if (a[m - 1] == b[n - 1])
        return editDistance(a, b, m - 1, n - 1);

    return 1 + min({editDistance(a, b, m, n - 1), // Insert
                    editDistance(a, b, m - 1, n), // Remove
                    editDistance(a, b, m - 1, n - 1) // Replace
                });
}
```

```cpp
// My own
ll editDistance(string &s, string &t) {
    ll n = s.size();
    ll m = t.size();
    vvl dp(n+1, vl(m+1, 0));
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= m; j++) {
            if (min(i, j) == 0) dp[i][j] = max(i, j);
            else if (s[i-1] == t[j-1]) dp[i][j] = dp[i-1][j-1];
            else dp[i][j] = min(dp[i-1][j], min(dp[i][j-1],
                dp[i-1][j-1])) + 1;
        }
    }
    return dp[n][m];
}
```

3.11.3 Elevator Problem

```cpp
// Given n <= 20 persons, print the minimum number of travels
// to move everyone in a elevator with capacity k.

ll n, k;
vl nums;
vector<pair<ll,ll>> dp;

// minimum travels, last travel with minimum weight.
// use f((1 << n) - 1).F
pair<ll,ll> f(ll mask) {
    if (dp[mask] != make_pair(-1ll, -1ll)) {
        return dp[mask];
    }
    if (mask == 0) {
        return dp[mask] = {0, k};
    }
    dp[mask] = {n + 1, 0}; // one person in a travel, or use
        popcount.
    for (int i = 0; i < n; i++) {
        // person i is the last to enter to elevator.
        if ((mask >> i) & 1) {
```

```
            auto actual = f(mask ^ (1 << i)); // best option
                without this last person.
            if (actual.S + nums[i] <= k) {
                actual.S += nums[i];
                // what happened if there are a better minimum.
                // well in that case the last person should be
                    other one.
                // so we are trying all options that last person
                    will be better.
            } else {
                actual.S = nums[i];
                actual.F++;
            }
            dp[mask] = min(dp[mask], actual);
        }
    }
    return dp[mask];
}


// Iterative
void test_case() {
    ll n, k;
    cin >> n >> k;
    vl nums(n);
    vector<pair<ll,ll>> dp(1 << n, {n+1, 0});
    for (int i =0; i < n; i++) cin >> nums[i];
    dp[0] = {0, k};
    for (int i = 1; i < (1 << n); i++) {
        for (int j = 0; j < n; j++) {
            if (i& (1 << j)) {
                auto actual = dp[i ^ (1 << j)];
                if (actual.S + nums[j] <= k) {
                    actual.S += nums[j];
                } else {
                    actual.F++;
                    actual.S = nums[j];
                }
                dp[i] = min(dp[i], actual);
            }
        }
    }
    cout << dp[(1 << n) -1].F << "\n";
}
```

3.11.4 Longest Common Subsequence 3

```
string X = "AGGT12";
string Y = "12TXAYB";
string Z = "12XBA";
bool calc[100][100][100];
int dp[100][100][100];
//lcsOf3(X.size() - 1, Y.size() - 1, Z.size() - 1);
int lcsOf3(int i, int j,int k) {
    if(i==-1||j==-1||k==-1) // outbounds
        return 0;
    if(calc[i][j][k]) //memo
        return dp[i][j][k];
    calc[i][j][k] = true;
    if(X[i]==Y[j] && Y[j]==Z[k]) // same
        return dp[i][j][k] = 1+lcsOf3(i-1,j-1,k-1);
    else // best of reducine any
        return dp[i][j][k] = max(max(lcsOf3(i-1,j,k),
                            lcsOf3(i,j-1,k)),lcsOf3(i,j,k-1));
}
```

3.11.5 Longest Common Subsequence

```
const int M_MAX = 20;
const int N_MAX = 20;
int m, n;
string X;
string Y;
int memo[M_MAX + 1][N_MAX + 1];


// Encuetra el Longest Common Subsequence de string X e Y. m y n
    son sus tamaos
// lcs de abfgh aeeeeiiiiigh = agh
```

```
int lcs (int m, int n) {
  for (int i = 0; i <= m; i++) {
    for (int j = 0; j <= n; j++) {
      if (i == 0 || j == 0) memo[i][j] = 0;
      else if (X[i - 1] == Y[j - 1]) memo[i][j] = memo[i - 1][j -
          1] + 1;
      else memo[i][j] = max(memo[i - 1][j], memo[i][j - 1]);
    }
  }
  return memo[m][n];
}
```

3.11.6 Max Sum 1D

```
int maxRangeSum(vector<int> a){
       int sum = 0, ans = 0;
       for (int i = 0; i < a.size(); i++){
              if (sum + a[i] >= 0) {
                     sum += a[i];
               ans = max(ans, sum);
         } else sum = 0;
       }
       return ans;
}
```

3.11.7 Max Sum 2D

```
int INF = -100000007; // minimo valor
int n, m; //filas y columnas
const int MAX_N = 105, MAX_M = 105;
int values[MAX_N][MAX_M];

int max_range_sum2D(){
  for(int i=0; i<n;i++){
    for(int j=0; j<m; j++){
      if(i>0) values[i][j] += values[i-1][j];
      if(j>0) values[i][j] += values[i][j-1];
      if(i>0 && j>0) values[i][j] -= values[i-1][j-1];
    }
  }
  int max_mat = INF;
  for(int i=0; i<n;i++){
    for(int j=0; j<m; j++){
      for(int h = i; h<n; h++){
        for(int k = j; k<m; k++){
          int sub_mat = values[h][k];
          if(i>0) sub_mat -= values[i-1][k];
          if(j>0) sub_mat -= values[h][j-1];
          if(i>0 && j>0) sub_mat += values[i-1][j-1];
          max_mat = max(sub_mat, max_mat);
        }
      }
    }
  }
  return max_mat;
}
```

3.11.8 Max Sum 3D

```
long long a=20, b=20, c=20;
long long acum[a][b][c];
long long INF = -100000000007;

long long max_range_3D(){
  for(int x=0; x<a; x++){
    for(int y = 0; y<b; y++){
      for(int z = 0; z<c; z++){
        if(x>0) acum[x][y][z] += acum[x-1][y][z];
        if(y>0) acum[x][y][z] += acum[x][y-1][z];
        if(z>0) acum[x][y][z] += acum[x][y][z-1];
        if(x>0 && y>0) acum[x][y][z] -= acum[x-1][y-1][z];
        if(x>0 && z>0) acum[x][y][z] -= acum[x-1][y][z-1];
        if(y>0 && z>0) acum[x][y][z] -= acum[x][y-1][z-1];
        if(x>0 && y>0 && z>0) acum[x][y][z] += acum[x-1][y-1][z-1];
      }
    }
  }
  long long max_value = INF;
```

```
  for(int x=0; x<a; x++){
    for(int y = 0; y<b; y++){
      for(int z = 0; z<c; z++){
        for(int h = x; h<a; h++){
          for(int k = y; k<b; k++){
            for(int l = z; l<c; l++){
              long long aux = acum[h][k][l];
              if(x>0) aux -= acum[x-1][k][l];
              if(y>0) aux -= acum[h][y-1][l];
              if(z>0) aux -= acum[x][k][z-1];
              if(x>0 && y>0) aux += acum[x-1][y-1][l];
              if(x>0 && z>0) aux += acum[x-1][k][z-1];
              if(z>0 && y>0) aux += acum[h][y-1][z-1];
              if(x>0 && y>0 && z>0) aux -= acum[x-1][y-1][z-1];
              max_value = max(max_value, aux);
            }
          }
        }
      }
    }
  }
  return max_value;
}
```

3.11.9 Traveling Sales Man (Cycle)

```
// Given directed weighted graph, gets the minimun halmilton
    cycle.
// Use dfs(0, 1<<(initial)), if 1e9 then it impossible, otherwise
    get the min.
const int MAX_SIZE = 15;
const ll IMPOSSIBLE = 1e9;
ll INITIAL = 0; // initial node
vpl adj[MAX_SIZE];
vvl dp(MAX_SIZE, vl(1 << MAX_SIZE, -1));
ll n, m;
ll target; // init as (1 << n) - 1, full visited

ll dfs(ll x, ll mask) {
    if (dp[x][mask] != -1) {
        return dp[x][mask];
    }
    if (mask == target) {
        each(yy, adj[x]) {
            if (yy.F == INITIAL) {
                return yy.S;
            }
        }
        return dp[x][mask] = IMPOSSIBLE;
    }
    ll ans = IMPOSSIBLE;
    each(yy, adj[x]) {
        ll y, d;
        tie(y, d) = yy;
        if ((mask >> y) & 1) continue;
        ll actual = dfs(y, mask | (1 << y)) + d;
        ans = min(ans, actual);
    }
    return dp[x][mask] = ans;
}
```

3.11.10 Traveling Sales Man (Path)

```
// Given directed weighted graph, gets the minimun halmilton path.
ll n;
vvl adj;

int dp[1<<20][20];
ll f(ll mask, ll x) {
    if (dp[mask][x] != -1) {
        return dp[mask][x];
    }
    if (x == n-1) {
        return (mask == 0);
    }
    ll ways = 0;
    for (auto y : adj[x]) {
        if ((mask>>y)&1) {
            ways = (ways + f(mask^(1<<y), y)) % MOD;
        }
```

```
    }
    return dp[mask][x] = ways;
}

void test_case() {
    ll m;
    cin >> n >> m;
    adj.assign(n,vl());
    memset(dp,-1,sizeof dp);
    for (int i=0;i<m;i++) {
        ll x, y;
        cin >>x >> y;
        x--,y--;
        adj[x].pb(y);
    }
    cout << f((1<<n)-1-1,0) << "\n";
}
```

4 Data Structures

4.1 Custom Hash Pair

```
// Example: unordered_set<pair<ll,ll>, HASH> exists;
// It's better to convine with other custom hash
struct HASH{
  size_t operator()(const pair<ll,ll>&x)const{
    return hash<ll>()(((ll)x.first)^(((ll)x.second)<<32));
  }
};
```

4.2 Custom Hash

```
// Avoid hashing hacks and improve performance of hash structures
// e.g. unordered_map<ll,ll,custom_hash>
struct custom_hash {
    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM =
            chrono::steady_clock::now().time_since_epoch().count();
        x ^= FIXED_RANDOM;
        return x ^ (x >> 16);
    }
};

struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM =
            chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};
```

4.3 Disjoint Set Union

```
/* Disjoint Set Union
This uses union by rank, it is longer to code but faster
than normal Disjoint Set Union. The time complexity per operation
is faster than O(log n)

It finds if 2 nodes in a graph are connected or not (have same
    'find(x)' value),
and the size of the connected component, the graph starts with no
    edges,
then you can add edges with 'group(nodeA, nodeB)' method.
*/

struct union_find {
    vi link, score, size;
    int n;
```

```
    void init(int nn) {
        link.resize(nn);
        score.resize(nn);
        size.resize(nn);
        this->n = nn;
        for (int i = 0; i < n; i++) {
            link[i] = i;
            score[i] = 0;
            size[i] = 1;
        }
    }
    int find(int x) {
        if (link[x] == x) return x;
        return (link[x] = find(link[x]));
    }
    void group(int a, int b) {
        int pa = find(a);
        int pb = find(b);
        if (pa != pb) {
            if (score[pa] >= score[pb]) {
                link[pb] = pa;
                size[pa] += size[pb];
                if (score[pa] == score[pb]) score[pa]++;
            } else {
                link[pa] = pb;
                size[pb] += size[pa];
            }
        }
    }
};
```

4.4 Fenwick Tree 2D

```
struct BIT2D { // 1-indexed
    vector<vl> bit;
    ll n, m;

    BIT2D(ll n, ll m) : bit(n+1, vl(m+1)), n(n), m(m) {}

    ll lsb(ll i) { return i & -i; }

    void add(int row, int col, ll x) {
        for (int i = row;i<=n;i+=lsb(i))
            for (int j = col;j<=m;j+=lsb(j))
                bit[i][j] += x;
    }

    ll sum(int row, int col) {
        ll res = 0;
        for (int i = row;i>0;i-=lsb(i))
            for (int j = col;j>0;j-=lsb(j))
                res += bit[i][j];
        return res;
    }

    ll sum(int x1, int y1, int x2, int y2) {
        return sum(x2,y2) - sum(x1-1,y2) - sum(x2,y1-1) +
            sum(x1-1,y1-1);
    }

    void set(int x, int y, ll val) {
        add(x,y,val-sum(x,y,x,y));
    }
};
```

4.5 Fenwick Tree

```
struct FenwickTree {
    vector<int> bit;
    int n;

    FenwickTree(int n) {
        this->n = n;
        bit.assign(n, 0);
    }

    FenwickTree(vector<int> a) : FenwickTree(a.size()) {
        for (size_t i = 0; i < a.size(); i++)
            add(i, a[i]);
    }
```

```
    int sum(int r) {
        int ret = 0;
        for (; r >= 0; r = (r & (r + 1)) - 1)
            ret += bit[r];
        return ret;
    }

    int sum(int l, int r) {
        return sum(r) - sum(l - 1);
    }

    void add(int idx, int delta) {
        for (; idx < n; idx = idx | (idx + 1))
            bit[idx] += delta;
    }

    // TODO: Lower Bound
};
```

4.6 General Iterative Segment Tree

```
// >>>>>>>>> Implement
struct Node { ll x = 0; };

