4gr manga 7gr

Contents

Miscellaneous

Basic

```
#include <bits/stdc++.h>
#define pb push_back
#define F first
#define S second
#define MP make_pair
#define all(x) x.begin(),x.end()
#define Fast
ios::sync_with_stdio(false);cout.tie(NULL);cin.tie(NULL
);

using namespace std;
using ll = long long;
using pi = pair<int, int>;
using vi = vector<int>;
using vl = vector<ll>;
using vpi = vector <pair<int, int>>;
using vvi = vector <vector<int>>;

const int OO = 1e9 + 5;
const int N = 2e5 + 5;

void TC(){

}

int32_t main() {
#ifndef ONLINE_JUDGE
    freopen("input.in", "r", stdin);
freopen("output.out", "w", stdout);
#endif
    Fast
    int t = 1;
    cin >> t;
    while (t--) {
        TC();
        cout << '\n';
    }
    return 0;
}
```

Optimizations

```
// Cmake files
//add_definitions(-D Clion)
//set(CMAKE_EXE_LINKER_FLAGS "${CMAKE_EXE_LINKER_FLAGS}
-Wl,--stack,1000000000")

#pragma GCC optimize("O3")
#pragma GCC optimize ("unroll-loops")
#pragma GCC optimize ("Ofast");
#pragma GCC target("avx2")
```

Modular

```
const int MOD = 998244353;

int add(ll a, ll b) {
    a %= MOD, b %= MOD;
    a += b;
    if (a >= MOD) a -= MOD;
    return a;
}

int sub(ll a, ll b) {
    a %= MOD, b %= MOD;
    a -= b;
    if (a < 0) a += MOD;
    return a;
}

int mul(ll a, ll b) { return (a % MOD) * (b % MOD) %
MOD; }

int powmod(ll x, ll y) {
    x %= MOD;
    int ans = 1;
    while (y) {
        if (y & 1) ans = mul(ans, x);
        x = mul(x, x);
        y >>= 1;
    }
    return ans;
}

int inv(ll a) { return powmod(a, MOD - 2); }
```

Compress

```
int arr[N],n;
 void compress() {
    vector<int> vals;
    for (int i = 0; i < n; ++i) {
        vals.push_back(arr[i]);
    }
    sort(all(vals));
    vals.erase(unique(vals.begin(), vals.end()),
vals.end());
    for (int i = 0; i < n; ++i) {
        arr[i] = lower_bound(all(vals), arr[i]) -
vals.begin();
    }
}
```

Ordered Set and Fast Map

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;
template<typename T>
using ordered_set = tree<T, null_type, less<T>,
rb_tree_tag, tree_order_statistics_node_update>;
template<typename T> using ordered_multiset = tree<T,
null_type,less_equal <T>, rb_tree_tag,
tree_order_statistics_node_update>;
struct chash {
    const int RANDOM = (long
long)(make_unique<char>().get()) ^
chrono::high_resolution_clock::now().time_since_epoch()
.count();
    static unsigned long long hash_f(unsigned long long
x) {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
```

```
    }
    static unsigned hash_combine(unsigned a, unsigned
b) { return a * 31 + b; }
    int operator()(int x) const { return
hash_f(x)^RANDOM; }
};

gp_hash_table<int, int, chash> table;
```

Random

```
void random() {
    mt19937
rng(chrono::steady_clock::now().time_since_epoch().coun
t());
    int n;
    vector<int>v(n);
    // shuffle 1
    shuffle(v.begin(), v.end(), rng);
    // shuffle 2
    for (int i = 1; i < n; i++)
        swap(v[i], v[uniform_int_distribution<int>(0,
i)(rng)]);
}
```

Fractions Up To N

```
vector<int> s;
    for (int i = 0; i <= n; i++) {
        unsigned long long k = a / b;
        a -= b * k;
        a *= 10;
        s.push_back(k);
    }
```

Kth Balanced Bracket Sequence

```
//O(n^2)
string kth_balanced(int n, int k) {
    vector<vector<int>> d(2*n+1, vector<int>(n+1, 0));
    d[0][0] = 1;
    for (int i = 1; i <= 2*n; i++) {
        d[i][0] = d[i-1][1];
        for (int j = 1; j < n; j++)
            d[i][j] = d[i-1][j-1] + d[i-1][j+1];
        d[i][n] = d[i-1][n-1];
    }

    string ans;
    int depth = 0;
    for (int i = 0; i < 2*n; i++) {
        if (depth + 1 <= n && d[2*n-i-1][depth+1] >= k)
{
            ans += '(';
            depth++;
        } else {
            ans += ')';
            if (depth + 1 <= n)
                k -= d[2*n-i-1][depth+1];
            depth--;
        }
    }
    return ans;
}
```

Next Balanced Bracket Sequence

```
//This function computes O(n) time the next balanced
bracket sequence, and returns false if there is no next
one.
bool next_balanced_sequence(string & s) {
    int n = s.size();
    int depth = 0;
    for (int i = n - 1; i >= 0; i--) {
        if (s[i] == '(')
            depth--;
        else
            depth++;

        if (s[i] == '(' && depth > 0) {
            depth--;
            int open = (n - i - 1 - depth) / 2;
            int close = n - i - 1 - open;
            string next = s.substr(0, i) + ')' +
string(open, '(') + string(close, ')');
            s.swap(next);
            return true;
        }
    }
    return false;
}
```

Notes

Removing Item From Knapsack:

Suppose there are n rocks, each with a weight wi. You are maintaining an array dp[i], where dp[i] is the number of ways to pick a subset of rocks with total weight exactly i.

Adding a new item is classical:

```
1 # we go from large to small so that the already updated dp values won't
affect any calculations
2 for (int i = dp.size() - 1; i >= weight; i--) {
3     dp[i] += dp[i - weight];
4 }
```

To undo what we just did, we can simply do everything backwards.

```
1 # this moves the array back to the state as it was before the item was
added
2 for (int i = weight; i < dp.size(); i++) {
3     dp[i] -= dp[i - weight];
4 }
```

Notice however, that the array dp does not in any way depend on the order the items were added. So in fact, the code above will correctly delete any one element with weight weight from the array — we can just pretend that it was the last one added to prove the correctness.

3k trick, square root optimization of knapsack:

Assume you have n rocks with nonnegative integer weights a1,a2,…,an such that a1+a2+⋯+an=m. You want to find out if there is a way to choose some rocks such that their total weight is w.

Suppose there are three rocks with equal weights a,a,a

. Notice that it doesn't make any difference if we replace these three rocks with two rocks with weights a,2a. We can repeat this process of replacing until there are at most two rocks of each weight. The sum of weights is still m, so there can be only O(m−−√) rocks (see next point). Now you can use a classical DP algorithm but with only O(m−−√)

elements, which can be lead to a better complexity in many cases.

This trick mostly comes up when the a1,a2,…,an

form a partition of some kind. For example, maybe they represent connected components of a graph. See the example.

Number Theory

Congruence Equation

```
ll extended_euclid(ll a, ll b, ll &x, ll &y) {
  if (b == 0) {
    x = 1; y = 0;
    return a;
  }
  ll x1, y1;
  ll d = extended_euclid(b, a % b, x1, y1);
  x = y1;
  y = x1 - y1 * (a / b);
  return d;
}
ll inverse(ll a, ll m) {
  ll x, y;
  ll g = extended_euclid(a, m, x, y);
  if (g != 1) return -1;
  return (x % m + m) % m;
}
// ax = b (mod m)
vector<ll> congruence_equation(ll a, ll b, ll m) {
  vector<ll> ret;
  ll g = gcd(a, m), x;
  if (b % g != 0) return ret;
  a /= g, b /= g;
  x = inverse(a, m / g) * b;
  for (int k = 0; k < g; ++k) { // exactly g solutions
    ret.push_back((x + m / g * k) % m);
  }
  // minimum solution = (m / g - (m - x) % (m / g)) %
(m / g)
  return ret;
}
```

Floor Values

```
//code to get all differnet values of floor(n/i)
for (ll l = 1, r = 1; (n/l); l = r + 1) {
    r = (n/(n/l));
    // q = (n/l), process the range [l, r]
}
```

Chinese Remainder Theorem

```
/// calculate each two congruences then solve with
next: sol(sol(sol(1, 2), 3), 4)

/// T = x mod N                  -> T = N * k + x
/// T = y mod M                  -> T = M * p + y
/// N * k + x = M * p + y        -> N * k - M * p = y -
x (LDE)
ll CRT(vector<ll> &rems, vector<ll> &mods){
    ll prevRem = rems[0], prevMod = mods[0]; /// first
congruence

    for(int i = 1; i < rems.size(); i++){
        ll x, y, c = rems[i] - prevRem;

        if(c % __gcd(prevMod, -mods[i])) /// LDE can't
be solved (no answer to system of congruences)
            return -1;

        ll g = eGCD(prevMod, -mods[i], x, y);
        x *= c / g;

        prevRem += prevMod * x;
        prevMod = prevMod / g * mods[i];
        prevRem = ((prevRem % prevMod) + prevMod) %
prevMod;
    }

    return prevRem;
}
```

Sieve

```
const int N = 1e6 + 5;
int SPF[N];
void sieve()
{
    for(int x=1; x<N; x++)
        SPF[x] = x;

    for(ll x=2; x< N; x++)
    {
        if(SPF[x] != x)
            continue;
        for(ll i = x*x; i<N; i+=x)
        {
            if(SPF[i] != i)
                continue;
            SPF[i] = (int)x;
        }
    }
}

map<int,int> factorize(int x)
{
    map<int,int> facts;
    while(x > 1)
    {
        int p = SPF[x];
        facts[p]++;
        x /= p;
    }
    return facts;
}
```

Long Division

```
string longDivision(string num, ll divisor){
    string ans;

    ll idx = 0;
    ll temp = num[idx] - '0';
    while (temp < divisor)
        temp = temp * 10 + (num[++idx] - '0');

    while (num.size() > idx) {
        ans += (temp / divisor) + '0';

        temp = (temp % divisor) * 10 + num[++idx] -
'0';
    }

    if (ans.length() == 0)
        return "0";
    return ans;
}
```

Linear Sieve and Mobius

```
vi prime;
bool isComp[N];
int mob[N];

void sieve(int n = N) {
```

```
    fill(isComp, isComp + n, false);
    mob[1] = 1;
    for (int i = 2; i < n; ++i) {
        if (!isComp[i]) {
            prime.push_back(i);
            mob[i] = -1;
        }
        for (int j = 0; j < prime.size() && i *
prime[j] < n; ++j) {
            isComp[i * prime[j]] = true;
            if (i % prime[j] == 0) {
                mob[i * prime[j]] = 0;
                break;
            } else
                mob[i * prime[j]] = mob[i] *
mob[prime[j]];
        }
    }
}
```

Discrete Logarithm

```
// Returns minimum x for which a ^ x % m = b % m.
int solve(int a, int b, int m) {
    a %= m, b %= m;
    int k = 1, add = 0, g;
    while ((g = gcd(a, m)) > 1) {
        if (b == k)
            return add;
        if (b % g)
            return -1;
        b /= g, m /= g, ++add;
        k = (k * 1ll * a / g) % m;
    }

    int n = sqrt(m) + 1;
    int an = 1;
    for (int i = 0; i < n; ++i)
        an = (an * 1ll * a) % m;

    unordered_map<int, int> vals;
    for (int q = 0, cur = b; q <= n; ++q) {
        vals[cur] = q;
        cur = (cur * 1ll * a) % m;
    }

    for (int p = 1, cur = k; p <= n; ++p) {
        cur = (cur * 1ll * an) % m;
        if (vals.count(cur)) {
            int ans = n * p - vals[cur] + add;
            return ans;
        }
    }
    return -1;
}
```

Linear Diophantine Equation

```
// Solves a*x + b*y = c where c is divisible by
gcd(a,b)
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}
bool find_any_solution(int a, int b, int c, int &x0,
int &y0, int &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }
    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}

void shift_solution(int & x, int & y, int a, int b, int
cnt) {
    x += cnt * b;
    y -= cnt * a;
}

int find_all_solutions(int a, int b, int c, int minx,
int maxx, int miny, int maxy) {
    int x, y, g;
    if (!find_any_solution(a, b, c, x, y, g))
        return 0;
    a /= g;
    b /= g;

    int sign_a = a > 0 ? +1 : -1;
    int sign_b = b > 0 ? +1 : -1;

    shift_solution(x, y, a, b, (minx - x) / b);
    if (x < minx)
        shift_solution(x, y, a, b, sign_b);
    if (x > maxx)
        return 0;
    int lx1 = x;

    shift_solution(x, y, a, b, (maxx - x) / b);
    if (x > maxx)
        shift_solution(x, y, a, b, -sign_b);
    int rx1 = x;

    shift_solution(x, y, a, b, -(miny - y) / a);
    if (y < miny)
        shift_solution(x, y, a, b, -sign_a);
    if (y > maxy)
        return 0;
    int lx2 = x;

    shift_solution(x, y, a, b, -(maxy - y) / a);
    if (y > maxy)
        shift_solution(x, y, a, b, sign_a);
    int rx2 = x;

    if (lx2 > rx2)
        swap(lx2, rx2);
    int lx = max(lx1, lx2);
    int rx = min(rx1, rx2);

    if (lx > rx)
        return 0;
    return (rx - lx) / abs(b) + 1;
}


/*
aX + bY = g
```

```
aXt + bYt = c = gt

t = c / g
x *= t, y *= t
xUnit = b / g, yUnit = a / g;
*/

// if you want to use with Y pass: (y, x, yUnit, xUnit,
bar, orEqual)

void raiseXOverBar(ll &x, ll &y, ll &xUnit, ll &yUnit,
ll bar, bool orEqual){
    if(x > bar or (x == bar and orEqual))
        return;

    ll shift = (bar - x + xUnit - orEqual) / xUnit;
    x += shift * xUnit;
    y -= shift * yUnit;
}

void lowerXUnderBar(ll &x, ll &y, ll &xUnit, ll &yUnit,
ll bar, bool orEqual){
    if(x < bar or (x == bar and orEqual))
        return;

    ll shift = (x - bar + xUnit - orEqual) / xUnit;
    x -= shift * xUnit;
    y += shift * yUnit;
}

void minXOverBar(ll &x, ll &y, ll &xUnit, ll &yUnit, ll
bar, bool orEqual){
    if(x < bar or (x == bar and !orEqual)){
        ll shift = (bar - x + xUnit - orEqual) / xUnit;
        x += shift * xUnit;
        y -= shift * yUnit;
    }
    else{
        ll shift = (x - bar - !orEqual) / xUnit;
        x -= shift * xUnit;
        y += shift * yUnit;
    }
}

void maxXUnderBar(ll &x, ll &y, ll &xUnit, ll &yUnit,
ll bar, bool orEqual){
    if(x < bar or (x == bar and orEqual)){
        ll shift = (bar - x - !orEqual) / xUnit;
        x += shift * xUnit;
        y -= shift * yUnit;
    }
    else{
        ll shift = (x - bar + xUnit - orEqual) / xUnit;
        x -= shift * xUnit;
        y += shift * yUnit;
    }
}
```

Primitive Root

```
// Ord(x) is the least positive number such that
x^ord(x) = 1 % n.
// Number of x with Ord(x) = y is Phi(y).
// all possible Ord(x) divide Phi(n).
// Ord(a^k) = Ord(a) / gcd(k,Ord(a))

int powmod (int a, int b, int p) {
    int res = 1;
    while (b)
        if (b & 1)
            res = int (res * 1ll * a % p),  --b;
        else
            a = int (a * 1ll * a % p),  b >>= 1;
    return res;
}

int generator (int p) {
    vector<int> fact;
    int phi = p-1,  n = phi;
    for (int i=2; i*i<=n; ++i)
        if (n % i == 0) {
            fact.push_back (i);
            while (n % i == 0)
                n /= i;
        }
    if (n > 1)
        fact.push_back (n);

    for (int res=2; res<=p; ++res) {
        bool ok = true;
        for (size_t i=0; i<fact.size() && ok; ++i)
            ok &= powmod (res, phi / fact[i], p) != 1;
        if (ok)  return res;
    }
    return -1;
}
```

Segmented Sieve

```
vector<char> segmentedSieve(long long L, long long R) {
    // generate all primes up to sqrt(R)
    long long lim = sqrt(R);
    vector<char> mark(lim + 1, false);
    vector<long long> primes;
    for (long long i = 2; i <= lim; ++i) {
        if (!mark[i]) {
            primes.emplace_back(i);
            for (long long j = i * i; j <= lim; j += i)
                mark[j] = true;
        }
    }

    vector<char> isPrime(R - L + 1, true);
    for (long long i : primes)
        for (long long j = max(i * i, (L + i - 1) / i *
i); j <= R; j += i)
            isPrime[j - L] = false;
    if (L == 1)
        isPrime[0] = false;
    return isPrime;
}
```

Primality Test

