Team Note of **hibye1217**

SolvedAC: hibye1217, CodeForces: hibye1217, AtCoder: hibye1217

Compiled on December 12, 2025

Contents

1 Have You Tried...

1.1 ● Soft Techniques

- Reading the problem once more? Again?
- Thinking whether or not your claim is actually true?
- Representing it as a formular?
- Finding some global monovariant that does not care about the locality?
- Finding some auxilary property?
- Examining small cases, by hands or using computer?
- Checking the possible range of the answer?
- Checking variants of the problem?

- Trying to come up with a counterexample of the claim? Why does it (or does it not) work?
- Thinking about why a certain hard technique you tried failed, and/or convincing yourself that it cannot work?
- Start from scratch, with completely different ideas?
- Stop whining and start implementing?

1.2 ● Hard Techniques

- Any of the algorithms listed below?
- Greedy? or a Network Flow?
- Randomization?
- Square Root Decomposition? Or other bucket size like $B = O(\sqrt{N \log N})$?
- Sparse Table to bypass $O(\log N)$ query time?
- Backtracking? With some pruning?
- Heuristics?
- Using smaller type? (e.g. double instead of long double)?

2 Mathematics - Number Theory

2.1 ● Extended Eucliean Algorithm

The solutions to $ax + by = g$ is $(x + bk/g, y - ak/g)$.

```
pl3 egcd(ll a, ll b){
  if (b == 0){ return {{1, 0}, a}; }
  pl3 p = egcd(b, a%b);
  ll x = p.fr.fr, y = p.fr.sc, g = p.sc;
  ll xx = y, yy = x - a/b*y;
  return {{xx, yy}, g};
} inline ll finv(ll a, ll b){ pl3 p = egcd(a, n); return (p.fr.fr%n+n)%n; }
```

2.2 ● Chinese Remainder Theorem

```
pl2 crt(pl2 f1, pl2 f2){
  if (f1.sc < f2.sc){ swap(f1, f2); }
  ll a1 = f1.fr, m1 = f1.sc; ll a2 = f2.fr, m2 = f2.sc;
  ll g = gcd(m1, m2); ll l = lcm(m1, m2);
  if ((a2-a1)%g != 0){ return {-1, -1}; }
  ll mg1 = m1/g, mg2 = m2/g, ag = (a2-a1)/g;
  ll y = ag *finv(mg1, mg2) %mg2;
  ll x = m1*y + a1; return {(x%l+l)%l, l};
}
```

2.3 ● Möbius Function

```
int phi[X+20], mob[X+20], prr[X+20];
memset(mob, -1, sizeof(mob)); mob[1] = 1;
for (int x = 1; x <= X; x++){ phi[x] = x; }
for (int x = 2; x <= X; x++){
  if (prr[x] == 0){
    for (int a = x; a <= X; a+=x){ prr[a] = x; phi[a] -= phi[a]/x; }
  } int p = prr[x]; mob[x] = (x/p%p == 0 ? 0 : mob[x/p]*mob[p]);
}
```

2.4 ● Miller-Rabin

```
const vector<ll> prr = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
bool prime(ll n){
  if (n <= 40){
    for (int p : prr){ if (n == p){ return 1; } }
    return 0;
  } if (n%2 == 0){ return 0; }
  int s = 0; ll d = n-1; while (d%2 == 0){ s += 1; d /= 2; }
```

```
for (int p : prr){
  ll res = 1, mul = p, bit = d; while (bit){
    if (bit&1){ res = (i128)res*mul % n; }
    mul = (i128)mul*mul % n; bit >>= 1;
  }
  bool chk = (res == 1); for (int r = 0; r < s; r++){
    chk |= (res == n-1); res = (i128)res*res % n;
  } if (!chk){ return 0; }
} return 1;
}
```

2.5 ● Pollard's Rho

```
vector<ll> ans;
inline ll f(ll x, ll c, ll mod){ return ((i128)x*x + c) % mod; }
void factor(ll n){
  if (n == 1){ return; }
  if (n%2 == 0){ ans.push_back(2); return factor(n/2); }
  if (prime(n)){ ans.push_back(n); return; }
  uniform_int_distribution<ll> rnd(1, n);
  ll x1 = rnd(gen); ll x2 = x1;
  ll c = rnd(gen); do{
    x1 = f(x1, c, n); x2 = f(f(x2, c, n), c, n);
  } while (gcd(abs(x1-x2), n) == 1);
  ll g = gcd(abs(x1-x2), n);
  if (g == n){ return factor(n); }
  else{ factor(g); factor(n/g); }
}
```

2.6 ● Power Tower

CUT $> \log_2 m$ is that $a^{\phi(m)} \not\equiv 1 \pmod{m}$ can happen. But for $d \geq \log_2 m$, $a^{d+\phi(m)} \equiv a^d \pmod{m}$.

```
const ll INF = 1e18; // INF > mod^2; might need i128
ll fpow(ll mul, ll bit, ll mod, ll lim = INF){
  ll res = 1; while (bit){
    if (bit&1){ res = min(res*mul, lim) % mod; }
    mul = mul*mul % mod; bit >>= 1;
  } return res;
}

const int CUT = 100, LEN = 4; // CUT >= log2(mod); A = [2 for 1..LEN] > CUT
int n; int arr[1000020];
int solve(int idx){
  int val = 1; for (int i = n; i >= idx; i--){
    val = fpow(arr[i], val, (CUT+1)*(CUT+2), CUT+1);
  } return val;
}
int solve(int idx, int mod){
  if (mod == 1){ return 0; }
  if (idx == n){ return arr[idx] % mod; }
  if (n-idx+1 <= LEN){
    if (solve(idx+1) <= CUT){ return fpow(arr[idx], solve(idx+1), mod); }
  }
  int m = phi[mod];
  return fpow(arr[idx], solve(idx+1, m) + CUT*m, mod);
}
```

3 Mathematics - Combinatorics

3.1 ● Lucas Theorem

$\binom{n}{r} \equiv \binom{n \bmod p}{r \bmod p} \times \binom{n/p}{r/p} \pmod{p}$.

```
int ncr(int n, int r){
  if (0 > r || r > n){ return 0; } if (r == 0){ return 1; }
  if (0 <= n && n < mod && 0 <= r && r < mod){ return fac[n] * inv[r] % mod * inv[n-r] %
  mod; }
  return ncr(n%mod, r%mod) * ncr(n/mod, r/mod) % mod;
}
```

3.2 ● Well-known Sequences

Derangement is the number of permutation with $P_i \neq i$ for all i.
$D_0 = 1$; $D_1 = 0$; $D_n = (n-1)(D_{n-1} + D_{n-2})$.
Catalan Number is the number of valid Regular Bracket Sequence with n pairs.
$C_0 = 1$; $C_{n+1} = (4n+2)C_n/(n+2)$.
Stirling Number of the First Kind is the number of permutation with n elements and k cycles.
$s_{0,0} = 1$; $s_{n,0} = s_{0,k} = 0$; $s_{n,k} = (n-1)s_{n-1,k} + s_{n-1,k-1}$.
Stirling Number of the Second Kind is the number of ways to partition a set of n different objects unto k non-empty subsets.
$S_{0,0} = 1$; $S_{n,0} = S_{0,k} = 0$; $S_{n,k} = kS_{n-1,k} + S_{n-1,k-1}$.
Bell Number is the number of ways to partition a set of n different objects.
$B_n = \sum_{k=0}^{n} S_{n,k}$.
Partition Number is the number of ways to partition n into k parts.
$p_k(n) = p_{k-1}(n-1) + p_k(n-k)$. $p(n) = \sum_{k=1}^{n} p_k(n)$.

3.3 ● Lindström-Gessel-Viennot Lemma

Let $w(P)$ be the product of the weights of the edges of the path P. Let D be the matrix with $D_{i,j}$ be the sum of $w(P)$s with $i \to j$.

Then, $\det(D) = \sum_{(P_1,\ldots,P_N): \, S \to T} \text{sign}(\sigma(P)) \prod_{i=1}^{N} w(P_i)$, where $(P_1, \ldots, P_N)$ is the N *non-intersecting* paths, with $P_i = A_i \to B_{\sigma(i)}$.

When every edge have weight 1, $w(P) = 1$. Therefore, the lemma above will just find the number of possible non-intersecting path (with parity of the permutation).

3.4 ● Generating Function

Let $A(x)$ be the GF of the sequence a_i. a is 0-indexed.
- If A is OGF, $a_n = A^{(n)}(0)/n!$.
- If A is EGF, $a_n = A^{(n)}(0)$.
- If $C(x) = kA(x) + lB(x)$, then $c_i = ka_i + lb_i$.
- If $B(x) = xA'(x)$, then $b_i = ia_i$.
- Let $\langle b_0, b_1, \ldots, b_{k-1}, b_k, b_{k+1}, \ldots \rangle = \langle 0, 0, \ldots, 0, a_0, a_1, \ldots \rangle$.
    - If A and B are both OGF, then $B(x) = x^k A(x)$.
    - If A and B are both EGF, then $B(x) = \int^{(k)} A(x)d^k x$.
- Let $\langle b_0, b_1, \ldots \rangle = \langle a_k, a_{k+1}, \ldots \rangle$.
    - If A and B are both OGF, then $B(x) = (A(x) - a_0 - a_1 x - \cdots - a_{k-1}x^{k-1})/x^k$.
    - If A and B are both EGF, then $B(x) = \frac{d^k}{d^k x} A(x)$.
- Let $C(x) = A(x)B(x)$.
    - If A and B are both OGF, then $c_i = \sum_{k=0}^{i} a_k b_{i-k} = (a*b)_i$.
    - If A and B are both EGF, then $c_i = \sum_{k=0}^{i} \binom{i}{k} a_k b_{i-k}$.
- Let k be a positive integer, and let $B(x) = A(x)^k$.
    - If A and B are both OGF, then $b_n = \sum_{i_1+i_2+\cdots+i_k=n} a_{i_1} a_{i_2} \cdots a_{i_k}$.
    - If A and B are both EGF, then $b_n = \sum_{i_1+i_2+\cdots+i_k=n} \frac{n!}{i_1! i_2! \cdots i_k!} a_{i_1} a_{i_2} \cdots a_{i_k}$.
- If A and B are both OGF with $B(x) = \frac{A(x)}{1-x}$, then $b_i = a_0 + a_1 + \cdots + a_i$.
- $(1+x)^k = \sum_{n=0}^{\infty} \binom{k}{n} x^n$.
- $\frac{1}{(1-x)^k} = \sum_{n=0}^{\infty} \binom{n+k-1}{n} x^n = \sum_{n=0}^{\infty} {}_k H_n x^n$.

