# Team Notebook

## Universidad Mayor de San Simón - Wasos

# 1 1. Math

## 1.1 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
*/
```

## 1.2 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);
}
```

## 1.3 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);
    }
}
```

## 1.4 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));
}
```

## 1.5 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;
}
```

## 1.6 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;
```

## 1.7 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);}
}
```

### 1.8 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;
    }
}
```

### 1.9 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 1. Math - Fft

### 2.1 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;
}
```

## 2.2 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];
    }
```

## 2.3 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;
}
```

### 2.4 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 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;
}
```

# 3 1. Math - Mobius

## 3.1 sigma^{n} {i=1} 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;
            }
        }
    }
}
```

### 3.2 [gcd(a i, a j) == k] text{ 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";
    }
}
```

### 3.3 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]++;
    }
}
```

### 3.4 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]];
    }
}
```

### 3.5 [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;
}
```

# 4 1. Math - Modular Arithmethics

## 4.1 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;
}
```

## 4.2 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;
    }
};
```

### 4.3 diophantine ecuations

```
Encuentra x, y en la ecuación 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); // sólo hay solución 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;
}
```

### 4.4 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};
}
```

### 4.5 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);
}
```

### 4.6 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";
}
```

# 5 2. Dp

## 5.1 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";
}
```

## 5.2 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";
}
```

## 5.3 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;
  }
};
```

### 5.4 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!!
 //note about dp[i][j] is that I set inf instead of cost(i,j)
and it got accepted, not sure why
            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";
}
// https://cses.fi/problemset/task/2088
// https://www.spoj.com/problems/BRKSTRNG/
```

### 5.5 sos dp

```
// F(mask) = op(F(x_1), F(x_2), ...) where x is submask of mask
// init sos[z]
for (int i = 0; i < M; i++) {
  for (int mask = 0; mask < (1 << M); mask++) {
    if (mask & (1 << i)) {
      sos[mask] += sos[mask ^ (1 << i)];
    }   }
}
```

### 5.6 divide and conquer optimization

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

### 5.7 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];
}
```

### 5.8 optimizing pragmas for bitset

```
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")
// popcnt is The difference to TLE and AC in https://cses.fi/
problemset/task/2137/
```

### 5.9 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)
```

### 5.10 sos dp reverse

```
// Useful to calculate reverse sos DP.
// DP[x] = op(DP[y_1], DP[y_2], ...), where x is submask of
y_i
    vector<ll> nums(n);
    for (int i =0;i<n;i++) {
        nums[i] = rd(); // read submask, and set init state
        dp[nums[i]] = min(dp[nums[i]], {0, i});
    }
    for (int mask = (1<<m)-1; mask >= 0; mask--) { // You can
switch fors depends of the problem
        if (dp[mask].first > 0) continue; // Don't propagate
        for (int i =0;i<m;i++) {
            if ((mask>>i)&1) { // Propagate to all submasks
                auto nxt = dp[mask]; nxt.first--;
                dp[mask^(1ll<<i)] = min(dp[mask^(1ll<<i)],
nxt);
```

```
            }
        }
    }
```

### 5.11 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];
}
```

### 5.12 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.
```

# 6 3. Data Structures

## 6.1 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
```

## 6.2 block tree

```
/*
SQRT on Trees.
For having queries and updates online to subtrees or sub
forests.
See: https://codeforces.com/blog/entry/46843
Solution to: https://codeforces.com/gym/100589/problem/A
Properties:
  - Each node will have atmost SQRT children nodes in Block
Tree
  - There will be atmost SQRT Blocks
  - There will be atmost SQRT nodes belongs to a block
  - Not completely sure, but for merging answers of childs it
should be fast
*/
struct Block { // Block storing info depending of problem!!!
    // O(n) memory here is possible, but better O(sqrt n) to
BlockTree creation be O(n)
    vector<int> depthCnt;
    ll sum = 0;
    void update(int lvl, ll v) { // !!! depends of problem
        assert(lvl < depthCnt.size());
        if (lvl >= depthCnt.size()) return;
        sum += v * depthCnt[lvl];
    }
    ll query() { return sum; }
};
struct BlockTree {
    const int SQRT = 320; // sqrt of max N
    ll n;
    vector<Block> blocks;
    vector<vector<int>> child; // if node < n,then normal node,
otherwise block node
    vector<int> depth;
    vector<int> leader; // for the sqrt group of nodes the
first ancestor that is not in the group
    vector<int> blockId;
    // ===== depends of the problem!!!
    vector<ll> sum;
    // =====
    BlockTree(ll n, vector<vector<ll>> &adj) : n(n),
child(2*n+5), depth(2*n+5),

leader(2*n+5), blockId(2*n+5) {
        sum = vector<ll>(n);
        build(adj); // adjacency should be x -> y, and y -> x
    }
    bool is_block(ll x) { return x >= n; }
    void build(vector<vector<ll>> &adj, ll root = 0) {
        auto dfs = [&](ll x, ll p, auto &&dfs) -> void {
            for (auto y : adj[x]) if (y != p) {
                depth[y] = depth[x]+1;
                dfs(y,x,dfs);
            }
        };
        dfs(root, -1, dfs); // !!! Don't forget
        ll idNewBlock = n;
        auto createBlock = [&](int parent, const vector<ll>
&nodes) {
            ll id = idNewBlock++;
            if (parent != -1) {
                child[parent].push_back(id);
                depth[id] = depth[parent]+1;
            }
            Block block; // Modify the creation of your block
here!!!
            leader[id] = parent;
            blockId[id] = id;
            block.depthCnt = vector<int>(n);
            auto dfsOverBlock = [&](ll x, auto &&dfsOverBlock)-
```

```
>void {
                leader[x] = parent;
                blockId[x] = id;
                block.depthCnt[depth[x]]++;
                for (auto y : child[x]) {
                    if (!is_block(y)) dfsOverBlock(y,
dfsOverBlock);
                    else child[id].push_back(y); // depending
of the problem, you can add
                    // relationship between block node -> node
or block node -> block node, like this
                }
            };
            for (auto & x : nodes) dfsOverBlock(x,
dfsOverBlock);
            blocks.push_back(block);
        };
        vector<vector<ll>> byDepth(n); // counting sort
        for (int i =0;i<n;i++) byDepth[n-1-depth[i]].pb(i);
        vector<ll> sz(n);
        for (auto & nodes : byDepth) {
            for (int x : nodes) {
                ll blockSz = 0;
                vector<ll> curBlock;
                sz[x] = 1;
                for (int y : adj[x]) if (depth[x] < depth[y]) {
                    sz[x] += sz[y];
                    blockSz += sz[y];
                    curBlock.pb(y);
                    if (blockSz > SQRT) {
                        createBlock(x, curBlock);
                        sz[x] -= blockSz;
                        blockSz = 0;
                        curBlock.clear();
                    }
                }
                child[x].insert(child[x].end(),
curBlock.begin(), curBlock.end());
            }
        }
        createBlock(-1, { root });
    }
    void update(int lvl, ll v) {
        sum[lvl] += v;
        for (auto & block : blocks) {
            block.update(lvl, v);
        }
    }
    ll query(ll x) { // dfs over node blocks and non node
blocks
        ll ans = is_block(x) ? blocks[x-n].query() :
sum[depth[x]];
        for (auto & y : child[x])
            ans += query(y);
        return ans;
    }
};
```

### 6.3 xor basis