```
using u64 = uint64_t;
using u128 = __uint128_t;

u64 binpower(u64 base, u64 e, u64 mod) {
    u64 result = 1;
    base %= mod;
    while (e) {
        if (e & 1)
            result = (u128)result * base % mod;
        base = (u128)base * base % mod;
        e >>= 1;
    }
    return result;
}

bool check_composite(u64 n, u64 a, u64 d, int s) {
```

```
    u64 x = binpower(a, d, n);
    if (x == 1 || x == n - 1)
        return false;
    for (int r = 1; r < s; r++) {
        x = (u128)x * x % n;
        if (x == n - 1)
            return false;
    }
    return true;
};

bool MillerRabin(u64 n) { // returns true if n is
prime, else returns false.
    if (n < 2)
        return false;

    int r = 0;
    u64 d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        r++;
    }

    for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
31, 37}) {
        if (n == a)
            return true;
        if (check_composite(n, a, d, r))
            return false;
    }
    return true;
}
```

Lagrange

```
struct LagrangePoly {
    vector<long long> y, den;
    void build(vector<long long> _a){
        //f(i) = _a[i]
        //f(x) has degree of y.size() - 1
        y = _a;
        den.resize(y.size());
        int n = (int) y.size();
        for (int i = 0; i < n; i++) {
            y[i] = (y[i] % MOD + MOD) % MOD;
            den[i] = inv[n - i - 1] * inv[i] % MOD;
            if ((n - i - 1) % 2 == 1) {
                den[i] = (MOD - den[i]) % MOD;
            }
        }
    }

    ll getVal(ll x) {
        int n = (int) y.size();
        x %= MOD;
        if (x < n) {
            return y[(int) x];
        }
        //O(N^2)
        /*long long ans = 0;
        for(int i = 0; i < n; i++) {
            long long cur = den[i];
            for(int j = 0; j < n; j++) {
                if(i == j) { continue; }
                cur = cur * (x - y[i] + MOD) % MOD;
            }
            ans = (ans + cur) % MOD;
        }
        return ans;*/
        // O(N)
        std::vector<long long> l, r;
        l.resize(n);
        l[0] = 1;
        for (int i = 1; i < n; i++) {
            l[i] = l[i - 1] * (x - (i - 1) + MOD) %
MOD;
        }
        r.resize(n);
        r[n - 1] = 1;
        for (int i = n - 2; i >= 0; i--) {
            r[i] = r[i + 1] * (x - (i + 1) + MOD) %
MOD;
        }
        long long ans = 0;
        for (int i = 0; i < n; i++) {
            long long coef = l[i] * r[i] % MOD;
            ans = (ans + coef * y[i] % MOD * den[i]) %
MOD;
        }
        return ans;
    }

};
```

FFT

```
using cd = complex<double>;
const double PI = acos(-1);

void fft(vector<cd> & a, bool invert) {
    int n = a.size();

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

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

    for (int len = 2; len <= n; len <<= 1) {
        double ang = 2 * PI / len * (invert ? -1 : 1);
        cd wlen(cos(ang), sin(ang));
        for (int i = 0; i < n; i += len) {
            cd w(1);
            for (int 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<int> multiply(vector<int> const& a, vector<int>
const& b) {
    vector<cd> fa(a.begin(), a.end()), fb(b.begin(),
b.end());
    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
```

```
    fa.resize(n);
    fb.resize(n);

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

    vector<int> result(n);
    for (int i = 0; i < n; i++)
        result[i] = round(fa[i].real());
    return result;
}
```

Higher Percision FFT (FFTMOD)

```
#define rep(aa, bb, cc) for(int aa = bb; aa < cc;aa++)
#define sz(a) (int)a.size()
typedef complex<double> C;
typedef vector<double> vd;
void fft(vector<C>& a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vector<complex<long double>> R(2, 1);
    static vector<C> rt(2, 1);  // (^ 10% faster if
double)
    for (static int k = 2; k < n; k *= 2) {
        R.resize(n); rt.resize(n);
        auto x = polar(1.0L, acos(-1.0L) / k);
        rep(i,k,2*k) rt[i] = R[i] = i&1 ? R[i/2] * x :
R[i/2];
    }
    vi rev(n);
    rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) /
2;
    rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
                // C z = rt[j+k] * a[i+j+k]; // (25%
faster if hand-rolled)  /// include-line
                auto x = (double *)&rt[j+k], y =
(double *)&a[i+j+k];        /// exclude-line
                C z(x[0]*y[0] - x[1]*y[1], x[0]*y[1] +
x[1]*y[0]);           /// exclude-line
                a[i + j + k] = a[i + j] - z;
                a[i + j] += z;
            }
}

template<int M> vi convMod(const vi &a, const vi &b) {
    if (a.empty() || b.empty()) return {};
    vi res(sz(a) + sz(b) - 1);
    int B=32-__builtin_clz(sz(res)), n=1<<B,
cut=int(sqrt(M));
    vector<C> L(n), R(n), outs(n), outl(n);
    rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i]
% cut);
    rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i]
% cut);
    fft(L), fft(R);
    rep(i,0,n) {
        int j = -i & (n - 1);
        outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 *
n);
        outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 *
n) / 1i;
    }
    fft(outl), fft(outs);
    rep(i,0,sz(res)) {
        ll av = ll(real(outl[i])+.5), cv =
ll(imag(outs[i])+.5);
        ll bv = ll(imag(outl[i])+.5) +
ll(real(outs[i])+.5);
        res[i] = ((av % M * cut + bv) % M * cut + cv) %
M;
    }
    return res;
}
```

NTT

```
#define rep(aa, bb, cc) for(int aa = bb; aa < cc;aa++)
#define sz(a) (int)a.size()

const ll mod = (119 << 23) + 1, root = 62; // =
998244353
// For p < 2^30 there is also e.g. 5 << 25, 7 << 26,
479 << 21
// and 483 << 21 (same root). The last two are > 10^9.


ll modpow(ll b, ll e) {
    ll ans = 1;
    for (; e; b = b * b % mod, e /= 2)
        if (e & 1) ans = ans * b % mod;
    return ans;
}

// Primitive Root of the mod of form 2^a * b + 1
int generator () {
    vector<int> fact;
    int phi = mod-1,  n = phi;
    for (int i=2; i*i<=n; ++i)
        if (n % i == 0) {
            fact.push_back (i);
            while (n % i == 0)
                n /= i;
        }
    if (n > 1)
        fact.push_back (n);

    for (int res=2; res<=mod; ++res) {
        bool ok = true;
        for (size_t i=0; i<fact.size() && ok; ++i)
            ok &= modpow (res, phi / fact[i]) != 1;
        if (ok)  return res;
    }
    return -1;
}

typedef vector<ll> vl;
void ntt(vl &a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vl rt(2, 1);
    for (static int k = 2, s = 2; k < n; k *= 2, s++) {
        rt.resize(n);
        ll z[] = {1, modpow(root, mod >> s)};
        rep(i,k,2*k) rt[i] = rt[i / 2] * z[i & 1] %
mod;
    }
    vi rev(n);
    rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) /
2;
    rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
                ll z = rt[j + k] * a[i + j + k] % mod,
&ai = a[i + j];
```

```
                a[i + j + k] = ai - z + (z > ai ? mod :
0);
                ai += (ai + z >= mod ? z - mod : z);
            }
}
vl conv(const vl &a, const vl &b) {
    if (a.empty() || b.empty()) return {};
    int s = sz(a) + sz(b) - 1,
B = 32 - __builtin_clz(s),
n = 1 << B;
    int inv = modpow(n, mod - 2);
    vl L(a), R(b), out(n);
    L.resize(n), R.resize(n);
    ntt(L), ntt(R);
    rep(i,0,n) out[-i & (n - 1)] = (ll)L[i] * R[i] %
mod * inv % mod;
    ntt(out);
    return {out.begin(), out.begin() + s};
}
```

Fast Walsh-Hadamard Transform (FWHT)

```
#define rep(aa, bb, cc) for(int aa = bb; aa < cc;aa++)
#define sz(a) (int)a.size()
template<int MOD>
struct FWHT {
    int fast(int b, int e) {
        int res = 1;
        for(;e;e>>=1,b=1ll*b*b%MOD)
            if(e & 1)
                res = 1ll * res * b % MOD;
        return res;
    }
    inline int add(int x, int y) {
        return x + y - (x + y >= MOD? MOD : 0);
    }
    inline int sub(int x, int y) {
        return x - y + (x - y < 0? MOD : 0);
    }
    void FST(vi& a, bool inv) {
        for (int n = sz(a), step = 1; step < n; step *=
2) {
            for (int i = 0; i < n; i += 2 * step)
rep(j,i,i+step) {
                int &u = a[j], &v = a[j + step];
tie(u, v) =
                //  inv ? pii(sub(v,u), u) : pii(v,
add(u,v)); // AND
                //  inv ? pii(v, sub(u,v)) :
pii(add(u,v), u); // OR /// include-line
                    pair<ll,ll>(add(u,v),
sub(u,v));           // XOR /// include-line
            }
        }
         if (inv) {
            int divisor = fast(sz(a), MOD - 2);
            for (int& x : a) x = 1ll * x * divisor %
MOD; // XOR only /// include-line
         }
    }
    vi conv(vi a, vi b) {
        FST(a, 0); FST(b, 0);
        rep(i,0,sz(a)) a[i] = 1ll * a[i] * b[i] % MOD;
        FST(a, 1); return a;
    }
};
```

Notes

```
Sum of squares of first n numbers:
n*(n+1)*(2*n+1)/6
Sum of squares of first n even numbers:
2*n*(n+1)*(2*n+1)/3
Sum of squares of first n odd numbers:
n*(2*n+1)*(2*n-1)/3

Number of ways to pick equal number of elements from
two sets : (n+m)C(m)

Sum of phi(d) for all d | n is equal to n.
Number of pairs (x,y) that satisfy x+y=n and
gcd(x,y)=1 is phi(n).

Game Theory:
Game splits into multplie possibilities (take MEX)
Game has multiple subgames (take XOR)

Fi = Fibonacci of i
F1^2 + F2^2 + ... + Fn^2 = Fn*Fn+1
F1 + F2 + ... Fn = Fn+2 - 1
F(x + y) = F(x) * F(y + 1) + F(x - 1) * F(y) --- F[0] =
0, F[1] = 1

Number of labelled rooted forests (n + 1)^(n-1)

Number of labeled trees with given degree sequence with
size n
(n - 2)! / ((d1 - 1)! * (d2 - 1)! * (dn - 1)!)

Number of labeled graphs Gn = 2^(n*(n-1)/2)
Number of connected labeled graphs
Cn = Gn - 1/n * Sum(k * nCk * Ck * Gn-k) k = [1,n-1]

Number of labeled graphs with k components
D[n][k] = Sum(n-1Cs-1 * Cs * D[n-s][k-1]) s = [1,n]

Number of Derangements of size n F(n)
F(0) = 0
F(1) = 1
F(n) = (n-1) * (F(n-2) + F(n-1))
```

Combinatorics

nCr

```
const int N = 1e5 + 5;
const int MOD = 1e9 + 7;
ll fact[N], modInv[N];

ll fastExp(ll x, ll n)
{
    if(n == 0)
        return 1;
    ll u = fastExp(x, n/2);
    u = u * u % MOD;
    if(n & 1)
        u = u * x % MOD;
    return u;
}

// modInv[i] = fact[i]^-1 % MOD
void preprocess()
{
    fact[0] = 1;
```

```
    for(ll i =1; i<N; i++)
        fact[i] = fact[i-1] * i % MOD;

    modInv[N-1] = fastExp(fact[N-1], MOD - 2) % MOD;
    for(ll i=N-2; i>=0;i--)
        modInv[i] = (i+1) * modInv[i+1] % MOD;
}

ll modInvF(ll x)
{
    return fastExp(x, MOD - 2);
}

ll nCr(int n, int r)
{
    if(r > n)
        return 0;

    // return ( n! / ((n-r)! * r!) ) % MOD
    return (fact[n] * modInv[n-r] % MOD) * modInv[r] %
MOD;
}
```

nCr Recursive

```
ll nCr(int n, int r) {
    if (r > n)
        return 0;

    ll &ret = dp[n][r];
    if (~ret)
        return ret;
    if (r == 0)return ret = 1;
    if (r == 1)return ret = n;
    if (n == 1)return ret = 1;
    return ret = nCr(n - 1, r - 1) + nCr(n - 1, r);
}
```

Notes

Taking k items is the same as choosing n-k:

$$\binom{n}{k} = \binom{n}{n-k}$$

Factoring in:

$$\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$$

Choosing any number is equal to the Number of Subsets:

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n$$

Sum over n:

$$\sum_{m=0}^{n} \binom{m}{k} = \binom{n+1}{k+1}$$

Sum Over n and k:

$$\sum_{k=0}^{m} \binom{n+k}{k} = \binom{n+m+1}{m}$$

Sum of Squares:

$$\binom{n}{0}^2 + \binom{n}{1}^2 + \cdots + \binom{n}{n}^2 = \binom{2n}{n}$$

Weighted Sum:

$$1\binom{n}{1} + 2\binom{n}{2} + \cdots + n\binom{n}{n} = n2^{n-1}$$

Connection To Fib Numbers:

$$\binom{n}{0} + \binom{n-1}{1} + \cdots + \binom{n-k}{k} + \cdots + \binom{0}{n} = F_{n+1}$$

Stirling Numbers of the Second Kind are the number of partitions of n distinct elements into exactly k groups. See KACTL for formula.

Stirling Numbers of the First kind are the number of permutations on n items with k cycles. See KACTL for formula.

Linear Algebra

XOR Basis

```
const int LG = 60 + 1;
//basis[i] contains a basis whose highest bit is i
ll basis[LG];

void insert(ll x) {
    for (int b = LG - 1; b >= 0; --b) {
        //dimension is 0
        if (((1ll << b) & x) == 0)
            continue;

        //basis is not occupied, just put it here
        if (basis[b] == 0) {
            basis[b] = x;
            return;
        }

        //subtract this basis from x
        x ^= basis[b];
    }
}
```

Matrix Exponentiation

```
vector <vector<ll>> IDN;

vector <vector<ll>> mul(vector <vector<ll>> &v1, vector
<vector<ll>> &v2) {
    int n = v1.size(), m = v2[0].size();
    vector <vector<ll>> prod(n, vector<ll>(m));
    int iters = v1[0].size();
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            for (int k = 0; k < iters; ++k) {
                (prod[i][j] += v1[i][k] * v2[k][j] %
MOD) %= MOD;
            }
        }
    }
    return prod;
}

vector <vector<ll>> fastPowMats(vector <vector<ll>> &a,
int n) {
    if (n == 0)
        return IDN;
    vector <vector<ll>> res = fastPowMats(a, n / 2);
    res = mul(res, res);
    if (n & 1)
        res = mul(res, a);
    return res;
}
```

Faster Matrix Exponentiation

```
const int M = 2;

int mul(const ll &a,const ll&b){
    return (a % MOD + MOD) * (b % MOD + MOD) % MOD;
}

int add(const ll &a,const ll&b){
    return (a + b + 2 * MOD)%MOD;
}

typedef array<array<int,M>,M> matrix;

matrix operator*(const matrix &lhs, const matrix &rhs)
{
    matrix ret{};
    for (int i = 0; i < M; ++i)
        for (int j = 0; j < M; ++j)
            for (int k = 0; k < M; ++k)
                ret[i][k] = add(ret[i][k],
mul(lhs[i][j],rhs[j][k]));
    return ret;
}

matrix Identity(int n) {
    matrix ret={};
    for (int i = 0; i < n; ++i) {
        ret[i][i] = 1;
    }
    return ret;
}

matrix mat_power(matrix x, ll p) {
    matrix res = Identity(x.size());
    while (p) {
        if (p & 1) res = (res * x);
        x = (x * x);
        p >>= 1;
    }
    return res;
}
```

Gauss