4   Mathematics - Algebra

4.1   ● Modeling - 2-SAT - Operations

- $(a \vee b) \rightarrow (\neg a \implies b) \wedge (\neg b \implies a)$.
- $(a \wedge b) \rightarrow (a \vee a) \wedge (b \vee b)$.
- $(a\overline{\wedge}b) \rightarrow \neg(a \wedge b) \rightarrow (\neg a \vee \neg b)$.
- $(a\overline{\vee}b) \rightarrow (\neg a \wedge \neg b)$.
- $(a \oplus b) \rightarrow (a \vee b) \wedge (\neg a \vee \neg b)$.
- $(a\overline{\oplus}b) \rightarrow (a \vee \neg b) \wedge (\neg a \vee b)$.

4.2   ● Modeling - 2-SAT - At most 1

3 variables (a, b, c) can be done with $(\neg a \vee \neg b) \wedge (\neg b \vee \neg c) \wedge (\neg c \vee \neg a)$. This can be expanded to make $\mathcal{O}(N^2)$ clauses.

N variables $(x_1, x_2, \ldots, x_N)$ can be done with the following method. Let y_i be true if one of $x_1, x_2, \ldots, x_i$ is true. This can be modeled as following:

- $x_i \implies y_i$.
- $y_i \implies y_{i+1}$.
- $y_i \implies \neg x_{i+1}$.

4.3   ● Modeling - Minimizing Quadratic Pseudo-Boolean Function

Given N boolean variables $x_1, x_2, \ldots, x_N$, minimize the cost $f(x_1, x_2, \ldots, x_N) = c + \sum_i b_i(x_i) + \sum_{i<j} a_{i,j}(x_i, x_j)$. Function a must satisfies the condition $a(0,0) + a(1,1) \leq a(0,1) + a(1,0)$.

This can be modeled as following. Let there be source s, sink t, and N vertices $x_1, x_2, \ldots, x_N$.

- c is trivial.
- $b_i(x_i = 0)$ can be modeled as $x_i \rightarrow t$ with weight $b_i(0)$.
- $b_i(x_i = 1)$ can be modeled as $s \rightarrow x_i$ with weight $b_i(1)$.
- $a_{i,j}(x_i, x_j)$ can be modeled as following:
  - $s \rightarrow v_i$ with weight $a_{i,j}(1,0)$.
  - $v_i \rightarrow t$ with weight $a_{i,j}(0,0)$.
  - $s \rightarrow v_j$ with weight $a_{i,j}(1,1) - a_{i,j}(1,0)$.
  - $v_i \rightarrow v_j$ with weight $a_{i,j}(0,1) + a_{i,j}(1,0) - a_{i,j}(0,0) - a_{i,j}(1,1)$.

4.4   ● Fast Fourier Transform & Number Theoretic Transform

```
void dft(vector<cpl>& arr, bool inv = false){ int n = arr.size();
  for (int j=0, i=1; i < n; i++){ int bit = n>>1;
    while (j & bit){ j ^= bit; bit >>= 1; } j ^= bit;
    if (i < j){ swap(arr[i], arr[j]); }
  }
  for (int l = 1; l < n; l <<= 1){
    if (complex){
      ld ang = PI / l; if (inv){ ang *= -1; }
      cpl w = {cos(ang), sin(ang)};
    }
    else{
      ll w = fpow(r, (mod-1)/(2*l)); if (inv){ w = finv(w); }
      // r is primitive root of mod. 998244353 -> 3
    }
    for (int i = 0; i < n; i += l<<1){
      cpl wp = 1; for (int j = 0; j < l; j++){
        cpl a = arr[i+j], b = arr[i+j + l] * wp;
        arr[i+j] = a+b; arr[i+j + l] = a-b; wp *= w;
      }
    }
  }
  if (inv){ for (int i = 0; i < n; i++){ arr[i] /= n; } }
}
void mul(vector<cpl>& arr, vector<cpl>& brr){
```

```
  int n = max(arr.size(), brr.size()); n = bitp(n) * 2; // power of 2
  arr.resize(n); brr.resize(n);
  dft(arr); dft(brr);
  for (int i = 0; i < n; i++){ arr[i] *= brr[i]; }
  dft(arr, -1);
}
```

4.5   ● Fast Welsh-Hadamard Transform

```
void fwht(vector<ll>& arr, bool inv = false){
  int n = arr.size();
  for (int l = 1; l < n; l*=2){
    for (int p = 0; p < n; p+=l*2){
      for (int i = 0; i < l; i++){
        ll a = arr[p+i], b = arr[p+l+i];
        if (XOR){ arr[p+i] = (a+b)%mod; arr[p+l+i] = (a-b+mod)%mod; }
        if (AND){ arr[p+i] = (a + (inv ? -b : b) + mod) % mod; }
        if (OR){ arr[p+l+i] = (b + (inv ? -a : a) + mod) % mod; }
      }
    }
  } if (inv && XOR){ for (int i = 0; i < n; i++){ arr[i] *= invn; } }
}
```

4.6   ● Gauss-Jordan Elimination

Gauss-Jordan Elimination is not stable. It is recommended that you use partial pivoting, which always choose the row that has the maximum absolute value on pivoting column.

For each column c: Let r be the argmax of $|A_{r,c}|$, $r \geq p$. swap r-th and p-th row, then apply $A_{r,c} = 1$, then apply $A_{i,c} = 0$ for all $i \neq p$.

4.7   ● Floor Sum of Arithmetic Progression

Let $f(a, b, c, n) = \sum_{x=0}^{n} \left\lfloor \frac{ax+b}{c} \right\rfloor$.

If $a \geq c$ or $b \geq c$, then $f(a, b, c, n) = \frac{n(n+1)}{2}\left\lfloor\frac{a}{c}\right\rfloor + (n+1)\left\lfloor\frac{b}{c}\right\rfloor + f(a \bmod c, b \bmod c, c, n)$.

Otherwise, we have $f(a, b, c, n) = nm - f(c, c - b - 1, a, m - 1)$ where $m = \left\lfloor\frac{an+b}{c}\right\rfloor$.

4.8   ● Linear Programming

If possible, minimize the number of variables.

Primal: Maximize $\mathbf{c}^T\mathbf{x}$ subject to $A\mathbf{x} \leq \mathbf{b}$ and $\mathbf{x} \geq 0$.

Dual: Maximize $-\mathbf{b}^T\mathbf{y}$ subject to $-A^T\mathbf{y} \leq -\mathbf{c}$ and $\mathbf{y} \geq 0$.

Time Complexity: $O(NM)$ per pivot, $O(2^N)$ at worst. $O(N^3)$?

```
int n, m; vector<int> slk, piv;
matrix<ld> mat;
void init(const matrix<ld> &a, const vector<ld> &b, const vector<ld> &c){
  n = c.size(); m = b.size();
  slk = vector<int>(n+1); piv = vector<int>(m);
  mat = matrix<ld>(m+2, vector<ld>(n+2));
  for (int i = 0; i < m; i++){
    for (int j = 0; j < n; j++){ mat[i][j] = a[i][j]; }
  }
  for (int i = 0; i < m; i++){
    piv[i] = n+i; mat[i][n] = -1; mat[i][n+1] = b[i];
  }
  for (int j = 0; j < n; j++){
    slk[j] = j; mat[m][j] = -c[j];
  } slk[n] = -1; mat[m+1][n] = 1;
}
void pivot(int r, int s){
```

```cpp
    ld *a = mat[r].data(); ld inva = 1/a[s];
    for (int i = 0; i < m+2; i++){
      if (i == r || abs(mat[i][s]) <= eps){ continue; }
      ld *b = mat[i].data(); ld invb = b[s]*inva;
      for (int j = 0; j < n+2; j++){ b[j] -= a[j]*invb; }
      b[s] = a[s]*invb;
    }
    for (int j = 0; j < n+2; j++){ if (j != s){ mat[r][j] *= inva; } }
    for (int i = 0; i < m+2; i++){ if (i != r){ mat[i][s] *= -inva; } }
    mat[r][s] = inva; swap(piv[r], slk[s]);
}
bool simplex(int typ){
    int x = m+typ-1; while (1){
      int s = -1, r = -1; for (int j = 0; j < n+1; j++){
        if (slk[j] == -typ){ continue; }
        if (s == -1 || mkp(mat[x][j], slk[j]) < mkp(mat[x][s], slk[s])){ s = j; }
      } if (mat[x][s] >= -eps){ return 1; }
      for (int i = 0; i < m; i++){
        if (mat[i][s] <= eps){ continue; }
        if (r == -1 || mkp(mat[i][n+1]/mat[i][s], piv[i]) < mkp(mat[r][n+1]/mat[r][s],
        piv[r])){ r = i; }
      } if (r == -1){ return 0; }
      pivot(r, s);
    }
}
pair<ld, vector<ld>> solve(){
    int r = 0; for (int i = 1; i < m; i++){
      if (mat[i][n+1] < mat[r][n+1]){ r = i; }
    }
    if (mat[r][n+1] < -eps){
      pivot(r, n); if (!simplex(2)){ return {-inf, {}}; }
      if (mat[m+1][n+1] < -eps){ return {-inf, {}}; }
      for (int i = 0; i < m; i++){
        if (piv[i] != -1){ continue; }
        int s = 0; for (int j = 1; j < n+1; j++){
          if (s == -1 || mkp(mat[i][j], slk[j]) < mkp(mat[i][s], slk[s])){ s = j; }
        } pivot(i, s);
      }
    }
    bool chk = simplex(1);
    vector<ld> x(n); for (int i = 0; i < m; i++){
      if (piv[i] < n){ x[piv[i]] = mat[i][n+1]; }
    } return {chk ? mat[m][n+1] : inf, x};
}
```

5   Mathematics - Geometry

Note that (`first`, `second`) is (y, x). Polygon is always given as counter-clockwise order, with no 3 points in a line.