Node e() { return Node(); } // null element

Node op(Node &a, Node &b) { // operation
    Node c;
    c.x = a.x + b.x;
    return c;
}
// <<<<<<<

struct segtree {
    vector<Node> t;
    ll n;

    void init(int n) {
        t.assign(n * 2, e());
        this->n = n;
    }

    void init(vector<Node>& s) {
        n = s.size();
        t.assign(n * 2, e());
        for (int i = 0; i < n; i++) {
            t[i+n] = s[i];
        }
        build();
    }

    void build() { // build the tree
        for (int i = n - 1; i > 0; --i) t[i] = op(t[i<<1],
            t[i<<1|1]);
    }

    // set value at position p
    void update(int p, const Node& value) {
        for (t[p += n] = value; p >>= 1; ) t[p] = op(t[p<<1],
            t[p<<1|1]);
    }

    // sum on interval [l, r]
    Node query(int l, int r) {
        r++; // make this inclusive
        Node resl=e(), resr=e(); // null element
        for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
            if (l&1) resl = op(resl, t[l++]);
            if (r&1) resr = op(t[--r], resr);
        }
        return op(resl, resr);
    }

    Node get(int i) {
        return query(i, i); // improve to o(1)
    }

    // TODO: implement left, right binary search
};
```

4.7 General Lazy Tree

```
// TODO: Make this iterative
struct Node { // Structure
    ll mn;
    ll size = 1;

    Node(ll mn):mn(mn) {
    }
};

struct Func { // Applied function
    ll a = 0;
};

Node e() { // op(x, e()) = x
    Node a(INT64_MAX); // neutral element
    return a;
};

Func id() { // mapping(x, id()) = x
    Func l = {0}; // identify func
    return l;
}

Node op(Node &a, Node &b) { // associative property
    Node c = e();         // binary operation
    c.size = a.size + b.size;
    c.mn = min(a.mn, b.mn);
    return c;
}

Node mapping(Node node, Func &lazy) {
    node.mn += lazy.a; // appling function
    return node;
}

Func composicion(Func &prev, Func &actual) {
    prev.a = prev.a + actual.a; // composing funcs
    return prev;
}
```

```
struct lazytree {
    int n;
    vector<Node> nodes;
    vector<Func> lazy;

    void init(int nn) {
        n = nn;
        int size = 1;
        while (size < n) {
            size *= 2;
        }
        ll m = size *2;
        nodes.assign(m, e());
        lazy.assign(m, id());
    }

    void push(int i, int sl, int sr) {
        nodes[i] = mapping(nodes[i], lazy[i]);
        if (sl != sr) {
            lazy[i * 2 + 1] = composicion(lazy[i*2+1],lazy[i]);
            lazy[i * 2 + 2] = composicion(lazy[i*2+2],lazy[i]);
        }
        lazy[i] = id();
    }

    void apply(int i, int sl, int sr, int l, int r, Func f) {
        push(i, sl, sr);
        if (l <= sl && sr <= r) {
            lazy[i] = f;
            push(i,sl,sr);
        } else if (sr < l || r < sl) {
        } else {
            int mid = (sl + sr) >> 1;
            apply(i * 2 + 1, sl, mid, l, r, f);
            apply(i * 2 + 2, mid + 1, sr, l, r, f);
            nodes[i] = op(nodes[i*2+1],nodes[i*2+2]);
        }
    }

    void apply(int l, int r, Func f) {
```

```
    assert(l <= r);
    assert(r < n);
    apply(0, 0, n - 1, l, r, f);
}

void update(int i, Node node) {
    assert(i < n);
    update(0, 0, n-1, i, node);
}

void update(int i, int sl, int sr, int pos, Node node) {
    if (sl <= pos && pos <= sr) {
        push(i,sl,sr);
        if (sl == sr) {
            nodes[i] = node;
        } else {
            int mid = (sl + sr) >> 1;
            update(i * 2 + 1, sl, mid, pos, node);
            update(i * 2 + 2, mid + 1, sr, pos, node);
            nodes[i] = op(nodes[i*2+1], nodes[i*2+2]);
        }
    }
}

Node query(int i, int sl, int sr, int l, int r) {
    push(i,sl,sr);
    if (l <= sl && sr <= r) {
        return nodes[i];
    } else if (sr < l || r < sl) {
        return e();
    } else {
        int mid = (sl + sr) >> 1;
        auto a = query(i * 2 + 1, sl, mid, l, r);
        auto b = query(i * 2 + 2, mid + 1, sr, l, r);
        return op(a,b);
    }
}

Node query(int l, int r) {
    assert(l <= r);
    assert(r < n);
    return query(0, 0, n - 1, l, r);
    }
};
```

4.8 Lazy Sum Tree

```
struct lazytree {
    int n;
    vl sum;
    vl lazySum;

    void init(int nn) {
        sum.clear();
        n = nn;
        int size = 1;
        while (size < n) {
            size *= 2;
        }
        sum.resize(size * 2);
        lazySum.resize(size * 2);
    }

    void update(int i, int sl, int sr, int l, int r, ll diff) {
        if (lazySum[i]) {
            sum[i] += (sr - sl + 1) * lazySum[i];
            if (sl != sr) {
                lazySum[i * 2 + 1] += lazySum[i];
                lazySum[i * 2 + 2] += lazySum[i];
            }
            lazySum[i] = 0;
        }
        if (l <= sl && sr <= r) {
            sum[i] += (sr - sl + 1) * diff;
            if (sl != sr) {
                lazySum[i * 2 + 1] += diff;
                lazySum[i * 2 + 2] += diff;
            }
        } else if (sr < l || r < sl) {
        } else {
            int mid = (sl + sr) >> 1;
```

```
        update(i * 2 + 1, sl, mid, l, r, diff);
        update(i * 2 + 2, mid + 1, sr, l, r, diff);
        sum[i] = sum[i * 2 + 1] + sum[i * 2 + 2];
    }
}

void update(int l, int r, ll diff) {
    assert(l <= r);
    assert(r < n);
    update(0, 0, n - 1, l, r, diff);
}

ll query(int i, int sl, int sr, int l, int r) {
    if (lazySum[i]) {
        sum[i] += lazySum[i] * (sr - sl + 1);
        if (sl != sr) {
            lazySum[i * 2 + 1] += lazySum[i];
            lazySum[i * 2 + 2] += lazySum[i];
        }
        lazySum[i] = 0;
    }
    if (l <= sl && sr <= r) {
        return sum[i];
    } else if (sr < l || r < sl) {
        return 0;
    } else {
        int mid = (sl + sr) >> 1;
        return query(i * 2 + 1, sl, mid, l, r) + query(i * 2 +
            2, mid + 1, sr, l, r);
    }
}

ll query(int l, int r) {
    assert(l <= r);
    assert(r < n);
    return query(0, 0, n - 1, l, r);
}
};
```

4.9 Min Sparse Table

```
using Type = int;

// Gets the minimum in a range [l,r] in O(1)
// Preprocesing is O(n log n)
struct min_sparse {

    int log;
    vector<vector<Type>> sparse;

    void init(vector<Type> &nums) {
        int n = nums.size();
        log = 0;
        while (n) log++, n/=2;
        n = nums.size();
        sparse.assign(n, vector<Type>(log, 0));
        for (int i = 0; i < n; i++) sparse[i][0] = nums[i];
        for (int l = 1; l < log; l++) {
            for (int j = 0; j + (1 << l) - 1 < n; j++) {
                sparse[j][l] = min(sparse[j][l-1], sparse[j+(1 <<
                    (l-1))][l-1]);
            }
        }
    }

    Type query(int x, int y) {
        int n = y - x + 1;
        int logg = -1;
        while (n) logg++, n/=2; // TODO: improve this with fast
            builtin
        return min(sparse[x][logg], sparse[y-(1 << logg)+1][logg]);
    }
};
```

4.10 Mo's Hilbert Curve

```
/*
Hilbert Curve for Mo's,
```

```
This is a better ordering for Mo (review Mo's algorithm),
sometimes it take the half time, other times it takes
more time. So it needs a revision to identify when it's better!!!
https://codeforces.com/blog/entry/61203
*/

inline int64_t gilbertOrder(int x, int y, int pow, int rotate) {
	if (pow == 0) {
		return 0;
	}
	int hpow = 1 << (pow-1);
	int seg = (x < hpow) ? (
		(y < hpow) ? 0 : 3
	) : (
		(y < hpow) ? 1 : 2
	);
	seg = (seg + rotate) & 3;
	const int rotateDelta[4] = {3, 0, 0, 1};
	int nx = x & (x ^ hpow), ny = y & (y ^ hpow);
	int nrot = (rotate + rotateDelta[seg]) & 3;
	int64_t subSquareSize = int64_t(1) << (2*pow - 2);
	int64_t ans = seg * subSquareSize;
	int64_t add = gilbertOrder(nx, ny, pow-1, nrot);
	ans += (seg == 1 || seg == 2) ? add : (subSquareSize - add
	    - 1);
	return ans;
}

struct Query {
	int l, r, idx;
	int64_t ord;

	inline void calcOrder() {
		ord = gilbertOrder(l, r, 21, 0); // n,q <= 1e5
	}
};

inline bool operator<(const Query &a, const Query &b) {
	return a.ord < b.ord;
}
```

4.11 Mo's

```
/*
Mo's algorithm O((N + Queries)*SQRT(N))

It answers queries offline using SQRT Descomposition,
sometimes get TLE when you have Set, Map, various iteration when
    adding,
So you need to implement efficiently (with arrays) in O(1). You
    can answer
questions like in a range, the number of distincts values.
*/
const int BLOCK_SIZE = 430; // For se 1e5=310, for 2e5=430

struct query {
    int l, r, idx;

    bool operator <(query &other) const {
        return MP(l / BLOCK_SIZE, r) < MP(other.l / BLOCK_SIZE,
            other.r);
    }
};


void add(int idx);
void remove(int idx);
ll getAnswer();

vector<ll> mo(vector<query> queries) {
    vector<ll> answers(queries.size());
    int l = 0;
    int r = -1;
    sort(all(queries));
    each(q, queries) {
        while (q.l < l) add(--l);
        while (r < q.r) add(++r);
        while (l < q.l) remove(l++);
        while (q.r < r) remove(r--);
        answers[q.idx] = getAnswer();
    }
    return answers;
```

```
}

vl nums; //init
ll ans = 0;
int cnt[1000001];
void add(int idx) {
  // update ans, when adding an element
}
void remove(int idx) {
  // update ans, when removing an element
}
ll getAnswer() { return ans; }
```

4.12 MultiOrderedSet

```
#include <bits/stdc++.h>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>

using namespace __gnu_pbds;

struct multiordered_set {
    tree<ll,
        null_type,
        less_equal<ll>, // this is the trick
        rb_tree_tag,
        tree_order_statistics_node_update> oset;

    //this function inserts one more occurrence of (x) into the
        set.
    void insert(ll x) {
        oset.insert(x);
    }

    //this function checks weather the value (x) exists in the set
        or not.
    bool exists(ll x) {
        auto it = oset.upper_bound(x);
        if (it == oset.end()) {
            return false;
        }
        return *it == x;
    }

    //this function erases one occurrence of the value (x).
    void erase(ll x) {
        if (exists(x)) {
            oset.erase(oset.upper_bound(x));
        }
    }

    //this function returns the value at the index (idx)..(0
        indexing).
    ll find_by_order(ll pos) {
        return *(oset.find_by_order(pos));
    }

    //this function returns the first index of the value (x)..(0
        indexing).
    int first_index(ll x) {
        if (!exists(x)) {
            return -1;
        }
        return (oset.order_of_key(x));
    }

    //this function returns the last index of the value (x)..(0
        indexing).
    int last_index(ll x) {
        if (!exists(x)) {
            return -1;
        }
        if (find_by_order(size() -1) == x) {
            return size() - 1;
        }
        return first_index(*oset.lower_bound(x)) -1;
    }

    //this function returns the number of occurrences of the value
        (x).
    int count(ll x) {
```

```
        if (!exists(x)) {
            return -1;
        }
        return last_index(x) - first_index(x) + 1;
    }

    // Count the numbers between [l, r]
    int count(ll l, ll r) {
        auto left = oset.upper_bound(l);
        if (left == oset.end() || *left>r) {
            return 0;
        }
        auto right = oset.upper_bound(r);
        if (right != oset.end()) {
            right = oset.find_by_order(oset.order_of_key(*right));
        }
        if (right == oset.end() || *right >r) {
            if (right == oset.begin()) return 0;
            right--;
        }
        return last_index(*right)-first_index(*left)+1;
    }

    //this function clears all the elements from the set.
    void clear() {
        oset.clear();
    }

    //this function returns the size of the set.
    ll size() {
        return (ll)oset.size();
    }
};
```

4.13 OrderedSet

```
// Same as set, but you can get the order of an element or the
// element in given sorted position in O(log n)
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

#define oset tree<ll, null_type,less<ll>,
    rb_tree_tag,tree_order_statistics_node_update>
//find_by_order(k) order_of_key(k)
```

4.14 PersistantSegmentTree

```
struct Vertex {
    Vertex *l, *r;
    ll sum;
    Vertex(int val) : l(nullptr), r(nullptr), sum(val) {}
    Vertex(Vertex *l, Vertex *r) : l(l), r(r), sum(0) {
        if (l) sum += l->sum;
        if (r) sum += r->sum;
    }
};
Vertex* build(vector<ll>& a, int tl, int tr) {
    if (tl == tr)
        return new Vertex(a[tl]);
    int tm = (tl + tr) / 2;
    return new Vertex(build(a, tl, tm), build(a, tm+1, tr));
}
ll get_sum(Vertex* v, int tl, int tr, int l, int r) {
    if (l > r)
        return 0;
    if (l == tl && tr == r)
        return v->sum;
    int tm = (tl + tr) / 2;
    return get_sum(v->l, tl, tm, l, min(r, tm))
         + get_sum(v->r, tm+1, tr, max(l, tm+1), r);
}
Vertex* update(Vertex* v, int tl, int tr, int pos, int new_val) {
    if (tl == tr)
        return new Vertex(new_val);
    int tm = (tl + tr) / 2;
    if (pos <= tm)
        return new Vertex(update(v->l, tl, tm, pos, new_val),
            v->r);
    else
```