```
#define i23 long long
vector<i23> basis;
void add(i23 x) {
  for (int i = 0; i < (int)basis.size(); i++) {
    // reduce x using the current basis vectors
    x = min(x, x ^ basis[i]);
  }
  if (x != 0) { basis.push_back(x); }
}
// Other implement
// To count number of xor subsets
void test_case() {
    ll n;cin >> n;
    vector<ll> basis(32,0);
    ll cnt = 0;
    auto add = [&](ll x) {
        for (ll i = 31;i>=0;i--) {
            if ((x>>i)&1) {
                if (basis[i]==0) {
                    basis[i]=x;
                    cnt++;
                    return;
                }
```

```
                x ^= basis[i];
            }
        }
    };
    for (int i =0;i<n;i++) {
        ll x;
        cin >> x;
        add(x);
    }
    ll ans = 1ll << (cnt);
    cout << ans << "\n";
}
```

### 6.4 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);
    }
};
```

### 6.5 multiorderedset

```cpp
#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();
    }
};
```

### 6.6 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);
}
```

### 6.7 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));
```

```
    }
};
```

### 6.8 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];
            }
        }
    }
};
```

### 6.9 custom hash pair

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

### 6.10 fast hash map

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
const int RANDOM =
chrono::high_resolution_clock::now().time_since_epoch().count();
struct chash {
    long long operator()(const ll &x) const {
        return x ^ RANDOM;
    };
};
// There is no mp.count(x), so use mp.find(x) != mp.end()
gp_hash_table<ll,ll,chash> mp;
```

### 6.11 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/AzKAZ2
RDiVTjeWSBolwoC5zl/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
  F0R(i,sz(v)) v.mutable_reference_at(i) = i+1;
  F0R(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; F0R(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).
*/
```

### 6.12 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
};
```

### 6.13 dynamic connectivity

```
struct union_find {
    struct uf_save { ll a, rnka, b, rnkb; };
    vector<uf_save> saves;
    vector<ll> p, rnk;
    ll n, comps;
    union_find(ll _n) : n(_n), p(_n), rnk(_n) {
        for (int i = 0;i<n;i++) p[i] = i;
        comps = n;
    };
    ll find(ll x) { return p[x] == x ? x : find(p[x]); }
    int merge(ll a, ll b) {
        a = find(a);
        b = find(b);
        if (a == b) return 0;
        if (rnk[b] > rnk[a]) swap(a,b);
        comps--;
        saves.pb({a, rnk[a], b, rnk[b]});
        p[b] = a;
        if (rnk[a] == rnk[b]) rnk[a]++;
        return 1;
    };
    void rollback() {
```

```
        if (saves.empty()) return;
        uf_save lst = saves.back();
        saves.pop_back();
        comps++;
        p[lst.a] = lst.a, rnk[lst.a] = lst.rnka;
        p[lst.b] = lst.b, rnk[lst.b] = lst.rnkb;
    }
};
struct query { ll a, b; };
struct segtree {
    int n; // size of queries!!!
    vector<vector<query>> t; // node si a vector<query>
    segtree(int queries_n) : n(queries_n), t(2*n) {}
    void addQuery(int l, int r, const query &q) {
        l += n, r += n+1;
        for (; l < r; l >>= 1, r >>= 1) {
            if (l&1) t[l++].pb(q);
            if (r&1) t[--r].pb(q);
        }
    }
    void dfs(int id, vector<ll> &ans, union_find &uf) {
        ll cnt = 0;
        for (auto & q : t[id]) cnt += uf.merge(q.a, q.b);
        if (id < n) dfs(id*2, ans, uf), dfs(id*2+1,ans,uf);
        else ans[id-n] = uf.comps;
        for (int i =0;i<cnt;i++) uf.rollback();
    }
    vector<ll> solve(union_find &uf) {
        vector<ll> ans(n);
        dfs(1, ans, uf);
        return ans;
    }
};
void test_case() {
    ll n, m, q; cin >> n >> m >> q;
    segtree tree(q+1);
    vector<map<int,int>> lst(n); // to manage living time!
    auto addEdge = [&](int x, int y, int time) {
        if (y<x) swap(x,y);
        lst[x][y] = time;
    };
    auto erEdge = [&](int x, int y, int time) {
        if (y<x) swap(x,y);
        tree.addQuery(lst[x][y], time-1, {x,y});
        lst[x].erase(y);
    };
    for (int i = 0;i<m;i++) {
        ll x, y; cin >> x >> y; x--,y--;
        addEdge(x,y,0);
    }
    for (int time =1;time<=q;time++) {
        ll t, x, y; cin >> t >> x >> y;
        x--,y--;
        if (t==1) addEdge(x,y,time);
        else erEdge(x,y,time);
    }
    for (int i = 0;i < n;i++)
        for (auto &[j,v] : lst[i]) // be carefull with it and
erasing
            tree.addQuery(v,q,{i,j});
    union_find uf(n);
    auto ans = tree.solve(uf);
    for (int i = 0;i < ans.size(); i++) {
        cout << ans[i] << " \n" [i+1 == ans.size()];
    }
} // https://cses.fi/problemset/task/2133/
```

### 6.14 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]);
    }
};
```

### 6.15 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)
```

### 6.16 general iterative segment tree

```
const int inf = 1e9;
// >>>>>>>>> Implement
struct Node { ll mn = inf, mx = -inf; };
Node e() { return Node(); } // op(a, e()) = a
Node op(const Node &a, const Node &b) { // asociative property
    Node c;
    c.mn = min(a.mn, b.mn);
    c.mx = max(a.mx, b.mx);
    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*2],
t[i*2+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*2],
t[p*2+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 t[i+n];
    }
    // Maximum r that satisfy g(op(t[l], t[l+1], ..., t[r-1]))
= True
    // if there's no g() that returns true, then r = l
    template <class F> int max_right(int l, F f) const {
        assert(0 <= l && l <= n);
        assert(f(e()));
        if (l == n) return n;
        ll size = n;
        l += size;
```

```
        Node sm = e();
        do {
            while (l % 2 == 0) l >>= 1;
            if (!f(op(sm, t[l]))) {
                while (l < size) {
                    l = (2 * l);
                    if (f(op(sm, t[l]))) {
                        sm = op(sm, t[l]);
                        l++;
                    }
                }
                return l - size;
            }
            sm = op(sm, t[l]);
            l++;
        } while ((l & -l) != l);
        return n;
    }
    // If g is monotone, this is the minimum l that satisfies
    // g(op(a[l], a[l + 1], ..., a[r - 1])) = true.
    template <class F> int min_left(int r, F f) const {
        assert(0 <= r && r <= n);
        assert(f(e()));
        if (r == 0) return 0;
        ll size = n;
        r += size;
        Node sm = e();
        do {
            r--;
            while (r > 1 && (r % 2)) r >>= 1;
            if (!f(op(t[r], sm))) {
                while (r < size) {
                    r = (2 * r + 1);
                    if (f(op(t[r], sm))) {
                        sm = op(t[r], sm);
                        r--;
                    }
                }
                return r + 1 - size;
            }
            sm = op(t[r], sm);
        } while ((r & -r) != r);
        return 0;
    }
};
```