5.1   ● Line Intersection

```cpp
inline ll cross(const pl2& v1, const pl2& v2){ return v1.sc*v2.fr - v2.sc*v1.fr; }
inline pl2 ltov(const pl4& l){ return {l.sc.fr-l.fr.fr, l.sc.sc-l.fr.sc}; }
int intersect(pl4 l1, pl4 l2){
    pl2 p1 = l1.fr, p2 = l1.sc; pl2 p3 = l2.fr, p4 = l2.sc;
    int c123 = ccw(p1, p2, p3), c124 = ccw(p1, p2, p4);
    int c341 = ccw(p3, p4, p1), c342 = ccw(p3, p4, p2);
    if (c123 == 0 && c124 == 0){
```

```cpp
      if (p1 > p2){ swap(p1, p2); } if (p3 > p4){ swap(p3, p4); }
      if (p2 < p3 || p4 < p1){ /* No Intersection */ }
      if (p2 == p3 || p4 == p1){ /* Endpoint */ }
      /* Infinitely Many */
    }
    int c12 = c123*c124, c34 = c341*c342;
    if (c12 > 0 || c34 > 0){ /* No Intersection */ }
    if (c12 == 0 || c34 == 0){ /* Endpoint */ }
    /* Mid-line */
}
pd2 intersection(const pl4& l1, const pl4& l2){
    pl2 p1 = l1.fr, p2 = l1.sc; pl2 p3 = l2.fr, p4 = l2.sc;
    if (p1 == p3 || p1 == p4){ return p1; } if (p2 == p3 || p2 == p4){ return p2; }
    pl2 v1 = ltov(l1), v2 = ltov(l2); pl2 d = {p3.fr-p1.fr, p3.sc-p1.sc};
    ld a = (ld)cross(d, v2) / cross(v1, v2);
    return {p1.fr + v1.fr*a, p1.sc + v1.sc*a};
}
```

5.2   ● Convex Hull

```cpp
for (int i = 1; i <= n; i++){
    if (arr[1] > arr[i]){ swap(arr[1], arr[i]); }
} pl2 p0 = arr[1]; sort(arr+2, arr+n+1, [&p0](const pl2& p1, const pl2& p2){
    int res = ccw(p0, p1, p2); if (res != 0){ return res > 0; }
    return dis(p0, p1) < dis(p0, p2);
});
vector<pl2> stk; for (int i = 1; i <= n; i++){
    pl2 p3 = arr[i]; while (stk.size() >= 2){
      pl2 p1 = *prev(prev(stk.end())), p2 = *prev(stk.end());
      if (ccw(p1, p2, p3) <= 0){ stk.pop_back(); } else{ break; } // < 0 if linear
    } stk.push_back(p3);
}
for (int i = n-1; i > 1; i--){ // if linear only
    if (ccw(stk.front(), arr[i], stk.back()) == 0){ stk.push_back(arr[i]); }
    else{ break; }
}
```

5.3   ● Point in Convex Polygon

```cpp
int st = 2, ed = n; while (st+1 <= ed-1){
    int mid = st+ed >> 1;
    if (ccw(arr[1], arr[mid], p) == +1){ st = mid; } else{ ed = mid; }
} pl2 p1 = arr[1], p2 = arr[st], p3 = arr[st+1];
int r12 = ccw(p1, p2, p), r23 = ccw(p2, p3, p), r31 = ccw(p3, p1, p);
if (r12 < 0 || r23 < 0 || r31 < 0){ /* Outside */ }
else{
    if (r23 == 0 || r12 == 0 && st == 2 || r31 == 0 && st == n-1){ /* Line */ }
    else{ /* Inside */ }
}
```

5.4   ● Point in Non-convex Polygon

If the point is on the line, then whatever do what the problem says.

OTherwise, send the straight line (x, y) to $(x + X, y + X + 1)$ with X being sufficiently large integer. Then, count the number of intersections.

5.5   ● Rotating Calipers

```cpp
int j = 0; for (int i = 0; i < sp; i++){
    while (j < sp){
      int ip = (i+1) % sp, jp = (j+1) % sp;
```

```cpp
    pl2 l = { arr[ip].fr - arr[i].fr, arr[ip].sc - arr[i].sc };
    pl2 r = { arr[jp].fr - arr[j].fr, arr[jp].sc - arr[j].sc };
    if (ccw({0, 0}, l, r) < 0){ break; }
    ans = max(ans, dist(i, j)); j += 1;
  } if (j < sp){ ans = max(ans, dist(i, j)); }
}
```

5.6 ● Tangent Line on a Convex Polygon

Let i_1 and i_2 be the result of `tangent(p, -1)` and `tangent(p, +1)` respectively. The visible vertices from p is P_{i_1} to P_{i_2}. Note that the index is cyclic i.e. if $i_1 > i_2$ then the visible vertices are P_{i_1} to P_n and P_1 to P_{i_2}.

If there are multiple tangent line on the single direction, it returns the nearest one. Test `prv` and `nxt` resp. for farthest in dir -1 and $+1$, resp.

```cpp
int n; pl2 arr[100020];
inline int nxt(int i){ return i==n ? 1 : i+1; }
inline int prv(int i){ return i==1 ? n : i-1; }
inline bool locate(const pl2& p, const pl2& p1, const pl2& p2, const pl2& p3, int dir){
  return dir*ccw(p, p1, p2) <= 0 && dir*ccw(p, p2, p3) >= 0;
}
int fix(const pl2& p, int idx, int dir){
  if (dir > 0 && ccw(p, arr[idx], arr[prv(idx)]) == 0){
    pl2 p1 = arr[idx], p2 = arr[prv(idx)]; if (p1 > p2){ swap(p1, p2); }
    if (p1 <= p && p <= p2){ return idx; } else{ return prv(idx); }
  }
  if (dir < 0 && ccw(p, arr[idx], arr[nxt(idx)]) == 0){
    pl2 p1 = arr[idx], p2 = arr[nxt(idx)]; if (p1 > p2){ swap(p1, p2); }
    if (p1 <= p && p <= p2){ return idx; } else{ return nxt(idx); }
  }
  return idx;
}
int tangent(const pl2& p, int dir){
  if (locate(p, arr[prv(1)], arr[1], arr[nxt(1)], dir)){ return fix(p, 1, dir); }
  int st = 1, ed = n+1; while (st+1 <= ed-1){
    int mid = st+ed >> 1;
    if (locate(p, arr[prv(mid)], arr[mid], arr[nxt(mid)], dir)){ return fix(p, mid, dir); }
    if (dir*ccw(p, arr[st], arr[nxt(st)]) < 0){
      if (dir*ccw(p, arr[mid], arr[nxt(mid)]) > 0){ ed = mid; }
      else if (dir*ccw(p, arr[mid], arr[st]) > 0){ st = mid; }
      else{ ed = mid; }
    } else{
      if (dir*ccw(p, arr[mid], arr[nxt(mid)]) < 0){ st = mid; }
      else if (dir*ccw(p, arr[mid], arr[st]) < 0){ st = mid; }
      else{ ed = mid; }
    }
  }
  if (st != 1 && locate(p, arr[prv(st)], arr[st], arr[nxt(st)], dir)){ return fix(p, st,
  dir); }
  if (ed != n+1 && locate(p, arr[prv(ed)], arr[ed], arr[nxt(ed)], dir)){ return fix(p, ed,
  dir); }
  return -1;
}
```

5.7 ● Minimum Enclosing Circle

```cpp
inline pd2 circumcenter(const pd2& p1, const pd2& p2, const pd2& p3){
  ld d1 = dis(p2, p3), d2 = dis(p3, p1), d3 = dis(p1, p2);
  d1 *= d1; d2 *= d2; d3 *= d3;
  ld w1 = d1*(d2+d3-d1), w2 = d2*(d3+d1-d2), w3 = d3*(d1+d2-d3); ld w = w1+w2+w3;
  w1 /= w; w2 /= w; w3 /= w;
  return pd2(w1*p1.fr + w2*p2.fr + w3*p3.fr, w1*p1.sc + w2*p2.sc + w3*p3.sc);
}

shuffle(arr+1, arr+n+1, gen);
pd2 o = {0, 0}; ld r = 0; for (int i = 1; i <= n; i++){
  if (inside(o, r, arr[i])){ continue; }
  o = arr[i]; r = 0;
  for (int j = 1; j < i; j++){
    if (inside(o, r, arr[j])){ continue; }
    o = {(arr[i].fr+arr[j].fr)/2, (arr[i].sc+arr[j].sc)/2}; r = dis(o, arr[i]);
    for (int k = 1; k < j; k++){
      if (inside(o, r, arr[k])){ continue; }
      o = circumcenter(arr[i], arr[j], arr[k]); r = dis(o, arr[i]);
    }
  }
} // center O, radius R.
```