```
        return new Vertex(v->l, update(v->r, tm+1, tr, pos,
            new_val));
}
Vertex* build(vector<ll> &a) {
    return build(a,0,a.size()-1);
}
ll get_sum(Vertex *v,ll n,int l, int r) {
    return get_sum(v,0,n-1,l,r);
}
Vertex* update(Vertex* v,ll n, int pos, int newV) {
    return update(v,0,n-1,pos,newV);
}
Vertex* copy(Vertex* v) {
    return new Vertex(v->l,v->r);
}
```

4.15 Priority Queue

```
template<class T> using pql =
    priority_queue<T,vector<T>,greater<T>>;// less first
template<class T> using pqg = priority_queue<T>; // greater first
```

4.16 Rope

```
/**
 * Description: insert element at $i$-th position, cut a
     substring and
       * re-insert somewhere else. At least 2 times slower than
           handwritten treap.
 * Time: O(\log N) per operation? not well tested
 * Source: https://codeforces.com/blog/entry/10355
 * Verification: CEOI 2018 Day 2 Triangles
       *
           https://szkopul.edu.pl/problemset/problem/AzKAZ2RDiVTjeWSBolwoC5zl/site/?key=statement
       * vector is faster for this problem ...
 */

#include <ext/rope>
using namespace __gnu_cxx;
void ropeExample() {
       rope<int> v(5,0); // initialize with 5 zeroes
       FOR(i,sz(v)) v.mutable_reference_at(i) = i+1;
       FOR(i,5) v.pb(i+1); // constant time pb
       rope<int> cur = v.substr(1,2);
       v.erase(1,3); // erase 3 elements starting from 1st element
       for (rope<int>::iterator it = v.mutable_begin();
              it != v.mutable_end(); ++it) pr((int)*it,' ');
       ps(); // 1 5 1 2 3 4 5
       v.insert(v.mutable_begin()+2,cur); // or just 2
       v += cur; FOR(i,sz(v)) pr(v[i],' ');
       ps(); // 1 5 2 3 1 2 3 4 5 2 3
}

/*
Tested in cses https://cses.fi/problemset/task/1749/
https://cses.fi/problemset/result/10105636/ 2 times slower than
    ordered set
It's better to create rope with an given size and value

TLE with 2*10^5
Need revision!!

push_back() - O(log N).
pop_back() - O(log N)
insert(int x, crope r1): O(log N) and Worst O(N)
substr(int x, int l): O(log N)
replace(int x, int l, crope r1): O(log N).
*/
```

5 Graphs

5.1 BFS

```
2. BFS

vector<int> adj[n + 1];
bool visited[n + 1];
```

```
void bfs() {
	queue<int> q;
	q.push(0); // initial node
	visited[0] = true;
	while(q.size() > 0) {
		int c = q.front();
		q.pop();
		for (int a : adj[c]) {
			if (visited[a]) continue;
			q.push(a);
			visited[a] = true;
		}
	}
}
```

5.2 Cycle Detection

```
vector<vector<ll>> adj(2e5+5);
vector<ll> visited(2e5);
bool ok = false; // if cycle was found ok is true
vector<ll> cycle; // the found cycle
void dfs(ll x, vector<ll> &st) {
    if (ok || visited[x] == 2) {
        return;
    } else if (visited[x] == 1) {
        cycle.pb(x);
        while (st.back() != x) {
            cycle.pb(st.back());
            st.pop_back();
        }
        cycle.pb(x);
        reverse(all(cycle));
        ok = true;
        return;
    }
    visited[x] = 1;
    st.pb(x);
    for (auto y : adj[x]) {
        dfs(y, st);
    }
    st.pop_back();
    visited[x] = 2;
}

void test_case() {
    ll n, m;
    cin >> n >> m;

    for (int i =0; i < m;i ++) {
        ll x, y;
        cin >> x >> y;
        adj[x].pb(y);
    }

    vector<ll> st;
    for (int i = 1; i <= n; i++) {
        dfs(i, st);
    }

    if (ok) {
        cout << cycle.size() << "\n";
        for (int i = 0; i < cycle.size(); i++) {
            cout << cycle[i] << " \n" [i == cycle.size() - 1];
        }
    } else {
        cout << "IMPOSSIBLE\n";
    }
}
```

5.3 DFS

```
const int n = 1e6;
vector<int> adj[n + 1];
bool visited[n + 1];

void dfs(int x) {
	if (visited[x]) return;
	visited[x] = true;
	for (int &a : adj[x]) {
```

```
            dfs(x);
        }
}
```

5.4 Kruskal

```
/* Kruskal O(|edges|*Log|edges|)
   Finds the max/min spanning tree of an undirected graph
   provide the undirected edges with its costs vector<{cost(a, b),
       a, b}>
   and the size
*/
struct union_find {
    vl p;
    union_find(int n) : p(n,-1) {}

    ll find(ll x) {
        if (p[x] == -1) return x;
        return p[x] = find(p[x]);
    }

    bool group(ll a, ll b) {
        a = find(a);
        b = find(b);
        if (a == b) return false;
        p[a] = b;
        return true;
    }
};

ll kruskal(vector<tuple<ll,ll,ll>> &edges, ll nodes) {
    union_find uf(nodes+1);
    sort(all(edges));
    reverse(all(edges)); // for max
    ll answer = 0;
    for (auto edge : edges) {
        ll cost, a, b;
        tie(cost, a, b) = edge;
        if (uf.group(a, b))
            answer += cost;
    }
    return answer;
}
```

5.5 Shortest Path

5.5.1 BellmanFord Find Negative Cycle

```
// This uses Bellmanford algorithm to find a negative cycle
// O(n*m) m=edges, n=nodes
void test_case() {
    ll n, m;
    cin >> n >> m;
    vector<ll> dist(n+1);
    vector<ll> p(n+1);
    vector<tuple<ll,ll,ll>> edges(m);
    for (int i =0; i < m; i ++) {
        ll x, y, z;
        cin >> x >> y >> z;
        edges[i] = {x, y, z};
    }

    ll efe = -1;
    for (int i = 0; i < n; i++) {
        efe = -1;
        for (auto pp : edges) {
            ll x,y,z;
            tie(x,y,z) = pp;
            if (dist[x] + z < dist[y]) {
                dist[y] = dist[x] + z;
                p[y] = x;
                efe = y;
            }
        }
    }
    if (efe == -1) {
        cout << "NO\n";
    } else {
        cout << "YES\n";
        ll x = efe;
```

```
        for (int i = 0; i < n; i++) {
            x = p[x];
        }
        vector<ll> cycle;
        ll y = x;
        while (cycle.size() == 0 || y != x) {
            cycle.pb(y);
            y = p[y];
        }
        cycle.pb(x);
        reverse(all(cycle));
        for (int i =0; i < cycle.size(); i++) {
            cout << cycle[i] << " \n" [i == cycle.size() -1];
        }
    }
}
```

5.5.2 BellmanFord

```
/* BellmanFord O(|Nodes| * |Edges|)
Finds shortest path in a directed or undirected graph with
    negative weights.
Also you can find if the graph has negative cycles.
*/
const int inf = 1e9; // Check max possible distance value!!!
vector<tuple<int, int, int>> edges;
ll distance[n];

void bellmanFord() {
        for (int i = 0; i < n; i++) {
                distance[i] = inf;
        }
        distance[start] = 0;
        for (int i = 0; i < n - 1; i++) {
                //bool changed = false;
                // add one iteration (i < n) to valide negative
                    cicles
                for (auto& edge : edges) {
                        int a, b, w;
                        tie(a, b, w) = edge;
                        if (distance[a] + w < distance[b]) {
                                distance[b] = distance[a] + w;
                                //changed = true;
                        }
                }
                // if changed after all iterations, then exists
                    negative cycle
        }
}
```

5.5.3 Dijkstra K-Shortest Path

```
// Using djisktra, finds the k shortesth paths from 1 to n
// 2  n10  ^5, 1   m210   ^5, 1  weight10  ^9, 1  k10
// complexity seems O(k*m)
#define P pair<ll,ll>
void test_case() {
    ll n, m, k;
    cin >> n >> m >> k;
    vector<ll> visited(n+1, 0);
    vector<vector<pair<ll,ll>>> adj(n+1);
    for (int i = 0; i < m; i++) {
        ll a, b, c;
        cin >> a >> b >> c;
        adj[a].pb({b, c});
    }
    vector<ll> ans;
    priority_queue<P,vector<P>, greater<P>> q;
    q.push({0, 1});
    ll kk = k;
    while (q.size()) {
        ll x = q.top().S;
        ll z = q.top().F;
        q.pop();
        if (visited[x] >= kk) {
            continue;
        }
        visited[x]++;
        if (x == n) {
            ans.pb(z);
```

```
            k--;
            if (k == 0) break;
        }
        for (auto yy : adj[x]) {
            q.push({yy.S + z, yy.F});
        }
    }
    for (int i = 0; i < ans.size(); i++) {
        cout << ans[i] << " \n" [i == ans.size() - 1];
    }
}
```

5.5.4 Dijkstra

```
/* Dijkstra O(M + N * log M)
Finds the shortest path in a directed or undirected
graph with non-negative weights. */

const int inf = 1e9; // check max possible dist!!!!
vector<pair<int, int>> adj[n];
bool processed[n];
ll distance[n]; // Distance from Start to 'i'

void dijkstra() {
        priority_queue<pair<int, int>> q;
        for (int i = 0; i < n; i++) {
                distance[i] = inf;
        }
        int start = 0;
        distance[start] = 0;
        q.push({0, start});
        while (q.size() > 0) {
                int c = q.top().second;
                q.pop();
                if (processed[c]) continue;
                processed[c] = true;
                for (auto& a : adj[c]) {
                        int u = a.first;
                        int w = a.second;
                        if (distance[c] + w < distance[u]) {
                                distance[u] = distance[c] + w;
                                q.push({-distance[u], u});
                        }
                }
        }
}
```

5.5.5 Floyd Warshall Negative Weights

```
// Find the minimum distance from any i to j, with negative
    weights.
// dist[i][j] == -inf, there some negative loop from i to j
// dist[i][j] == inf, from i cannot reach j
// otherwise the min dist from i to j

// take care of the max a path from i to j, it has to be less
    than inf
const ll inf = INT32_MAX;
void test_case() {
    ll n, m; // nodes, edges
    vector<vector<ll>> dist(n, vector<ll>(n, inf));
    for (int i = 0; i < n; i++) dist[i][i] = 0;
    for (int i = 0; i < m; i++) {
        ll a, b, w;
        cin >> a >> b >> w; // negative weights
        dist[a][b] = min(dist[a][b], w);
    }
    // floid warshall
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (dist[i][k] == inf || dist[k][j] == inf)
                    continue;
                dist[i][j] = min(dist[i][j], dist[i][k] +
                    dist[k][j]);
            }
        }
    }
    // find negative cycles for a node
    for (int i = 0; i < n; i++) {
```

```
        if (dist[i][i] < 0) dist[i][i] = -inf;
    }
    // find negative cycles betweens a routes from i to j
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n ; j++) {
            for (int k = 0; k < n; k++) {
                if (dist[k][k] < 0 && dist[i][k] != inf &&
                    dist[k][j] != inf) {
                    dist[i][j] = -inf;
                }
            }
        }
    }
}
```

5.6 Strongly Connected Components

```
/*
 Tarjan t(graph); provides you the SCC of that graph
 passing the adjancency list of the graph (as vector<vl>)

 This is 0-indexed, (but you can have node 0 as dummy-node)
 Use t.comp[x] to get the component of node x
 SCC is the total number of components
 adjComp() gives you the adjacency list of strongly components
*/
struct Tarjan {
    vl low, pre, comp;
    ll cnt, SCC, n;
    vvl g;
    const int inf = 1e9;

    Tarjan(vvl &adj) {
        n = adj.size();
        g = adj;
        low = vl(n);
        pre = vl(n,-1);
        cnt = SCC = 0;
        comp = vl(n,-1);
        for (int i = 0;i<n;i++)
            if (pre[i] == -1) tarjan(i);
    }

    stack<int> st;
    void tarjan(int u) {
        low[u] = pre[u] = cnt++;
        st.push(u);
        for (auto &v : g[u]) {
            if (pre[v] == -1) tarjan(v);
            low[u] = min(low[u],low[v]);
        }
        if (low[u] == pre[u]) {
            while (true) {
                int v = st.top();st.pop();
                low[v] = inf;
                comp[v] = SCC;
                if (u == v) break;
            }
            SCC++;
        }
    }

    vvl adjComp() {
        vvl adj(SCC);
        for (int i = 0;i<n;i++) {
            for (auto j : g[i]) {
                if (comp[i] == comp[j]) continue;
                adj[comp[i]].pb(comp[j]);
            }
        }
        for (int i = 0;i<SCC;i++) {
            sort(all(adj[i]));
            adj[i].erase(
                unique(all(adj[i])),
                adj[i].end());
        }
        return adj;
    }
};
```

```
/* Another way is with with Kosaraju:
    1. Find topological order of G
    2. Run dfs in topological order in reverse Graph
       to find o connected component
*/
```

5.7 Topological Sort