## 6.17 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;
}
```

### 6.18 sqrt mod

```
/*
Trick using SQRT for problems using de diferent floor values o
a num.
A num has only around SQRT(n) possible values when it is floor
divided, for 2e5 is around 890
e.g. floor(n/i)=j. j has around sqrt(n) values.
e.g. floor(10/i)=j. j only can be 1, 2, 3, 5, 10 for any i
The following function calculates when a n/i changes if you
iterate over i increasing.
You can solve https://codeforces.com/gym/105009/problem/L using
this.
*/
const int MXN = 2e5 + 10;
int nums[MXN];
vector<int> ch[MXN];
for (int i =0;i<n;i++) {
  for (int d = 1; d <= nums[i]; d = nums[i]/(nums[i]/d)+1) {
    ch[d].pb(i);
  }
  ch[nums[i]+1].pb(i);
}
```

### 6.19 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; }
```

### 6.20 iterative lazytree

```
/*
Lazy Iterative Seg Tree.
This example is for https://cses.fi/problemset/task/1735/
*/
struct Node { ll x; };
const ll SET = 1;
```

```
const ll ADD = 2;
struct Func { ll type, val; }; // applied Function
Node e() { return { 0 }; }; // op(x, e()) = x
Func id() { return {ADD, 0}; } // mapping(x, id()) = x
Node op(Node &a, Node &b) { // associative property
    return {a.x + b.x};
}
Node mapping(Node node, ll sz, const Func &lazy) {
    if (lazy.type == SET) node.x = lazy.val * sz;
    else node.x += lazy.val * sz;
    return node;
}
Func comp(Func prev, const Func &actual) {
    if (actual.type==SET) return actual;
    if (prev.type == SET) return {SET, prev.val + actual.val};
    return {ADD, prev.val + actual.val };
}
struct lazyseg {
    ll n, h;
    vector<Node> t;
    vector<Func> lz;
    vector<int> sz; // if sz of node needed
    lazyseg(ll n) : n(n), t(2*n,e()), lz(n, id()), sz(2*n,1) {
        h = 32-__builtin_clz(n);
        for (int i = n-1;i>0;i--) sz[i] = sz[i*2]+sz[i*2+1];
    }
    void all_apply(ll x, const Func & f) {
        t[x] = mapping(t[x], sz[x], f);
        if (x<n) lz[x] = comp(lz[x], f);
    }
    void push(ll x) {
        all_apply(x*2, lz[x]);
        all_apply(x*2+1, lz[x]);
        lz[x] =id();
    }
    void build(ll x) {
        t[x] = op(t[x*2], t[x*2+1]);
    }
    bool ok(ll x, ll i) {
        return ((x>>i)<<i)!=x;
    }
    void update(ll p, Node v) {
        p += n;
        for (int i =h;i>0;i--) push(p>>i);
        t[p] = v;
        for (int i =1;i<=h;i++) build(p>>i);
    }
    Node query(ll l, ll r) {
        if (l>r) return e();
        l += n, r += n+1;
        for (int i=h;i>0;i--) {
            if (ok(l,i)) push(l>>i);
            if (ok(r,i)) push((r-1)>>i);
        }
        Node ansl = e(), ansr = e();
        for (;l<r;l>>=1,r>>=1) {
            if (l&1) ansl = op(ansl, t[l++]);
            if (r&1) ansr = op(t[--r], ansr);
        }
        return op(ansl, ansr);
    }
    void apply(ll l, ll r, const Func &f) {
        if (l>r) return;
        l += n, r += n+1;
        for (int i =h;i>0;i--) {
            if (ok(l,i)) push(l>>i);
            if (ok(r,i)) push((r-1)>>i);
        }
        {
            int l2 = l, r2 = r;
            for (;l<r;l>>=1,r>>=1) {
                if (l&1) all_apply(l++, f);
                if (r&1) all_apply(--r,f);
            }
            l = l2, r= r2;
        }
        for (int i =1;i<=h;i++) {
            if (ok(l,i)) build(l>>i);
            if (ok(r,i)) build((r-1)>>i);
        }
    };
    // Maximum r that satisfy g(op(t[l], t[l+1], ..., t[r-1]))
```

```
= True
    // if there's no g() that returns true, then r = l
    template <class G> int max_right(int l, G g) {
        assert(0 <= l && l <= n);
        assert(g(e()));
        if (l == n) return n;
        l += n;
        for (int i = h; i >= 1; i--) push(l >> i);
        Node sm = e();
        do {
            while (l % 2 == 0) l >>= 1;
            if (!g(op(sm, t[l]))) {
                while (l < n) {
                    push(l);
                    l = (2 * l);
                    if (g(op(sm, t[l]))) {
                        sm = op(sm, t[l]);
                        l++;
                    }
                }
                return l - n;
            }
            sm = op(sm, t[l]);
            l++;
        } while ((l & -l) != l);
        return n;
    }
    // If g is monotone, this is the minimum l that satisfies
    // g(op(a[l], a[l + 1], ..., a[r - 1])) = true.
    template <class G> int min_left(int r, G g) {
        assert(0 <= r && r <= n);
        assert(g(e()));
        if (r == 0) return 0;
        ll size = n;
        r += size;
        for (int i = h; i >= 1; i--) push((r - 1) >> i);
        Node sm = e();
        do {
            r--;
            while (r > 1 && (r % 2)) r >>= 1;
            if (!g(op(t[r], sm))) {
                while (r < size) {
                    push(r);
                    r = (2 * r + 1);
                    if (g(op(t[r], sm))) {
                        sm = op(t[r], sm);
                        r--;
                    }
                }
                return r + 1 - size;
            }
            sm = op(t[r], sm);
        } while ((r & -r) != r);
        return 0;
    }
};
```

## 6.21 segment tree 2d

```
// Tested: https://www.spoj.com/problems/MATSUM/
struct Node { ll x = 0; };
Node e() { return Node(); } // Identity element, op(x,e()) = x
// Binary Operation, Associative. (a+b)+c = a+(b+c)
Node op(const Node &a, const Node &b) {
    Node c;
    c.x = a.x + b.x;
    return c;
}
struct segtree2d {
    int n, m;
    vector<vector<Node>> t;
    void init(int n, int m) {
        this->n = n;
        this->m = m;
        t.assign(n*2, vector<Node>(m*2, e()));
    }
    void init(int n, int m, vector<vector<Node>> &nums) {
        init(n, m);
        for (int i = 0;i<n;i++) { // build leaf segtrees
            for (int j=0;j<m;j++) t[i+n][j+m] = nums[i][j];
            for (int j=m-1;j>0;j--) t[i+n][j] = op(t[i+n][j*2],
t[i+n][j*2+1]);
        }
```

```
        for (int i = n-1;i>0;i--) { // build non leaf segtrees
            for (int j=0;j<2*m;j++) t[i][j] = op(t[i*2][j],
t[i*2+1][j]);
        }
    }
    Node query_y(int x, int l, int r) {
        l += m, r += m+1;
        Node resl = e(), resr = e();
        for (; l < r; l>>=1, r>>=1) {
            if (l&1) resl = op(resl, t[x][l++]);
            if (r&1) resr = op(t[x][--r], resr);
        }
        return op(resl, resr);
    }
    Node query(int x1, int y1, int x2, int y2) {
        ll l = x1+n, r = x2+n+1;
        Node resl = e(), resr = e();
        for (; l < r; l>>=1, r>>=1) {
            if (l&1) resl = op(resl, query_y(l++, y1, y2));
            if (r&1) resr = op(query_y(--r,y1,y2), resr);
        }
        return op(resl,resr);
    }
    void update(int x, int y, const Node &v) {
        x += n, y += m;
        t[x][y] = v; // fix leaf segtrees
        for (int j = y; j>>=1; ) {
            t[x][j] = op(t[x][j*2], t[x][j*2+1]);
        }
        for (; x >>= 1; ) { // fix non leaf segtree
            t[x][y] = op(t[x*2][y], t[x*2+1][y]);
            for (int j = y; j>>=1; ) {
                t[x][j] = op(t[x*2][j], t[x*2+1][j]);
            }
        }
    };
    Node get(int x, int y) { return t[x+n][y+m]; }
};
```