5.8 ● Bulldozer Trick

`update(pi)`: updates pi-th element on the data structure.

```cpp
sort(arr+1, arr+n+1); for (int i = 1; i <= n; i++){ pos[i] = i; } // min y first
for (int i = 1; i <= n; i++){
  for (int j = i+1; j <= n; j++){
    pl2 p = {arr[i].fr.fr-arr[j].fr.fr, arr[i].fr.sc-arr[j].fr.sc};
    p = {p.sc, -p.fr};
    if (p.sc < 0 || p.sc==0 && p.fr > 0){ p = {-p.fr, -p.sc}; }
    lin.push_back({p, {i, j}}); // p as directional vector, min y first
  }
}
sort(lin.begin(), lin.end(), [](pl2i2 p1, pl2i2 p2){
  int res = ccw(pl2(0, 0), p1.fr, p2.fr); // ccw, if same then index
  if (res != 0){ return res > 0; } else{ return p1.sc < p2.sc; }
});
int ans = solve(state); for (pl2i2 p : lin){
  int i = p.sc.fr, j = p.sc.sc;
  int pi = pos[i], pj = pos[j];
  swap(arr[pi], arr[pj]); swap(pos[i], pos[j]); update(pi); update(pj);
  ans = ans or newState;
}
```

5.9 ● Halfplane Intersection

`v` contains $(y_1, x_1), (y_2, x_2)$. Left side of $\mathbf{v} = p_2 - p_1$ is the halfplane. Counterclockwise convex polygon $P = \langle p_1, p_2, \ldots, p_n \rangle$ will have intersection equivalent to P.

```cpp
const pl2 O = {0, 0};
const ld eps = 1e-9;
const ll INF = 1e9;
inline bool eqf(ld a, ld b){ return abs(a-b)/max<ld>({abs(a),abs(b),1}) <= eps; }
inline int sgn(ll x){ return (x > 0) - (x < 0); }
inline int sgn(ld x){ return eqf(x, 0) ? 0 : (x > 0) - (x < 0); }

inline int ccw(const pl2& p1, const pl2& p2, const pl2& p3){
  ll a = p1.sc*p2.fr + p2.sc*p3.fr + p3.sc*p1.fr;
  ll b = p1.sc*p3.fr + p3.sc*p2.fr + p2.sc*p1.fr;
  return sgn(a-b);
}
inline int ccwd(const pd2& p1, const pd2& p2, const pd2& p3){
  ld a = p1.sc*p2.fr + p2.sc*p3.fr + p3.sc*p1.fr;
```

```
  ld b = p1.sc*p3.fr + p3.sc*p2.fr + p2.sc*p1.fr;
  return sgn(a-b);
}
inline ld area(const pd2& p1, const pd2& p2, const pd2& p3){
  ld a = p1.sc*p2.fr + p2.sc*p3.fr + p3.sc*p1.fr;
  ld b = p1.sc*p3.fr + p3.sc*p2.fr + p2.sc*p1.fr;
  return abs(a-b)/2;
}
inline ll cross(const pl2& v1, const pl2& v2){ return v1.sc*v2.fr - v2.sc*v1.fr; }

inline pl2 ltov(const pl4& l){ return pl2{l.sc.fr-l.fr.fr, l.sc.sc-l.fr.sc}; }
pair<bool, pd2> intersection(const pl4& l1, const pl4& l2){
  pl2 v1 = ltov(l1), v2 = ltov(l2);
  if (ccw(0, v1, v2) == 0){ return {0, {0, 0}}; }
  pl2 p1 = l1.fr, p2 = l2.fr; pl2 dp = {p2.fr-p1.fr, p2.sc-p1.sc};
  ld a = (ld)cross(dp, v2) / cross(v1, v2);
  return {1, {p1.fr + v1.fr*a, p1.sc + v1.sc*a}};
}
bool out(const pd2& p, const pl4& l){ return ccwd(l.fr, l.sc, p) <= 0; }
bool out(const pl4& l1, const pl4& l2, const pl4& l){
  auto res = intersection(l1, l2); if (!res.fr){ return 0; }
  return out(res.sc, l);
}

vector<pd2> halfplane(vector<pl4> v){
  {
    pl2 p1 = {-INF, -INF}, p2 = {-INF, INF};
    pl2 p3 = {INF, INF}, p4 = {INF, -INF};
    v.push_back({p1, p2}); v.push_back({p2, p3});
    v.push_back({p3, p4}); v.push_back({p4, p1});
  }
  sort(v.begin(), v.end(), [](const pl4& l1, const pl4& l2){
    pl2 v1 = ltov(l1), v2 = ltov(l2);
    if ((v1 > 0) != (v2 > 0)){ return (v1 > 0) > (v2 > 0); }
    return ccw(0, v1, v2) > 0;
  });
  deque<pl4> dq; for (const pl4& l : v){
    while (dq.size() >= 2){
      int dql = dq.size();
      if (out(dq[dql-2], dq[dql-1], l)){ dq.pop_back(); }
      else{ break; }
    }
    while (dq.size() >= 2){
      int dql = dq.size();
      if (out(dq[0], dq[1], l)){ dq.pop_front(); }
      else{ break; }
    }
    int dql = dq.size(); if (dql >= 1){
      pl2 v1 = ltov(l), v2 = ltov(dq[dql-1]);
      if (ccw(0, v1, v2) == 0){
        if ((v1 > 0) == (v2 > 0)){
          pl2 p1 = l.fr;
          if (!out(p1, dq[dql-1])){ dq.pop_back(); dq.push_back(l); }
          continue;
        } else{ return {}; }
      }
    } if (dql < 2 || !out(dq[dql-1], l, dq[0])){ dq.push_back(l); }
```

```
} int dql = dq.size(); if (dql < 3){ return {}; }
  vector<pd2> ans; for (int i = 0; i < dql; i++){
    int j = i+1; if (j == dql){ j = 0; }
    pl4 l1 = dq[i], l2 = dq[j]; auto res = intersection(l1, l2);
    if (res.fr){ ans.push_back(res.sc); }
  } return ans;
}
```

5.10 ● Shamos-Hoey

Simple Polygon version. For Full non-intersection follow the comments instead.

```
inline ld f(const pl4& l, ll x){
  return (ld)(l.sc.fr-l.fr.fr)/(l.sc.sc-l.fr.sc) * (x-l.fr.sc) + l.fr.fr;
}
ll ptr; bool cmp4(const pl4& l1, const pl4& l2){
  ld p1 = f(l1, ptr), p2 = f(l2, ptr);
  return p1 < p2;
} bool cmp(const pl5& p1, const pl5& p2){ return cmp4(p1.fr, p2.fr); }

// usual No Intersection / Yes Intersection if fully nonintersected.
int intersect(const pl4& l1, const pl4& l2){
  pl2 p1 = l1.fr, p2 = l1.sc, p3 = l2.fr, p4 = l2.sc;
  int c123 = ccw(p1, p2, p3), c124 = ccw(p1, p2, p4);
  int c341 = ccw(p3, p4, p1), c342 = ccw(p3, p4, p2);
  if (c123 == 0 && c124 == 0){
    if (p1 > p2){ swap(p1, p2); } if (p3 > p4){ swap(p3, p4); }
    if (p2 <= p3 || p4 <= p1){ return 0; } else{ return 1; }
  }
  if (p1 == p3 || p2 == p4){ return p1 == p3 ? -1 : -2; }
  return c123 != c124 && c341 != c342;
} int intersect(const pl5& l1, const pl5& l2){ return intersect(l1.fr, l2.fr); }

pl4 prr[100020]; pl4 pos[200020];
multiset<pl5, decltype(&cmp)> lin(cmp);
pi2 solve(int n){
  sort(pos+1, pos+n+n+1); lin.clear();
  for (int i = 1; i <= n+n; i++){
    ll x = pos[i].fr.fr; bool ed = pos[i].fr.sc;
    ll y = pos[i].sc.fr; int idx = pos[i].sc.sc;
    if (prr[idx].fr.sc == prr[idx].sc.sc){ continue; }
    pl5 l = {prr[idx], idx}; ptr = x;
    if (!ed){
      // auto it = lin.insert(l);
      // if (it != lin.begin()){ if (intersect(*it, *prev(it))){ return {it->sc,
      prev(it)->sc}; } }
      // if (next(it) != lin.end()){ if (intersect(*it, *next(it))){ return {it->sc,
      next(it)->sc}; } }
      bool flg = 0; auto it = lin.insert(l); again:
      if (it != lin.begin()){
        int res = intersect(*it, *prev(it));
        if (res == 1){ return {it->sc, prev(it)->sc}; }
        if (res == -1){
          lin.erase(it); ptr += 1; it = lin.insert(l); ptr -= 1;
          if (!flg){ flg = 1; goto again; }
        }
      }
      if (next(it) != lin.end()){
        int res = intersect(*it, *next(it));
```

```
          if (res == 1){ return {it->sc, next(it)->sc}; }
          if (res == -1){
            lin.erase(it); ptr += 1; it = lin.insert(l); ptr -= 1;
            if (!flg){ flg = 1; goto again; }
          }
        }
    } else{
      // auto it = lin.lower_bound(l);
      // if (it != lin.begin() && next(it) != lin.end()){
      //   if (intersect(*prev(it), *next(it))){ return {prev(it)->sc, next(it)->sc}; }
      // } lin.erase(it);
      auto it = lin.lower_bound(l); if (*it != l){
        ptr -= 1; it = lin.lower_bound(l); ptr += 1;
      }
      if (it != lin.begin() && next(it) != lin.end()){
        if (intersect(*prev(it), *next(it)) == 1){ return {prev(it)->sc, next(it)->sc}; }
      } lin.erase(it);
    }
  } return {-1, -1};
}

pl4 arr[100020]; // (2y, 2x). // (y, x) if fully nonintersected.
for (int t = 1; t <= 3; t++){
  for (int i = 1; i <= n; i++){
    pl2 p1 = arr[i].fr, p2 = arr[i].sc;
    if (t == 2){ swap(p1.fr, p1.sc); swap(p2.fr, p2.sc); }
    if (t == 3){ p1 = pl2(p1.fr+p1.sc, p1.fr-p1.sc); p2 = pl2(p2.fr+p2.sc, p2.fr-p2.sc); }
    if (pl2(p1.sc, p1.fr) > pl2(p2.sc, p2.fr)){ swap(p1, p2); } prr[i] = {p1, p2};
    pos[2*i-1] = { {p1.sc, 0}, {p1.fr, i} };
    pos[2*i] = { {p2.sc, -1}, {p2.fr, i} }; // +1 if fully nonintersected.
  } pi2 res = solve(n); if (res != pi2(-1, -1)){ cout << "NO"; goto done; }
} cout << "YES";
```