```
ll pw(ll b , ll p,ll MOD)
{
    if (!p)
        return 1;
    ll ans = pw(b, p / 2,MOD);
    ans = (ans * ans) % MOD;
    if (p%2) ans = (ans * b) % MOD;
    return ans;
}
ll inv(ll x, ll MOD) { return pw(x, MOD - 2,MOD); }

vector<ll> gauss(vector<vector<ll> > &a, ll MOD)
{
    int n = a.size(), m = a[0].size() - 1;

    for(int i = 0; i < n; i++)
        for(int j = 0; j <= m; j++)
            a[i][j] = (a[i][j] % MOD + MOD) % MOD;

    vector<int> where(m, -1);
    for(int col = 0, row = 0; col < m && row < n;
col++)
    {
        int sel = row;
        for(int i = row; i < n; i++)
            if(a[i][col] > a[sel][col])
                sel = i;

        if(a[sel][col] == 0) { where[col] = -1;
continue;
        }

        for(int i = col; i <= m; i++)
            swap(a[sel][i], a[row][i]);
        where[col] = row;

        ll c_inv = inv(a[row][col], MOD);
        for(int i = 0; i < n; i++)
            if(i != row)
            {
                if(a[i][col] == 0) continue;
                ll c = (a[i][col] * c_inv) % MOD;
                for(int j = 0; j <= m; j++)
                    a[i][j] = (a[i][j] - c * a[row][j]
% MOD + MOD) % MOD;
            }

        row++;
    }
    vector<ll> ans(m, 0);
    ll ways = 1;

    for(int i = 0; i < m; i++)
        if(where[i] != -1) ans[i] = (a[where[i]][m] *
inv(a[where[i]][i], MOD)) % MOD;
        else ways = (ways * MOD) % MOD;

    for(int i = 0; i < n; i++)
    {
        ll sum = a[i][m] % MOD;
        for(int j = 0; j < m; j++)
            sum = (sum + MOD - (ans[j] * a[i][j]) %
MOD) % MOD;
if(sum != 0) return {}; //Has No Sol}
return ans;}
```

Data Structures

BIT

```
template<typename T>
class FenwickTree {
public:
    vector<T> tree;
    int n;

    void init(int n) {
        tree.assign(n + 2, 0);
        this->n = n;
    }

    T merge(T &x, T &y) { return x + y; }

    void update(int x, T val) {
        for (; x <= n; x += x & -x) {
            tree[x] = merge(tree[x], val);
        }
    }

    T getPrefix(int x) {
        if (x <= 0)return 0;
        T ret = 0;
        for (; x; x -= x & -x) {
```

```
            ret = merge(ret, tree[x]);
        }
        return ret;
    }

    T getRange(int l, int r) {
        return getPrefix(r) - getPrefix(l - 1);
    }

    int lowerBound(ll x) {
        int pos = 0;
        for (int sz = (1 << __lg(n)); sz > 0 && x; sz
>>= 1) {
            if (pos + sz <= n && tree[pos + sz] < x) {
                x -= tree[pos + sz];
                pos += sz;
            }
        }
        return pos + 1;
    }
};
```

2D BIT

```
template<typename T>
class FenwickTree2D {
public:
    vector<vector<T>> tree;
    int n, m;

    void init(int n, int m) {
        tree.assign(n + 2, vector<T>(m + 2, 0));
        this->n = n;
        this->m = m;
    }

    T merge(T &x, T &y) { return x + y; }

    void update(int x, int y, T val) {
        for (; x <= n; x += x & -x) {
            for (int z = y; z <= m; z += z & -z) {
                tree[x][z] = merge(tree[x][z], val);
            }
        }
    }

    T getPrefix(int x,int y){
        if(x <= 0)return 0;
        T ret = 0;
        for (; x ; x-=x&-x) {
            for (int z = y; z ; z-=z&-z) {
                ret = merge(ret,tree[x][z]);
            }
        }
        return ret;
    }

    T getSquare(int xl,int yl,int xr,int yr){
        return getPrefix(xr,yr) + getPrefix(xl - 1,yl -
1) -
               getPrefix(xr,yl - 1) - getPrefix(xl -
1,yr);
    }

};
```

DSU

// 0-based

```
struct DSU {
    vector<int> par, sz;

    DSU(int n) : par(n), sz(n, 1) { iota(par.begin(),
par.end(), 0); }

    int find(int x) {
        if(x == par[x])return x;
        return par[x] = find(par[x]);
    }

    bool same(int x, int y) { return find(x) ==
find(y); }

    bool join(int x, int y) {
        x = find(x);
        y = find(y);
        if (x == y) return false;
        if (sz[x] < sz[y])
            swap(x, y);
        sz[x] += sz[y];
        par[y] = x;
        return true;
    }
 int size(int x) { return sz[find(x)]; }
};
```

Bipartite DSU

// Maintains whether each component is bipartite

```
struct BipartiteDSU {
    vector<int> sz,bipartite;
    vector<pair<int, int>>par;

    BipartiteDSU(int n) : par(n), sz(n, 1),bipartite(n)
{
        for (int i = 0; i < n; ++i) {
            par[i] = {i,0};
        }
    }

    pair<int, int> find(int u) {
        if (u == par[u].fi)return {u, 0};
        int parity = par[u].se;
        par[u] = find(par[u].first);
        par[u].se ^= parity;
        return par[u];
    }

    bool same(int x, int y) { return find(x).first ==
find(y).first; }

    bool join(int u, int v) {
        pair<int,int>pu = find(u);
        pair<int,int>pv = find(v);
        u = pu.first;
        v = pv.first;
        int x = pu.second,y = pv.second;
        if (u == v) {
            if(x == y)
                bipartite[u] = false;
            return false;
        }
        if (sz[u] < sz[v])
            swap(u, v);
        par[v] = {u, x ^ y ^ 1};
        bipartite[u] &= bipartite[v];
        sz[u] += sz[v];
        return true;
    }
```

```
    int size(int x) { return sz[find(x).first]; }
};
```

Rollback DSU

```
struct RollbackDSU {
    vector<int> par; vector<pair<int,int>> st;
    int comps;
    RollbackDSU(int n) : par(n, -1), comps(n) {}
    int size(int x) { return -par[find(x)]; }
    int find(int x) { return par[x] < 0 ? x :
find(par[x]); }
    int time() { return st.size(); }
    void rollback(int t) {
        comps += (time() - t)/2;
        for (int i = time(); i-- > t;)
            par[st[i].first] = st[i].second;
        st.resize(t);
    }
    // a : leader par[a] = -sz(a)
    // a : not par[a] = leader(a)
    bool join(int a, int b) {
        a = find(a), b = find(b);
        if (a == b) return false;
        if (-par[a] < -par[b]) swap(a, b);
        st.emplace_back(a, par[a]);
        st.emplace_back(b, par[b]);
        par[a] += par[b]; par[b] = a;
        return true;
    }
};
```

Sparse Table

```
#define sz(aa) (int)aa.size()
template<class T>
struct sparseTable {
    vector<vector<T>> jmp;
    void build(const vector<T>& V){
        jmp.resize(1,V);
        for (int pw = 1, k = 1; pw * 2 <= sz(V); pw *=
2, ++k) {
            jmp.emplace_back(sz(V) - pw * 2 + 1);
            for (int j = 0; j < sz(jmp[k]); ++j) {
                jmp[k][j] = max(jmp[k - 1][j], jmp[k -
1][j + pw]);
            }
        }
    }
    T query(int l, int r) {
        assert(l <= r);
        int dep = 31 - __builtin_clz(r - l + 1);
        return max(jmp[dep][l], jmp[dep][r - (1 << dep)
+ 1]);}
};
```

MonoQueue

```
template<class T>
struct Mono_stack{
    stack<pair<T,T>>st;
    void push(const T& val){
        if(st.empty())
            st.emplace(val,val);
        else
st.emplace(val,std::max(val,st.top().second));
    }
    void pop(){
        st.pop();
    }
    bool empty(){
        return st.empty();
    }
    int size(){
        return st.size();
    }
    T top(){
        return st.top().first;
    }
    T max(){
        return st.top().second;
    }
};
template<class T>
struct Mono_queue{
    Mono_stack<T>pop_st,push_st;
    void push(const T& val){
        push_st.push(val);
    }
    void move(){
        if(pop_st.size())
            return;
        while(!push_st.empty())
            pop_st.push(push_st.top()),push_st.pop();
    }
    void pop(){
        move();
        pop_st.pop();
    }
    bool empty(){
        return pop_st.empty()&&push_st.empty();
    }
    int size(){
        return pop_st.size()+push_st.size();
    }
    T top(){
        move();
        return pop_st.top();
    }
    T max(){
        if(pop_st.empty())
            return push_st.max();
        if(push_st.empty())
            return pop_st.max();
        return std::max(push_st.max(),pop_st.max());
    }

};
```

Seg Tree

```
struct SegTree {
    vector<ll> tree;
    int n;
    const ll IDN = OO;

    ll combine(ll a, ll b) {
        return min(a, b);
    }

    void build(int inputN, vector<ll>& a) {
        n = inputN;
        if (__builtin_popcount(n) != 1)
            n = 1 << (__lg(n) + 1);
        tree.resize(n << 1, IDN);
        for (int i = 0; i < inputN; i++)
            tree[i + n] = a[i];
        for (int i = n - 1; i >= 1; i--)
```

```
            tree[i] = combine(tree[i << 1], tree[i << 1
| 1]);
    }

    void update(int ql, int qr, ll v, int k, int sl,
int sr) {
        if (qr < sl || sr < ql || ql > qr) return;
        if (ql <= sl && qr >= sr) {
            tree[k] = v;
            return;
        }

        int mid = (sl + sr) / 2;
        update(ql, qr, v, k << 1, sl, mid);
        update(ql, qr, v, (k << 1) | 1, mid + 1, sr);
        tree[k] = combine(tree[k << 1], tree[k << 1 |
1]);
    }

    ll query(int ql, int qr, int k, int sl, int sr) {
        if (qr < sl || sr < ql || ql > qr) return IDN;
        if (ql <= sl && qr >= sr) return tree[k];

        int mid = (sl + sr) / 2;
        ll left = query(ql, qr, k << 1, sl, mid);
        ll right = query(ql, qr, k << 1 | 1, mid + 1,
sr);
        return combine(left, right);
    }

    void update(int ql, int qr, ll v){
        update(ql, qr, v, 1, 0, n-1);
    }

    ll query(int ql, int qr){
        return query(ql, qr, 1, 0, n-1);
    }
};
```

Seg Tree Lazy

```
struct SegTree {
    vector <ll> tree;
    vector <ll> lazy;
    int n;
    const ll IDN = OO;
    const ll LAZY_IDN = 0;

    ll combine(ll a, ll b) {
        return min(a, b);
    }

    void build(int inputN, const vector<ll>& a) {
        n = inputN;
        if (__builtin_popcount(n) != 1)
            n = 1 << (__lg(n) + 1);
        tree.resize(n << 1, IDN);
        lazy.resize(n << 1, LAZY_IDN);
        for (int i = 0; i < inputN; i++)
            tree[i + n] = a[i];
        for (int i = n - 1; i >= 1; i--)
            tree[i] = combine(tree[i << 1], tree[i << 1
| 1]);
    }

    void propagate(int k, int sl, int sr) {
        if (lazy[k] != LAZY_IDN) {
            tree[k] += lazy[k];
            if (sl != sr) {
                lazy[k << 1] += lazy[k];
                lazy[k << 1 | 1] += lazy[k];
            }
        }
        lazy[k] = LAZY_IDN;
    }

    void update(int ql, int qr, ll v, int k, int sl,
int sr) {
        propagate(k, sl, sr);
        if (qr < sl || sr < ql || ql > qr) return;
        if (ql <= sl && qr >= sr) {
            lazy[k] = v;
            propagate(k, sl, sr);
            return;
        }

        int mid = (sl + sr) / 2;
        update(ql, qr, v, k << 1, sl, mid);
        update(ql, qr, v, (k << 1) | 1, mid + 1, sr);
        tree[k] = combine(tree[k << 1], tree[k << 1 |
1]);
    }

    ll query(int ql, int qr, int k, int sl, int sr) {
        propagate(k, sl, sr);
        if (qr < sl || sr < ql || ql > qr) return IDN;
        if (ql <= sl && qr >= sr) return tree[k];

        int mid = (sl + sr) / 2;
        ll left = query(ql, qr, k << 1, sl, mid);
        ll right = query(ql, qr, k << 1 | 1, mid + 1,
sr);
        return combine(left, right);
    }

    void update(int ql, int qr, ll v){
        update(ql, qr, v, 1, 0, n-1);
    }

    ll query(int ql, int qr){
        return query(ql, qr, 1, 0, n-1);
    }
};
```

Persistent Segment Tree

```
struct Vertex {
    Vertex *l, *r;
    int sum = 0;

    Vertex(int val) : l(nullptr), r(nullptr), sum(val)
{}
    Vertex() : l(nullptr), r(nullptr) {}

    Vertex(Vertex *l, Vertex *r) : l(l), r(r), sum(0) {
        if (l) sum += l->sum;
        if (r) sum += r->sum;
    }
    void addChild(){
        l = new Vertex();
        r = new Vertex();
    }
};
struct Seg {
    int n;

    Seg(int n) {
        this->n = n;
    }
```

```
    Vertex merge(Vertex x, Vertex y) {
        Vertex ret;
        ret.sum = x.sum + y.sum;
        return ret;
    }


    Vertex *update(Vertex *v, int i, int lx, int rx) {
        if (lx == rx)
            return new Vertex(v->sum + 1);
        int mid = (lx + rx) / 2;
        if(!v->l)v->addChild();
        if (i <= mid) {
            return new Vertex(update(v->l, i, lx, mid),
v->r);
        } else {
            return new Vertex(v->l, update(v->r, i, mid
+ 1, rx));
        }
    }

    Vertex *update(Vertex *v, int i) {
        return update(v, i, 0, n - 1);
    }

    Vertex query(Vertex *v, int l, int r, int lx, int
rx) {
        if (l > rx || r < lx)
            return {};
        if (l <= lx && r >= rx)
            return *v;
        if(!v->l)v->addChild();
        int mid = (lx + rx) / 2;
        return merge(query(v->l, l, r, lx, mid),
query(v->r, l, r, mid + 1, rx));
    }

    Vertex query(Vertex *v, int l, int r) {
        return query(v, l, r, 0, n - 1);
    }

    int getKth(Vertex *a, Vertex *b, int k, int lx, int
rx) {
        if (lx == rx) {
            return lx;
        }
        if(!a->l)a->addChild();
        if(!b->l)b->addChild();
        int rem = b->l->sum - a->l->sum;
        int mid = (lx + rx) / 2;
        if (rem >= k)
            return getKth(a->l, b->l, k, lx, mid);
        else
            return getKth(a->r, b->r, k - rem, mid + 1,
rx);
    }

    int getKth(Vertex *a, Vertex *b, int k) {
        return getKth(a, b, k, 0, n - 1);
    }
}
```

Dynamic Li-Chao Tree

```
const ll OO = 1e18 + 5;
const ll maxN = 1e6 + 5;

struct Line {
    ll m, c;
    Line() : m(0), c(OO) {}

    Line(ll m, ll c) : m(m), c(c) {}
};

ll sub(ll x, Line l) {
    return x * l.m + l.c;
}

// Li Chao sparse
struct node {
    // range I am responsible for
    Line line;
    node *left, *right;

    node() {
        left = right = NULL;
    }

    node(ll m, ll c) {
        line = Line(m, c);
        left = right = NULL;
    }

    void extend(int l, int r) {
        if (left == NULL && l != r) {
            left = new node();
            right = new node();
        }
    }

    void add(Line toAdd, int l, int r) {
        assert(l <= r);
        int mid = (l + r) / 2;
        if (l == r) {
            if (sub(l, toAdd) < sub(l, line))
                swap(toAdd, line);
            return;
        }
        bool lef = sub(l, toAdd) < sub(l, line);
        bool midE = sub(mid+1, toAdd) < sub(mid+1,
line);
        if(midE)
            swap(line, toAdd);
        extend(l, r);
        if(lef != midE)
            left->add(toAdd, l, mid);
        else
            right->add(toAdd, mid+1, r);
    }

    void add(Line toAdd) {
        add(toAdd, 0, maxN-1);
    }

    ll query(ll x, int l, int r) {
        int mid = (l + r) / 2;
        if (l == r || left == NULL)
            return sub(x, line);
        extend(l, r);
        if (x <= mid)
            return min(sub(x, line), left->query(x, l,
mid));
        else
            return min(sub(x, line), right->query(x,
mid+1, r));
    }

    ll query(ll x) {
```

```
        return query(x, 0, maxN-1);
    }

    void clear() {
        if (left != NULL) {
            left->clear();
            right->clear();
        }
        delete this;
    }
};
```

Dynamic Persistent Li-Chao Tree

```
// Not well tested
const ll OO = 1e18 + 5;
const ll maxN = 1e9 + 5;
struct Line {
    ll m, c;
    Line() : m(0), c(OO) {}
    Line(ll m, ll c) : m(m), c(c) {}
};

ll sub(ll x, Line l) {
    return x * l.m + l.c;
}

// Persistent Li Chao
struct Node {
    // range I am responsible for
    Line line;
    Node *left, *right;