## 6.22 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);
    }
};
```

## 6.23 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);
    }
};
```

# 7 4. Graphs

## 7.1 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];
        }
    }
}
```

## 7.2 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";
    }
}
```

### 7.3 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
*/
```

### 7.4 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;
  }
};
```

### 7.5 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;
}
```

# 8 4. Graphs - Shortest Path

## 8.1 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];
        }
    }
}
```

## 8.2 dijkstra k shortest path

```
// Using djisktra, finds the k shortesth paths from 1 to n
// 2≤n≤10^5, 1≤m≤2·10^5, 1≤weight≤10^9, 1≤k≤10
// 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];
    }
}
```

## 8.3 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
  }
}
```

## 8.4 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});
      }
    }
  }
}
```

### 8.5 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;
                }
            }
        }
    }
}
```

## 9 5. Flow

### 9.1 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];
    }
};
```

## 9.2 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;
    // Añade 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});
            }
        }
    }
}
```

## 9.3 blossom

```
Halla el máximo match en un grafo general O(E * v ^2)
struct network {
    struct struct_edge {
        int v; struct_edge * n;
    };
    typedef struct_edge* edge;
    int n;
    struct_edge pool[MAXE]; // 2*n*n;
    edge top;
    vector<edge> adj;
    queue<int> q;
    vector<int> f, base, inq, inb, inp, match;
    vector<vector<int>> ed;
    network(int n) : n(n), match(n, -1), adj(n), top(pool),
f(n), base(n),
        inq(n), inb(n), inp(n), ed(n, vector<int>(n)) {}
    void add_edge(int u, int v) {
        if (ed[u][v]) return;
        ed[u][v] = 1;
        top->v = v, top->n = adj[u], adj[u] = top++;
        top->v = u, top->n = adj[v], adj[v] = top++;
    }
    int get_lca(int root, int u, int v) {
        fill(inp.begin(), inp.end(), 0);
        while (1) {
            inp[u = base[u]] = 1;
            if (u == root) break;
            u = f[match[u]];
        }
        while (1) {
            if (inp[v = base[v]]) return v;
            else v = f[match[v]];
        }
    }
    void mark(int lca, int u) {
        while (base[u] != lca) {
            int v = match[u];
            inb[base[u]] = 1;
            inb[base[v]] = 1;
            u = f[v];
            if (base[u] != lca) f[u] = v;
        }
    }
    void blossom_contraction(int s, int u, int v) {
        int lca = get_lca(s, u, v);
```

```
        fill(inb.begin(), inb.end(), 0);
        mark(lca, u); mark(lca, v);
        if (base[u] != lca) f[u] = v;
        if (base[v] != lca) f[v] = u;
        for (int u = 0; u < n; u++) {
            if (inb[base[u]]) {
                base[u] = lca;
                if (!inq[u]) {
                    inq[u] = 1;
                    q.push(u);
                }
            }
        }
    }
    int bfs(int s) {
        fill(inq.begin(), inq.end(), 0);
        fill(f.begin(), f.end(), -1);
        for (int i = 0; i < n; i++) base[i] = i;
        q = queue<int>();
        q.push(s);
        inq[s] = 1;
        while (q.size()) {
            int u = q.front(); q.pop();
            for (edge e = adj[u]; e; e = e->n) {
                int v = e->v;
                if (base[u] != base[v] && match[u] != v) {
                    if ((v == s) || (match[v] != -1 &&
f[match[v]] != -1)) {
                        blossom_contraction(s, u, v);
                    } else if (f[v] == -1) {
                        f[v] = u;
                        if (match[v] == -1) return v;
                        else if (!inq[match[v]]) {
                            inq[match[v]] = 1;
                            q.push(match[v]);
                        }
                    }
                }
            }
        }
        return -1;
    }
    int doit(int u) {
        if (u == -1) return 0;
        int v = f[u];
        doit(match[v]);
        match[v] = u; match[u] = v;
        return u != -1;
    }
    // (i < net.match[i]) => means match
    int maximum_matching() {
        int ans = 0;
        for (int u = 0; u < n; u++)
            ans += (match[u] == -1) && doit(bfs(u));
        return ans;
    }
};
```

### 9.4 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";
    }
}
```

### 9.5 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};
    }
};
```

# 10 6. Tree

## 10.1 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";
        }
    }
}
```

### 10.2 isomorfismo

```
Permite hashear árboles para comparar su estructura
rápidamente.
Agregar Random Integer para ll
map<ll, ll> table;
ll get(ll x) {
    if (table.count(x)) return table[x];
    return table[x] = rand(0, 1e18);
}
ll hashes[N], sum[N];
void dfs(int u, int p) {
    sum[u] = 0;
    for (auto &v : g[u]) {
        if (v == p) continue;
        dfs(v, u);
        sum[u] += hashes[v];
    }
    hashes[u] = get(sum[u]);
}
```

### 10.3 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";
    }
}
```

### 10.4 heavy light

```
vector<vector<int>> adj; // INIT this, use init()!!!
vector<int> parent, depth, heavy, head, pos;
int cur_pos;
int dfs(int v) {
    int size = 1, max_size = 0;
    for (int c : adj[v]) if (c != parent[v]) {
        parent[c] = v, depth[c] = depth[v] + 1;
        int c_size = dfs(c);
        size += c_size;
        if (c_size > max_size)
            max_size = c_size, heavy[v] = c;
    }
    return size;
}
void decompose(int v, int h) {
    head[v] = h, pos[v] = cur_pos++;
    if (heavy[v] != -1) // check when copy head and heavy
        decompose(heavy[v], h);
    for (int c : adj[v]) {
        if (c != parent[v] && c != heavy[v])
            decompose(c, c);
    }
}
segtree tree; // !!ADD SegTree
vector<ll> values;
void init() {
    int n = adj.size();
    parent = depth = head = pos = vector<int>(n);
    heavy = vector<int>(n, -1);
    cur_pos = 0;
    dfs(0);
    decompose(0, 0);
    tree.init(n);
    for (int i = 0;i<n;i++) tree.update(pos[i], { values[i] });
}
int seg_query(int a, int b) {
    return tree.query(a, b).x; // !! depends of segtree
}
void update_query(int node, int val) {
    tree.update(pos[node], { val }); // depends of segtree
}
int query(int a, int b) {
    int res = 0;
    for (; head[a] != head[b]; b = parent[head[b]]) {
        if (depth[head[a]] > depth[head[b]])
            swap(a, b);
        int cur_heavy_path_max = seg_query(pos[head[b]],
pos[b]);
        res = max(res, cur_heavy_path_max);
    }
    if (depth[a] > depth[b])
        swap(a, b);
    int last_heavy_path_max = seg_query(pos[a], pos[b]); //
consider pos[a]+1 when query edges
    res = max(res, last_heavy_path_max); // Change to Segment
Tree Operation
    return res;
}
```

### 10.5 online centroid