5.11 ● Faces of the Planar Graph

As always, the polygon are in counterclockwise, Except the outer face which is inner. If you want to include the inner face, remove val > 0.

```
pl2 arr[100020]; vector<int> adj[100020];
vector<int> chk[100020];

vector< vector<int> > res;

vector<int> vtx;
void dfs(int u, int ui){
  chk[u][ui] = 1; if (vtx.size() >= 2 && *prev(prev(vtx.end())) == u){
    vtx.pop_back();
  } else{ vtx.push_back(u); }
  int v = adj[u][ui]; int vi = lower_bound(adj[v].begin(), adj[v].end(), u, [&v](int w1, int
  w2){
    if ((arr[v] < arr[w1]) != (arr[v] < arr[w2])){ return (arr[v] < arr[w1]) > (arr[v] <
    arr[w2]); }
    int res = ccw(arr[v], arr[w1], arr[w2]); return res > 0;
  }) - adj[v].begin(); int vl = adj[v].size();
  vi = (vi == 0 ? vl-1 : vi-1); int w = adj[v][vi];
  int p = (vtx.size() >= 2 ? *prev(prev(vtx.end())) : -1);
  if (!chk[v][vi]){ dfs(v, vi); }
}
```

```
for (int v = 1; v <= n; v++){
  chk[v].resize(adj[v].size()); for (int& x : chk[v]){ x = 0; }
  sort(adj[v].begin(), adj[v].end(), [&v](int w1, int w2){
    if ((arr[v] < arr[w1]) != (arr[v] < arr[w2])){ return (arr[v] < arr[w1]) > (arr[v] <
    arr[w2]); }
    int res = ccw(arr[v], arr[w1], arr[w2]); return res > 0;
  });
}
for (int v = 1; v <= n; v++){
  int vl = adj[v].size(); for (int vi = 0; vi < vl; vi++){
    if (chk[v][vi]){ continue; }
    vtx.clear(); dfs(v, vi); if (!vtx.empty()){
      int vl = vtx.size(); ll val = 0; for (int i = 0; i < vl; i++){
        int j = (i+1 == vl ? 0 : i+1);
        int p1 = vtx[0], p2 = vtx[i], p3 = vtx[j];
        val += area2(arr[p1], arr[p2], arr[p3]);
      } if (val > 0){ res.push_back(vtx); }
    }
  }
}
```

5.12 ● Polygon Raycast

```
inline ll dot(pl2 p1, pl2 p2){ return p1.sc*p2.sc + p1.fr*p2.fr; }
inline ll cross(pl2 p1, pl2 p2){ return p1.sc*p2.fr - p1.fr*p2.sc; }

inline pl2 norm(pl2 p){
  if (p == pl2(0, 0)){ return p; }
  ll g = gcd(p.fr, p.sc); p.fr /= g; p.sc /= g;
  if (p.sc < 0){ p = {-p.fr, -p.sc}; } return p;
} inline pl2 norm(ll a, ll b){ return norm({a, b}); }

using pl3 = pair<pl2, int>;
vector<pl3> res; // out->line / in->line / line->out / line->in
               //    convex(out->out) / concave(in->in) / out->in / in->out
pl2 ray(const vector<pl2>& arr, pl2 p, pl2 d){
  ll g = gcd(d.fr, d.sc); d.fr /= g; d.sc /= g;
  int n = arr.size();
  vector<pl3> tmp; for (int i = 0; i < n; i++){
    int j = (i+1 == n ? 0 : i+1); pl2 p1 = arr[i], p2 = arr[j];
    pl2 v = {p2.fr-p1.fr, p2.sc-p1.sc};
    pl2 d1 = {p1.fr-p.fr, p1.sc-p.sc}, d2 = {p2.fr-p.fr, p2.sc-p.sc};
    int s1 = sgn(cross(d, d1)), s2 = sgn(cross(d, d2));
    if (s1 == 0){ tmp.push_back({norm(dot(d, d1), dot(d, d)), s2}); }
    if (s2 == 0){ tmp.push_back({norm(dot(d, d2), dot(d, d)), s1}); }
    if (s1*s2 == -1){ tmp.push_back({norm(cross({-d1.fr, -d1.sc}, v), cross(v, d)), 6}); }
  } sort(tmp.begin(), tmp.end(), [](const pl3& p1, const pl3& p2){
    pl2 f1 = p1.fr, f2 = p2.fr; return (i128)f1.fr*f2.sc < (i128)f2.fr*f1.sc;
  }); res.clear();
  int l = tmp.size(); for (int i = 0; i < l; i++){
    if (i+1 < l && tmp[i].fr == tmp[i+1].fr && tmp[i].sc != 6){
      int s1 = tmp[i].sc, s2 = tmp[i+1].sc;
      res.push_back({tmp[i].fr, s1*s2==0 ? (1-s1-s2)/2 : s1*s2 > 0 ? 4 : 6}); i += 1;
    } else{ res.push_back(tmp[i]); }
  }
  int pos = 0; // out / in / line ccw / line cw
  for (pl3& p : res){
    if (p.sc == 4){ p.sc ^= pos; }
```

```
    else if (p.sc == 6){ p.sc ^= pos; pos ^= 1; }
    else if (p.sc == 0){ p.sc = pos; pos ^= 2; }
    else if (p.sc == 1){ p.sc = pos^(pos>>1); pos ^= 3; }
  } return {g, 1};
}

bool fcmp(pl2 p1, pl2 p2){ return (i128)p1.fr*p2.sc < (i128)p2.fr*p1.sc; }
// Assuming that p1 and p2 are not inside; tangent point/line permitted
bool visible(const vector<pl2>& arr, pl2 p1, pl2 p2){
  if (p1 == p2){ return 1; }
  pl2 st = {0, 1}, ed = ray(arr, p1, {p2.fr-p1.fr, p2.sc-p1.sc});
  vector<pl2> pos; for (pl3 e : res){
    int ei = e.sc; //cout << "event " << e.fr.fr << '/' << e.fr.sc << ' ' << e.sc << endl <<
    flush;
    if (ei == 1 || ei == 5 || ei == 7){ pos.push_back(e.fr); }
    if (ei == 3 || ei == 5 || ei == 6){ pos.push_back(e.fr); }
  } int el = pos.size(); //cout << el << endl << flush;
  for (int i = 0; i < el; i+=2){
    pl2 f1 = pos[i], f2 = pos[i+1];
    if (fcmp(st, f2) && fcmp(f1, ed)){ return 0; }
  } return 1;
}
```

6   Graph Theory - Connectivity

6.1   ● Strongly Connected Component

Time Complexity: $\mathcal{O}(V + E)$.

```
stack<int> stk; int scc[N+20];
int ord[N+20], ont;
int dfs(int now){
  stk.push(now); int res = ord[now] = ++ont;
  for (int nxt : adj[now]){
    if (ord[nxt] == 0){ res = min(res, dfs(nxt)); }
    else if (scc[nxt] == 0){ res = min(res, ord[nxt]); }
  }
  if (res == ord[now]){
    while (!stk.empty()){
      int vtx = stk.top(); stk.pop();
      scc[vtx] = now; if (vtx == now){ break; }
    }
  } return res;
}
```

6.2   ● Solution of the 2-SAT

```
for (int i = -n; i <= +n; i++){
  if (i == 0){ continue; }
  if (ord[i+Z] == 0){ dfs(i+Z); }
}
for (int i = 1; i <= n; i++){
  if (scc[i+Z] == scc[-i+Z]){ cout << -1; return; }
}
for (int i = -n; i <= +n; i++){
  if (i == 0){ continue; }
  int v = i+Z; for (int w : adj[v]){
    int pv = scc[v], pw = scc[w]; if (pv == pw){ continue; }
    dag[pv].push_back(pw); ind[pw] += 1;
  }
}
```

```
queue<int> q; for (int i = -n; i <= +n; i++){
  if (i == 0){ continue; }
  int v = i+Z; if (scc[v] != v){ continue; }
  if (ind[v] == 0){ q.push(v); }
} while (!q.empty()){
  int v = q.front(); q.pop();
  for (int p : sccv[v]){
    if (ans[p] != 0){ continue; }
    ans[p] = -1; ans[-(p-Z)+Z] = +1;
  }
  for (int w : dag[v]){
    ind[w] -= 1; if (ind[w] == 0){ q.push(w); }
  }
}
```

6.3   ● Biconnected Component

Time Complexity: $\mathcal{O}(V + E)$.

```
pi3 edg[100020]; // {{v, w}, color}
vector<int> adj[100020]; // Edge Index
stack<int> stk;
int ord[100020], ont = 0;
int dfs(int now, int pre){
  int res = ord[now] = ++ont;
  for (int ei : adj[now]){
    int nxt = edg[ei].fr.fr ^ edg[ei].fr.sc ^ now;
    if (nxt == pre){ continue; }
    if (ord[now] > ord[nxt]){ stk.push(ei); }
    if (ord[nxt] > 0){ res = min(res, ord[nxt]); }
    else{
      int val = dfs(nxt, now); res = min(res, val);
      if (val >= ord[now]){
        vector<int> v; while (!stk.empty()){
          int e = stk.top(); stk.pop();
          edg[e].sc = ei; if (e == ei){ break; }
        }
      }
    }
  }
  return res;
}
```

6.4   ● Articulation Point & Edge

- Articulation Point: Vertex with 2 or more colors.
- Articulation Edge: Edge that has unique color.