    Node() {
        left = right = NULL;
    }

    Node(ll m, ll c) {
        line = Line(m, c);
        left = right = NULL;
    }

    void extend(int l, int r) {
        if (left == NULL && l != r) {
            left = new Node();
            right = new Node();
        }
    }

    Node* copy(Node* node){
        Node* newNode = new Node;
        newNode->left = node->left;
        newNode->right = node->right;
        newNode->line = node->line;
        return newNode;
    }

    Node* add(Line toAdd, int l, int r) {
        assert(l <= r);
        int mid = (l + r) / 2;
        Node* cur = copy(this);
        if (l == r) {
            if (sub(l, toAdd) < sub(l, cur->line))
                swap(toAdd, cur->line);
            return cur;
        }
        bool lef = sub(l, toAdd) < sub(l, cur->line);
        bool midE = sub(mid+1, toAdd) < sub(mid+1, cur-
>line);
        if(midE)
            swap(cur->line, toAdd);
        cur->extend(l, r);
        if(lef != midE)
            cur->left = cur->left->add(toAdd, l, mid);
        else
            cur->right = cur->right->add(toAdd, mid+1,
r);
        return cur;
    }

    Node* add(Line toAdd) {
        return add(toAdd, 0, maxN-1);
    }

    ll query(ll x, int l, int r) {
        int mid = (l + r) / 2;
        if (l == r || left == NULL)
            return sub(x, line);
        extend(l, r);
        if (x <= mid)
            return min(sub(x, line), left->query(x, l,
mid));
        else
            return min(sub(x, line), right->query(x,
mid+1, r));
    }

    ll query(ll x) {
        return query(x, 0, maxN-1);
    }

    void clear() {
        if (left != NULL) {
            left->clear();
            right->clear();
        }
        delete this;
    }
};
Node* tree[N];
```

General Binary Walk on SegTree

```
//query leftmost element not less than v
int binWalk(int ql, int qr, int v, int k = 1, int sl =
0, int sr = n - 1) {
    propagate(k, sl, sr);
    if (qr < sl || sr < ql)
        return -1;
    int mid = (sl + sr) / 2;
    if (ql <= sl && qr >= sr) {
        if (sl == sr)
            return tree[k] >= v ? sl : -1;

        propagate(k << 1, sl, mid);
        propagate(k << 1 | 1, mid + 1, sr);
        if (tree[k << 1] >= v)
            return binWalk(ql, qr, v, k << 1, sl, mid);
        if (tree[k << 1 | 1] >= v)
            return binWalk(ql, qr, v, k << 1 | 1, mid +
1, sr);
        return -1;
    }

    int left = binWalk(ql, qr, v, k << 1, sl, mid);
    if (left != -1)
        return left;
    int right = binWalk(ql, qr, v, k << 1 | 1, mid + 1,
sr);
    if (right != -1)
```

```
        return right;
    return -1;
}
```

Treap

```
template <typename T, class Allocator =
std::allocator<T> >
class treap {
private:
    struct node;
    using pnode = struct node *;
    using node_allocator_t = typename
std::allocator_traits<Allocator>::template
rebind_alloc<node>;

    std::mt19937_64 * rng_;
    node_allocator_t node_allocator_;
    bool rng_owner_;
    bool is_sorted_;
    bool stop_; // for priority regeneration
    pnode root_;

    using priority_t = std::mt19937_64::result_type;

    priority_t next_priority () {
        priority_t priority = (*rng_)();
        return priority;
    }

    void regenerate_priorities_recursive (std::vector
<int> & new_priors, pnode & t, int l, int r) {
        if (!t)
            return;
        t->priority = new_priors[r - 1];
        regenerate_priorities_recursive(new_priors, t-
>l, l, l + cnt(t->l));
        regenerate_priorities_recursive(new_priors, t-
>r, l + cnt(t->l), r - 1);
    }

    void regenerate_priorities () {
        int sz = size();
        std::vector <int> new_priors(sz);
        for (int i = 0; i < sz; i++)
            new_priors[i] = next_priority();
        std::sort(new_priors.begin(),
new_priors.end());
        for (int i = 0; i < sz; i++)
            new_priors[i] += i;
        regenerate_priorities_recursive(new_priors,
root_, 0, sz);
    }

    struct node {
        priority_t priority;
        int cnt, rev;
        T key, add, fsum;
        pnode l, r;

        node (T x, priority_t p) {
            add = 0 * x;
            key = fsum = x;
            cnt = 1;
            rev = 0;
            l = r = nullptr;
            priority = p;
        }
    };

    pnode create_node(T x) {
        auto place = node_allocator_.allocate(1);

std::allocator_traits<node_allocator_t>::construct(node
_allocator_, place, x, next_priority());
        return place;
    }

    void destroy_node(pnode t) {

std::allocator_traits<node_allocator_t>::destroy(node_a
llocator_, t);
        node_allocator_.deallocate(t, 1);
    }

    int cnt (pnode t) {
        return t ? t->cnt : 0;
    }

    void upd_cnt (pnode t) {
        if (t)
            t->cnt = cnt(t->l) + cnt(t->r) + 1;
    }

    void upd_sum (pnode t) {
        if (t) {
            t->fsum = t->key;
            if (t->l)
                t->fsum += t->l->fsum;
            if (t->r)
                t->fsum += t->r->fsum;
        }
    }

    void update (pnode t, T add, int rev) {
        if (!t)
            return;
        t->add = t->add + add;
        t->rev = t->rev ^ rev;
        t->key = t->key + add;
        t->fsum = t->fsum + cnt(t) * add;
    }

    void push (pnode t) {
        if (!t || (t->add == 0 * T() && t->rev == 0))
            return;
        update(t->l, t->add, t->rev);
        update(t->r, t->add, t->rev);
        if (t->rev)
            std::swap(t->l, t->r);
        t->add = 0 * T();
        t->rev = 0;
    }

    void merge (pnode & t, pnode l, pnode r) {
        push(l);
        push(r);
        if (!l || !r)
            t = l ? l : r;
        else if (l->priority > r->priority) {
            merge(l->r, l->r, r);
            t = l;
        }
        else {
            merge(r->l, l, r->l);
            t = r;
        }
        upd_cnt(t);
        upd_sum(t);
```

```
    }

    void split (pnode t, pnode & l, pnode & r, int
index) { // split at position
        if (!t) {
            l = r = 0;
            return;
        }
        push(t);
        if (index <= cnt(t->l)) {
            split(t->l, l, t->l, index);
            r = t;
        }
        else {
            split(t->r, t->r, r, index - 1 - cnt(t-
>l));
            l = t;
        }
        upd_cnt(t);
        upd_sum(t);
    }

    void split_at (pnode t, pnode & l, pnode & r, T &
key, bool & eq) { // split by key
        if (!t) {
            l = r = 0;
            return;
        }
        push(t);
        if (key == t->key) {
            eq = true;
            return;
        }
        if (key < t->key) {
            split_at(t->l, l, t->l, key, eq);
            if (!eq)
                r = t;
        }
        else {
            split_at(t->r, t->r, r, key, eq);
            if (!eq)
                l = t;
        }
        if (!eq)
            upd_cnt(t);
        upd_sum(t);
    }

    void insert (pnode & t, pnode it, int index) { //
insert at position
        push(t);
        if (!t)
            t = it;
        else if (it->priority == t->priority) {
            stop_ = true;
            regenerate_priorities();
        }
        else if (it->priority > t->priority) {
            split(t, it->l, it->r, index);
            t = it;
        }
        else if (index <= cnt(t->l))
            insert(t->l, it, index);
        else
            insert(t->r, it, index - cnt(t->l) - 1);
        if (stop_)
            return;
        upd_cnt(t);
        upd_sum(t);
    }

    void insert_at (pnode & t, pnode it, bool & eq) {
// insert by key
        push(t);
        if (!t)
            t = it;
        else if (it->key == t->key) {
            eq = true;
            return;
        }
        else if (it->priority == t->priority) {
            stop_ = true;
            regenerate_priorities();
        }
        else if (it->priority > t->priority) {
            split_at(t, it->l, it->r, it->key, eq);
            if (!eq)
                t = it;
        }
        else if (it->key < t->key)
            insert_at(t->l, it, eq);
        else
            insert_at(t->r, it, eq);
        if (stop_)
            return;
        if (!eq)
            upd_cnt(t);
        upd_sum(t);
    }

    void erase (pnode & t, int index) {
        push(t);
        if (cnt(t->l) == index) {
            pnode l = t->l, r = t->r;
            destroy_node(t);
            t = nullptr;
            merge(t, l, r);
        }
        else if (index < cnt(t->l))
            erase(t->l, index);
        else
            erase(t->r, index - cnt(t->l) - 1);
        upd_cnt(t);
        upd_sum(t);
    }

    void erase_at (pnode & t, T key, bool & found) {
        push(t);
        if (key == t->key) {
            pnode l = t->l, r = t->r;
            destroy_node(t);
            t = nullptr;
            merge(t, l, r);
            found = true;
        }
        else if (key < t->key)
            erase_at(t->l, key, found);
        else
            erase_at(t->r, key, found);
        upd_cnt(t);
        upd_sum(t);
    }

    T get (pnode t, int index) {
        push(t);
        if (index < cnt(t->l))
            return get(t->l, index);
        else if (index > cnt(t->l))
```

```
            return get(t->r, index - cnt(t->l) - 1);
        return t->key;
    }

    int find (pnode t, T key) {
        push(t);
        if (!t || key == t->key)
            return cnt(t->l);
        if (key < t->key)
            return get(t->l, key);
        else
            return get(t->r, key) + 1 + cnt(t->l);
    }

    std::pair <T, int> lower_bound (pnode t, T key, int
index) {
        push(t);
        if (!t)
            return {T(), size()};
        if (key == t->key)
            return {key, index + cnt(t->l)};
        if (key < t->key) {
            std::pair <T, int> ret = lower_bound(t->l,
key, index);
            if (ret.second == size())
                ret = {t->key, index + cnt(t->l)};
            return ret;
        }
        return lower_bound(t->r, key, index + 1 +
cnt(t->l));
    }

    std::pair <T, int> upper_bound (pnode t, T key, int
index) {
        push(t);
        if (!t)
            return {T(), size()};
        if (key < t->key) {
            std::pair <T, int> ret = upper_bound(t->l,
key, index);
            if (ret.second == size())
                ret = {t->key, index + cnt(t->l)};
            return ret;
        }
        return upper_bound(t->r, key, index + 1 +
cnt(t->l));
    }

    void shift (pnode & t, int l, int r, T add) {
        pnode l1, r1;
        split(t, l1, r1, r + 1);
        pnode l2, r2;
        split(l1, l2, r2, l);
        update(r2, add, 0);
        pnode t2;
        merge(t2, l2, r2);
        merge(t, t2, r1);
    }

    void reverse (pnode & t, int l, int r) {
        pnode l1, r1;
        split(t, l1, r1, r + 1);
        pnode l2, r2;
        split(l1, l2, r2, l);
        update(r2, 0 * T(), 1);
        pnode t2;
        merge(t2, l2, r2);
        merge(t, t2, r1);
    }

    void move (pnode & t, int left, int right, int
shift) {
        // [l, r) becomes [l+shift, r+shift)
        if (shift == 0)
            return;
        int l = left + std::min(0, shift);
        int r = right + std::max(0, shift);
        int m = (shift > 0) ? right : left;
        pnode prefix, tmp;
        split(root_, prefix, tmp, l);
        pnode suffix, middle;
        split(tmp, middle, suffix, r - l);
        pnode mid_prefix, mid_suffix;
        split(middle, mid_prefix, mid_suffix, m - l);
        merge(middle, mid_suffix, mid_prefix);
        merge(tmp, middle, suffix);
        merge(root_, prefix, tmp);
    }

    T get_sum (pnode & t, int l, int r) {
        pnode l1, r1;
        split(t, l1, r1, r + 1);
        pnode l2, r2;
        split(l1, l2, r2, l);
        T ret = r2->fsum;
        pnode t2;
        merge(t2, l2, r2);
        merge(t, t2, r1);
        return ret;
    }

    void clear (pnode & t) {
        if (!t)
            return;
        clear(t->l);
        clear(t->r);
        destroy_node(t);
        t = nullptr;
    }

public:
    treap (std::mt19937_64 * rng = nullptr) {
        is_sorted_ = true;
        stop_ = false;
        root_ = nullptr;
        if (rng) {
            rng_owner_ = false;
            rng_ = rng;
        }
        else {
            rng_owner_ = true;
            rng_ = new std::mt19937_64;
            rng_-
>seed(std::chrono::steady_clock::now().time_since_epoch
().count());
        }
    }

    ~treap () {
        if (rng_owner_)
            delete rng_;
        clear(root_);
    }

    int size () { return cnt(root_); }

    bool empty () { return (cnt(root_) == 0); }
```

```
bool is_sorted () { return is_sorted_; }

void srand (std::mt19937_64::result_type seed) {
    // optional
    rng_->seed(seed);
}

bool insert (T x) {
    bool eq = false;
    pnode t = create_node(x);
    stop_ = false;
    insert_at(root_, t, eq);
    while (stop_) {
        stop_ = false;
        eq = false;
        insert_at(root_, t, eq);
    }
    if (eq)
        destroy_node(t);
    return !eq;
}

void insert_at (int pos, T x) {
    if (pos > size())
        return;
    pnode t = create_node(x);
    stop_ = false;
    insert(root_, t, pos);
    while (stop_) {
        stop_ = false;
        insert(root_, t, pos);
    }
    if (pos > 0 && is_sorted_) {
        if (get(root_, pos - 1) >= get(root_, pos))
            is_sorted_ = false;
    }
    if (pos < size() - 1 && is_sorted_) {
        if (get(root_, pos) >= get(root_, pos + 1))
            is_sorted_ = false;
    }
}

bool erase (T x) {
    bool found = false;
    erase_at(root_, x, found);
    return found;
}

void erase_at (int pos) {
    if (pos >= size())
        return;
    erase(root_, pos);
}

void clear () {
    clear(root_);
}

int get_index (T key) {
    if (!is_sorted_)
        return size();
    pnode t = root_;
    int index = 0;
    while (t && t->key != key) {
        if (t->key > key)
            t = t->l;
        else {
            index += cnt(t->l) + 1;
            t = t->r;
        }
    }
    if (!t)
        return size();
    index += cnt(t->l);
    return index;
}

T operator[] (int index) {
    return get(root_, index);
}

std::pair <T, int> lower_bound (T x) {
    if (!is_sorted_)
        return {T(), size()};
    return lower_bound(root_, x, 0);
}

std::pair <T, int> upper_bound (T x) {
    if (!is_sorted_)
        return {T(), size()};
    return upper_bound(root_, x, 0);
}

void shift (int left, int right, T x) {
    left = std::max(left, 0);
    right = std::min(right, size() - 1);
    shift(root_, left, right, x);
    if (left > 0 && is_sorted_) {
        if (get(root_, left - 1) >= get(root_,
left))
            is_sorted_ = false;
    }
    if (right < size() - 1 && is_sorted_) {
        if (get(root_, right) >= get(root_, right +
1))
            is_sorted_ = false;
    }
}

void reverse (int left, int right) {
    left = std::max(left, 0);
    right = std::min(right, size() - 1);
    reverse(root_, left, right);
    if (left != right)
        is_sorted_ = false;
}

void move (int left, int right, int shift) {
    move(root_, left, right, shift);
}

T get_sum (int left, int right) {
    return get_sum(root_, left, right);
}
};
```

Graph

Bellman Ford

```
void solve()
{
    vector<int> d(n, INF);
    d[v] = 0;
    vector<int> p(n, -1);
    int x;
    for (int i = 0; i < n; ++i) {
```

```
        x = -1;
        for (Edge e : edges)
            if (d[e.a] < INF)
                if (d[e.b] > d[e.a] + e.cost) {
                    d[e.b] = max(-INF, d[e.a] +
e.cost);
                    p[e.b] = e.a;
                    x = e.b;
                }
    }

    if (x == -1)
        cout << "No negative cycle from " << v;
    else {
        int y = x;
        for (int i = 0; i < n; ++i)
            y = p[y];

        vector<int> path;
        for (int cur = y;; cur = p[cur]) {
            path.push_back(cur);
            if (cur == y && path.size() > 1)
                break;
        }
        reverse(path.begin(), path.end());

        cout << "Negative cycle: ";
        for (int u : path)
            cout << u << ' ';
    }
}
```

Dijkstra