```
// Find the topological order of a graph in O(n)
const int N = 1e5;
vector<vector<ll>> adj(N + 10);
vector<ll> visited(N +10);
bool cycle = false; // reports if doesn't exists a topological
    sort
vector<ll> topo;

void dfs(ll x) {
    if (visited[x] == 2) {
        return;
    } else if (visited[x] == 1) {
        cycle = true;
        return;
    }
    visited[x] = 1;
    for (auto y : adj[x]) dfs(y);
    visited[x] = 2;
    topo.pb(x);
}

void test_case() {
    ll n, m; cin >> n >> m;
    for (int i =0; i < m; i++) {
        ll x, y; cin >> x >> y;
        adj[x].pb(y);
    }
    for (int i = 1; i <= n; i++) dfs(i);
    reverse(topo.begin(), topo.end());
    if (cycle) {
        cout << "IMPOSSIBLE\n";
    } else {
        for (int i =0; i < n; i++) {
            cout << topo[i] << " \n" [i == n - 1];
        }
    }
}
```

5.8 Transitive Closure

```
/* Transitive Closure O(n^3)
Check if from A can reach B in an undirected graph.
*/
const int inf = 1e9;
vector<int> adj[n];
ll distance[n][n];

void floydWarshall() {
        for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                        distance[i][j] = false;
                }
        }
        for (int i = 0; i < n; i++) {
                for (int b : adj[i]) {
                        distance[i][b] = true;
                }
        }
        for (int k = 0; k < n; k++) {
                for (int i = 0; i < n; i++) {
                        for (int j = 0; j < n; j++) {
                                distance[i][j] |= distance[i][k] &
                                    distance[k][j];
                        }
                }
        }
}
```

5.9 Two Sat

```
/*
2-Sat (Boolean satisfiability problem with 2-clause literals)
Complexity: O(n)
Tested: https://cses.fi/problemset/task/1684

To find a solution that makes this true with N boolean vars as
    form:
  (x or y) and (~x or y) and (z or ~x) and d and (x => y)

Call s.satisfiable() to see if solution, and sat2.value to see
values of variables
*/

struct sat2 {
  int n;
  vector<vector<vector<int>>> g;
  vector<int> tag;
  vector<bool> seen, value;
  stack<int> st;
  sat2(int n) : n(n), g(2, vector<vector<int>>(2*n)), tag(2*n),
      seen(2*n), value(2*n) { }
  int neg(int x) { return 2*n-x-1; }
  void add_or(int u, int v) { implication(neg(u), v); }
  void make_true(int u) { add_edge(neg(u), u); }
  void make_false(int u) { make_true(neg(u)); }
  void eq(int u, int v) {
    implication(u, v);
    implication(v, u);
  }
  void diff(int u, int v) { eq(u, neg(v)); }
  void implication(int u, int v) {
    add_edge(u, v);
    add_edge(neg(v), neg(u));
  }
  void add_edge(int u, int v) {
    g[0][u].push_back(v);
    g[1][v].push_back(u);
  }
  void dfs(int id, int u, int t = 0) {
    seen[u] = true;
    for(auto& v : g[id][u])
      if(!seen[v])
        dfs(id, v, t);
    if(id == 0) st.push(u);
    else tag[u] = t;
  }
  void kosaraju() {
    for(int u = 0; u < n; u++) {
      if(!seen[u]) dfs(0, u);
      if(!seen[neg(u)]) dfs(0, neg(u));
    }
    fill(seen.begin(), seen.end(), false);
    int t = 0;
    while(!st.empty()) {
      int u = st.top(); st.pop();
      if(!seen[u]) dfs(1, u, t++);
    }
  }
  bool satisfiable() {
    kosaraju();
    for(int i = 0; i < n; i++) {
      if(tag[i] == tag[neg(i)]) return false;
      value[i] = tag[i] > tag[neg(i)];
    }
    return true;
  }
};
```

6 Flow

6.1 Hungarian

```
// Halla el maximo match en un grafo bipartito con pesos (min
    cost) O(V ^ 3)

typedef ll T;
const T inf = 1e18;

struct hung {
    int n, m;
```

```
    vector<T> u, v; vector<int> p, way;
    vector<vector<T>> g;

    hung(int n, int m):
        n(n), m(m), g(n+1, vector<T>(m+1, inf-1)),
        u(n+1), v(m+1), p(m+1), way(m+1) {}

    void set(int u, int v, T w) { g[u+1][v+1] = w; }

    T assign() {
        for (int i = 1; i <= n; ++i) {
            int j0 = 0; p[0] = i;
            vector<T> minv(m+1, inf);
            vector<char> used(m+1, false);
            do {
                used[j0] = true;
                int i0 = p[j0], j1; T delta = inf;
                for (int j = 1; j <= m; ++j) if (!used[j]) {
                    T cur = g[i0][j] - u[i0] - v[j];
                    if (cur < minv[j]) minv[j] = cur, way[j] = j0;
                    if (minv[j] < delta) delta = minv[j], j1 = j;
                }
                for (int j = 0; j <= m; ++j)
                    if (used[j]) u[p[j]] += delta, v[j] -= delta;
                    else minv[j] -= delta;
                j0 = j1;
            } while (p[j0]);
            do {
                int j1 = way[j0]; p[j0] = p[j1]; j0 = j1;
            } while (j0);
        }
        return -v[0];
    }
};
```

6.2 Max Flow

```
//#define int long long // take care int overflow with this
//#define vi vector<long long>
struct Dinitz{
const int INF = 1e9 + 7;
Dinitz(){}
Dinitz(int n, int s, int t) {init(n, s, t);}

void init(int n, int s, int t)
{
    S = s, T = t;
    nodes = n;
    G.clear(), G.resize(n);
    Q.resize(n);
}
struct flowEdge
{
    int to, rev, f, cap;
};

vector<vector<flowEdge> > G;

// Aade  arista (st -> en) con su capacidad
void addEdge(int st, int en, int cap) {
    flowEdge A = {en, (int)G[en].size(), 0, cap};
    flowEdge B = {st, (int)G[st].size(), 0, 0};
    G[st].pb(A);
    G[en].pb(B);
}

int nodes, S, T; // asignar estos valores al armar el grafo G
                 // nodes = nodos en red de flujo. Hacer
                     G.clear(); G.resize(nodes);
vi work, lvl;
vi Q;

bool bfs() {
    int qt = 0;
    Q[qt++] = S;
    lvl.assign(nodes, -1);
    lvl[S] = 0;
    for (int qh = 0; qh < qt; qh++) {
        int v = Q[qh];
        for (flowEdge &e : G[v]) {
            int u = e.to;
```

```
            if (e.cap <= e.f || lvl[u] != -1) continue;
            lvl[u] = lvl[v] + 1;
            Q[qt++] = u;
        }
    }
    return lvl[T] != -1;
}

int dfs(int v, int f) {
    if (v == T || f == 0) return f;
    for (int &i = work[v]; i < G[v].size(); i++) {
        flowEdge &e = G[v][i];
        int u = e.to;
        if (e.cap <= e.f || lvl[u] != lvl[v] + 1) continue;
        int df = dfs(u, min(f, e.cap - e.f));
        if (df) {
            e.f += df;
            G[u][e.rev].f -= df;
            return df;
        }
    }
    return 0;
}

int maxFlow() {
    int flow = 0;
    while (bfs()) {
        work.assign(nodes, 0);
        while (true) {
            int df = dfs(S, INF);
            if (df == 0) break;
            flow += df;
        }
    }
    return flow;
}
};

void test_case() {
    ll n, m, s, t;
    cin >> n >> m >> s >> t;
    Dinitz flow;
    flow.init(n, s, t);
    for (int i =0; i < m; i++) {
        ll a, b, c;
        cin >> a >> b >> c;
        flow.addEdge(a, b, c);
    }
    ll f = flow.maxFlow(); // max flow

    vector<tuple<ll,ll,ll>> edges; // edges used with flow
    for (int i = 0; i < n; i++) {
        for (auto edge : flow.G[i]) {
            if (edge.f > 0) {
                edges.pb({i, edge.to, edge.f});
            }
        }
    }
}
```

6.3 Min Cost Max Flow

```
// O(min(E^2*V^2, E*V*FLOW))
// Min Cost Max Flow Dinits
struct CheapDinitz{
    const int INF = 1e9 + 7;

    CheapDinitz() {}
    CheapDinitz(int n, int s, int t) {init(n, s, t);}

    int nodes, S, T;
    vi dist;
    vi pot, curFlow, prevNode, prevEdge, Q, inQue;

    struct flowEdge{
        int to, rev, flow, cap, cost;
    };
    vector<vector<flowEdge>> G;
    void init(int n, int s, int t)
    {
        nodes = n, S = s, T = t;
```

```
    curFlow.assign(n, 0), prevNode.assign(n, 0),
        prevEdge.assign(n, 0);
    Q.assign(n, 0), inQue.assign(n, 0);
    G.clear();
    G.resize(n);
}

void addEdge(int s, int t, int cap, int cost)
{
    flowEdge a = {t, (int)G[t].size(), 0, cap, cost};
    flowEdge b = {s, (int)G[s].size(), 0, 0, -cost};
    G[s].pb(a);
    G[t].pb(b);
}

void bellmanFord()
{
    pot.assign(nodes, INF);
    pot[S] = 0;
    int qt = 0;
    Q[qt++] = S;
    for (int qh = 0; (qh - qt) % nodes != 0; qh++)
    {
        int u = Q[qh % nodes];
        inQue[u] = 0;
        for (int i = 0; i < (int)G[u].size(); i++)
        {
            flowEdge &e = G[u][i];
            if (e.cap <= e.flow) continue;
            int v = e.to;
            int newDist = pot[u] + e.cost;
            if (pot[v] > newDist)
            {
                pot[v] = newDist;
                if (!inQue[v])
                {
                    Q[qt++ % nodes] = v;
                    inQue[v] = 1;
                }
            }
        }
    }
}

ii MinCostFlow()
{
    bellmanFord();
    int flow = 0;
    int flowCost = 0;
    while (true) // always a good start for an algorithm :v
    {
        set<ii> s;
        s.insert({0, S});
        dist.assign(nodes, INF);
        dist[S] = 0;
        curFlow[S] = INF;
        while (s.size() > 0)
        {
            int u = s.begin() -> s;
            int actDist = s.begin() -> f;
            s.erase(s.begin());
            if (actDist > dist[u]) continue;
            for (int i = 0; i < (int)G[u].size(); i++)
            {
                flowEdge &e = G[u][i];
                int v = e.to;
                if (e.cap <= e.flow) continue;
                int newDist = actDist + e.cost + pot[u] -
                    pot[v];
                if (newDist < dist[v])
                {
                    dist[v] = newDist;
                    s.insert({newDist, v});
                    prevNode[v] = u;
                    prevEdge[v] = i;
                    curFlow[v] = min(curFlow[u], e.cap - e.flow);
                }
            }
        }
        if (dist[T] == INF)
            break;
        for (int i = 0; i < nodes; i++)
```

```
            pot[i] += dist[i];
        int df = curFlow[T];
        flow += df;
        for (int v = T; v != S; v = prevNode[v])
        {
            flowEdge &e = G[prevNode[v]][prevEdge[v]];
            e.flow += df;
            G[v][e.rev].flow -= df;
            flowCost += df * e.cost;
        }
    }
    return {flow, flowCost};
  }
};
```

6.4 Min Cut

```
/*
Minimum Cut, need revision for general cases!!!

Tested in https://cses.fi/problemset/task/1695/
  with undirected graph with capacity 1

1. Run Max Flow Algorithm
2. Get reachable vertices from source in the residual graph!
   using BFS or DFS, call this reachable vertices = S
3. Iterate S, iterate edges of S in normal graph that not belongs
    to S,
   thats the mincut edges of the answer.
*/

void test_case() {
    ll n, m; cin >> n >> m;
    vvl g(n);
    Dinitz f(n,0,n-1); // add Dinitz for maxFlow!!
    for (int i =0;i<m;i++) {
        ll x, y;cin >>x >> y;x--,y--;
        g[x].pb(y); g[y].pb(x); //this example the graph is
            bidirectional
        f.addEdge(x,y,1); f.addEdge(y,x,1);
    }
    f.maxFlow(); // step 1

    vvl residual(n); // Step 2
    for (int i =0;i<n;i++) for (auto j:f.G[i])
        if ((j.f<0 && j.cap==0) || (j.f==0 && j.cap > 0))
            residual[i].pb(j.to); // residual graph

    set<ll> vis;
    function<void(int)> dfs = [&](int x) {
        vis.insert(x); // dfs on residual from source
        for (auto y : residual[x]) if (!vis.count(y)) dfs(y);
    };
    dfs(0);

    vector<pair<ll,ll>> ans; // step 3
    for (auto x : vis)
        for (auto y : g[x])
            if (!vis.count(y))
                ans.pb({x,y});

    cout << ans.size() << "\n";
    for (auto e : ans) {
        cout << e.F + 1 << " " << e.S + 1 << "\n";
    }
}
```

7 Tree

7.1 Euler Tour - Sum

```
/* Euler Tour Sum Path O(n log n)
Given a Tree, and values in each node, process this queries:
 - Calculate the sum of values in the Path 1 to a 'given node'.
 - Update value of a node

Also you can extend this to sum path from A to B with updates.
Just add LCA and sum(0,A) + sum(0,B) - 2*(sum(0,lca)-values[lca])
```