7   Graph Theory - Network Flow & Matching

7.1   ● Hall's Marriage Theorem

A bipartite graph $G = (X + Y, E)$ has a perfect X-matching iff for all subset $V \subseteq X$, $|V| \leq |N_G(V)|$.
A bipartite graph $G = (X + Y, E)$ have a maximum X-matching of size $|X| - \max_{V \subseteq X} |V| - |N_G(V)|$.

7.2   ● Dinitz

```
int n, m; int src, snk, nc; ll adj[220][220];
int dis[220];
bool bfs(int st, int ed){
  memset(dis, -1, sizeof(dis)); dis[st] = 0;
  queue<int> q; q.push(st); while (!q.empty()){
    int now = q.front(); q.pop();
```

```
    for (int nxt = 0; nxt < nc; nxt++){
      if (adj[now][nxt] == 0){ continue; }
      if (dis[nxt] != -1){ continue; }
      dis[nxt] = dis[now]+1; q.push(nxt);
    }
  } return dis[ed] != -1;
}
bool chk[220]; int ptr[220];
int pth[220], pc;
bool dfs(int now, int idx){
  pth[idx] = now; chk[now] = 1; if (now == snk){ pc = idx; return 1; }
  for (int& nxt = ptr[now]; nxt < nc; nxt++){
    if (adj[now][nxt] == 0){ continue; }
    if (dis[now]+1 != dis[nxt]){ continue; }
    if (chk[nxt]){ continue; }
    if (dfs(nxt, idx+1)){ return 1; }
  } return 0;
}

memset(ptr, 0, sizeof(ptr)); memset(chk, 0, sizeof(chk));
ll ans = 0; while (bfs(src, snk)){
  memset(ptr, 0, sizeof(ptr)); memset(chk, 0, sizeof(chk));
  while (dfs(src, 0)){
    memset(chk, 0, sizeof(chk));
    ll res = INF; for (int i = 1; i <= pc; i++){
      int v = pth[i-1], w = pth[i];
      res = min(res, adj[v][w]);
    } ans += res;
    for (int i = 1; i <= pc; i++){
      int v = pth[i-1], w = pth[i];
      adj[v][w] -= res; adj[w][v] += res;
    }
  }
}
```

7.3 ● Modeling - Flow with Demands

- req_i = required flow ith vertex need to send
    - $\text{req}_i < 0$: Need to get it from source
    - $\text{req}_i > 0$: Need to send it to sink
- $v \to w$ with lower/upper bound of l and u.
    - $\text{req}_v = l$; $\text{req}_w = -l$
    - Capacity of $v \to w = u - l$
- Create new source and sink, and connect the v with source or sink with capacity $|\text{req}_v|$.
- From the old sink to old source, connect an edge with the capacity ∞. If there were none, ignore this.
- MaxFlow should be equal to $\sum_{\text{req}_v > 0} \text{req}_v$ in order to satisfy the original graph.

7.4 ● Minimum Cost Maximum Flow w/ SPFA

```
int src, snk, nc; pi2 adj[220][220]; // (flow, cost)
int dis[220]; bool chk[220]; int pre[220];
int pth[220]; int pc;
bool spfa(int st, int ed){
  memset(dis, 0x3f, sizeof(dis)); memset(chk, 0, sizeof(chk));
  queue<int> q; dis[st] = 0; chk[st] = 1; q.push(st);
  while (!q.empty()){
    int now = q.front(); q.pop(); chk[now] = 0;
```

```
    for (int nxt = 1; nxt <= nc; nxt++){
      if (adj[now][nxt].fr == 0){ continue; }
      if (dis[nxt] <= dis[now] + adj[now][nxt].sc){ continue; }
      dis[nxt] = dis[now] + adj[now][nxt].sc; pre[nxt] = now;
      if (!chk[nxt]){ chk[nxt] = 1; q.push(nxt); }
    }
  }
  if (dis[ed] == INF){ return 0; }
  int ptr = ed; pc = 0;
  while (ptr != st){ pth[pc++] = ptr; ptr = pre[ptr]; } pth[pc] = ptr;
  reverse(pth, pth+pc+1); return 1;
}

int ans = 0, res = 0; while (spfa(src, snk)){
  int cnt = INF; for (int i = 1; i <= pc; i++){
    int v = pth[i-1], w = pth[i];
    cnt = min(cnt, adj[v][w].fr);
  } res += cnt;
  for (int i = 1; i <= pc; i++){
    int v = pth[i-1], w = pth[i];
    adj[v][w].fr -= cnt; adj[w][v].fr += cnt;
    ans += cnt*adj[v][w].sc;
  }
}
```

7.5 ● Stable Marriage Problem

For i in 1 to N, try to match the best possible matching j, knocking other matchings in process. This matching is best for proposer i while worst for proposee j.

```
int adj[1020][1020]; int rnk[1020][1020]; // adj[i][r] -> j // rnk[j][i] -> r
int ptr[1020]; int res[1020]; // ptr[i] = r // res[j] = i
bool f(int i, int j){
  if (res[j] == 0){ res[j] = i; return 1; }
  int p = res[j]; if (rnk[j][p] > rnk[j][i]){
    res[j] = i; ptr[p] += 1; for (int& r = ptr[p]; ; r++){
      if (f(p, adj[p][r])){ break; }
    } return 1;
  } return 0;
}

for (int i = 1; i <= n; i++){ ptr[i] = 1; }
for (int i = 1; i <= n; i++){
  for (int& r = ptr[i]; ; r++){
    if (f(i, adj[i][r])){ break; }
  }
}
```

7.6 ● Hungarian Algorithm

```
// Max version. for Min, adj[v][w] *= -1, then print -ans.
ll adj[520][520];
ll l1[520], l2[520]; ll d[520]; int di[520];
bool c1[520], c2[520]; int m1[520], m2[520];
int par[520];
inline ll cost(int v, int w){ return l1[v] + l2[w] - adj[v][w]; }
void psh(int n, int v, int p){ // p -> m1[v] -> v
  c1[v] = 1; par[v] = p;
  for (int j = 1; j <= n; j++){
    if (cost(v, j) < d[j]){ d[j] = cost(v, j); di[j] = v; }
```

```
    }
}

for (int i = 1; i <= n; i++){ l1[i] = *max_element(adj[i]+1, adj[i]+n+1); }
for (int j = 1; j <= n; j++){ l2[j] = 0; }
for (int cnt = 0; cnt < n; cnt++){
  memset(c1, 0, sizeof(c1)); memset(c2, 0, sizeof(c2));
  memset(par, 0, sizeof(par));
  queue<int> q; int st = 0;
  for (int i = 1; i <= n; i++){
    if (m1[i] == 0){
      st = i; q.push(i);
      par[i] = -1; c1[i] = 1; break;
    }
  }
  for (int j = 1; j <= n; j++){ d[j] = cost(st, j); di[j] = st; }
  int v = 0, w = 0; while (1){
    while (!q.empty()){
      v = q.front(); q.pop();
      for (int j = 1; j <= n; j++){
        if (cost(v, j) == 0 && !c2[j]){
          if (m2[j] == 0){ w = j; goto augment; }
          c2[j] = 1; q.push(m2[j]); psh(n, m2[j], v);
        }
      }
    }
    ll val = INF; for (int j = 1; j <= n; j++){
      if (!c2[j]){ val = min(val, d[j]); }
    }
    for (int i = 1; i <= n; i++){ if (c1[i]){ l1[i] -= val; } }
    for (int j = 1; j <= n; j++){
      if (c2[j]){ l2[j] += val; } else{ d[j] -= val; }
    }
    while (!q.empty()){ q.pop(); }
    for (int j = 1; j <= n; j++){
      if (!c2[j] && d[j] == 0){
        if (m2[j] == 0){ v = di[j]; w = j; goto augment; }
        else{
          c2[j] = 1; if (!c1[m2[j]]){
            q.push(m2[j]); psh(n, m2[j], di[j]);
          }
        }
      }
    }
  }
  } augment:
  while (v != -1){
    int pv = m2[w], pw = m1[v];
    m1[v] = w; m2[w] = v; v = par[v]; w = pw;
  }
}
ll ans = 0; for (int i = 1; i <= n; i++){ ans += adj[i][m1[i]]; }
```