```
const ll OO = 1e18;
const int N = 1e5 + 5;

vector<pair<int, ll>> adj[N];
ll dist[N];
int n, m;

void dijkstra(int src) {
    for (int i = 1; i <= n; i++)
        dist[i] = OO;

    priority_queue<pair<ll, int>, vector<pair<ll,
int>>, greater<pair<ll,int>>> pq;
    dist[src] = 0;
    pq.push({0, src});
    while(!pq.empty()){
        int u;ll w;
        tie(w, u) = pq.top();
        pq.pop();
        if(dist[u] < w)
            continue;

        for(auto e:adj[u]){
            if(dist[u] + e.S < dist[e.F]){
                dist[e.F] = dist[u] + e.S;
                pq.push({dist[e.F], e.F});
            }
        }
    }
}
```

Floyd Warshall

```
for (int k = 0; k < n; ++k) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (d[i][k] < INF && d[k][j] < INF)
                d[i][j] = min(d[i][j], d[i][k] +
d[k][j]);
        }
    }
}

// can check for every pair if there is infinite path

// d[t][t] < 0  t is in a negative cycle
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
        for (int t = 0; t < n; ++t) {
            if (d[i][t] < INF && d[t][t] < 0 && d[t][j]
< INF)
                d[i][j] = - INF;
        }
    }
}
```

SPFA

```
const ll OO = 1e15;
const int N = 2500 + 5;
vector<pair<int,ll>> adj[N];
//st and par are optional, just for finding negative
cycles
vi st;
int n;
bool SPFA(int src, vector<ll>& d){
    fill(d.begin(), d.end(), OO);
    vi cnt(n+1), par(n+1, -1);
    vector<bool> inQ(n+1, false);
    queue<int> q;

    d[src] = 0;
    q.push(src);
    inQ[src] = true;
    while(!q.empty()){
        int p = q.front(); q.pop();
        inQ[p] = false;

        for(auto e:adj[p]){
            int to = e.F;ll w = e.S;
            if(d[p] + w < d[to]){
                d[to] = max(-OO, d[p] + w);
                par[to] = p;
                if(!inQ[to]){
                    inQ[to] = true;
                    if(++cnt[to] > n)
                        st.pb(to);
                    else
                        q.push(to);
                }
            }
        }
    }

    sort(st.begin(), st.end());
    st.erase(unique(st.begin(), st.end()), st.end());
    for(auto &e:st)for(int i=0; i<n; i++)e = par[e];

    return st.empty();
}
```

Kosaraju

```
vector<int> adj[N], adjr[N], scc[N];
int vis[N], head[N], n;
```

```
stack<int> topo;

void dfs(int u) {
    vis[u] = 1;
    for (auto v: adj[u]) if (!vis[v]) dfs(v);
    topo.push(u);
}

void dfs2(int u, int g) {
    if (~head[u]) return;
    head[u] = g;
    for (auto v: adjr[u]) dfs2(v, g);
}

void kosaraju() {
    for (int i = 0; i < n; ++i) {
        vis[i] = false;
        head[i] = -1;
    }
    int comps = 0;
    for (int i = 0; i < n; i++)if (!vis[i]) dfs(i);

    while (!topo.empty()) dfs2(topo.top(), comps++),
topo.pop();

    for (int u = 0; u < n; ++u) {
        for(auto v:adj[u]){
            if(head[u] == head[v])continue;
            scc[head[u]].push_back(head[v]);
        }
    }
}
```

SCC and TwoSat

```
#include <bits/stdc++.h>

#define pb push_back
#define F first
#define S second
#define MP make_pair
#define all(x) x.begin(),x.end()
#define Hagry
ios::sync_with_stdio(false);cout.tie(NULL);cin.tie(NULL
);

using namespace std;
using ll = long long;
using pi = pair<int, int>;
using vi = vector<int>;
using vb = vector<bool>;
using vll = vector<ll>;
using vpi = vector<pair<int, int>>;
using vvi = vector<vector<int>>;

// assuming nodes are zero based
struct SCC {
    vvi adj, adjRev, comps;
    vpi edges;
    vi revOut, compOf;
    vb vis;
    int N;

    void init(int n) {
        N = n;
        adj.resize(n);
        adjRev.resize(n);
        vis.resize(n);
        compOf.resize(n);
    }

    void addEdge(int u, int v) {
        edges.pb(make_pair(u, v));
        adj[u].pb(v);
        adjRev[v].pb(u);
    }

    void dfs1(int u) {
        vis[u] = true;
        for (auto v:adj[u])
            if (!vis[v])
                dfs1(v);
        revOut.pb(u);
    }

    void dfs2(int u) {
        vis[u] = true;
        comps.back().pb(u);
        compOf[u] = comps.size() - 1;
        for (auto v:adjRev[u])
            if (!vis[v])dfs2(v);
    }

    void gen() {
        fill(all(vis), false);
        for (int i = 0; i < N; ++i) {
            if (!vis[i])
                dfs1(i);
        }
        reverse(all(revOut));
        fill(all(vis), false);
        for (auto node:revOut) {
            if (vis[node])continue;
            comps.pb(vi());
            dfs2(node);
        }
    }

    vvi generateCondensedGraph() {
        vvi adjCon(comps.size());
        for (auto edge:edges)
            if (compOf[edge.F] != compOf[edge.S])

adjCon[compOf[edge.F]].pb(compOf[edge.S]);
        return adjCon;
    }
};

// usage: for negating varaibles pass ~x
// -1-2*x transforms ~x into 2*x + 1
struct TwoSat {
    int N;
    vpi edges;

    void init(int _N) {
        N = _N;
    }

    int addVar() { return N++; }

    // x or y, edges will be refined in the end
    void either(int x, int y) {
        x = max(2 * x, -1 - 2 * x);
        y = max(2 * y, -1 - 2 * y);
        edges.pb({x, y});
    }

    void implies(int x, int y) {
        either(~x, y);
```

```
    }

    void must(int x) {
        either(x, x);
    }

    void XOR(int x, int y) {
        either(x, y);
        either(~x, ~y);
    }

    // void atMostOne exists in kactl
    vb solve(int _N = -1) {
        if (_N != -1) N = _N;
        SCC scc;
        scc.init(2 * N);
        for (auto e:edges) {
            scc.addEdge(e.F ^ 1, e.S);
            scc.addEdge(e.S ^ 1, e.F);
        }
        scc.gen();
        for (int i = 0; i < 2 * N; ++i) {
            if (scc.compOf[i] == scc.compOf[i ^
1])return {};
        }
        vvi &comps = scc.comps;
        vi &compOf = scc.compOf;
        vi tmp(comps.size());
        for (int i = comps.size()-1; ~i; --i) {
            if (!tmp[i]) {
                tmp[i] = 1;
                for (auto e:comps[i])
                    tmp[compOf[e ^ 1]] = -1;
            }
        }
        vb ans(N);
        for (int i = 0; i < N; ++i)
            ans[i] = tmp[compOf[2 * i]] == 1;
        return ans;
    }
};
```

Dinic

```
#define rep(aa, bb, cc) for(int aa = bb; aa < cc;aa++)
struct Dinic {
    struct Edge {
        int to, rev,idx;
        ll c, oc;
        ll flow() { return max(oc - c, 0LL); } // if
you need flows
    };
    vi lvl, ptr, q;
    vector<vector<Edge>> adj;
    Dinic(int n) : lvl(n), ptr(n), q(n), adj(n) {}
    void addEdge(int a, int b, ll c, ll rcap = 0) {
        adj[a].push_back({b, (int)adj[b].size(),1, c,
c});
        adj[b].push_back({a, (int)adj[a].size() - 1,-1,
rcap, rcap});
    }
    ll dfs(int v, int t, ll f) {
        if (v == t || !f) return f;
        for (int& i = ptr[v]; i < (int)adj[v].size();
i++) {
            Edge& e = adj[v][i];
            if (lvl[e.to] == lvl[v] + 1)
                if (ll p = dfs(e.to, t, min(f, e.c))) {
                    e.c -= p, adj[e.to][e.rev].c += p;
                    return p;
                }
        }
        return 0;
    }
    ll calc(int s, int t) {
        ll flow = 0; q[0] = s;
        rep(L,0,31) do { // 'int L=30' maybe faster for
random data
            lvl = ptr = vi((int)q.size());
            int qi = 0, qe = lvl[s] = 1;
            while (qi < qe && !lvl[t]) {
                int v = q[qi++];
                for (Edge e : adj[v])
                    if (!lvl[e.to] && e.c >> (30 -
L))
                        q[qe++] = e.to, lvl[e.to] =
lvl[v] + 1;
            }
            while (ll p = dfs(s, t, LLONG_MAX))
flow += p;
        } while (lvl[t]);
        return flow;
    }
    bool leftOfMinCut(int a) { return lvl[a] != 0; }
};
```

MinCost-MaxFlow

```
struct Edge {
    int to;
    int cost;
    int cap, flow, backEdge;
};

struct MCMF
{

    const int inf = 1000000010;
    int n;
    vector<vector<Edge>> g;

    MCMF(int _n) {
        n = _n + 1;
        g.resize(n);
    }

    void addEdge(int u, int v, int cap, int cost) {
        Edge e1 = {v, cost, cap, 0, (int) g[v].size()};
        Edge e2 = {u, -cost, 0, 0, (int) g[u].size()};
        g[u].push_back(e1);
        g[v].push_back(e2);
    }

    pair<int, int> minCostMaxFlow(int s, int t) {
        int flow = 0;
        int cost = 0;
        vector<int> state(n), from(n), from_edge(n);
        vector<int> d(n);
        deque<int> q;
        while (true) {
            for (int i = 0; i < n; i++)
                state[i] = 2, d[i] = inf, from[i] = -1;
            state[s] = 1;
            q.clear();
            q.push_back(s);
            d[s] = 0;
            while (!q.empty()) {
                int v = q.front();
                q.pop_front();
```

```
                state[v] = 0;
                for (int i = 0; i < (int) g[v].size();
i++) {
                    Edge e = g[v][i];
                    if (e.flow >= e.cap || (d[e.to] <=
d[v] + e.cost))
                        continue;
                    int to = e.to;
                    d[to] = d[v] + e.cost;
                    from[to] = v;
                    from_edge[to] = i;
                    if (state[to] == 1) continue;
                    if (!state[to] || (!q.empty() &&
d[q.front()] > d[to]))
                        q.push_front(to);
                    else q.push_back(to);
                    state[to] = 1;
                }
            }
            if (d[t] == inf) break;
            int it = t, addflow = inf;
            while (it != s) {
                addflow = min(addflow,

g[from[it]][from_edge[it]].cap
                                -
g[from[it]][from_edge[it]].flow);
                it = from[it];
            }
            it = t;
            while (it != s) {
                g[from[it]][from_edge[it]].flow +=
addflow;

g[it][g[from[it]][from_edge[it]].backEdge].flow -=
addflow;
                cost += g[from[it]][from_edge[it]].cost
* addflow;
                it = from[it];
            }
            flow += addflow;
        }
        return {cost, flow};
    }
};
```

MinCost-MaxFlow with Negative Cycles

```
template<typename flow_t = int, typename cost_t = int>
struct mcSFlow {
  struct Edge {
    cost_t c;
    flow_t f;
    int to, rev;
    Edge(int _to, cost_t _c, flow_t _f, int _rev):
c(_c), f(_f), to(_to), rev(_rev) {}
  };
  static constexpr cost_t INFCOST =
numeric_limits<cost_t>::max() / 2;
  cost_t eps;
  int N, S, T;
  vector<vector<Edge> > G;
  vector<unsigned int> isq, cur;
  vector<flow_t> ex;
  vector<cost_t> h;
  mcSFlow(int _N, int _S, int _T): eps(0), N(_N),
S(_S), T(_T), G(_N) {}
  void add_edge(int a, int b, cost_t cost, flow_t cap)
{
    assert(cap >= 0);
    assert(a >= 0 && a < N && b >= 0 && b < N);
    if(a == b) {
      assert(cost >= 0);
      return;
    }
    cost *= N;
    eps = max(eps, abs(cost));
    G[a].emplace_back(b, cost, cap, G[b].size());
    G[b].emplace_back(a, -cost, 0, G[a].size() - 1);
  }
  void add_flow(Edge& e, flow_t f) {
    Edge &back = G[e.to][e.rev];
    if (!ex[e.to] && f)
      hs[h[e.to]].push_back(e.to);
    e.f -= f;
    ex[e.to] += f;
    back.f += f;
    ex[back.to] -= f;
  }
  vector<vector<int> > hs;
  vector<int> co;
  flow_t max_flow() {
    ex.assign(N, 0);
    h.assign(N, 0);
    hs.resize(2 * N);
    co.assign(2 * N, 0);
    cur.assign(N, 0);
    h[S] = N;
    ex[T] = 1;
    co[0] = N - 1;
    for(auto &e : G[S]) add_flow(e, e.f);
    if(hs[0].size())
      for (int hi = 0; hi >= 0;) {
        int u = hs[hi].back();
        hs[hi].pop_back();
        while (ex[u] > 0) { // discharge u
          if (cur[u] == G[u].size()) {
            h[u] = 1e9;
            for(unsigned int i = 0; i < G[u].size();
++i) {
              auto &e = G[u][i];
              if (e.f && h[u] > h[e.to] + 1) {
                h[u] = h[e.to] + 1, cur[u] = i;
              }
            }
            if (++co[h[u]], !--co[hi] && hi < N)
              for(int i = 0; i < N; ++i)
                if (hi < h[i] && h[i] < N) {
                  --co[h[i]];
                  h[i] = N + 1;
                }
            hi = h[u];
          } else if (G[u][cur[u]].f && h[u] ==
h[G[u][cur[u]].to] + 1)
            add_flow(G[u][cur[u]], min(ex[u],
G[u][cur[u]].f));
          else ++cur[u];
        }
        while (hi >= 0 && hs[hi].empty()) --hi;
      }
    return -ex[S];
  }
  void push(Edge &e, flow_t amt) {
    if(e.f < amt) amt = e.f;
    e.f -= amt;
    ex[e.to] += amt;
    G[e.to][e.rev].f += amt;
    ex[G[e.to][e.rev].to] -= amt;
  }
```

```
  void relabel(int vertex) {
    cost_t newHeight = -INFCOST;
    for(unsigned int i = 0; i < G[vertex].size(); ++i)
{
      Edge const&e = G[vertex][i];
      if(e.f && newHeight < h[e.to] - e.c) {
        newHeight = h[e.to] - e.c;
        cur[vertex] = i;
      }
    }
    h[vertex] = newHeight - eps;
  }
  static constexpr int scale = 2;
  pair<flow_t, cost_t> minCostMaxFlow() {
    cost_t retCost = 0;
    for(int i = 0; i < N; ++i)
      for(Edge &e : G[i])
        retCost += e.c * (e.f);
    //find max-flow
    flow_t retFlow = max_flow();
    h.assign(N, 0);
    ex.assign(N, 0);
    isq.assign(N, 0);
    cur.assign(N, 0);
    queue<int> q;
    for(; eps; eps >>= scale) {
      //refine
      fill(cur.begin(), cur.end(), 0);
      for(int i = 0; i < N; ++i)
        for(auto &e : G[i])
          if(h[i] + e.c - h[e.to] < 0 && e.f) push(e,
e.f);
      for(int i = 0; i < N; ++i) {
        if(ex[i] > 0) {
          q.push(i);
          isq[i] = 1;
        }
      }
      // make flow feasible
      while(!q.empty()) {
        int u = q.front();
        q.pop();
        isq[u] = 0;
        while(ex[u] > 0) {
          if(cur[u] == G[u].size())
            relabel(u);
          for(unsigned int &i = cur[u], max_i =
G[u].size(); i < max_i; ++i) {
            Edge &e = G[u][i];
            if(h[u] + e.c - h[e.to] < 0) {
              push(e, ex[u]);
              if(ex[e.to] > 0 && isq[e.to] == 0) {
                q.push(e.to);
                isq[e.to] = 1;
              }
              if(ex[u] == 0) break;
            }
          }
        }
      }
      if(eps > 1 && eps >> scale == 0) {
        eps = 1 << scale;
      }
    }
    for(int i = 0; i < N; ++i) {
      for(Edge &e : G[i]) {
        retCost -= e.c * (e.f);
      }
    }
    return make_pair(retFlow, retCost / 2 / N);
  }
  flow_t getFlow(Edge const &e) {
    return G[e.to][e.rev].f;
  }
};
```

Hopcroft-Karp

```
// Gets maximum bipartite matching
struct HopcroftKarp {
    vector<int> leftMatch, rightMatch, dist, cur;
    vector<vector<int> > a;
    int n, m;