```
Tested: https://cses.fi/problemset/task/1138/
*/
void test_case() {
    ll n, m; cin >> n >> m;
    vector<vl> adj(n+1); // 1-indexed
    vl nums(n+1), tin(n+1), tout(n+1);
    FenwickTree tree(2*n+2); // Add Fenwick Tree 0-indexed

    for (int i = 1; i <= n; i++) cin >> nums[i];
    for (int i = 0; i < n-1; i++) {
        ll x, y; cin >> x >> y;
        adj[x].pb(y); adj[y].pb(x);
    }

    ll time = 0;
    function<void(int,int)> dfs =[&](int x, int p) {
        tin[x] = time++;
        for (auto y : adj[x]) if (y != p) dfs(y, x);
        tout[x] = time++;
        tree.add(tin[x], nums[x]);
        tree.add(tout[x], -nums[x]);
    };
    dfs(1, 0);
    for (int i =0; i < m; i++) {
        ll t, x;
        cin >> t >> x;
        if (t == 1) { // update
            ll y; cin >> y;
            ll diff = y - nums[x];
            nums[x] = y;
            tree.add(tin[x], diff);
            tree.add(tout[x], -diff);
        } else { // query
            cout << tree.sum(0, tin[x]) << "\n";
        }
    }
}
```

7.2 K-th Parent

```
/* K-th Parent.cpp
Given and Tree, and Q queries to see the K-Parent of a node
*/
const int LOG = 20;
vvl parent(LOG, vl(2e5 + 10, -1));

void test_case() { // 1-based indexed
    ll n, q; cin >> n >> q;

    for (int i = 0; i < n - 1; i++)
        cin >> parent[0][i+2];

    for (int j = 1; j < LOG; j++) {
        for (int i = 1; i <= n; i++) {
            if (parent[j-1][i] == -1) continue;
            parent[j][i] = parent[j-1][parent[j-1][i]];
        }
    }
    for (int i = 0; i < q; i++) { // Queries
        ll x, k;
        cin >> x >> k;
        ll ans = x;
        ll y = 0;
        while (k) {
            if (k&1) {
                ans = parent[y][ans];
            }
            if (ans == -1) break;
            k /= 2;
            y++;
        }
        cout << ans << "\n";
    }
}
```

7.3 Lowest Common Ancestor

```
// Give a rooted tree, find the Lowest Common Ancestor node
// of A and B.
```

```
// Tested https://cses.fi/problemset/task/1688/
vector<vector<ll>> up;
vector<ll> depth;
const int LOG = 18; // for 2e5

void init(vector<vector<ll>> children, ll n) {
    up.assign(LOG,vector<ll>(n,0));
    depth.assign(n,0);
    function<void(int)> dfs = [&](int x) {
        for (auto y : children[x]) {
            up[0][y] = x;
            depth[y] = depth[x] +1;
            dfs(y);
        }
    };
    int root = 0; dfs(root);

    for (int i = 1; i < LOG; i++)
        for (int j = 0; j < n; j++)
            up[i][j] = up[i-1][up[i-1][j]];
}

ll kParent(ll x, ll k) {
    ll i = 0;
    while (k) {
        if (k & 1) x = up[i][x];
        k >>= 1;
        i++;
    }
    return x;
}

ll query(ll x, ll y) {
    if (depth[x] < depth[y]) swap(x, y);
    x = kParent(x, depth[x] - depth[y]);
    if (x == y) {
        return x;
    }
    for (int i = LOG - 1; i >= 0; i--) {
        if (up[i][x] != up[i][y]) {
            x = up[i][x];
            y = up[i][y];
        }
    }
    return up[0][x];
}

void test_case() {
    ll n, q; cin >> n >> q;
    vvl children(n);
    for (int i = 1; i < n; i++) {
        ll p; cin >> p; p--;
        children[p].pb(i);
    }
    init(children, n);

    for (int i = 0; i < q; i++) {
        ll x, y;cin >> x >> y;
        x--,y--;
        cout << query(x, y) + 1 << "\n";
    }
}
```

8 Strings

8.1 Fast Hashing

```
const int N = 1e6 + 9; // Max size

int power(long long n, long long k, const int mod) {
  int ans = 1 % mod;
  n %= mod;
  if (n < 0) n += mod;
  while (k) {
    if (k & 1) ans = (long long) ans * n % mod;
    n = (long long) n * n % mod;
    k >>= 1;
  }
  return ans;
}
```

```
const int MOD1 = 127657753, MOD2 = 987654319;
const int p1 = 137, p2 = 277;
int ip1, ip2;
pair<int, int> pw[N], ipw[N];
void init() { // Call init() first!!!
  pw[0] = {1, 1};
  for (int i = 1; i < N; i++) {
    pw[i].first = 1LL * pw[i - 1].first * p1 % MOD1;
    pw[i].second = 1LL * pw[i - 1].second * p2 % MOD2;
  }
  ip1 = power(p1, MOD1 - 2, MOD1);
  ip2 = power(p2, MOD2 - 2, MOD2);
  ipw[0] = {1, 1};
  for (int i = 1; i < N; i++) {
    ipw[i].first = 1LL * ipw[i - 1].first * ip1 % MOD1;
    ipw[i].second = 1LL * ipw[i - 1].second * ip2 % MOD2;
  }
}
struct Hashing {
  int n;
  string s; // 0 - indexed
  vector<pair<int, int>> hs; // 1 - indexed
  Hashing() {}
  Hashing(string _s) {
    n = _s.size();
    s = _s;
    hs.emplace_back(0, 0);
    for (int i = 0; i < n; i++) {
      pair<int, int> p;
      p.first = (hs[i].first + 1LL * pw[i].first * s[i] % MOD1) %
          MOD1;
      p.second = (hs[i].second + 1LL * pw[i].second * s[i] % MOD2)
          % MOD2;
      hs.push_back(p);
    }
  }
  pair<int, int> get_hash(int l, int r) { // 1-indexed
    assert(1 <= l && l <= r && r <= n);
    pair<int, int> ans;
    ans.first = (hs[r].first - hs[l - 1].first + MOD1) * 1LL *
        ipw[l - 1].first % MOD1;
    ans.second = (hs[r].second - hs[l - 1].second + MOD2) * 1LL *
        ipw[l - 1].second % MOD2;
    return ans;
  }
  pair<int,int> get(int l, int r) { // 0-indexed
    return get_hash(l+1,r+1);
  }
  pair<int, int> get_hash() {
    return get_hash(1, n);
  }
};
```

8.2 KMP Automaton

```
// Very useful for some DP's with strings
// aut[i][j], you are in 'i' position, and choose character 'j',
    the next position.

const int MAXN = 1e5 + 5, alpha = 26;
const char L = 'A';
int aut[MAXN][alpha]; // aut[i][j] = a donde vuelvo si estoy en i
    y pongo una j

void build(string &s) {
    int lps = 0;
    aut[0][s[0]-L] = 1;
    int n = s.size();
    for (int i = 1; i < n+1; i++) {
        for (int j = 0; j < alpha; j++) aut[i][j] = aut[lps][j];
        if (i < n) {
            aut[i][s[i]-L] = i + 1;
            lps = aut[lps][s[i]-L];
        }
    }
}
```

8.3 KMP

```
// Given string s, and patter p, count and find
// occurences of p in s. O(n)
struct KMP {
    int kmp(vector<ll> &s, vector<ll> &p) {
        int n = s.size(), m = p.size(), cnt = 0;
        vector<int> pf = prefix_function(p);
        for(int i = 0, j = 0; i < n; i++) {
            while(j && s[i] != p[j]) j = pf[j-1];
            if(s[i] == p[j]) j++;
            if(j == m) {
                cnt++;
                j = pf[j-1];
            }
        }
        return cnt;
    }

    vector<int> prefix_function(vector<ll> &s) {
        int n = s.size();
        vector<int> pf(n);
        pf[0] = 0;
        for (int i = 1, j = 0; i < n; i++) {
            while (j && s[i] != s[j]) j = pf[j-1];
            if (s[i] == s[j]) j++;
            pf[i] = j;
        }
        return pf;
    }
};
```

9 Geometry

9.1 Closest Pair Of Points

```
// It seems O(n log n), not sure but it worked for 50000
// This algorithms is not the best, TLE in CSES
// https://cses.fi/problemset/task/2194
#define x first
#define y second
long long dist2(pair<int, int> a, pair<int, int> b) {
        return 1LL * (a.x - b.x) * (a.x - b.x) + 1LL * (a.y - b.y)
            * (a.y - b.y);
}
pair<int, int> closest_pair(vector<pair<int, int>> a) {
        int n = a.size();
        assert(n >= 2);
        vector<pair<pair<int, int>, int>> p(n);
        for (int i = 0; i < n; i++) p[i] = {a[i], i};
        sort(p.begin(), p.end());
        int l = 0, r = 2;
        long long ans = dist2(p[0].x, p[1].x);
        pair<int, int> ret = {p[0].y, p[1].y};
        while (r < n) {
                while (l < r && 1LL * (p[r].x.x - p[l].x.x) *
                    (p[r].x.x - p[l].x.x) >= ans) l++;
                for (int i = l; i < r; i++) {
                        long long nw = dist2(p[i].x, p[r].x);
                        if (nw < ans) {
                                ans = nw;
                                ret = {p[i].y, p[r].y};
                        }
                }
                r++;
        }
        return ret;
}

// Tested:
    https://vjudge.net/solution/52922194/ccPUXODAMWTzpzCEvXbV
void test_case() {
    ll n;
    cin >> n;
    vector<pair<int,int>> points(n);
    for (int i = 0;i<n;i++) cin >> points[i].x >> points[i].y;
    auto ans = closest_pair(points);
    cout << fixed << setprecision(6);
    if (ans.F > ans.S) swap(ans.F,ans.S);
```

```
    ld dist = sqrtl(dist2(points[ans.F],points[ans.S]));
    cout << ans.F << " " << ans.S << " " << dist << endl;
}
```

9.2 Convex Hull

```
// Given a Polygon, find its convex hull polygon
// O(n)
struct pt {
    ll x, y;

    pt operator - (pt p) { return {x-p.x, y-p.y}; }

    bool operator == (pt b) { return x == b.x && y == b.y; }
    bool operator != (pt b) { return !((*this) == b); }
    bool operator < (const pt &o) const { return y < o.y || (y ==
        o.y && x < o.x); }
};

ll cross(pt a, pt b) { return a.x*b.y - a.y*b.x; } // x = 180 ->
    sin = 0
ll orient(pt a, pt b, pt c) { return cross(b-a,c-a); }//
    clockwise = -
ld norm(pt a) { return a.x*a.x + a.y*a.y; }
ld abs(pt a) { return sqrt(norm(a)); }


struct polygon {
    vector<pt> p;
    polygon(int n) : p(n) {}

    void delete_repetead() {
        vector<pt> aux;
        sort(p.begin(), p.end());
        for(pt &i : p)
            if(aux.empty() || aux.back() != i)
              aux.push_back(i);
        p.swap(aux);
    }

    int top = -1, bottom = -1;
    void normalize() { /// polygon is CCW
        bottom = min_element(p.begin(), p.end()) - p.begin();
        vector<pt> tmp(p.begin()+bottom, p.end());
        tmp.insert(tmp.end(), p.begin(), p.begin()+bottom);
        p.swap(tmp);
        bottom = 0;
        top = max_element(p.begin(), p.end()) - p.begin();
    }

    void convex_hull() {
        sort(p.begin(), p.end());
        vector<pt> ch;
        ch.reserve(p.size()+1);
        for(int it = 0; it < 2; it++) {
            int start = ch.size();
            for(auto &a : p) {
                /// if colineal are needed, use < and remove
                    repeated points
                while(ch.size() >= start+2 &&
                    orient(ch[ch.size()-2], ch.back(), a) <= 0)
                    ch.pop_back();
                ch.push_back(a);
            }
            ch.pop_back();
            reverse(p.begin(), p.end());
        }
        if(ch.size() == 2 && ch[0] == ch[1]) ch.pop_back();
        /// be careful with CH of size < 3
        p.swap(ch);
    }

    ld perimeter() {
        ld per = 0;
        for(int i = 0, n = p.size(); i < n; i++)
          per += abs(p[i] - p[(i+1)%n]);
        return per;
    }
};
```

9.3 Heron Formula

```
ld triangle_area(ld a, ld b, ld c) {
    ld s = (a + b + c) / 2;
    return sqrtl(s * (s - a) * (s - b) * (s - c));
}
```

9.4 Point In General Polygon

```
// Use insidepoly(poly, point)
// Returns if a point is inside=0, outside=1, onedge=2
// tested
    https://vjudge.net/solution/45869791/BIPDAUMWyupUW18AlWgd
// Seems to be O(n)??
int inf = 1 << 30;
int INSIDE = 0;
int OUTSIDE = 1;
int ONEDGE = 2;
int COLINEAR = 0;
int CW = 1;
int CCW = 2;
typedef long double ld;
struct point {
    ld x, y;
    point(ld xloc, ld yloc) : x(xloc), y(yloc) {}
    point() {}
    point& operator= (const point& other) {
        x = other.x, y = other.y;
        return *this;
    }
    int operator == (const point& other) const {
        return (abs(other.x - x) < .00001 && abs(other.y - y) <
            .00001);
    }
    int operator != (const point& other) const {
        return !(abs(other.x - x) < .00001 && abs(other.y - y) <
            .00001);
    }
    bool operator< (const point& other) const {
        return (x < other.x ? true : (x == other.x && y <
            other.y));
    }
};

struct vect { ld i, j; };

struct segment {
    point p1, p2;
    segment(point a, point b) : p1(a), p2(b) {}
    segment() {}
};

long double crossProduct(point A, point B, point C) {
    vect AB, AC;
    AB.i = B.x - A.x;
    AB.j = B.y - A.y;
    AC.i = C.x - A.x;
    AC.j = C.y - A.y;
    return (AB.i * AC.j - AB.j * AC.i);
}

int orientation(point p, point q, point r) {
    int val = int(crossProduct(p, q, r));
    if(val == 0) {
        return COLINEAR;
    }
    return (val > 0) ? CW : CCW;
}

bool onSegment(point p, segment s) {
    return (p.x <= max(s.p1.x, s.p2.x) && p.x >= min(s.p1.x,
        s.p2.x) &&
            p.y <= max(s.p1.y, s.p2.y) && p.y >= min(s.p1.y,
                s.p2.y));
}

vector<point> intersect(segment s1, segment s2) {
    vector<point> res;
    point a = s1.p1, b = s1.p2, c = s2.p1, d = s2.p2;
```