7.7 ● General Matching

```
int n, m; vector<int> adj[520]; // Input: Graph
int res[520]; // Matched Vertex

int par[520]; // BFS path, before v.
```

```
int chk[520]; // Visited? (0: no, 1: yes+odd, 2: yes+even) \
        odd/even comes from bipartite... kinda. \
        all v in q satisfies chk[v] = 1.
void pro(int r, int v){ // Re-Match. a.k.a. Augmenting
  int p = v; do{
    p = par[v]; int q = res[p];
    res[v] = p; res[p] = v; v = q;
  } while (r != p);
}

int bls[520]; // Position of a Blossom
bool vst[520]; // Used for lca.
int lca(int v, int w){ // LCA on a BFS tree.
  memset(vst, 0, sizeof(vst));
  while (1){
    if (v != 0){
      if (vst[v]){ return v; }
      vst[v] = 1; v = bls[par[res[v]]];
    } swap(v, w);
  }
}
void cyc(int v, int w, queue<int>& q){ // Cycle (fancy term = Blossom) Merging.
  int l = lca(bls[v], bls[w]);
  while (bls[v] != l){
    par[v] = w; w = res[v];
    if (chk[w] == 2){ q.push(w); chk[w] = 1; }
    bls[v] = bls[w] = l; v = par[w];
  }
}
bool bfs(int r){ // main BFS.
  memset(chk, 0, sizeof(chk)); memset(par, 0, sizeof(par));
  for (int v = 1; v <= n; v++){ bls[v] = v; }
  queue<int> q; q.push(r); chk[r] = 1; while (!q.empty()){
    int v = q.front(); q.pop();
    for (int w : adj[v]){
      if (chk[w] == 0){
        par[w] = v; chk[w] = 2;
        if (res[w] == 0){ pro(r, w); return 1; }
        q.push(res[w]); chk[res[w]] = 1;
      }
      else if (chk[w] == 1 && bls[v] != bls[w]){
        cyc(w, v, q); cyc(v, w, q);
      }
    }
  } return 0;
}

void Main(){
  cin >> n >> m; while (m--){
    int v, w; cin >> v >> w;
    adj[v].push_back(w); adj[w].push_back(v);
  }
  int ans = 0; for (int v = 1; v <= n; v++){
    if (res[v] == 0 && bfs(v)){ ans += 1; }
  } cout << ans;
}
```

8   Graph Theory - Miscellaneous

8.1   ● Eulerian Path

Directed version. For undirected use $e_i = (v_i, w_i)$ and save i like BCC.

```cpp
vector<int> adj[200020]; int ptr[200020];
int pth[200020]; int len;
void dfs(int now){
  pth[++len] = now;
  for (int& p = ptr[now]; p < adj[now].size();){
    int nxt = adj[now][p++];
    dfs(nxt);
  }
}
```

8.2   ● Tree Isomorphism

```cpp
int dep[2][100020]; vector<int> vtx[2][100020]; int par[2][100020];
int dfs(int t, int v, int p){
  dep[t][v] = dep[t][p]+1; vtx[t][dep[t][v]].push_back(v);
  par[t][v] = p; erase(adj[t][v], p);
  int res = dep[t][v]; for (int w : adj[t][v]){
    if (w == p){ continue; }
    res = max(res, dfs(t, w, v));
  } return res;
}


int val[2][100020]; int ans[100020];
using pvi = pair<vector<int>, int>;

centroid(0, 1); pi2 c0 = cent; centroid(1, 1); pi2 c1 = cent;
if ((c0.sc == -1) != (c1.sc == -1)){ cout << "EI"; return; }
for (int t = 0; t < 2; t++){
  pi2& c = (t==0 ? c0 : c1); if (c.sc == -1){ continue; }
  int v = c.fr, w = c.sc; erase(adj[t][v], w); erase(adj[t][w], v);
  adj[t][v].push_back(n+1); adj[t][w].push_back(n+1);
  adj[t][n+1].push_back(v); adj[t][n+1].push_back(w);
  c = {n+1, 0};
}
int d0 = dfs(0, c0.fr, c0.fr); int d1 = dfs(1, c1.fr, c1.fr);
if (d0 != d1){ cout << "EI"; return; }
for (int d = d0; d > 1; d--){
  vector<pvi> res[2];
  for (int t = 0; t < 2; t++){
    for (int v : vtx[t][d-1]){
      res[t].emplace_back(); res[t].back().sc = v;
      for (int w : adj[t][v]){ res[t].back().fr.push_back(val[t][w]); }
    } for (pvi& p : res[t]){ sort(p.fr.begin(), p.fr.end()); }
    sort(res[t].begin(), res[t].end());
  }
  int l0 = res[0].size(), l1 = res[1].size(); if (l0 != l1){ cout << "EI"; return; }
  int num = 0; for (int i = 0; i < l0; i++){
    if (res[0][i].fr != res[1][i].fr){ cout << "EI"; return; }
    if (i > 0 && res[0][i-1].fr != res[0][i].fr){ num += 1; }
    int v0 = res[0][i].sc, v1 = res[1][i].sc;
    val[0][v0] = val[1][v1] = num;
  }
}
queue<pi2> q; q.push({c0.fr, c1.fr}); while (!q.empty()){
```

```cpp
  int r0 = q.front().fr, r1 = q.front().sc; q.pop(); ans[r0] = r1;
  sort(adj[0][r0].begin(), adj[0][r0].end(), [](int v1, int v2){ return val[0][v1] <
  val[0][v2]; });
  sort(adj[1][r1].begin(), adj[1][r1].end(), [](int v1, int v2){ return val[1][v1] <
  val[1][v2]; });
  int l = adj[0][r0].size(); for (int i = 0; i < l; i++){ q.push({adj[0][r0][i],
  adj[1][r1][i]}); }
}
cout << "JAH" << endl; for (int i = 1; i <= n; i++){ cout << ans[i]-1 << endl; }
```

9   Dynamic Programming

9.1   ● Rerooting

Tree DP, with changing root.
Time Complexity: $\mathcal{O}(N)$.

```cpp
void reroot(int now, int pre){
  ans[now] = dp[now]; for (int nxt : adj[now]){
    if (nxt == pre){ continue; }
    int dpv = dp[now], dpw = dp[nxt];
    dp[now] -= (dp[nxt]?); dp[nxt] += (dp[now]?);
    reroot(nxt, now);
    dp[now] = dpv; dp[nxt] = dpw;
  }
} // dpf(root, root); reroot(root, root)
```

9.2   ● Sum over Subset

$$D_S = \sum_{T \subseteq S} A_T.$$

Time Complexity: $\mathcal{O}(N2^N)$.

```cpp
for (int i = 0; i < X; i++){ dp[i] = arr[i]; }
for (int b = 0; b < B; b++){
  for (int bit = 0; bit < X; bit++){
    if (bit>>b & 1){ dp[bit] += dp[bit ^ 1<<b]; }
  }
}
```

9.3   ● Convex Hull Trick

$$D_i = \max_{j<i} a_j x_i + b_j.$$

Time Complexity: $\mathcal{O}(N)$ for N insertions, $\mathcal{O}(\log N)$ for query.

```cpp
inline ld crs(pl2 l1, pl2 l2){ return (ld)(l2.sc-l1.sc)/(l1.fr-l2.fr); }
pair<pl2, ld> stk[1000020]; int sp = 0;
inline void psh(pl2 p){
  stk[sp].fr = p; stk[sp].sc = (sp==0 ? 0 : crs(stk[sp-1].fr, stk[sp].fr));
  sp += 1;
}
inline void pop(){ sp -= 1; }

// query: x
int st = 0, ed = sp; while (st+1 <= ed-1){
  int mid = st+ed >> 1;
  if (x < stk[mid].sc){ ed = mid; } else{ st = mid; }
} pl2 p = stk[st].fr; // A(j)x + B(j)
dp[i] = p.fr*x + p.sc + C(i);
// update: p = (A(j), B(j))
while (sp > 0){
  if (crs(stk[sp-1].fr, p) <= stk[sp-1].sc){ pop(); } else{ break; }
} psh(p);
```

9.4 ● Divide and Conquer Optimization

$D_{k,i} = \max j < i D_{k-1,j} + A_{j,i}$. $\text{opt}_i < \text{opt}_{i+1}$.
Time Complexity: $\mathcal{O}(KN \log N)$.

```cpp
ll dp[2][N+20]; // Sliding Window on first index
void dnc_opt(ll* dp1, ll* dp0, int is, int ie, int os, int oe){
  if (is > ie){ return; } int im = is+ie >> 1;
  int o = os; for (int p = os; p <= min(oe, im-1); p++){
    if (f(dp0, p, im) < f(dp0, o, im)){ o = p; }
  } dp1[im] = f(dp0, o, im);
  dnc_opt(dp1, dp0, is, im-1, os, o); dnc_opt(dp1, dp0, im+1, ie, o, oe);
}
```

9.5 ● WQS Binary Search (Aliens Trick)

$D_{k,i} = \min j < i D_{k-1,j} + A_{j,i}$. $D_{k,N}$ must be convex.
Time Complexity: $\mathcal{O}(T(N) \log X)$.

```cpp
// solve(n, mid) returns the number of used elements
// Binary Search over Lambda
ll st = 0, ed = 1e18; ll ans = 1e18;
while (st <= ed){
  ll mid = st+ed >> 1;
  int cnt = solve(n, mid); ans = min<i128>(ans, dp[n] + (i128)mid*k);
  if (cnt <= k){ ed = mid-1; } else{ st = mid+1; }
} cout << ans;
```

9.6 ● Monotone Queue Optimization

$D_i = \min_{j<i} D_j + A_{j,i}$. For all pair $i < j$, there exists a point p that $D_i + C_{i,p}$ and $D_j + C_{j,p}$ changes. So $k < p$ means $<$, and $\geq$ otherwise.
Time Complexity: $\mathcal{O}(N \log N)$.
Code from *BOJ 17439 - Flower Shop*.

```cpp
ll arr[50020]; ll prf[50020];

deque<int> dq;
pl2 dp[50020];
inline ll f(int j, int i){ return (prf[i]-prf[j])*(i-j); }
inline ll dpf(int j, int i){ return dp[j].fr + f(j, i); }
inline int crx(int a, int b, int n){ // Assume a < b.
  int st = b+1, ed = n; int i = n+1; while (st <= ed){
    int mid = st+ed >> 1;
    if (dpf(a, mid) >= dpf(b, mid)){ i = mid; ed = mid-1; }
    else{ st = mid+1; }
  } return i;
}
int solve(int n, ll x){
  dp[0] = {0, 0}; for (int i = 1; i <= n; i++){
    while (dq.size() >= 2){
      int dql = dq.size();
      int a = dq[dql-2], b = dq[dql-1];
      if (crx(a, b, n) >= crx(b, i-1, n)){ dq.pop_back(); }
      else{ break; }
    } dq.push_back(i-1);
    while (dq.size() >= 2){
      int a = dq[0], b = dq[1];
      if (crx(a, b, n) <= i){ dq.pop_front(); }
      else{ break; }
    } int j = dq.front();
    dp[i] = {dpf(j, i), dp[j].sc+1};
    //cout << dp[i].fr << " \n"[i==n];
  } return dp[n].sc;
}
```

9.7 ● Slope Trick

I think this one is easier to just say. Let f be a convex piecewise linear function with integer coefficient. We can save this function by the points where the line changes, with one linear function.