    HopcroftKarp() {}

    HopcroftKarp(int n, int m) {
        this->n = n;
        this->m = m;
        a = vector<vector<int> >(n);
        leftMatch = vector<int>(m, -1);
        rightMatch = vector<int>(n, -1);
        dist = vector<int>(n, -1);
        cur = vector<int>(n, -1);
    }

    void addEdge(int x, int y) {
        a[x].push_back(y);
    }

    int bfs() {
        int found = 0;
        queue<int> q;
        for (int i = 0; i < n; i++)
            if (rightMatch[i] < 0) dist[i] = 0,
q.push(i);
            else dist[i] = -1;

        while (!q.empty()) {
            int x = q.front();
            q.pop();
            for (int i = 0; i < int(a[x].size()); i++)
{
                int y = a[x][i];
                if (leftMatch[y] < 0) found = 1;
                else if (dist[leftMatch[y]] < 0)
                    dist[leftMatch[y]] = dist[x] + 1,
q.push(leftMatch[y]);
            }
        }

        return found;
    }

    int dfs(int x) {
        for (; cur[x] < int(a[x].size()); cur[x]++) {
            int y = a[x][cur[x]];
            if (leftMatch[y] < 0 || (dist[leftMatch[y]]
== dist[x] + 1 && dfs(leftMatch[y]))) {
                leftMatch[y] = x;
                rightMatch[x] = y;
                return 1;
            }
        }
        return 0;
    }

    int maxMatching() {
        int match = 0;
```

```
        while (bfs()) {
            for (int i = 0; i < n; i++) cur[i] = 0;
            for (int i = 0; i < n; i++)
                if (rightMatch[i] < 0) match += dfs(i);
        }
        return match;
    }
};
```

Flows With Lower Bounds

```
void solve() {
    int n, m;
    cin >> n >> m;
    int src = n, sink = n + 1;
    Dinic flw(n + 2);
    int sum_lower = 0;
    vector<int> ans(m + 1);
    for (int i = 1; i <= m; ++i) {
        int u, v, lower, upper;
        cin >> u >> v >> lower >> upper;
        u--,v--;

        flw.addEdge(u, v, upper - lower, i);
        flw.addEdge(src, v, lower, 0);
        flw.addEdge(u, sink, lower, 0);

        sum_lower += lower;

        ans[i] = lower;
    }

    int flow = flw.calc(src,sink);

    if (flow != sum_lower) {
        cout << "NO\n";
        return;
    }

    cout << "YES\n";
    for (int i = 0; i < flw.adj.size(); ++i) {
        for(auto &edge:flw.adj[i]){
            ans[edge.id] += edge.flow();
            flow += edge.flow();
        }
    }

    for (int i = 1; i <= m; ++i) {
        cout << ans[i] << '\n';
    }
}
```

Trees

LCA

```
vector<int> adj[N];
int depth[N], up[N][LOG], n, timer, tin[N], tout[N];


void dfs(int u, int p) {
    tin[u] = timer++;
    for (auto v: adj[u]) {
        if (v == p)continue;
        depth[v] = depth[u] + 1;
        up[v][0] = u;
        dfs(v, u);
    }
    tout[u] = timer - 1;
}

bool isAncestor(int u, int v) {
    return tin[u] <= tin[v] && tout[u] >= tout[v];
}

int LCA(int u, int v) {
    if (depth[u] < depth[v])
        swap(u, v);
    int k = depth[u] - depth[v];
    for (int i = 0; i < LOG; ++i) {
        if ((1 << i) & k) {
            u = up[u][i];
        }
    }
    if (u == v)
        return u;
    for (int i = LOG - 1; i >= 0; --i) {
        if (up[u][i] != up[v][i]) {
            u = up[u][i];
            v = up[v][i];
        }
    }
    return up[u][0];
}

int Kthancestor(int u,int k){
    if(k > depth[u])return 0;
    for (int j = LOG - 1; j >= 0; --j) {
        if(k&(1<<j)){
            u = up[u][j];
        }
    }
    return u;
}

void build() {
    dfs(0, 0);
    for (int j = 1; j < LOG; ++j) {
        for (int i = 0; i < n; ++i) {
            up[i][j] = up[up[i][j - 1]][j - 1];
        }
    }
}
```

Tree Hashing

```
vector<int>adj[N];
map<vector<int>,int>mp;
int dfs(int u,int par){
    vector<int>cur;
    for(auto v:adj[u]){
        if(v == par)continue;
        cur.push_back(dfs(v,u));}
    sort(all(cur));
    if(!mp.count(cur))mp[cur] = mp.size();
    return mp[cur];
}
```

Tree Hashing 2

```
unsigned long long pw(unsigned long long b, unsigned
long long p) {
    if (!p) return 1ULL;
    unsigned long long ret = pw(b, p >> 1ULL);
    ret *= ret;
    if (p & 1ULL)
        ret = ret * b;
    return ret;
```

```
}
int n;
vector<int> adj[N];
unsigned long long dfs(int u, int par) {
    vector<unsigned long long> child;
    for (auto v: adj[u]) {
        if (v == par)continue;
        child.push_back(dfs(v, u));
    }
    sort(all(child));
    unsigned long long ret = 0;
    for (int i = 0; i < child.size(); ++i) {
        ret += child[i] * child[i] + child[i] * pw(31,
i + 1) + (unsigned long long) 42;
    }
    return ret;
}
```

HLD

```
class HLD {
public:
    vector<int> par, sz, head, tin, tout, who, depth;

    int dfs1(int u, vector<vector<int>> &adj) {
        for (int &v: adj[u]) {
            if (v == par[u])continue;
            depth[v] = depth[u] + 1;
            par[v] = u;
            sz[u] += dfs1(v, adj);
            if (sz[v] > sz[adj[u][0]] || adj[u][0] ==
par[u]) swap(v, adj[u][0]);
        }
        return sz[u];
    }

    void dfs2(int u, int &timer, const
vector<vector<int>> &adj) {
        tin[u] = timer++;
        for (int v: adj[u]) {
            if (v == par[u])continue;
            head[v] = (timer == tin[u] + 1 ? head[u] :
v);
            dfs2(v, timer, adj);
        }
        tout[u] = timer - 1;
    }
    HLD(vector<vector<int>> adj, int r = 0)
            : par(adj.size(), -1), sz(adj.size(), 1),
head(adj.size(), r), tin(adj.size()), who(adj.size()),
tout(adj.size()),
              depth(adj.size()){
        dfs1(r, adj);
        int x = 0;
        dfs2(r, x, adj);
        for (int i = 0; i < adj.size(); ++i)
who[tin[i]] = i;
    }

    vector<pair<int, int>> path(int u, int v) {
        vector<pair<int, int>> res;
        for (;; v = par[head[v]]) {
            if(depth[head[u]] >
depth[head[v]])swap(u,v);
            if(head[u] != head[v]){
                res.emplace_back(tin[head[v]], tin[v]);
            }
            else{
                if(depth[u] > depth[v])swap(u,v);
                res.emplace_back(tin[u],tin[v]);
                return res;
            }
        }
    }

    pair<int, int> subtree(int u) {
        return {tin[u], tout[u]};
    }

    int dist(int u, int v) {
        return depth[u] + depth[v] - 2 * depth[lca(u,
v)];
    }

    int lca(int u, int v) {
        for (;; v = par[head[v]]) {
            if(depth[head[u]] >
depth[head[v]])swap(u,v);
            if(head[u] == head[v]){
                if(depth[u] > depth[v])swap(u,v);
                return u;
            }
        }
    }

    bool isAncestor(int u, int v) {
        return tin[u] <= tin[v] && tout[u] >= tout[v];
    }
};
```

Centroid Decomposition

```
int sz[N], n, k, freq[N];
vi adj[N];
bool rem[N];

void preSize(int i, int par) {
    sz[i] = 1;
    for (auto e: adj[i]) {
        if (e == par || rem[e])
            continue;
        preSize(e, i);
        sz[i] += sz[e];
    }
}

int getCen(int u, int p, int curSz) {
    for (auto v: adj[u]) {
        if (rem[v] || v == p)continue;
        if (sz[v] * 2 > curSz)
            return getCen(v, u, curSz);
    }
    return u;
}

ll solve(int v, int par, int d) {
    ll ans = k >= d ? freq[k - d] : 0;
    for (auto u: adj[v]) {
        if (rem[u] || u == par)
            continue;
        ans += solve(u, v, d + 1);
    }
    return ans;
}

void update(int v, int par, int d, int inc) {
    freq[d] += inc;
    for (auto u: adj[v]) {
        if (rem[u] || u == par)
```

```
            continue;
        update(u, v, d + 1, inc);
    }
}

ll getAns(int v) {
    ll ans = 0;
    for (auto u: adj[v]) {
        if (rem[u])
            continue;
        ans += solve(u, v, 1);
        update(u, v, 1, 1);
    }
    return ans;
}

ll decompose(int v) {
    preSize(v, 0);
    int cen = getCen(v, 0, sz[v]);
    freq[0]++;
    ll ans = getAns(cen);
    update(cen, 0, 0, -1);
    rem[cen] = true;
    for (auto u: adj[cen]) {
        if (rem[u])
            continue;
        ans += decompose(u);
    }
    return ans;
}
```

DSU On Tree

```
int dep[N], sz[N], big[N];
vi adj[N];

void dfs(int v, int p) {
    dep[v] = dep[p] + 1;
    sz[v] = 1;
    for (auto u: adj[v]) {
        if (u == p)
            continue;
        dfs(u, v);
        sz[v] += sz[u];
        if(big[v] == -1 || sz[u] > sz[big[v]])
            big[v] = u;
    }
}

vi *cols[N];
int col[N], freq[N], distinct, ans[N];

void smallToLarge(int v, int p, bool keep) {
    for (auto u: adj[v]) {
        if (u == p || u == big[v])
            continue;
        smallToLarge(u, v, false);
    }
    if (~big[v]){
        smallToLarge(big[v], v, true), cols[v] =
cols[big[v]];
    }
    else
        cols[v] = new vi;

    cols[v]->pb(col[v]);
    freq[col[v]]++;
    if (freq[col[v]] == 1)
        distinct++;
    for (auto u: adj[v]) {
        if (u == p || u == big[v])
            continue;
        for (auto e: *cols[u]) {
            cols[v]->pb(e);
            freq[e]++;
            if (freq[e] == 1)
                distinct++;
        }
    }
    ans[v] = distinct;
    if (keep)
        return;
    for (auto e: *cols[v]) {
        freq[e]--;
        if (!freq[e])
            --distinct;
    }
}
```

Mo On Trees

```

struct Query {
    int l, r, ind, lca;

    Query(int _l, int _r, int _ind, int _lca = -1) :
l(_l), r(_r), ind(_ind), lca(_lca) {}

    bool operator<(const Query &q2) {
        return (l / B < q2.l / B) || (l / B == q2.l / B
&& r < q2.r);
    }
};


struct MoTree {
    vi in, out, flat, dep, freqV;
    vvi anc;
    int n;

    MoTree(vvi& adj, int n, vi& col, int r = 1)
            : n(n), in(n+1), out(n+1), flat((n+1) * 2),
              dep(n+1), freqV(n+1), anc(n+1, vi(LG)),
    {
        int x = 0;
        flatten(r, r, x, adj);
        preLCA();
    }

    void flatten(int v, int p, int& timer, const vvi&
adj) {
        anc[v][0] = p;
        dep[v] = dep[p] + 1;
        in[v] = timer, flat[timer] = v, ++timer;
        for (auto u: adj[v]) if(u != p){
                flatten(u, v, timer, adj);
            }
        out[v] = timer, flat[timer] = v, ++timer;
    }

    void preLCA() {
        for (int k = 1; k < LG; k++)
            for (int i = 1; i <= n; i++)
                anc[i][k] = anc[anc[i][k - 1]][k - 1];
    }

    int binaryLift(int x, int jump) {
        for (int b = 0; b < LG; b++) {
            if (jump & (1 << b))
                x = anc[x][b];
```

```
        }
        return x;
    }

    int LCA(int a, int b) {
        if (dep[a] > dep[b])
            swap(a, b);
        int diff = dep[b] - dep[a];
        b = binaryLift(b, diff);
        if (a == b)
            return a;

        for (int bit = LG - 1; bit >= 0; bit--) {
            if (anc[a][bit] == anc[b][bit])
                continue;
            a = anc[a][bit];
            b = anc[b][bit];
        }
        return anc[a][0];
    }

    void upd(int ind, int inc){
        int v = flat[ind];
        freqV[v] += inc;
        if (freqV[v] == 1) {
            // add()
        }
        else {
            // remove()
        }
    }

    vi takeQueries(int q){
        vi ans(q);
        vector<Query> queries;
        int x, y;
        for(int i = 0; i < q; i++)
        {
            cin >> x >> y;
            if (in[x] > in[y])
                swap(x, y);
            int lca = LCA(x, y);
            if (lca == x)
                queries.emplace_back(in[x], in[y], i);
            else
                queries.emplace_back(out[x], in[y], i,
lca);
        }
        sort(all(queries));

        int l = 0, r = 0;
        upd(0, 1);
        for(auto query:queries)
        {
            while (r < query.r)
                upd(++r, 1);
            while (l > query.l)
                upd(--l, 1);
            while (l < query.l)
                upd(l++, -1);
            while (r > query.r)
                upd(r--, -1);

            if(~query.lca) ;//addLCA
            //ans[query.ind] = ;
            if(~query.lca) ;//removeLCA
        }

        return ans;
    }
};
```

Strings

Trie

```
const int K = 26;

struct Trie {
    struct Node {
        int go[K];
        int freq;

        Node() {
            fill(go, go + K, -1);
            freq = 0;
        }
    };

    vector<Node> aut;

    Trie(vector<string> &pats) {
        aut.resize(1);
        for (auto &e:pats)
            add_string(e);
    }

    void add_string(string &s) {
        int u = 0; //cur node
        for (auto ch:s) {
            int c = ch - 'a';
            if (aut[u].go[c] == -1) {
                aut[u].go[c] = (int) aut.size();
                aut.emplace_back();
            }
            u = aut[u].go[c];
            aut[u].freq++;
        }
    }
};
```

Trie For Numbers

```
struct Trie{
    vector<vector<int>>trie;
    vector<int>cnt;
//    vector<int>leaves;
    int mxBit,sz;

    int addNode(){
        trie.emplace_back(2,-1);
        cnt.emplace_back();
//        leaves.emplace_back();
        sz++;
        return sz - 1;
    }

    Trie(int mx = 60): mxBit(mx),sz(0){
        addNode();
    };

    // insert or remove
    void insert(ll x,int type = 1){
        int cur = 0;
        cnt[cur] += type;
        for (int i = mxBit; i >= 0; --i) {
            int t = (x >> i)&1;
            if(trie[cur][t] == -1)
```

```
                trie[cur][t] = addNode();
            cur = trie[cur][t];
            cnt[cur] += type;
        }
//        leaves[cur] += type;
    }

    ll maxXor(ll x){
        // no elements in trie
        int cur = 0;
        if(!cnt[cur])return -1e9;
        for (int i = mxBit; i >= 0; --i) {
            int t = (x >> i)&1^1;
            if(trie[cur][t] == -1 ||
!cnt[trie[cur][t]])t ^= 1;
            cur = trie[cur][t];
            if(t)x ^= 1ll << i;
        }
        return x;
    }
};
```

ACA

```
struct AhoCorasick
{
    int states = 0;
    vector<int> pi;
    vector<vector<int>> trie, patterns;

    AhoCorasick(int n, int m = 26)
    {
        pi = vector<int>(n + 10, -1);
        patterns = vector<vector<int>>(n + 10);
        trie = vector<vector<int>>(n + 10,
vector<int>(m, -1));
    }

    AhoCorasick(vector<string> &p, int n, int m = 26)
    {
        /*
         * MAKE SURE THAT THE STRINGS IN P ARE UNIQUE
         * N is the summation of sizes of p
         * M is the number of used alphabet
        */

        pi = vector<int>(n + 10, -1);
        patterns = vector<vector<int>>(n + 10);
        trie = vector<vector<int>>(n + 10,
vector<int>(m, -1));

        for(int i = 0; i < p.size(); i++)
            insert(p[i], i);
        build();
    }

    void insert(string &s, int idx)
    {
        int cur = 0;
        for(auto &it: s)
        {
            if(trie[cur][it - 'a'] == -1)
                trie[cur][it - 'a'] = ++states;
            cur = trie[cur][it - 'a'];
        }
        patterns[cur].push_back(idx);
    }

    int nextState(int trieNode, int nxt)
    {
        int cur = trieNode;
        while(trie[cur][nxt] == -1)
            cur = pi[cur];
        return trie[cur][nxt];
    }

    void build()
    {
        queue<int> q;
        for(int i = 0; i < 26; i++)
        {
            if(trie[0][i] != -1)
                pi[trie[0][i]] = 0, q.push(trie[0][i]);
            else
                trie[0][i] = 0;
        }

        while(q.size())
        {
            int cur = q.front();
            q.pop();
            for(int i = 0; i < 26; i++)
            {
                if(trie[cur][i] == -1)
                    continue;
                int f = nextState(pi[cur],i);
                pi[trie[cur][i]] = f;

patterns[trie[cur][i]].insert(patterns[trie[cur][i]].en
d(), patterns[f].begin(), patterns[f].end());
                q.push(trie[cur][i]);
            }
        }
    }


    vector<vector<int>> search(string &s,
vector<string> &p, int n)
    {
        int cur = 0;
        vector<vector<int>> ret(n);
        for(int i = 0; i < s.length(); i++)
        {
            cur = nextState(cur, s[i] - 'a');
            if(cur == 0 || patterns[cur].empty())
                continue;