```
// !!! Add centroid Descomposition
// When you can get Answers for fixed root
// When you can get answer for all paths for fixed root
struct centroid_data {
    ll idx = 0, best_red = -1;
    void mark_red(int x, ll d) {
        if (best_red==-1) best_red = d;
        best_red = min(best_red, d);
    }
```

```
    int query(int x, ll d) {
        if (best_red == -1) return inf;
        return best_red + d;
    }
};
vector<vector<pair<ll,ll>>> belongs; // idx centroid, extra
data, init this!!
vector<centroid_data> centroids;
int id_centroid = 0;
void process_centroid(int root, int sz) { // creates centroids
    centroids.push_back({id_centroid++, -1});
    auto & c = centroids.back();
    auto dfs = [&](int x, int p, ll dist, auto &&dfs) -> void {
        belongs[x].pb({c.idx, dist});
        for (auto & y : adj[x]) if (y != p && alive[y])
            dfs(y, x, dist+1, dfs);
    };
    dfs(root, -1, 0, dfs);
}
void mark_red(int x) { // UPDATE O(log n) centroids belongs to
node
    for (auto [idx, dist] : belongs[x])
        centroids[idx].mark_red(x, dist);
}
int query(int x) { // iterates over each centroid belongs to
node O(logn)
    int best = 1e9;
    for (auto [idx, dist] : belongs[x])
        best = min(best, centroids[idx].query(x, dist));
    return best;
}
```

## 10.6 centroid descomposition

```
vector<vector<ll>> adj;
vector<bool> alive;
vector<int> sz;
void init(int n) {
    alive = vector<bool>(n+1, true);
    sz = vector<int>(n+1);
    adj = vector<vector<ll>>(n+1);
    // belongs = vector<vector<pair<ll,ll>>>(n+1); // !! for
online
}
int get_sz(int x, int p) {
    sz[x] = 1;
    for (auto y : adj[x]) if (y != p && alive[y])
        sz[x] += get_sz(y, x);
    return sz[x];
}
int find_centroid(int x, int p, int tree_sz) {
    for (auto y : adj[x]) if (y != p && alive[y]) {
        if (sz[y] * 2 > tree_sz) {
            return find_centroid(y, x, tree_sz);
        }
    }
    return x;
}
//!!! Implement O(sz) solution, use adj, only use alive nodes
check with alive[x]
// dont sz[x] because it is not rerooted to centroid
void process_centroid(int root, int sz);
void centroid_decomp(ll x) {
    int sz = get_sz(x, -1);
    int centroid = find_centroid(x, -1, sz);
    process_centroid(centroid, sz); // implement!!!
    alive[centroid] = false;
    for (auto y : adj[centroid]) if (alive[y]) {
        centroid_decomp(y);
    }
}
```

## 10.7 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";
    }
}
```

# 11 7. Strings

## 11.1 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;
    }
};
```

## 11.2 micky hashing

```
ll pot(ll x, ll y, ll m) {
    if (y==0) return 1;
    ll ans = pot(x,y/2,m);
    ans = (ans*ans)%m;
    if (y&1) ans = (ans*x)%m;
    return ans;
}
struct Hash {
    int p = 997, m[2], in[2];
    vl h[2],inv[2];
    Hash(string s) {
        m[0] = 998244353, m[1]=1e9+9;
        for (int i = 0;i<2;i++) {
            in[i] = pot(p,m[i]-2,m[i]);
            h[i].resize(s.size()+1);
            inv[i].resize(s.size()+1);
            ll acu = 1;
            h[i][0]=0,inv[i][0]=1;
            for (int j = 0;j<s.size();j++) {
                h[i][j+1]=(h[i][j]+acu*s[j])%m[i];
                inv[i][j+1]=(1ll*inv[i][j]*in[i])%m[i];
                acu = (acu * p) % m[i];
            }
        }
    }
    ll get(int b, int e) {
        e++;
```

```
        ll ha[2];
        for (int i =0;i<2;i++) {
      ha[i] = ((((h[i][e] - h[i][b]) * (ll)inv[i][b]) % m[i]) +
m[i]) % m[i];
        }
        return ((ha[0]<<32)|ha[1]);
    }
};
```

### 11.3 manacher

```
Devuelve un vector p donde, para cada i, p[i] es igual al largo
del palindromo mas largo con centro en i.
string parse(string &s) {
    string t = "%";
    for (auto &c : s) t.pb('#'), t.pb(c);
    t += "#$";
    return t;
}
vector<int> manacher(string &s) {
    string t = parse(s);
    int n = t.size(), c = 0, r = 0;
    vector<int> p(n, 0);
    for (int i = 1; i < n-1; i++) {
        int j = c - (i-c) ;
        if (r > i) p[i] = min(r-i, p[j]);
        while (t[i+1+p[i]] == t[i-1-p[i]])
            p[i]++;
        if (i+p[i] > r) {
            c = i;
            r = i+p[i];
        }
    }
    return p;
    // si p[i] > 0 entonces, s[l, r) es palíndromo
    // donde l = (i-1-p[i])/2 y r = l+p[i];
}
```

### 11.4 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);
  }
};
```

### 11.5 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];
        }
    }
}
```

## 12 8. Geometry

### 12.1 ufps line

```
struct line {
    pt v; T c; // v:direction c: pos in y axis
    line(pt v, T c) : v(v), c(c) {}
    line(T a, T b, T c) : v({b, -a}), c(c) {} // ax + by = c
    line(pt p, pt q) : v(q-p), c(cross(v, p)) {}
    T side(pt p) { return cross(v, p)-c; }
    lf dist(pt p) { return abs(side(p)) / abs(v); }
    lf sq_dist(pt p) { return side(p)*side(p) / (lf)norm(v); }
    line perp_through(pt p) { return {p, p + rot90ccw(v)}; } //
line perp to v passing through p
    bool cmp_proj(pt p, pt q) { return dot(v, p) < dot(v,
q); } // order for points over the line
    line translate(pt t) { return {v, c + cross(v, t)}; }
    line shift_left(double d) { return {v, c + d*abs(v)}; }
    pt proj(pt p) { return p - rot90ccw(v)*side(p)/
norm(v); } // pt proyected on the line
    pt refl(pt p) { return p - rot90ccw(v)*2*side(p)/
norm(v); } // pt reflected on the other side of the line
};
bool inter_ll(line l1, line l2, pt &out) {
    T d = cross(l1.v, l2.v);
    if (d == 0) return false;
    out = (l2.v*l1.c - l1.v*l2.c) / d; // floating points
    return true;
}
// bisector divides the angle in 2 equal angles
// interior line goes on the same direction as l1 and l2
line bisector(line l1, line l2, bool interior) {
    assert(cross(l1.v, l2.v) != 0); // l1 and l2 cannot be
parallel!
    lf sign = interior ? 1 : -1;
    return {l2.v/abs(l2.v) + l1.v/abs(l1.v) * sign,
            l2.c/abs(l2.v) + l1.c/abs(l1.v) * sign};
}
```

## 12.2 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));
}
```