```
    if(orientation(a, b, c) == 0 && orientation(a, b, d) == 0 &&
       orientation(c, d, a) == 0 && orientation(c, d, b) == 0) {
        point min_s1 = min(a, b), max_s1 = max(a, b);
        point min_s2 = min(c, d), max_s2 = max(c, d);

        if(min_s1 < min_s2) {
            if(max_s1 < min_s2) {
                return res;
            }
        }
        else if(min_s2 < min_s1 && max_s2 < min_s1) {
            return res;
        }

        point start = max(min_s1, min_s2), end = min(max_s1,
            max_s2);
        if(start == end) {
            res.push_back(start);
        }
        else {
            res.push_back(min(start, end));
            res.push_back(max(start, end));
        }
        return res;
    }
    ld x1 = (b.x - a.x);
    ld y1 = (b.y - a.y);
    ld x2 = (d.x - c.x);
    ld y2 = (d.y - c.y);
    ld u1 = (-y1 * (a.x - c.x) + x1 * (a.y - c.y)) / (-x2 * y1 +
        x1 * y2);
    ld u2 = (x2 * (a.y - c.y) - y2 * (a.x - c.x)) / (-x2 * y1 + x1
        * y2);

    if(u1 >= 0 && u1 <= 1 && u2 >= 0 && u2 <= 1) {
        res.push_back(point((a.x + u2 * x1), (a.y + u2 * y1)));
    }
    return res;
}

int insidepoly(vector<point> poly, point p) {
    bool inside = false;
    point outside(inf, p.y);
    vector<point> intersection;

    for(unsigned int i = 0, j = poly.size()-1; i < poly.size();
        i++, j = i-1) {
        if(p == poly[i] || p == poly[j]) {
            return ONEDGE;
        }
        if(orientation(p, poly[i], poly[j]) == COLINEAR &&
            onSegment(p, segment(poly[i], poly[j]))) {
            return ONEDGE;
        }
        intersection = intersect(segment(p, outside),
            segment(poly[i], poly[j]));
        if(intersection.size() == 1) {
            if(poly[i] == intersection[0] && poly[j].y <= p.y) {
                continue;
            }
            if(poly[j] == intersection[0] && poly[i].y <= p.y) {
                continue;
            }
            inside = !inside;
        }
    }

    return inside ? INSIDE : OUTSIDE;
}
//
```

9.5 Point in Convex Polygon

```
// Check if a point is in, on, or out a convex Polygon
// in O(log n)
ll IN = 0;
ll ON = 1;
ll OUT = 2;
vector<string> ANS = {"IN", "ON", "OUT"};

#define pt pair<ll,ll>
```

```
#define x first
#define y second

pt sub(pt a, pt b) { return {a.x - b.x, a.y - b.y}; }
ll cross(pt a, pt b) { return a.x*b.y - a.y*b.x; } // x = 180 ->
    sin = 0
ll orient(pt a, pt b, pt c) { return cross(sub(b,a),sub(c,a));
    }// clockwise = -

// poly is in clock wise order
ll insidePoly(vector<pt> &poly, pt query) {
    ll n = poly.size();
    ll left = 1;
    ll right = n - 2;
    ll ans = -1;
    if (!(orient(poly[0], poly[1], query) <= 0
         && orient(poly[0], poly[n-1], query) >= 0)) {
        return OUT;
    }
    while (left <= right) {
        ll mid = (left + right) / 2;
        if (orient(poly[0], poly[mid], query) <= 0) {
            left = mid + 1;
            ans = mid;
        } else {
            right = mid - 1;
        }
    }
    left = ans;
    right = ans + 1;
    if (orient(poly[left], query, poly[right]) < 0) {
        return OUT;
    }
    if (orient(poly[left], poly[right], query) == 0
       || (left == 1 && orient(poly[0], poly[left], query) == 0)
       || (right == n-1 && orient(poly[0], poly[right], query) ==
           0)) {
        return ON;
    }
    return IN;
}
```

9.6 Polygon Diameter

```
// Given a set of points, it returns
// the diameter (the biggest distance between 2 points)
// tested: https://open.kattis.com/submissions/13937489
const double eps = 1e-9;
int sign(double x) { return (x > eps) - (x < -eps); }

struct PT {
    double x, y;
    PT() { x = 0, y = 0; }
    PT(double x, double y) : x(x), y(y) {}
    PT operator - (const PT &a) const { return PT(x - a.x, y -
        a.y); }
    bool operator < (PT a) const { return sign(a.x - x) == 0 ? y <
        a.y : x < a.x; }
    bool operator == (PT a) const { return sign(a.x - x) == 0 &&
        sign(a.y - y) == 0; }

};
inline double dot(PT a, PT b) { return a.x * b.x + a.y * b.y; }
inline double dist2(PT a, PT b) { return dot(a - b, a - b); }
inline double dist(PT a, PT b) { return sqrt(dot(a - b, a - b)); }
inline double cross(PT a, PT b) { return a.x * b.y - a.y * b.x; }
inline int orientation(PT a, PT b, PT c) { return sign(cross(b -
    a, c - a)); }

double diameter(vector<PT> &p) {
    int n = (int)p.size();
    if (n == 1) return 0;
    if (n == 2) return dist(p[0], p[1]);
    double ans = 0;
    int i = 0, j = 1;
    while (i < n) {
        while (cross(p[(i + 1) % n] - p[i], p[(j + 1) % n] - p[j])
            >= 0) {
                ans = max(ans, dist2(p[i], p[j]));
                j = (j + 1) % n;
        }
        ans = max(ans, dist2(p[i], p[j]));
        i++;
```

```
    }
    return sqrt(ans);
}
vector<PT> convex_hull(vector<PT> &p) {
        if (p.size() <= 1) return p;
        vector<PT> v = p;
    sort(v.begin(), v.end());
    vector<PT> up, dn;
    for (auto& p : v) {
        while (up.size() > 1 && orientation(up[up.size() - 2],
            up.back(), p) >= 0) {
            up.pop_back();
        }
        while (dn.size() > 1 && orientation(dn[dn.size() - 2],
            dn.back(), p) <= 0) {
            dn.pop_back();
        }
        up.push_back(p);
        dn.push_back(p);
    }
    v = dn;
    if (v.size() > 1) v.pop_back();
    reverse(up.begin(), up.end());
    up.pop_back();
    for (auto& p : up) {
        v.push_back(p);
    }
    if (v.size() == 2 && v[0] == v[1]) v.pop_back();
    return v;
}

void test_case() {
    ll n;
    cin >>n;
    vector<PT> p(n);
    for (int i = 0;i<n;i++) cin >> p[i].x >> p[i].y;
    p = convex_hull(p);
    cout << fixed<<setprecision(10) << diameter(p) << "\n";
}
```

9.7 Segment Intersection

```
// No the best algorithm, find a better one!!
// Given two segment, finds the intersection point.

// LINE if they are parallel and mulitple intersection??
// POINT with the intersection point
// NONE if not intersection
struct line {
    ld a, b; // first point
    ld x, y; // second point

    ld m() { return (a - x)/(b - y); }
    bool horizontal() { return b == y; }
    bool vertical() { return a == x; }

    void intersects(line &o) {
        if (horizontal() && o.horizontal()) {
            if (y == o.y) cout << "LINE\n";
            else cout << "NONE\n";
            return;
        }
        if (vertical() && o.vertical()) {
            if (x == o.x) cout << "LINE\n";
            else cout << "NONE\n";
            return;
        }
        if (!horizontal() && !o.horizontal()) {
            ld ma = m();
            ld mb = o.m();
            if (ma == mb) {
                ld someY = (o.x - x)/ma + y;
                if (abs(someY - o.y) <= 0.000001) {
                    cout << "LINE\n";
                } else {
                    cout << "NONE\n";
                }
            } else {
                ld xx = (x*mb - o.x*ma + ma*mb*(o.y - y))/(mb - ma);
                ld yy = (xx - x)/ma + y;
```

```
            cout << "POINT " << fixed << setprecision(2) << xx
                << " " << yy << "\n";
        }
    } else {
        if (!horizontal()) {
            ld xx;
            if (x == a) {
                xx = x;
            } else {
                xx = (o.y - y)/m() + x;
            }
            ld yy = o.y;
            cout << "POINT "<< fixed << setprecision(2) << xx
                << " " << yy << "\n";
        } else {
            ld xx;
            if (x == a) {
                xx = x;
            } else {
                xx = (y - o.y)/o.m() + o.x;
            }
            ld yy = y;
            cout << "POINT "<< fixed << setprecision(2) << xx
                << " " << yy << "\n";
        }
    }

};

void test_case() {
    line l[2];
    for (int i = 0; i < 2; i++) {
        ld x, y, a, b;
        cin >> x >> y >> a >> b;
        l[i].a = x;
        l[i].b = y;
        l[i].x = a;
        l[i].y = b;
    }
    l[0].intersects(l[1]);
}
```

9.8 Some Formulas

Volume of Glass with Water (Volumen del Vaso)

Given:

- p is the height of the water
- r_1 is the big radius at the water's surface
- r_2 is the small radius of the base

The volume of the glass with water is given by:

$$\text{Volume} = p \cdot \pi \cdot \frac{\left(r_1^2 + r_2^2 + r_1 \cdot r_2\right)}{3}$$

Heron's Formula

Finding the area of a triangle by the length of its sides, also applicable for points using Euclidean distance. You can use the following code to get the area; if the square root is negative, then the triangle is not valid.

```
long double triangle_area(long double a, long double b, long
    double c) {
    long double s = (a + b + c) / 2;
    return sqrtl(s * (s - a) * (s - b) * (s - c));
}
```

Sine and Cosine Laws

Let a, b, and c be the sides of the triangle, and A, B, and C the angles opposite to these sides, respectively.

The Sine Law:

$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C}$$

The Cosine Law:

$$c^2 = a^2 + b^2 - 2ab \cdot \cos C$$

10 Utils

10.1 Bit Tricks

```
y = x & (x-1) // Turn off rightmost 1bit
y = x & (-x)  // Isolate rightmost 1bit
y = x | (x-1) // Right propagate rightmost 1bit(fill in 1s)
y = x | (x+1) // Turn on rightmost 0bit
y = ~x & (x+1) // Isolate rightmost 0bit
// If x is of long type, use __builtin_popcountl(x)
// If x is of long long type, use __builtin_popcountll(x)
// 1. Counts the number of ones(set bits) in an integer.
__builtin_popcount(x)
// 2. Checks the Parity of a number. Returns true(1) if the
// number has odd number of set bits, else it returns
// false(0) for even number of set bits.
 __builtin_parity(x)
// 3. Counts the leading number of zeros of the integer.
__builtin_clz(x)
// 4. Counts the trailing number of zeros of the integer.
__builtin_ctz(x)
// 5. Returns 1 + the index of the least significant 1-bit.
__builtin_ffs(x) // If x == 0, returns 0.
// Iterate over non empty subsets of bitmask
for(int s=m;s;s=(s-1)&m) // Decreasing order
for(int s=0;s=s-m&m;) // Increasing order
```

10.2 Decimal Precisition Python

```
"""
For problems that needs more decimal precisition

You can try this code but in C++,
it will not pass in this problem
    https://cses.fi/problemset/task/1728/
"""
from decimal import *
getcontext().prec = 200 #The decimal precisition

n = int(input())
nums = [int(x) for x in input().split(" ")]

ans = Decimal(0)
for i in range(0,len(nums)):
    for j in range(i+1,len(nums)):
        for k in range(1,nums[j]+1):
            ans += max(0,nums[i]-k)/Decimal(nums[i]*nums[j])
            # Also for reduce getcontext().prec = 100
            # You can sum integer part, then apply the decimal
                division

print("{:.6f}".format(ans)) #The rounding half even to six
    decimals
```

10.3 Exponential Notation

```
// O(n) convert numbers to Exponential Notation
// (e.g 0102.150 -> 1.0215E2)
// only float numbers > 0
string exponential_notation(string s) {
    int firstPos = find_if(all(s), [&](char c) {
        return c != '0' && c != '.';
    }) - s.begin();
    int dotPos = find(all(s), '.') - s.begin();
    ll base = dotPos - (firstPos+(firstPos <= dotPos));
    s.erase(dotPos, 1);
    for (int i = 0; i < 2; i++) { //erase traveling zeros
        while (s.back() == '0') s.pop_back();
        reverse(all(s));
    }
    if (s.size() > 1) s.insert(1, ".");
    if (base != 0) s+= "E" + to_string(base);
    return s;
}
```

10.4 IO int128

```
// Read and Print integers of 128 bits
__int128 read() {
    __int128 x = 0, f = 1;
    char ch = getchar();
    while (ch < '0' || ch > '9') {
        if (ch == '-') f = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9') {
        x = x * 10 + ch - '0';
        ch = getchar();
    }
    return x * f;
}
void print(__int128 x) {
    if (x < 0) {
        putchar('-');
        x = -x;
    }
    if (x > 9) print(x / 10);
    putchar(x % 10 + '0');
}

void print(__int128 x) {
    if (x < 0) {
        cout << "-";
        x = -x;
    }
    if (x > 9) print(x / 10);
    cout << char((int)(x % 10) + '0');
}
```

10.5 Pragmas