            // patterns vector have every pattern that
is matched in this node
            // matched: the last index in the pattern
is index i
            for(auto &it: patterns[cur])
                ret[it].push_back(i - p[it].length() +
1);
        }
        return ret;
    }
};
```

Z-Algorithm

```
vector<int> z_function(string s) {
    int n = (int) s.length();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r)
            z[i] = min (r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]])
            ++z[i];
        if (i + z[i] - 1 > r)
```

```
            l = i, r = i + z[i] - 1;
    }
    return z;
}
```

String Hashing

```
// Right is most significant
const int p1 = 31, p2 = 37, MOD = 1e9 + 7;

const int N = 1e6 + 5;
int pw1[N], inv1[N], pw2[N], inv2[N];

ll powmod(ll x, ll y) {
    x %= MOD;
    ll ans = 1;
    while (y) {
        if (y & 1) ans = ans * x % MOD;
        x = x * x % MOD;
        y >>= 1;
    }
    return ans;
}

ll add(ll a, ll b) {
    a += b;
    if (a >= MOD) a -= MOD;
    return a;
}

ll sub(ll a, ll b) {
    a -= b;
    if (a < 0) a += MOD;
    return a;
}

ll mul(ll a, ll b) { return a * b % MOD; }

ll inv(ll a) { return powmod(a, MOD - 2); }

void pre() {
    pw1[0] = inv1[0] = 1;
    pw2[0] = inv2[0] = 1;
    int invV1 = inv(p1);
    int invV2 = inv(p2);
    for (int i = 1; i < N; ++i) {
        pw1[i] = mul(pw1[i - 1], p1);
        inv1[i] = mul(inv1[i - 1], invV1);
        pw2[i] = mul(pw2[i - 1], p2);
        inv2[i] = mul(inv2[i - 1], invV2);
    }
}

struct Hash {
    vector<pi> h;
    int n;

    Hash(string &s) {
        n = s.size();
        h.resize(n);
        h[0].F = h[0].S = s[0] - 'a' + 1;
        for (int i = 1; i < n; ++i) {
            h[i].F = add(h[i-1].F,mul((s[i] - 'a' + 1),
pw1[i]));
            h[i].S = add(h[i-1].S,mul((s[i] - 'a' + 1),
pw2[i]));
        }
    }
    pi getRange(int l, int r) {
        assert(l <= r);
        assert(r < n);
        return {
                mul(sub(h[r].F, l ? h[l - 1].F : 0),
inv1[l]),
                mul(sub(h[r].S,  l ? h[l - 1].S : 0),
inv2[l])
        };
    }
};
```

String Hashing 2

```
// Left is most significant
const int N = 1e6 + 5;
int pw1[N], pw2[N];

ll powmod(ll x, ll y) {
    x %= MOD;
    ll ans = 1;
    while (y) {
        if (y & 1) ans = ans * x % MOD;
        x = x * x % MOD;
        y >>= 1;
    }
    return ans;
}

ll add(ll a, ll b) {
    a += b;
    if (a >= MOD) a -= MOD;
    return a;
}

ll sub(ll a, ll b) {
    a -= b;
    if (a < 0) a += MOD;
    return a;
}

ll mul(ll a, ll b) { return a * b % MOD; }

ll inv(ll a) { return powmod(a, MOD - 2); }

void pre() {
    pw1[0] = 1;
    pw2[0] = 1;
    for (int i = 1; i < N; ++i) {
        pw1[i] = mul(pw1[i - 1], p1);
        pw2[i] = mul(pw2[i - 1], p2);
    }
}

struct Hash {
    vector<pi> h;
    int n;

    Hash(string &s) {
        n = s.size();
        h.resize(n);
        h[0].F = h[0].S = s[0] - 'a' + 1;
        for (int i = 1; i < n; ++i) {
            h[i].F = add(mul(h[i-1].F, p1), s[i] - 'a'
+ 1);
            h[i].S = add(mul(h[i-1].S, p2), s[i] - 'a'
+ 1);
        }
    }
```

```
    pi getRange(int l, int r) {
        assert(l <= r);
        assert(r < n);
        return {
                sub(h[r].F, mul(l ? h[l-1].F : 0,
pw1[r-l+1])),
                sub(h[r].S, mul(l ? h[l-1].S : 0,
pw2[r-l+1]))
        };
    }
};
```

Manacher

```
vi manacher_odd(string& s) {
    int n = s.size();

    string t = '^' + s  + '$';
    vi p(n+2);
    int l = 1, r = 1;
    for (int i = 1; i <= n; ++i) {
        int &len = p[i];
        int j = l + r-i;
        len = max(0, min(r - i, p[j]));

        while (t[i + len] == t[i - len])
            ++len;

        if(i + len > r){
            r = i + len;
            l = i - len;
        }
    }

    return vi(p.begin() + 1, p.begin() + n + 1);
}

vector<pi> manacher(string& s){
    int n = (int)s.size();
    string t;
    for(int i=0; i<n; ++ i){
        t.pb('#');
        t.pb(s[i]);
    }
    t.pb('#');

    vi p = manacher_odd(t);
    vector<pi> ret(n);
    //odd then even
    for(int i=0; i<n; ++i){
        ret[i].F = (p[2*i+1])/2;
        ret[i].S = (p[2*i]-1)/2;
    }
    return ret;
}
```

KMP

```
void KMP(string &s, vi &fail) {
    int n = (int) s.size();
    for (int i = 1; i < n; i++) {
        int j = fail[i - 1];
        while (j > 0 && s[j] != s[i])
            j = fail[j - 1];
        if (s[j] == s[i])
            ++j;
        fail[i] = j;
    }
}
```

```
void constructAut(string &s, vi &fail) {
    int n = s.size();
    // for each fail function value (i is not an index)
    for (int i = 0; i < n; i++) {
        // for each each possible transition
        for (int c = 0; c < ALPHA; c++) {
            if (i > 0 && s[i] != 'a' + c)
                aut[i][c] = aut[fail[i - 1]][c];
            else
                aut[i][c] = i + (s[i] == 'a' + c);
        }
    }
}
```

Palindromic Tree

```
class PalindromeTree {
public:
    int n, id, cur, tot;
    vector<array<int, 26>> go;
    vector<int> suflink, len, cnt;

    PalindromeTree() {};

    PalindromeTree(const string &s) {
        n = s.length();
        go.assign(n + 2, {});
        suflink.assign(n + 2, 0);
        len.assign(n + 2, 0);
        cnt.assign(n + 2, 0);
        suflink[0] = suflink[1] = 1;
        len[1] = -1;
        id = 2;
        cur = 0;
        tot = 0;
        for (int i = 0; i < n; i++) {
            add(s, i);
        }
    }

    int get(const string &s, int i, int v) {
        while (i - len[v] - 1 < 0 || s[i - len[v] - 1]
!= s[i]) {
            v = suflink[v];
        }
        return v;
    }

    void add(const string &s, int i) {
        int ch = s[i] - 'a';
        cur = get(s, i, cur);
        if (go[cur][ch] == 0) {
            len[id] = 2 + len[cur];
            suflink[id] = go[get(s, i,
suflink[cur])][ch];
            tot++;
            go[cur][ch] = id++;
        }
        cur = go[cur][ch];
        cnt[cur]++;
    }

    void countAll(){
        for (int i = id - 1; i >= 2; --i) {
            cnt[suflink[i]] += cnt[i];
        }
    }

    int cntDistinct() {
        return tot;
```

```
    }
};
```

Suffix Array

// Look up Suffix Array in MIT KACTL instead, much shorter

```
struct SuffixArray {
    string S;
    // sa is the suffix array with the empty suffix
being sa[0]
    // lcp[i] holds the lcp between sa[i], sa[i - 1]
    vector<int> logs, sa, lcp, rank;
    vector<vector<int>> table;

    SuffixArray() {};

    SuffixArray(string &s, int lim = 256) {
        S = s;
        int n = s.size() + 1, k = 0, a, b;
        vector<int> c(s.begin(), s.end() + 1), tmp(n),
frq(max(n, lim));
        c.back() = 0; //0 is less than any character
        sa = lcp = rank = tmp, iota(sa.begin(),
sa.end(), 0);
        for (int j = 0, p = 0; p < n; j = max(1, j *
2), lim = p) {
            p = j, iota(tmp.begin(), tmp.end(), n - j);
            for (int i = 0; i < n; i++) {
                if (sa[i] >= j)
                    tmp[p++] = sa[i] - j;
            }

            fill(frq.begin(), frq.end(), 0);

            for (int i = 0; i < n; i++) frq[c[i]]++;
            for (int i = 1; i < lim; i++) frq[i] +=
frq[i - 1];
            for (int i = n; i--;) sa[--frq[c[tmp[i]]]]
= tmp[i];

            swap(c, tmp), p = 1, c[sa[0]] = 0;
            for (int i = 1; i < n; i++)
                a = sa[i - 1], b = sa[i], c[b] =
(tmp[a] == tmp[b] && tmp[a + j] == tmp[b + j]) ? p - 1
: p++;
        }

        for (int i = 1; i < n; i++) rank[sa[i]] = i;
        for (int i = 0, j; i < n - 1; lcp[rank[i++]] =
k)
            for (k &&k--, j = sa[rank[i] - 1];
                    s[i + k] == s[j + k];
        k++);
    }

    void preLcp() {
        int n = S.size() + 1;
        logs = vector<int>(n + 5);
        for (int i = 2; i < n + 5; ++i) {
            logs[i] = logs[i / 2] + 1;
        }
        table = vector<vector<int>>(n,
vector<int>(20));
        for (int i = 0; i < n; ++i) {
            table[i][0] = lcp[i];
        }

        for (int j = 1; j <= logs[n]; ++j) {
            for (int i = 0; i <= n - (1 << j); ++i) {
                table[i][j] = min(table[i][j - 1],
table[i + (1 << (j - 1))][j - 1]);
            }
        }
    }
    int queryLcp(int i, int j) {
//        if (i == j)return (int) S.size() - i;
//        i = rank[i], j = rank[j];
        if (i == j)return (int) S.size() - sa[i];
        if (i > j)
            swap(i, j);
        i++;
        int len = logs[j - i + 1];
        return min(table[i][len], table[j - (1 << len)
+ 1][len]);
    }
};
```

Suffix Automaton

```
const int M = 26, N = 1000005;

struct suffixAutomaton {
    struct state {
        int len;        // length of longest string in
this class
        int link;        // pointer to suffix link
        int next[M];    // adjacency list
        ll cnt;    // number of times the strings in
this state occur in the original string

        bool terminal;    // by default, empty string
is a suffix
        // a state is terminal if it corresponds to a
suffix

        state() {
            len = 0, link = -1, cnt = 0;
            terminal = false;
            for (int i = 0; i < M; i++)
                next[i] = -1;
        }
    };

    vector<state> st;
    int sz, last, l;
    char offset = 'A';    // Careful!

    suffixAutomaton(string &s) {

        int l = s.length();
        st.resize(2 * l);
        for (int i = 0; i < 2 * l; i++)
            st[i] = state();

        sz = 1, last = 0;
        st[0].len = 0;
        st[0].link = -1;

        for (int i = 0; i < l; i++)
            addChar(s[i] - offset);

        for (int i = last; i != -1; i = st[i].link)
            st[i].terminal = true;
    }

    void addChar(int c) {
        int cur = sz++;
        assert(cur < N * 2);
        st[cur].len = st[last].len + 1;
```

```
        st[cur].cnt = 1;

        int p = last;
        while (p != -1 && st[p].next[c] == -1) {
            st[p].next[c] = cur;
            p = st[p].link;
        }

        last = cur;

        if (p == -1) {
            st[cur].link = 0;
            return;
        }

        int q = st[p].next[c];

        if (st[q].len == st[p].len + 1) {
            st[cur].link = q;
            return;
        }

        int clone = sz++;

        for (int i = 0; i < M; i++)
            st[clone].next[i] = st[q].next[i];
        st[clone].link = st[q].link;
        st[clone].len = st[p].len + 1;
        st[clone].cnt = 0;                  // cloned
states initially have cnt = 0

        while (p != -1 and st[p].next[c] == q) {
            st[p].next[c] = clone;
            p = st[p].link;
        }

        st[q].link = st[cur].link = clone;
    }

    bool contains(string &t) {
        int cur = 0;
        for (int i = 0; i < t.length(); i++) {
            cur = st[cur].next[t[i] - offset];
            if (cur == -1)
                return false;
        }
        return true;
    }

    // alternatively, compute the number of paths in a
DAG
    // since each substring corresponds to one unique
path in SA
    ll numberOfSubstrings() {
        ll res = 0;
        for (int i = 1; i < sz; i++)
            res += st[i].len - st[st[i].link].len;
        return res;
    }

    void numberOfOccPreprocess() {
        vector<pii> v;
        for (int i = 1; i < sz; i++)
            v.emplace_back(st[i].len, i);

        sort(v.begin(), v.end(), greater<>());

        for (int i = 0; i < sz - 1; i++) {
            int suf = st[v[i].second].link;
            st[suf].cnt += st[v[i].second].cnt;
        }
    }

    ll numberOfOcc(string &t) {
        int cur = 0;
        for (int i = 0; i < t.length(); i++) {
            cur = st[cur].next[t[i] - offset];
            if (cur == -1)
                return 0;
        }
        return st[cur].cnt;
    }

    ll totLenSubstrings() {
        // different Substrings
        ll tot = 0;
        for (int i = 1; i < sz; i++) {
            ll shortest = st[st[i].link].len + 1;
            ll longest = st[i].len;
            ll num_strings = longest - shortest + 1;
            ll cur = num_strings * (longest + shortest)
/ 2;
            tot += cur;
        }
        return tot;
    }
};
```

Geometry

// Look up KACTL for the rest of the algorithms

Point

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

template<class T>
struct P {
    T x, y;

    P() { ; }

    P(T x, T y) : x(x), y(y) {};

    P operator+(const P b) { return P(x + b.x, y +
b.y); }
    P operator-(const P b) { return P(x - b.x, y -
b.y); }
    P operator*(const T v) { return P(x * v, y * v); }
    P operator/(const T v) { return P(x / v, y / v); }

    bool operator== (const P b){ return MP(x, y) ==
MP(b.x, b.y);}
    T cross(P b) const{ return x * b.y - y * b.x; };
    T dot(P b) const{ return x * b.x + y * b.y; };
    T cross(P b,  P c) const { return (b -
*this).cross(c - *this); }

    T norm() { return x * x + y * y; }
    long double abs() { return sqrt(x * x + y * y); }

    P unit() { return *this / abs(); }

    friend istream &operator>>(istream &is, P &pt) {
        is >> pt.x >> pt.y;
        return is;
    }
```

```
    friend ostream &operator<<(ostream &os, P pt) {
        os << "(" << pt.x << ", " << pt.y << ")";
        return os;
    }
};
```

Distance Operations

```
template <class T>
long double lineDist(P<T>& x, P<T>& a, P<T>& b){
    return abs(a.cross(b, x)) / (b-a).abs();
}

template <class T>
long double rayDist(P<T> x, P<T> s1, P<T> s2){
    long double distSeg = lineDist(x, s1, s2);
    P<T> v1 = s1 - x;
    P<T> v2 = s2 - s1;
    if(v1.dot(v2) > 0)
        return v1.abs();
    return distSeg;
}

template <class T>
long double segDist(P<T> x, P<T> s1, P<T> s2){
    long double distLine = lineDist(x, s1, s2);
    P<T> v1 = s1 - x;
    P<T> v3 = s2 - s1;
    P<T> v2 = s2 - x;
    P<T> v4 = s1 - s2;
    if(v1.dot(v3) > 0  || v2.dot(v4) > 0)
        return min(v1.abs(), v2.abs());
    return distLine;
}

template <class P>
long double onSeg(P s1, P s2, P x){
    return segDist(x, s1, s2) < - EPS;
}

template <class P>
vector<P> segInter(P a, P b, P c, P d){
    auto oa = c.cross(d, a), ob = c.cross(d, b),
        oc = a.cross(b, c), od = a.cross(b, d);

    //very complicated formula, don't try to understand
from here, only for quick writing
    if(oa * ob < 0 && oc * od < 0)
        return {(a * ob - b * oa) / (ob - oa)};

    set<P> s;
    if(onSeg(c, d, a))s.insert(a);
    if(onSeg(c, d, b))s.insert(b);
    if(onSeg(a, b, c))s.insert(c);
    if(onSeg(a, b, d))s.insert(d);
    return {s.begin(), s.end()};
}

template<class P>
pair<int, P> lineInter(P a, P b, P c, P d) {
    auto dir = (b - a).cross(d - c);
    if (dir == 0)
        return {-(a.cross(b, c) == 0), P(0, 0)};
    auto p = c.cross(b, d), q = c.cross(d, a);
    return {1, (a * p + b * q) / dir};
}
```