## 12.3 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;
}
```

## 12.4 ufps circle

```
struct circle {
    pt c; T r;
};
// (x-xo)^2 + (y-yo)^2 = r^2
// circle that passes through abc
circle center(pt a, pt b, pt c) {
    b = b-a, c = c-a;
    assert(cross(b, c) != 0); // no circumcircle if A, B, C
aligned
    pt cen = a + rot90ccw(b*norm(c) - c*norm(b))/cross(b, c)/2;
    return {cen, abs(a-cen)};
}
// centers of the circles that pass through ab and have radius
r
vector<pt> centers(pt a, pt b, T r) {
    if (abs(a-b) > 2*r + eps) return {};
    pt m = (a+b)/2;
    double f = sqrt(r*r/norm(a-m) - 1);
    pt c = rot90ccw(a-m)*f;
    return {m-c, m+c};
}
int inter_cl(circle c, line l, pair<pt, pt> &out) {
    lf h2 = c.r*c.r - l.sq_dist(c.c);
    if (h2 >= 0) { // line touches circle
        pt p = l.proj(c.c);
        pt h = l.v*sqrt(h2)/abs(l.v); // vector of len h
```

```
parallel to line
        out = {p-h, p+h};
    }
    return 1 + sign(h2); // if 1 -> out.F == out.S
}
int inter_cc(circle c1, circle c2, pair<pt, pt> &out) {
    pt d = c2.c-c1.c;
    double d2 = norm(d);
    if (d2 == 0) { assert(c1.r != c2.r); return 0; } //
concentric circles (identical)
    double pd = (d2 + c1.r*c1.r - c2.r*c2.r)/2; // = |O_1P| * d
    double h2 = c1.r*c1.r - pd*pd/d2; // = h^2
    if (h2 >= 0) {
        pt p = c1.c + d*pd/d2, h = rot90ccw(d)*sqrt(h2/d2);
        out = {p-h, p+h};
    }
    return 1 + sign(h2);
}
// circle-line inter = 1
int tangents(circle c1, circle c2, bool inner, vector<pair<pt,
pt>> &out) {
    if (inner) c2.r = -c2.r; // inner tangent
    pt d = c2.c-c1.c;
    double dr = c1.r-c2.r, d2 = norm(d), h2 = d2-dr*dr;
    if (d2 == 0 || h2 < 0) { assert(h2 != 0); return 0; } //
(identical)
    for (double s : {-1, 1}) {
        pt v = (d*dr + rot90ccw(d)*sqrt(h2)*s)/d2;
        out.pb({c1.c + v*c1.r, c2.c + v*c2.r});
    }
    return 1 + (h2 > 0); // if 1: circle are tangent
}
// circle targent passing through pt p
int tangent_through_pt(pt p, circle c, pair<pt, pt> &out) {
    double d = abs(p - c.c);
    if (d < c.r) return 0;
    pt base = c.c-p;
    double w = sqrt(norm(base) - c.r*c.r);
    pt a = {w, c.r}, b = {w, -c.r};
    pt s = p + base*a/norm(base)*w;
    pt t = p + base*b/norm(base)*w;
    out = {s, t};
    return 1 + (abs(c.c-p) == c.r);
}
```

## 12.5 ufps polygon

```
enum {IN, OUT, ON};
struct polygon {
    vector<pt> p;
    polygon(int n) : p(n) {}
    int top = -1, bottom = -1;
    void delete_repetead() {
        vector<pt> aux;
        sort(p.begin(), p.end());
        for (pt &i : p)
            if (aux.empty() || aux.back() != i)
              aux.pb(i);
        p.swap(aux);
    }
    bool is_convex() {
        bool pos = 0, neg = 0;
        for (int i = 0, n = p.size(); i < n; i++) {
            int o = orient(p[i], p[(i+1)%n], p[(i+2)%n]);
            if (o > 0) pos = 1;
            if (o < 0) neg = 1;
        }
        return !(pos && neg);
    }
    lf area(bool s = false) { // better on clockwise order
        lf ans = 0;
        for (int i = 0, n = p.size(); i < n; i++)
            ans += cross(p[i], p[(i+1)%n]);
        ans /= 2;
        return s ? ans : abs(ans);
    }
    lf perimeter() {
        lf per = 0;
        for (int i = 0, n = p.size(); i < n; i++)
           per += abs(p[i] - p[(i+1)%n]);
        return per;
    }
    bool above(pt a, pt p) { return p.y >= a.y; }
```

```
    bool crosses_ray(pt a, pt p, pt q) { // pq crosses ray from
a
        return (above(a, q)-above(a, p))*orient(a, p, q) > 0;
    }
    int in_polygon(pt a) {
        int crosses = 0;
        for (int i = 0, n = p.size(); i < n; i++) {
            if (on_segment(p[i], p[(i+1)%n], a)) return ON;
            crosses += crosses_ray(a, p[i], p[(i+1)%n]);
        }
        return (crosses&1 ? IN : OUT);
    }
    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();
    }
    int in_convex(pt a) {
        assert(bottom == 0 && top != -1);
        if (a < p[0] || a > p[top]) return OUT;
        T orientation = orient(p[0], p[top], a);
        if (orientation == 0) {
            if (a == p[0] || a == p[top]) return ON;
            return top == 1 || top + 1 == p.size() ? ON : IN;
        } else if (orientation < 0) {
            auto it = lower_bound(p.begin()+1, p.begin()+top,
a);
            T d = orient(*prev(it), a, *it);
            return d < 0 ? IN : (d > 0 ? OUT: ON);
        } else {
            auto it = upper_bound(p.rbegin(), p.rend()-top-1,
a);
            T d = orient(*it, a, it == p.rbegin() ? p[0] :
*prev(it));
            return d < 0 ? IN : (d > 0 ? OUT: ON);
        }
    }
    polygon cut(pt a, pt b) { // cuts polygon on line ab
        line l(a, b);
        polygon new_polygon(0);
        for (int i = 0, n = p.size(); i < n; ++i) {
            pt c = p[i], d = p[(i+1)%n];
            lf abc = cross(b-a, c-a), abd = cross(b-a, d-a);
            if (abc >= 0) new_polygon.p.pb(c);
            if (abc*abd < 0) {
              pt out; inter_ll(l, line(c, d), out);
              new_polygon.p.pb(out);
            }
        }
        return new_polygon;
    }
    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.pb(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);
    }
    vector<pii> antipodal() {
        vector<pii> ans;
        int n = p.size();
        if (n == 2) ans.pb({0, 1});
        if (n < 3) return ans;
        auto nxt = [&](int x) { return (x+1 == n ? 0 : x+1); };
        auto area2 = [&](pt a, pt b, pt c) { return cross(b-a,
```

```
c-a); };
        int b0 = 0;
        while (abs(area2(p[n - 1], p[0], p[nxt(b0)])) >
abs(area2(p[n - 1], p[0], p[b0]))) ++b0;
        for (int b = b0, a = 0; b != 0 && a <= b0; ++a) {
            ans.pb({a, b});
            while (abs(area2(p[a], p[nxt(a)], p[nxt(b)])) >
abs(area2(p[a], p[nxt(a)], p[b]))) {
                b = nxt(b);
                if (a != b0 || b != 0) ans.pb({ a, b });
                else return ans;
            }
            if (abs(area2(p[a], p[nxt(a)], p[nxt(b)])) ==
abs(area2(p[a], p[nxt(a)], p[b]))) {
                if (a != b0 || b != n-1) ans.pb({ a, nxt(b) });
                else ans.pb({ nxt(a), b });
            }
        }
        return ans;
    }
    pt centroid() {
        pt c{0, 0};
        lf scale = 6. * area(true);
        for (int i = 0, n = p.size(); i < n; ++i) {
            int j = (i+1 == n ? 0 : i+1);
            c = c + (p[i] + p[j]) * cross(p[i], p[j]);
        }
        return c / scale;
    }
    ll pick() {
        ll boundary = 0;
        for (int i = 0, n = p.size(); i < n; i++) {
            int j = (i+1 == n ? 0 : i+1);
            boundary += __gcd((ll)abs(p[i].x - p[j].x),
(ll)abs(p[i].y - p[j].y));
        }
        return area() + 1 - boundary/2;
    }
    pt& operator [] (int i) { return p[i]; }
};
```

### 12.6 ufps segment