```
//#pragma GCC target("popcnt")
//It's worth noting that after adding __builtin_popcount() is
    replaced to corresponding machine instruction (look at the
    difference). In my test this maked x2 speed up.
    bitset::count() use __builtin_popcount() call in
    implementation, so it's also affected by this.
#pragma GCC target ("avx2")
#pragma GCC optimization ("O3")
#pragma GCC optimization ("unroll-loops")
#pragma GCC target("popcnt")
#pragma GCC
    target("avx,avx2,sse3,ssse3,sse4.1,sse4.2,tune=native")
#pragma GCC optimize(3)
#pragma GCC optimize("O3")
#pragma GCC optimize("inline")
#pragma GCC optimize("-fgcse")
#pragma GCC optimize("-fgcse-lm")
#pragma GCC optimize("-fipa-sra")
#pragma GCC optimize("-ftree-pre")
#pragma GCC optimize("-ftree-vrp")
#pragma GCC optimize("-fpeephole2")
#pragma GCC optimize("-fsched-spec")
#pragma GCC optimize("-falign-jumps")
#pragma GCC optimize("-falign-loops")
#pragma GCC optimize("-falign-labels")
#pragma GCC optimize("-fdevirtualize")
#pragma GCC optimize("-fcaller-saves")
#pragma GCC optimize("-fcrossjumping")
#pragma GCC optimize("-fthread-jumps")
#pragma GCC optimize("-freorder-blocks")
#pragma GCC optimize("-fschedule-insns")
#pragma GCC optimize("inline-functions")
#pragma GCC optimize("-ftree-tail-merge")
#pragma GCC optimize("-fschedule-insns2")
#pragma GCC optimize("-fstrict-aliasing")
#pragma GCC optimize("-falign-functions")
#pragma GCC optimize("-fcse-follow-jumps")
#pragma GCC optimize("-fsched-interblock")
#pragma GCC optimize("-fpartial-inlining")
#pragma GCC optimize("no-stack-protector")
#pragma GCC optimize("-freorder-functions")
#pragma GCC optimize("-findirect-inlining")
#pragma GCC optimize("-fhoist-adjacent-loads")
#pragma GCC optimize("-frerun-cse-after-loop")
```

```
#pragma GCC optimize("inline-small-functions")
#pragma GCC optimize("-finline-small-functions")
#pragma GCC optimize("-ftree-switch-conversion")
#pragma GCC optimize("-foptimize-sibling-calls")
#pragma GCC optimize("-fexpensive-optimizations")
#pragma GCC optimize("inline-functions-called-once")
#pragma GCC optimize("-fdelete-null-pointer-checks")
```

10.6 Randoms

```
// Get random numbers between [a, b]
mt19937
    mt_rng(chrono::steady_clock::now().time_since_epoch().count());
// also for ll exists mt19937_64
ll randint(ll a, ll b) {
    return uniform_int_distribution<ll>(a, b)(mt_rng);
}
```

10.7 String Streams

```
// For some complex reading of input
// st is the same as a cin, but you pass the string
string line;
getline(cin, line);
stringstream st(line);
vl in;
ll x;
while (st >> x) {
    in.pb(x);
}
```

11 To Order

11.1 1

This section in not sorted yet, probably many of these algorithms will be replaced. This section has also some solved problems

11.2 Bultin functions (out dated)

```
# Sum the values of a iterable
# Very important to put 0ll to avoid overflows
accumulate(v.begin(),v.end(),0ll)/n;
```

11.3 Fermat

```
// ll fermatFactors(ll n) {
//     ll a = ceil(sqrt(n)) ;
//     if(a * a == n){
//         return a;
//     }
//     ll b;
//     while(true) {
//         ll b1 = a * a - n ;
//         b = (ll)sqrt(b1) ;

//         if(b * b == b1)
//             break;
//         else
//             a += 1;
//     }
//     return min(a - b, a + b);
// }
```

11.4 Fraction Modular

```
const ll MOD = 998244353;
```

```
struct frac : public pair<ll, ll> {
    using pair<ll, ll>::pair;

    frac simplify() {
        if (first == 0) {
            return frac(0, 1);
        }
        ll gcd_val = __gcd(first, second);
        return frac(first / gcd_val, second / gcd_val);
    }

    frac operator*(const frac &other) {
        // a * invmod(b) = a / b
        // a * invmod(b) * a2 * invmod(b2) = (a * a2) / (b * b2)
        return frac((first * other.first) % MOD , (second *
            other.second) % MOD ).simplify();
    }

    frac operator+(const frac &other) {
        // opertaor + with module
        // a * invmod(b) + a2 * invmod(b2) = (a * b2 + a2 * b) /
            (b * b2)
        ll up = (first * other.second) % MOD + (other.first *
            second) % MOD;
        ll down = (second * other.second) % MOD;
        // TODO: check if simplify should be here
        return frac(up, down).simplify();
    }
};

// Expected Value is the sum of all the possible values
    multiplied by their probability
// Geometrica is reversed.
```

11.5 General Recursive Segtree

```
// >>>>>>> Implement
// Example of a Segment tree of Xor
struct Node {
    ll a = 0;
};

Node e() {
    Node node;
    return node;
}

Node op(Node a, Node b) {
    Node node;
    node.a = a.a ^ b.a;
    return node;
}
// >>>>>>>> Implement

struct segtree {
    vector<Node> nodes;
    ll n;

    void init(int n) {
        auto a = vector<Node>(n, e());
        init(a);
    }

    void init(vector<Node>& initial) {
        nodes.clear();
        n = initial.size();
        int size = 1;
        while (size < n) {
            size *= 2;
        }
        nodes.resize(size * 2);
        build(0, 0, n-1, initial);
    }

    void build(int i, int sl, int sr, vector<Node>& initial) {
        if (sl == sr) {
            nodes[i] = initial[sl];
        } else {
            ll mid = (sl + sr) >> 1;
            build(i*2+1, sl, mid, initial);
            build(i*2+2, mid+1,sr,initial);
```

```
        nodes[i] = op(nodes[i*2+1], nodes[i*2+2]);
    }
}

void update(int i, int sl, int sr, int pos, Node node) {
    if (sl <= pos && pos <= sr) {
        if (sl == sr) {
            nodes[i] = node;
        } else {
            int mid = (sl + sr) >> 1;
            update(i * 2 + 1, sl, mid, pos, node);
            update(i * 2 + 2, mid + 1, sr, pos, node);
            nodes[i] = op(nodes[i*2+1], nodes[i*2+2]);
        }
    }
}

void update(int pos, Node node) {
    update(0, 0, n - 1, pos, node);
}

Node query(int i, int sl, int sr, int l, int r) {
    if (l <= sl && sr <= r) {
        return nodes[i];
    } else if(sr < l || r < sl) {
        return e();
    } else {
        int mid = (sl + sr) / 2;
        auto a = query(i * 2 + 1, sl, mid, l, r);
        auto b = query(i * 2 + 2, mid + 1, sr, l, r);
        return op(a, b);
    }
}

Node query(int l, int r) {
    return query(0, 0, n - 1, l, r);
}

Node get(int i) {
    return query(i, i);
}
};
```

11.6 Get All Divisors

```
// user getDivisors to get all divisors of a number in aprox
    O(n^(1/3))
// Add fact method of factorization, miller rabin one.

void iterate(ll num, ll idx, vector<pair<ll,ll>> &facts,
    vector<ll> &divs) {
    if (idx == facts.size()) {
        divs.pb(num);
        return;
    }
    iterate(num, idx+1, facts, divs);
    ll f = 1;
    for (int i = 0; i < facts[idx].S; i++) {
        f *= facts[idx].F;
        iterate(num * f, idx + 1, facts, divs);
    }
}

// n^(1/3)
vector<ll> getDivisors(ll n) {
    map<ll,int> f;
    fact(n, f);
    vector<pair<ll,ll>> facts;
    for (auto p : f) facts.pb({p.F, p.S});
    vl divs;
    iterate(1, 0, facts,divs);
    return divs;
}
```

11.7 Macros (outdated)

```
#define MP make_pair
#define MT make_tuple
#define PB push_back
```

```
#define F first
#define S second
#define all(x) (x).begin(), (x).end()
#define sortt(x) sort(all(x))
#define sortn(x, n) sort((x), (x) + (n));
#define SQ(a) ((a) * (a))
#define max3(a, b, c) max((a), max((b), (c)))
#define max4(a, b, c, d) max(max3(a, b, c), d)
#define min3(a, b, c) min((a), min((b), (c)))
#define min4(a, b, c, d) min(min3(a, b, c), d)
#define fastIO() cin.tie(0); ios::sync_with_stdio(0);

// loops
#define FOR(i, a, b) for (ll (i) = (a); (i) < (b); (i)++)
#define ROF(i, a, b) for (ll (i) = (a); (i) >= (b); (i)--)
#define REP(i, a, b) for (ll (i) = (a); (i) <= (b); (i)++)
#define EACH(a, x) for (auto &(a) : (x))

typedef long long ll;
typedef pair<int, int> pii;
typedef tuple<long long, long long, long long> tiii;
typedef pair<long long, long long> pll;
typedef unsigned long long ull;
typedef long double ld;
typedef vector<int> vi;
typedef vector<bool> vb;
typedef vector<ll> vl;
typedef vector<pll> vpll;
typedef vector<vl> vvl;
typedef vector<vi> vvi;
typedef vector<string> vs;
typedef vector<ld> vld;

template<class T> using pql =
    priority_queue<T,vector<T>,greater<T>>;
template<class T> using pqg = priority_queue<T>;

const ld DINF=1e100;
const ld EPS = 1e-9;
const ld PI = acos(-1);
const ll infl = INT64_MAX;
const int inf = INT32_MAX;
const int dx[4]{1,0,-1,0}, dy[4]{0,1,0,-1};
const int MOD = 1e9 + 7;
```

11.8 Merge Sort Tree

```
// usage
// vector<node*> nodes;
// tree.query(l, r, nodes);

// returns log(n) sorted segments in a range (l, r)

struct node {
    ll l, r;
    vl nums;
    vl prefix;
};

struct segtree {
    int n;
    vector<node> tree;
    void init(int nn, vl& nodes) {
        tree.clear();
        n = nn;
        int size = 1;
        while (size < n) {
            size *= 2;
        }
        tree.resize(size * 2);
        build(0, 0, n - 1, nodes);
    }


    void query(ll i, ll sl, ll sr, ll l, ll r, vector<node*> &ans)
        {
        if (l <= sl && sr <= r) {
            ans.pb(&tree[i]);
        } else if (sr < l || r < sl) {

        } else {
```

```
        int mid = (sl + sr) >> 1;
        query(i * 2 + 1, sl, mid, l, r, ans);
        query(i * 2 + 2, mid + 1, sr, l, r, ans);
    }
}

void query(ll l, ll r, vector<node*> &ans) {
    return query(0, 0, n - 1, l, r, ans);
}

void build(int nodei, int l, int r, vl &nums) {
    if (l == r) {
        tree[nodei].nums = { nums[l] };
        tree[nodei].prefix = {nums[l]};
        tree[nodei].l = l;
        tree[nodei].r = r;
    } else {
        ll mid = (l + r) >> 1;
        build(nodei * 2 + 1, l, mid, nums);
        build(nodei * 2 + 2, mid + 1, r, nums);
        ll a = tree[nodei*2+1].nums.size();
        ll b = tree[nodei*2+2].nums.size();
        tree[nodei].nums.reserve(a + b);
        tree[nodei].prefix.resize(a+b);
        ll i = 0;
        ll j = 0;
        while (i < a && j < b) {
            ll simon = tree[nodei*2+1].nums[i];
            ll simon2 = tree[nodei*2+2].nums[j];
            if (simon <= simon2) {
                tree[nodei].nums.pb(simon);
                i++;
            } else {
                tree[nodei].nums.pb(simon2);
                j++;
            }
        }
        while (i < a) {
            tree[nodei].nums.pb(tree[nodei*2+1].nums[i]);
            i++;
        }
        while (j < b) {
            tree[nodei].nums.pb(tree[nodei*2+2].nums[j]);
            j++;
        }
        tree[nodei].prefix[0] = tree[nodei].nums[0];
        for (int i = 1; i < a + b; i++) {
            tree[nodei].prefix[i] = tree[nodei].prefix[i - 1] +
                tree[nodei].nums[i];
        }
        tree[nodei].l = l;
        tree[nodei].r = r;
    }
};
```

11.9 Min Segment Tree

```
// Max segment tree
struct segtree {
    int n;
    vl tree;

    void init(int nn) {
        tree.clear();
        n = nn;
        int size = 1;
        while (size < n) {
            size *= 2;
        }
        tree.resize(size * 2);
    }

    void update(int i, int sl, int sr, int pos, ll diff) {
        if (sl <= pos && pos <= sr) {
            if (sl == sr) {
                tree[i] += diff;
            } else {
                int mid = (sl + sr) / 2;
                update(i * 2 + 1, sl, mid, pos, diff);
                update(i * 2 + 2, mid + 1, sr, pos, diff);
```

```
            tree[i] = max(tree[i * 2 + 1], tree[i * 2 + 2]);
        }
    }
}

void update(int pos, ll diff) {
    update(0, 0, n - 1, pos, diff);
}

ll query(int i, int sl, int sr, int l, int r) {
    if (l <= sl && sr <= r) {
        return tree[i];
    } else if(sr < l || r < sl) {
        return INT64_MIN;
    } else {
        int mid = (sl + sr) / 2;
        auto a = query(i * 2 + 1, sl, mid, l, r);
        auto b = query(i * 2 + 2, mid + 1, sr, l, r);
        return max(a, b);
    }
}

ll query(int l, int r) {
    return query(0, 0, n - 1, l, r);
}
};
```

11.10 Mo's in Tree's