Convex Hull

```
//convex hull
void convex_hull(vector<P<ll>> &pts, bool inc_collinear
= false) {
    P<ll> p0 = *min_element(pts.begin(), pts.end(),
[](P<ll> &a, P<ll> &b) {
        return MP(a.y, a.x) < MP(b.y, b.x);
    });

    sort(pts.begin(), pts.end(), [&p0]( P<ll> &a,
P<ll> &b) {
        ll o = p0.cross(a, b);
        if (o != 0)return o > 0;
        return (a - p0).norm() < (b - p0).norm();
    });

    if(inc_collinear){
        int ind = pts.size() - 1;
        while(ind >= 0 && p0.cross(pts[ind],
pts.back()) == 0) --ind;
        reverse(pts.begin() + ind + 1, pts.end());
    }

    vector<P<ll>> ch;
    for(int i=0; i<(int)pts.size(); i++){
        int sz = ch.size();
        while(ch.size() > 1 &&
                (ch[sz-2].cross(ch[sz-1], pts[i]) < 0
||
                (!inc_collinear && ch[sz-
2].cross(ch[sz-1], pts[i]) == 0))){
            ch.pop_back();
            sz = ch.size();
        }
        ch.push_back(pts[i]);
    }

    pts = ch;
}
```

Hull Diameter and Width

```
template<class T>
ll hullDiameter(vector<T> S) {
    int n = S.size(), j = n < 2 ? 0 : 1;
    ll ret = 0;
    for (int i = 0; i < j; ++i) {
        for (;; j = (j + 1) % n) {
            ret = max(ret, (ll)(S[i] - S[j]).dist2());
            if ((S[(j + 1) % n] - S[j]).cross(S[i + 1]
- S[i]) >= 0)
                break;
        }
    }
    // returns the squared diameter
    return ret;
}

template<class T>
ld hullWidth(vector<T> S) {
    int n = S.size();
    if(n <= 2)return 0;
    int i = 0,j = 1;
    ld ret = 1e18;
    while (i < n){
        while((S[(i + 1) % n] - S[i]).cross(S[(j +
1)%n] - S[j]) >= 0)j = (j + 1) % n;
        ret = min(ret,lineDist(S[j], S[i],S[(i+1)%n]));
        i++;
    }
    return ret;}
```

Angle

```
template<class T>
// angle between [0, 2*pi]
ld angleBetween(T a,T b){
    ld ret = atan2(a.cross(b),a.dot(b));
    if(dcmp(ret,0) == -1){
        ret += 2 * PI;
    }
//    return min(ret,2 * PI - ret);    to return the
smaller angle
    return ret;
}
template<class T>
ld angleO(T a, T O, T b){ /// angle(aOb)
    assert(a.dist(O) > eps && b.dist(O) > eps); // nan
    T v1 = (a - O), v2 = (b - O);
    return angleBetween(v1,v2);}
```

Polygon Area

```
template<class T>
ld polygonArea(vector<T>&v){
    ld ret = 0;
    int n = v.size();
    for (int i = 0; i < n; ++i) {
        ret += v[i].cross(v[(i+1)%n]);
    }
    return 0.5 * abs(ret);
}
```

Half-Plane Intersection

```
template<class P>
pair<int, P> lineInter(P a, P b, P c, P d) {
    auto dir = (b - a).cross(d - c);
    if (dir == 0)
        return {-(a.cross(b, c) == 0), P(0, 0)};
    auto p = c.cross(b, d), q = c.cross(d, a);
    return {1, (a * p + b * q) / dir};
}


template <class P>
struct HalfPlane{
    P p, pq;
    long double angle;

    HalfPlane(){}
    HalfPlane(P& a, P& b):p(a), pq(b-a){
        angle = pq.angle();
    }

    bool out(P r){
        return pq.cross(r-p) < -EPS;
    }

    bool operator < (const HalfPlane<P>& e)const{
        return angle < e.angle;
    }

    P inter( HalfPlane<P>& s){
        return lineInter(s.p, s.p + s.pq, p, p + pq).S;
    }
};

template <class P>
vector<P> HalfPlaneInter(vector<HalfPlane<P>>& H) {
    //bounding box
    P box[4] = {P(-OO, -OO),
                P(OO, -OO),
                P(OO, OO),
                P(-OO, OO)};

    for (int i = 0; i < 4; i++) {
        HalfPlane<P> temp(box[i], box[(i + 1) % 4]);
        H.pb(temp);
    }

    sort(H.begin(), H.end());
    deque<HalfPlane<P>> dq;
    int len = 0;
    for (int i = 0; i < (int) H.size(); i++) {
        while (len > 1 && H[i].out(dq[len -
1].inter(dq[len - 2]))) {
            dq.pop_back();
            --len;
        }
        while (len > 1 && H[i].out(dq[0].inter(dq[1])))
{
            dq.pop_front();
            --len;
        }

        if (len > 0 &&
fabsl(H[i].pq.cross(dq.back().pq)) < EPS) {
            //opposite direction, no planes at all
            if (H[i].pq.dot(dq.back().pq) < 0.0)
                return vector<P>();

            if (H[i].out(dq[len - 1].p)) {
                dq.pop_back();
                --len;
            } else
                continue;
        }
        dq.push_back(H[i]);
        ++len;
    }

    while (len > 2 && dq[0].out(dq[len -
1].inter(dq[len - 2]))) {
        dq.pop_back();
        --len;
    }

    while (len > 2 && dq[len - 1].out(dq[0].inter(
dq[1]))) {
        dq.pop_front();
        --len;
    }

    if (len < 3)return vector<P>();

    vector<P> vec(len);
    for (int i = 0; i + 1 < len; i++)
        vec[i] = dq[i].inter(dq[i + 1]);

    vec[len - 1] = dq[len - 1].inter(dq[0]);
    return vec;
}
```

Circle From 3 Points

```
typedef Point<double> P;

bool isColliner(const P &A, const P &B, const P &C) {
    return dcmp(P(B - A).cross(P(C - A)), 0) == 0;
}
```

```
P ccCenter(const P &A, const P &B, const P &C) {
    P b = C - A, c = B - A;
    return A + (b * c.dist2() - c * b.dist2()).perp() /
b.cross(c) / 2;
}

double ccRadius(const P &A, const P &B, const P &C) {
    return (B - A).dist() * (C - B).dist() * (A -
C).dist() /
           abs((B - A).cross(C - A)) / 2;
}
```

Find Intersecting Segments

```
const double EPS = 1E-9;
struct pt {
    double x, y;
};
struct seg {
    pt p, q;
    int id;
    double get_y(double x) const {
        if (abs(p.x - q.x) < EPS)
            return p.y;
        return p.y + (q.y - p.y) * (x - p.x) / (q.x -
p.x);
    }
};
bool intersect1d(double l1, double r1, double l2,
double r2) {
    if (l1 > r1)
        swap(l1, r1);
    if (l2 > r2)
        swap(l2, r2);
    return max(l1, l2) <= min(r1, r2) + EPS;
}
int vec(const pt& a, const pt& b, const pt& c) {
    double s = (b.x - a.x) * (c.y - a.y) - (b.y - a.y)
* (c.x - a.x);
    return abs(s) < EPS ? 0 : s > 0 ? +1 : -1;
}
bool intersect(const seg& a, const seg& b)
{
    return intersect1d(a.p.x, a.q.x, b.p.x, b.q.x) &&
           intersect1d(a.p.y, a.q.y, b.p.y, b.q.y) &&
           vec(a.p, a.q, b.p) * vec(a.p, a.q, b.q) <= 0
&&
           vec(b.p, b.q, a.p) * vec(b.p, b.q, a.q) <=
0;
}
bool operator<(const seg& a, const seg& b)
{
    double x = max(min(a.p.x, a.q.x), min(b.p.x,
b.q.x));
    return a.get_y(x) < b.get_y(x) - EPS;
}
struct event {
    double x;
    int tp, id;
    event() {}
    event(double x, int tp, int id) : x(x), tp(tp),
id(id) {}
    bool operator<(const event& e) const {
        if (abs(x - e.x) > EPS)
            return x < e.x;
        return tp > e.tp;
    }
};
set<seg> s;
vector<set<seg>::iterator> where;
set<seg>::iterator prev(set<seg>::iterator it) {
    return it == s.begin() ? s.end() : --it;
}
set<seg>::iterator next(set<seg>::iterator it) {
    return ++it;
}
pair<int, int> solve(const vector<seg>& a) {
    int n = (int)a.size();
    vector<event> e;
    for (int i = 0; i < n; ++i) {
        e.push_back(event(min(a[i].p.x, a[i].q.x), +1,
i));
        e.push_back(event(max(a[i].p.x, a[i].q.x), -1,
i));
    }
    sort(e.begin(), e.end());
    s.clear();
    where.resize(a.size());
    for (size_t i = 0; i < e.size(); ++i) {
        int id = e[i].id;
        if (e[i].tp == +1) {
            set<seg>::iterator nxt =
s.lower_bound(a[id]), prv = prev(nxt);
            if (nxt != s.end() && intersect(*nxt,
a[id]))
                return make_pair(nxt->id, id);
            if (prv != s.end() && intersect(*prv,
a[id]))
                return make_pair(prv->id, id);
            where[id] = s.insert(nxt, a[id]);
        } else {
            set<seg>::iterator nxt = next(where[id]),
prv = prev(where[id]);
            if (nxt != s.end() && prv != s.end() &&
intersect(*nxt, *prv))
                return make_pair(prv->id, nxt->id);
            s.erase(where[id]);
        }
    }
    return make_pair(-1, -1);}
```

Lines

```
template<class T>
double lineDist(T p, T s, T e) {
    if (s == e) {
        return s.dist(p);
    }
    return fabs((p - s).cross(e - s) / (e - s).dist());
}

template<class T>
pair<int, T> lineInter(T s1, T e1, T s2, T e2) {
    // first = 0 no intersection
    // first = 1 intersection
    // first = -1 infinite intersection
    auto d = (e1 - s1).cross(e2 - s2);
    if (dcmp(d,0) == 0) // if parallel   same line
first = -1 else first = 0
        return {-(s1.cross(e1, s2) == 0), {0,0}};
    auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
    return {1, (s1 * p + e1 * q) / d};
}



template<class T>
bool onSegment(T p,T s, T e) {
    return dcmp(p.cross(s, e) ,0) == 0 && dcmp((s -
p).dot(e - p) , 0) <= 0;
```

```
}


template<class T>
double segDist(T p, T s, T e) {
    if (dcmp((p - s).dot(e - s),0) <= 0)return
s.dist(p);
    if (dcmp((p - e).dot(e - s),0) >= 0)return
e.dist(p);
    return lineDist(p, s, e);
}

template<class T>
pair<int, T> segInter(T s1, T e1, T s2, T e2) {
    // first = 0 no intersection
    // first = 1 intersection`
    // first = -1 infinite intersection
    pair<int, T> ret = lineInter(s1, e1, s2, e2);
    if (ret.first == 0)return ret;
    else if (ret.first == 1) {
        if (onSegment(ret.second, s1, e1) &&
onSegment(ret.second, s2, e2))
            return ret;
        else
            return {0, {0, 0}};
    } else {
        if (onSegment(s1, s2, e2) || onSegment(e1, s2,
e2))
            return {-1, (onSegment(s1, s2, e2) ? s1 :
e1)};
        else if (onSegment(s2, s1, e1) || onSegment(e2,
s1, e1))
            return {-1, (onSegment(s2, s1, e1) ? s2 :
e2)};
        else
            return {0, {0, 0}};
    }
}


template<class T>
T closestOnSegment(T p, T s, T e) {
    if ((p - s).dot(e - s) <= 0) return s;
    else if ((p - e).dot(e - s) >= 0) return e;
    else return p.projectOnLine(s, e);
}

template<class T>
double segSegDist(T s1, T e1, T s2, T e2) {
    if(segInter(s1,e1,s2,e2).first != 0)
        return 0;
    double ret = min({segDist(s1,s2,e2),
segDist(e1,s2,e2), segDist(s2,s1,e1),
segDist(e2,s1,e1)});
    return ret;
}



template<class T>
bool onRay(T p, T s, T e) {
    return dcmp(p.cross(s, e) , 0) == 0 && dcmp((p -
s).dot(e - s) , 0) >= 0;
}

template<class T>
double rayDist(T p, T s, T e) {
    if ((p - s).dot(e - s) <= 0) {
        return s.dist(p);
    }
    return lineDist(p, s, e);
}


template<class T>
pair<int, T> rayInter(T s1, T e1, T s2, T e2) {
    // first = 0 no intersection
    // first = 1 intersection
    // first = -1 infinite intersection
    pair<int, T> ret = lineInter(s1, e1, s2, e2);
    if (ret.first == 0)return ret;
    else if (ret.first == 1) {
        if (onRay(ret.second, s1, e1) &&
onRay(ret.second, s2, e2))
            return ret;
        else
            return {0, {0,0}};
    } else {
        if(onRay(s1, s2, e2) || onRay(s2, s1, e1))
            return {-1,onRay(s1, s2, e2) ? s1:s2};
        else
            return {0, {0,0}};
    }
}

template<class T>
double rayRayDist(T s1, T e1, T s2, T e2) {
    if(rayInter(s1,e1,s2,e2).first != 0)
        return 0;
    double ret = min(rayDist(s1,s2,e2),
rayDist(s2,s1,e1));
    return ret;
}
```

DP and DP Optimizations

LIS

```
int lis(vector<int> const& a) {
    int n = a.size();
    const int INF = 1e9;
    vector<int> d(n+1, INF);
    d[0] = -INF;

    for (int i = 0; i < n; i++) {
        int l = upper_bound(d.begin(), d.end(), a[i]) -
d.begin();
        if (d[l-1] < a[i] && a[i] < d[l])
            d[l] = a[i];
    }

    int ans = 0;
    for (int l = 0; l <= n; l++) {
        if (d[l] < INF)
            ans = l;
    }
    return ans;
}
```

Knuth

```
int solve() {
    int N;
    ... // read N and input
    int dp[N][N], opt[N][N];

    auto C = [&](int i, int j) {
        ... // Implement cost function C.
```

```
    };
    for (int i = 0; i < N; i++) {
        opt[i][i] = i;
        ... // Initialize dp[i][i] according to the
problem
    }
    for (int i = N-2; i >= 0; i--) {
        for (int j = i+1; j < N; j++) {
            int mn = INT_MAX;
            int cost = C(i, j);
            for (int k = opt[i][j-1]; k <= min(j-1,
opt[i+1][j]); k++) {
                if (mn >= dp[i][k] + dp[k+1][j] + cost)
{
                    opt[i][j] = k;
                    mn = dp[i][k] + dp[k+1][j] + cost;
                }
            }
            dp[i][j] = mn; } }
    cout << dp[0][N-1] << endl;
}
```

Divide and Conquer

```
int m, n;
vector<long long> dp_before, dp_cur;

long long C(int i, int j);

// compute dp_cur[l], ... dp_cur[r] (inclusive)
void compute(int l, int r, int optl, int optr) {
    if (l > r)
        return;

    int mid = (l + r) >> 1;
    pair<long long, int> best = {LLONG_MAX, -1};

    for (int k = optl; k <= min(mid, optr); k++) {
        best = min(best, {(k ? dp_before[k - 1] : 0) +
C(k, mid), k});
    }

    dp_cur[mid] = best.first;
    int opt = best.second;

    compute(l, mid - 1, optl, opt);
    compute(mid + 1, r, opt, optr);
}

long long solve() {
    dp_before.assign(n,0);
    dp_cur.assign(n,0);

    for (int i = 0; i < n; i++)
        dp_before[i] = C(0, i);

    for (int i = 1; i < m; i++) {
        compute(0, n - 1, 0, n - 1);
        dp_before = dp_cur;
    }

    return dp_before[n - 1];
}
```