```
bool in_disk(pt a, pt b, pt p) { // pt p inside ab disk
    return dot(a-p, b-p) <= 0;
}
bool on_segment(pt a, pt b, pt p) { // p on ab
    return orient(a, b, p) == 0 && in_disk(a, b, p);
}
// ab crossing cd
bool proper_inter(pt a, pt b, pt c, pt d, pt &out) {
    T oa = orient(c, d, a),
    ob = orient(c, d, b),
    oc = orient(a, b, c),
    od = orient(a, b, d);
    // Proper intersection exists iff opposite signs
    if (oa*ob < 0 && oc*od < 0) {
        out = (a*ob - b*oa) / (ob-oa);
        return true;
    }
    return false;
}
// intersection bwn segments
set<pt> inter_ss(pt a, pt b, pt c, pt d) {
    pt out;
    if (proper_inter(a, b, c, d, out)) return {out}; // if
cross -> 1
    set<pt> s;
    if (on_segment(c, d, a)) s.insert(a); // a in cd
    if (on_segment(c, d, b)) s.insert(b); // b in cd
    if (on_segment(a, b, c)) s.insert(c); // c in ab
    if (on_segment(a, b, d)) s.insert(d); // d in ab
    return s; // 0, 2
}
lf pt_to_seg(pt a, pt b, pt p) { // p to ab
    if (a != b) {
        line l(a, b);
        if (l.cmp_proj(a, p) && l.cmp_proj(p, b)) // if closest
to projection = (a, p, b)
            return l.dist(p); // output distance to line
    }
    return min(abs(p-a), abs(p-b)); // otherwise distance to A
or B
}
```

```
lf seg_to_seg(pt a, pt b, pt c, pt d) {
    pt dummy;
    if (proper_inter(a, b, c, d, dummy)) return 0; // ab
intersects cd
    return min({pt_to_seg(a, b, c), pt_to_seg(a, b, d),
pt_to_seg(c, d, a), pt_to_seg(c, d, b)}); // try the 4 pts
}
```

## 12.7 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]);
}
```

## 12.8 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/BIPDAUMWyupUW18
AlWgd
// 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;
}
//
```

## 12.9 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;
    }
};
```

## 12.10 ufps point

```
typedef double lf;
const lf eps = 1e-9;
const lf PI = acos(-1.0);
typedef double T;
struct pt {
    T x, y;
    pt operator + (pt p) { return {x+p.x, y+p.y}; }
    pt operator - (pt p) { return {x-p.x, y-p.y}; }
    pt operator * (pt p) { return {x*p.x-y*p.y, x*p.y+y*p.x}; }
    pt operator * (T d) { return {x*d, y*d}; }
    pt operator / (lf d) { return {x/d, y/d}; } // only for
floating point
    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); }
    bool operator > (const pt &o) const { return y > o.y || (y
== o.y && x > o.x); }
};
int cmp(lf a, lf b) { return (a + eps < b ? -1 :(b + eps < a ?
1 : 0)); } // double comparator
T norm(pt a) { return a.x*a.x + a.y*a.y; }
lf abs(pt a) { return sqrt(norm(a)); }
lf arg(pt a) { return atan2(a.y, a.x); }
pt unit(pt a) { return a/abs(a); }
T dot(pt a, pt b) { return a.x*b.x + a.y*b.y; } // x = 90 ->
cos = 0
T cross(pt a, pt b) { return a.x*b.y - a.y*b.x; } // x = 180 ->
sin = 0
T orient(pt a, pt b, pt c) { return cross(b-a, c-a); } //
clockwise = -
pt rot(pt p, lf a) { return {p.x*cos(a) - p.y*sin(a),
p.x*sin(a) + p.y*cos(a)}; }
pt rotate_to_b(pt a, pt b, lf ang) { return rot(a-b,
ang)+b; } // rotate by ang center b
pt rot90ccw(pt p) { return {-p.y, p.x}; }
pt rot90cw(pt p) { return {p.y, -p.x}; }
pt translate(pt p, pt v) { return p+v; }
pt scale(pt p, double f, pt c) { return c + (p-c)*f; } // c-
center
bool are_perp(pt v, pt w) { return dot(v, w) == 0; }
```

```
int sign(T x) { return (T(0) < x) - (x < T(0)); }
bool in_angle(pt a, pt b, pt c, pt x) { // x inside angle abc
(center in a)
    assert(orient(a, b, c) != 0);
    if (orient(a, b, c) < 0) swap(b, c);
    return orient(a, b, x) >= 0 && orient(a, c, x) <= 0;
}
// angle bwn 2 vectors [0, pi] -> [0, 180] and (-pi, 0) ->
(180, 360)
lf angle(pt a, pt b) { return acos(max(-1.0, min(1.0, dot(a,
b)/abs(a)/abs(b)))); }
lf angle(pt a, pt b) { return atan2(cross(a, b), dot(a, b)); }
lf angle360(pt a, pt b) { // [0, 360)
    lf ang = angle(a, b); return (ang < 0 ? ang+2*PI : ang) *
360/(2*PI);
}
// returns vector to transform points
pt get_linear_transformation(pt p, pt q, pt r, pt fp, pt fq) {
    pt pq = q-p, num{cross(pq, fq-fp), dot(pq, fq-fp)};
    return fp + pt{cross(r-p, num), dot(r-p, num)} / norm(pq);
}
bool half(pt p) { // true if is in (0, 180] (line is x axis)
    assert(p.x != 0 || p.y != 0); // the argument of (0, 0) is
undefined
    return p.y > 0 || (p.y == 0 && p.x < 0);
}
bool half_from(pt p, pt v = {1, 0}) { // line is v (above v is
true)
    return cross(v, p) < 0 || (cross(v, p) == 0 && dot(v, p) <
0);
}
bool polar_cmp(const pt &a, const pt &b) { // polar sort
    return make_tuple(half(a), 0) < make_tuple(half(b),
cross(a, b));
    // return make_tuple(half(a), 0, sq(a)) <
make_tuple(half(b), cross(a, b), sq(b)); // further ones appear
later
}
```

## 12.11 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";
}
```

## 12.12 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;
}
```

# 13 9. Utils

## 13.1 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 one's(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
```

## 13.2 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")
```

## 13.3 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);
}
```

### 13.4 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);
}
```

### 13.5 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');
}
```

### 13.6 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
```

### 13.7 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;
}
```

# 14 A. To Order

## 14.1 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;
}
```

## 14.2 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;
}
```

### 14.3 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);
    }
};
```

## 14.4 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)·(r−l+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;
}
```

## 14.5 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);
// }
```