```
#include <bits/stdc++.h>
using namespace std;
typedef vector<int> vi;
typedef vector<vi> vvi;

map<int, int> getID;
map<int, int>::iterator it;

const int LOGN = 20;
int id, bs, N;
int counter[50050];
int A[50050], P[100050];
int res[100050];
int st[50050], ed[50050];
int DP[20][50050], level[50050];
bool flag[50050];
bool seen[50050];
vvi edges;

struct Q {
  int l, r, p, id;
  bool operator < (const Q& other) const {
    return (l / bs < other.l / bs || (l / bs == other.l / bs &&
        r < other.r));
  }//operator <
} q[100050];

void DFS0(const int u) {
  seen[u] = 1;
  P[id] = u;
  st[u] = id++;
  for (auto& e : edges[u]) {
    if (!seen[e]) {
      DP[0][e] = u;
      level[e] = level[u] + 1;
      DFS0(e);
    }//if
  }//for
  P[id] = u;
  ed[u] = id++;
}//DFS0

void prep(const int r) {
  level[r] = 0;
  for (int i = 0; i < LOGN; i++)
    DP[i][r] = r;
  id = 0;
  DFS0(r);
  for (int i = 1; i < LOGN; i++)
    for (int j = 1; j <= N; j++)
      DP[i][j] = DP[i - 1][DP[i - 1][j]];
}//prep
```

```
int LCA(int a, int b) {
   if (level[a] > level[b])
      swap(a, b);
   int diff = level[b] - level[a];
   for (int i = 0; i < LOGN; i++)
      if (diff & (1 << i))
         b = DP[i][b]; //move 2^i parents upwards
   if (a == b)
      return a;
   for (int i = LOGN - 1; i >= 0; i--)
      if (DP[i][a] != DP[i][b])
         a = DP[i][a], b = DP[i][b];
   return DP[0][a];
}//LCA

int main() {
   int Q, n1, n2, L, R, a, v = 1, tot;
   scanf("%d %d", &N, &Q);
   edges.assign(N + 5, vi());
   bs = sqrt(N);
   for (int i = 1; i <= N; i++) {
      scanf("%d", &a);
      A[i] = ((it = getID.find(a)) != getID.end()) ? it->second :
          (getID[a] = v++);
   }//for
   for (int i = 0; i < N - 1; i++) {
      scanf("%d %d", &n1, &n2);
      edges[n1].push_back(n2);
      edges[n2].push_back(n1);
   }//for
   prep(1);
   for (int i = 0; i < Q; i++) {
      scanf("%d %d", &n1, &n2);
      if (st[n1] > st[n2])
         swap(n1, n2);
      q[i].p = LCA(n1, n2);
      if (q[i].p == n1)
         q[i].l = st[n1], q[i].r = st[n2];
      else
         q[i].l = ed[n1], q[i].r = st[n2];
      q[i].id = i;
   }//for
   sort(q, q + Q);
   L = 0; R = -1; tot = 0;
   for (int i = 0; i < Q; i++) {
      while (R < q[i].r) {
         if (!flag[P[++R]])
            tot += (++counter[A[P[R]]] == 1);
         else
            tot -= (--counter[A[P[R]]] == 0);
         flag[P[R]] = !flag[P[R]];
      }//while
      while (R > q[i].r) {
         if (!flag[P[R]])
            tot += (++counter[A[P[R]]] == 1);
         else
            tot -= (--counter[A[P[R]]] == 0);
         flag[P[R]] = !flag[P[R]];
         R--;
      }//while
      while (L < q[i].l) {
         if (!flag[P[L]])
            tot += (++counter[A[P[L]]] == 1);
         else
            tot -= (--counter[A[P[L]]] == 0);
         flag[P[L]] = !flag[P[L]];
         L++;
      }//while
      while (L > q[i].l) {
         if (!flag[P[--L]])
            tot += (++counter[A[P[L]]] == 1);
         else
            tot -= (--counter[A[P[L]]] == 0);
         flag[P[L]] = !flag[P[L]];
      }//while
      res[q[i].id] = tot + (q[i].p != P[q[i].l] &&
          !counter[A[q[i].p]]);
   }//for
   for (int i = 0; i < Q; i++)
      printf("%d\n", res[i]);
   return 0;
```

```
}//main
```

11.11 Nearest Selected Nodes Problem

```
// Given an order of selected nodes in a tree, you should print
    the miminum distance between two selected nodes after each
    operation.

// O(nlogn or n*sqrt(n)); n <= 2*10^5, 2.7 seconds.
// adj is the adjacency list, order is the selected nodes in order
// n is the numeber of nodes, returns the minimum after each
    operation
// note that operation 0 answer is 1e9
vl f(vvl &adj, vl &order, ll n) {
    vl answer;
    vl dist(n + 1, 1e9);
    ll best = 1e9;
    vl q(n + 1);
    ll sz = 0;
    for (int i = 0; i < n; i++) {
        best = min(best, dist[order[i]]);
        sz = 0;
        dist[order[i]] = 0;
        q[sz++] = order[i];
        ll idx = 0;
        while (idx < sz) {
            ll x = q[idx++];
            if (dist[x] + 1 >= best) break;
            for (auto &y : adj[x]) {
                if (dist[x] + 1 < dist[y]) {
                    dist[y] = dist[x] + 1;
                    q[sz++] = y;
                }
            }
        }
        answer.pb(best);
    }
    return answer;
}
```

11.12 Or Statements like 2 Sat Problem

```
// Return the smaller lexicographic array of size n that satities
    a_i | a_j = z
// a_i | a_i = z is allowed.
// there must exists a solution.
vector<ll> f(ll n, vector<tuple<ll,ll,ll>> &statements) {
    ll m = statements.size();
    vector<vector<pair<ll,ll>>> adj(n + 1);
    const ll bits = 30;
    vector<ll> taken(n+1, (1 << bits) - 1), answer(n+1, (1 <<
        bits) - 1);
    for (int i = 0; i < m; i++) {
        ll x, y, z;
        tie(x, y, z) = statements[i];

        answer[x] &= z;
        answer[y] &= z;
        if (x == y) {
            taken[x] = 0;
            continue;
        }
        taken[x] &= z;
        taken[y] &= z;
        adj[x].pb({y, z});
        adj[y].pb({x, z});
    }
    for (int x = 1; x <= n; x++) {
        for (int i = 0; i < bits; i++) {
            if (!((taken[x] >> i) & 1)) continue;
            ll allHave = true;
            for (auto y : adj[x]) {
                if ((y.S >> i) & 1) {
                    allHave &= ((taken[y.F] >> i) & 1) ||
                        ((answer[y.F] >> i) & 1);
                }
            }
            taken[x] -= 1 << i;
            if (allHave) {
                answer[x] -= 1 << i;
                for (auto y : adj[x]) {
```

```
                if ((y.S >> i) & 1) {
                    taken[y.F] |= 1 << i;
                    taken[y.F] ^= 1 << i;
                }
            }
        }
    }
    answer.erase(answer.begin());
    return answer;
}
```

11.13 Pi

```
const ld PI = acos(-1);
```

11.14 Poly Definitions

```
A(x) = Sum i=0 to n ( a_i * x^i ) y B(x) Sum i=0 to m ( b_i * x^i)

A(x)*B(x) Sum i=0 to (n+m) Sum j=0 to (n+m) (a_j)*(b_i-j))x^i
```

11.15 Polynomial Sum Lazy SegTree Problem

```
/* Polynomial Queries, queries
1. Increase [a,b] by 1, second by 2, third by 3, and so on
2. Sum of [a,b]
Use:

cin >> nums[i],tree.update(i, { nums[i] })
For 1: tree.apply(l,r,{0,1});
For 2: tree.query(l,r).sum
*/
struct Node { ll sum = 0; };
struct Func { ll add, ops; };
Node e() { return {0}; };
Func id() { return {0, 0}; }
Node op(Node a, Node b) { return {a.sum + b.sum }; }
ll f(ll x) { return x * (x+1)/2; }
Node mapping(Node node, Func lazy, ll sz) {
    return { node.sum + sz*lazy.add + lazy.ops*f(sz) };
}
Func composicion(Func prev, Func actual) {
    Func ans = { prev.add + actual.add, prev.ops + actual.ops };
    return ans;
}
Func sumF(Func f, ll x) { return {f.add + x*f.ops, f.ops }; }

struct lazytree {
    int n;
    vector<Node> nodes;
    vector<Func> lazy;
    void init(int nn) {
        n = nn;
        int size = 1;
        while (size < n) { size *= 2; }
        ll m = size * 2;
        nodes.assign(m, e());
        lazy.assign(m, id());
    }

    void push(int i, int sl, int sr) {
        nodes[i] = mapping(nodes[i], lazy[i], sr-sl+1);
        if (sl != sr) {
            ll cnt = (sr+sl)/2-sl+1; // changed
            lazy[i * 2 + 1] = composicion(lazy[i*2+1],lazy[i]);
            lazy[i * 2 + 2] =
                composicion(lazy[i*2+2],sumF(lazy[i],cnt));
        }
        lazy[i] = id();
    }

    void apply(int i, int sl, int sr, int l, int r, Func f) {
        push(i, sl, sr);
        if (l <= sl && sr <= r) {
            lazy[i] = sumF(f, abs(sl-l)); //Changed
            push(i,sl,sr);
```

```
    } else if (sr < l || r < sl) {
    } else {
        int mid = (sl + sr) >> 1;
        apply(i * 2 + 1, sl, mid, l, r, f);
        apply(i * 2 + 2, mid + 1, sr, l, r, f);
        nodes[i] = op(nodes[i*2+1],nodes[i*2+2]);
    }
}

void apply(int l, int r, Func f) {
    assert(l <= r);
    assert(r < n);
    apply(0, 0, n - 1, l, r, f);
}

void update(int i, Node node) {
    assert(i < n);
    update(0, 0, n-1, i, node);
}

void update(int i, int sl, int sr, int pos, Node node) {
    if (sl <= pos && pos <= sr) {
        push(i,sl,sr);
        if (sl == sr) {
            nodes[i] = node;
        } else {
            int mid = (sl + sr) >> 1;
            update(i * 2 + 1, sl, mid, pos, node);
            update(i * 2 + 2, mid + 1, sr, pos, node);
            nodes[i] = op(nodes[i*2+1], nodes[i*2+2]);
        }
    }
}

Node query(int i, int sl, int sr, int l, int r) {
    push(i,sl,sr);
    if (l <= sl && sr <= r) {
        return nodes[i];
    } else if (sr < l || r < sl) {
        return e();
    } else {
        int mid = (sl + sr) >> 1;
        auto a = query(i * 2 + 1, sl, mid, l, r);
        auto b = query(i * 2 + 2, mid + 1, sr, l, r);
        return op(a,b);
    }
}

Node query(int l, int r) {
    assert(l <= r);
    assert(r < n);
    return query(0, 0, n - 1, l, r);
}
};
```

11.16 Sum of xor Subarrays times its size Problem

```
// Given an array A
// calculate:
// ( l =0 to (n-1)  r =l to (n-1) f(l,r) ( rl +1)) % mod
// f(l,r) = a[l] ^ a[l+1] ^ ... ^ a[r]
// In other words, it calculate the sum
// of xor-subarrays multiplied by its size
// A or nums is 0-indexed
// sum(nums,n,998244353)
const int mod = 998244353;
ll sum(vl &nums, ll n) {
    vector<ll> pref(n+1);
    for (int i = 1; i <= n; i++) {
        pref[i] = pref[i-1] ^ nums[i-1];
    }
    ll ans =0;
    for (ll bit = 0; bit <= 60; bit++) {
        vl cnt(2);
        vl dist(2);
        ll sum = 0;
        for (int i = 0; i <= n; i++) {
            ll actual = (pref[i] >> bit) & 1;
            sum = (sum + dist[!actual]) % mod;
            cnt[actual]++;
            dist[actual] = (dist[actual] + cnt[actual]) % mod;
```

```
            dist[!actual] = (dist[!actual] + cnt[!actual]) % mod;
        }
        ans += (((1ll << bit) % mod) * sum) % mod;
        ans %= mod;
    }
    return ans;
}
```

11.17 Two Pieces on Tree (Problem)

```
// In a tree with 'n' nodes where 2 pieces starting from root 1
// must go to certain nodes each one and must not exceed 'd'
    between
// the two pieces, after they have to return to root 1
// two_pieces_on_tree() find the minimum quantity of moves
// My submittion in CF:
    https://codeforces.com/contest/1774/submission/189071201

ll n, d; // quantity of nodes, maximum distance between pieces
vvl children; // tree
vector<int> a, b; // nodes that must visit first and second
    piecesS

void dfs(ll x, vl &route) {
    route.pb(x);
    ll kParent = 1; //route
    if (route.size() - 1 >= d) {
        kParent = route[route.size() - 1 - d];
    }
    b[kParent] |= a[x];
    a[kParent] |= b[x];
    each(y, children[x]) {
        dfs(y, route);
        a[x] |= a[y];
        b[x] |= b[y];
    }
    route.pop_back();
}

ll two_pieces_on_tree() {
    ll root = 1;
    vl emptyRoute = vl();
    dfs(root, emptyRoute);
    ll total = 0;
    for (int i = 1; i <= n; i++) {
        total += a[i] + b[i];
    }
    return total * 2 - 4;
}
```