### 14.6 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.
```

### 14.7 mo's in tree's

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 40005;
const int MAXM = 100005;
const int LN = 19;
int N, M, K, cur, A[MAXN], LVL[MAXN], DP[LN][MAXN];
int BL[MAXN << 1], ID[MAXN << 1], VAL[MAXN], ANS[MAXM];
int d[MAXN], l[MAXN], r[MAXN];
bool VIS[MAXN];
vector < int > adjList[MAXN];
struct query{
  int id, l, r, lc;
  bool operator < (const query& rhs){
    return (BL[l] == BL[rhs.l]) ? (r < rhs.r) : (BL[l] <
BL[rhs.l]);
  }
}Q[MAXM];
// Set up Stuff
void dfs(int u, int par){
  l[u] = ++cur;
  ID[cur] = u;
  for (int i = 1; i < LN; i++) DP[i][u] = DP[i - 1][DP[i - 1]
[u]];
  for (int i = 0; i < adjList[u].size(); i++){
    int v = adjList[u][i];
    if (v == par) continue;
    LVL[v] = LVL[u] + 1;
    DP[0][v] = u;
    dfs(v, u);
  }
  r[u] = ++cur; ID[cur] = u;
}
// Function returns lca of (u) and (v)
inline int lca(int u, int v){
  if (LVL[u] > LVL[v]) swap(u, v);
  for (int i = LN - 1; i >= 0; i--)
    if (LVL[v] - (1 << i) >= LVL[u]) v = DP[i][v];
  if (u == v) return u;
  for (int i = LN - 1; i >= 0; i--){
    if (DP[i][u] != DP[i][v]){
      u = DP[i][u];
```

```
      v = DP[i][v];
    }
  }
  return DP[0][u];
}
inline void check(int x, int& res){
  // If (x) occurs twice, then don't consider it's value
  if ( (VIS[x]) and (--VAL[A[x]] == 0) ) res--;
  else if ( (!VIS[x]) and (VAL[A[x]]++ == 0) ) res++;
  VIS[x] ^= 1;
}
void compute(){
  // Perform standard Mo's Algorithm
  int curL = Q[0].l, curR = Q[0].l - 1, res = 0;
  for (int i = 0; i < M; i++){
    while (curL < Q[i].l) check(ID[curL++], res);
    while (curL > Q[i].l) check(ID[--curL], res);
    while (curR < Q[i].r) check(ID[++curR], res);
    while (curR > Q[i].r) check(ID[curR--], res);
    int u = ID[curL], v = ID[curR];
    // Case 2
    if (Q[i].lc != u and Q[i].lc != v) check(Q[i].lc, res);
    ANS[Q[i].id] = res;
    if (Q[i].lc != u and Q[i].lc != v) check(Q[i].lc, res);
  }
  for (int i = 0; i < M; i++) printf("%d\n", ANS[i]);
}
int main(){
  int u, v, x;
  while (scanf("%d %d", &N, &M) != EOF){
    // Cleanup
    cur = 0;
    memset(VIS, 0, sizeof(VIS));
    memset(VAL, 0, sizeof(VAL));
    for (int i = 1; i <= N; i++) adjList[i].clear();
    // Inputting Values
    for (int i = 1; i <= N; i++) scanf("%d", &A[i]);
    memcpy(d + 1, A + 1, sizeof(int) * N);
    // Compressing Coordinates
    sort(d + 1, d + N + 1);
    K = unique(d + 1, d + N + 1) - d - 1;
    for (int i = 1; i <= N; i++) A[i] = lower_bound(d + 1, d +
K + 1, A[i]) - d;
    // Inputting Tree
    for (int i = 1; i < N; i++){
      scanf("%d %d", &u, &v);
      adjList[u].push_back(v);
      adjList[v].push_back(u);
    }
    // Preprocess
    DP[0][1] = 1;
    dfs(1, -1);
    int size = sqrt(cur);
    for (int i = 1; i <= cur; i++) BL[i] = (i - 1) / size + 1;
    for (int i = 0; i < M; i++){
      scanf("%d %d", &u, &v);
      Q[i].lc = lca(u, v);
      if (l[u] > l[v]) swap(u, v);
      if (Q[i].lc == u) Q[i].l = l[u], Q[i].r = l[v];
      else Q[i].l = r[u], Q[i].r = l[v];
      Q[i].id = i;
    }
    sort(Q, Q + M);
    compute();
  }
}
```

### 14.8 pi

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

### 14.9 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
```

## 15 Problems

### 15.1 polynomialsumlazysegtree

```
/* 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);
    }
};
```

## 15.2 shift poly

```
// ADD ntt, or karatsuba.
// A poly: c_0 + c_1*x + c_2*x^2 + ... is transformed to
// c_0 + c_1*(x+k) + c_)*(x+k)^2 + ...
vl fact, ifact;
vl ki, iki;
void initShifts(ll n, ll k) {
    k = (k%MOD + MOD) % MOD;
    fact = ifact = vl(n+1);
    ki = iki = vl(n+1);
    fact[0] = ifact[0] = 1;
    ki[0] = iki[0] = 1;
    for (int i = 1; i <= n; i++) {
        fact[i] = (fact[i-1]*i)%MOD;
        ifact[i] = inv(fact[i]);
        ki[i] = (ki[i-1]*k) % MOD;
        iki[i] = inv(ki[i]);
    }
}
// P(x + k)
vl shift(vl &a, ll k) {
    if (k == 0) return a;
    ll n = a.size();
    initShifts(n, k);
    vl l(n), r(n);
    for (int i = 0; i < n; i++) {
        l[i] = mulmod(a[i], fact[i]);
        r[i] = mulmod(ki[n-1-i], ifact[n-1-i]);
    }
    vl c = multiply(l,r);
    vl ans(n);
    for (int i = 0; i < n; i++) {
        ans[i] = mulmod(c[n-1+i], ifact[i]);
    }
    return ans;
}
```

## 15.3 hashing max frequent subarray

```
// Find the max sub array that has same freq
// Of elements from 1 to K.
// nums <= 4*10^5
// 1 <= nums[i] <= k, 4*10^5
/*
n:6 k:2
nums:2 2 1 1 2 2
ans: 4
*/
#define bint __int128
ll findMaxFreqSubarray(ll n, ll k, vl nums) {
    bint MOD=212345678987654321LL; // prime
    bint  PI=1e9 + 7; // prime
    vector<bint> pows(k+1, 1);
    for (int i = 0; i < k; i++) {
        pows[i+1] = (pows[i] * PI) % MOD;
    }
    bint oneHash = 0;
    for (int i = 0; i < k; i++) {
        oneHash += pows[i];
        oneHash %= MOD;
    }
    vector<bint> hashes = {0}; // hashes with same freq
    for (int i = 0; i <= n/k; i++) {
        hashes.pb((hashes.back() + oneHash) % MOD);
    }
    map<bint,ll> prefixes;
    prefixes[0] = -1;
    bint actual = 0;
    set<pair<ll,ll>> freqs;
```

```
    for (int i = 1; i <= k; i++) {
        freqs.insert({0, i});
    }
    vl cnt(k+1);
    ll ans = 0;
    for (int i =0; i < n; i++) { // n
        freqs.erase({cnt[nums[i]], nums[i]});
        cnt[nums[i]]++;
        freqs.insert({cnt[nums[i]], nums[i]});
        ll mn = freqs.begin()->F;
        actual = (actual + pows[nums[i]-1]) % MOD;
        bint needed = (actual - hashes[mn] + MOD) % MOD;
        if (prefixes.count(needed)) {
            ans = max(ans, i - prefixes[needed]);
        }
        if (prefixes.count(needed)) continue;
        prefixes[needed] = i;
    }
    return ans;
}
```