We can add two functions by merging the points without removing duplicants, and adding the linear function.

9.8 ● Knuth Optimization

$D_{i,j} = D_{i,k} + D_{k+1,j} + A_{i,j}$ where $\text{opt}_{i,j-1} \leq \text{opt}_{i,j} \leq \text{opt}_{i+1,j}$. This is satisfied if C is monge.
Time Complexity: $\mathcal{O}(N^2)$.

```cpp
for (int d = 0; d < n; d++){
  for (int s=1, e=1+d; e <= n; s++, e++){
    if (d == 0){ dp[s][e] = 0; opt[s][e] = s; }
    else{
      opt[s][e] = s; dp[s][e] = INF;
      for (int p = opt[s][e-1]; p <= opt[s+1][e]; p++){
        ll res = dp[s][p] + dp[p+1][e] + prf[e]-prf[s-1];
        if (dp[s][e] > res){ dp[s][e] = res; opt[s][e] = p; }
      }
    }
  }
} cout << dp[1][n] << endl;
```

9.9 ● Kitamasa

$A_n = \prod_{i=1}^{k} A_{n-d} C_d$. The code below uses `arr` as initial terms with index 0 to $n-1$. And `f` as coefficient of $x^n - \sum_{d=1}^{k} C_d x^{n-d}$.

```cpp
const ll mod = 104857601; const ll r = 3;

ll fpow(ll mul, ll bit){
  ll res = 1; while (bit){
    if (bit&1){ res = res*mul % mod; }
    mul = mul*mul % mod; bit >>= 1;
  } return res;
} inline ll finv(ll x){ return fpow(x, mod-2); }

inline void normp(vector<ll>& arr){
  int l = arr.size(); while (l > 0){
    if (arr[l-1] == 0){ l -= 1; } else{ break; }
  } arr.resize(l);
}
inline vector<ll> trim(vector<ll> arr, int mx){
  int l = min<int>(arr.size(), mx); return vector<ll>(arr.begin(), arr.begin()+l);
}
void dft(vector<ll>& arr, bool inv = false){
  int n = arr.size();
  for (int j=0, i=1; i < n; i++){
    int bit = n>>1;
    while (j&bit){ j ^= bit; bit >>= 1; } j ^= bit;
    if (i < j){ swap(arr[i], arr[j]); }
  }
  for (int l = 1; l < n; l*=2){
    ll w = fpow(r, (mod-1)/(2*l)); if (inv){ w = finv(w); }
    for (int i = 0; i < n; i += l*2){
      ll wp = 1; for (int j = 0; j < l; j++){
        ll a = arr[i+j], b = arr[i+j+l]*wp % mod;
        arr[i+j] = (a+b) % mod; arr[i+j+l] = (a-b+mod)%mod;
        wp = wp*w % mod;
```

```
      }
     }
    }
   if (inv){
     ll invn = finv(n);
     for (int i = 0; i < n; i++){ arr[i] = arr[i]*invn % mod; }
   }
}
vector<ll> mulp(vector<ll> arr, vector<ll> brr){ // A \times B
  normp(arr); normp(brr);
  int al = arr.size(), bl = brr.size();
  int n = max(al, bl); n = 1 << (bits(n) + 1 - ((n&-n)==n));
  arr.resize(n); brr.resize(n);
  dft(arr); dft(brr);
  for (int i = 0; i < n; i++){ arr[i] = arr[i]*brr[i] % mod; }
  dft(arr, -1); arr = trim(arr, al+bl-1); return arr;
}
vector<ll> invp(vector<ll> arr, int k){ // A^{-1} \pmod{x^k}. Assume A[0] \neq 0.
  arr = trim(arr, k);
  vector<ll> res = vector<ll>{finv(arr[0])}; int m = 1;
  while (m < k){
    vector<ll> val = trim(mulp(trim(arr, m*2), res), m*2); int vl = val.size();
    for (int i = 0; i < vl; i++){
      val[i] = ((i==0)*2 + mod-val[i]) % mod;
    } res = trim(mulp(res, val), m*2); m *= 2;
  } res = trim(res, k); return res;
}
vector<ll> divp(vector<ll> arr, vector<ll> brr){ // D, where A = BQ + R with \deg R < \deg
B.
  normp(arr); normp(brr);
  int al = arr.size(), bl = brr.size(); if (al < bl){ return vector<ll>{}; }
  reverse(arr.begin(), arr.end()); reverse(brr.begin(), brr.end());
  arr = trim(arr, al-bl+1); brr = trim(brr, al-bl+1);
  vector<ll> qrr = mulp(arr, invp(brr, al-bl+1)); qrr.resize(al-bl+1);
  reverse(qrr.begin(), qrr.end()); normp(qrr); return qrr;
}
vector<ll> modp(vector<ll> arr, vector<ll> brr){ // R, where A = BQ + R with \deg R < \deg
B.
  normp(arr); normp(brr);
  int al = arr.size(), bl = brr.size(); if (al < bl){ return arr; }
  vector<ll> qrr = divp(arr, brr); vector<ll> drr = mulp(brr, qrr);
  for (int i = 0; i < al; i++){ arr[i] = (arr[i]-drr[i]+mod) % mod; }
  normp(arr); return arr;
}

ll arr[30020], crr[30020];

ll kth(ll k, const vector<ll>& arr, const vector<ll>& f){ // x^k \pmod f
  int n = arr.size(); if (k < n){ return arr[k]; }
  vector<ll> res{1}, mul{0, 1}; while (k){
    if (k&1){ res = modp(mulp(res, mul), f); }
    mul = modp(mulp(mul, mul), f); k >>= 1;
  }
  ll ans = 0; for (int i = 0; i < n; i++){ ans += res[i]*arr[i] % mod; }
  return ans%mod;
}
```

9.10   • Hirschberg

```
string s1, s2;

#define ENDL ""

int dp[3][7010];
void solve(int i1, int j1, int i2, int j2, bool top = true){
  //cout << "solve " << i1 << ' ' << j1 << ' ' << i2 << ' ' << j2 << endl;
  if (i1 > j1 || i2 > j2){ return; }
  if (i1 == j1){
    for (int p2 = i2; p2 <= j2; p2++){
      if (s1[i1] == s2[p2]){
        if (top){ cout << 1 << endl; }
        cout << s1[j1] << ENDL; return;
      }
    }
    if (top){ cout << 0 << endl; } return;
  }
  int m1 = i1+j1 >> 1;
  for (int p1 = i1-1; p1 <= m1; p1++){
    int b1 = p1&1; int b0 = 1-b1;
    for (int p2 = i2-1; p2 <= j2; p2++){
      if (p1 == i1-1 || p2 == i2-1){ dp[b1][p2] = 0; }
      else{
        dp[b1][p2] = max(dp[b0][p2], dp[b1][p2-1]);
        if (s1[p1] == s2[p2]){ dp[b1][p2] = max(dp[b1][p2], dp[b0][p2-1]+1); }
      }
    }
  }
  if ((m1&1) == 0){
    for (int p2 = i2-1; p2 <= j2; p2++){ dp[1][p2] = dp[0][p2]; }
  }
  for (int p1 = j1; p1 > m1; p1--){
    int b1 = p1&1; int b0 = 1-b1; b1 <<= 1; b0 <<= 1;
    for (int p2 = j2; p2 >= i2-1; p2--){
      if (p1 == j1 || p2 == j2){ dp[b1][p2] = 0; }
      else{
        dp[b1][p2] = max(dp[b0][p2], dp[b1][p2+1]);
        if (s1[p1+1] == s2[p2+1]){ dp[b1][p2] = max(dp[b1][p2], dp[b0][p2+1]+1); }
      }
    }
  }
  if ((m1+1&1) == 0){
    for (int p2 = i2-1; p2 <= j2; p2++){ dp[2][p2] = dp[0][p2]; }
  }
  int mx = 0; pi2 mxp = {0, 0}; for (int p2 = i2-1; p2 <= j2; p2++){
    int x = dp[1][p2]+dp[2][p2];
    if (x > mx){ mx = x; mxp = {0, p2}; }
  }
  for (int p2 = i2; p2 <= j2; p2++){
    if (s1[m1+1] == s2[p2]){
      int x = dp[1][p2-1]+dp[2][p2]+1;
      if (x > mx){ mx = x; mxp = {1, p2}; }
    }
  }
  if (top){ cout << mx << endl; }
  if (mx == 0){ return; }
```

```
    if (mxp.fr == 0){
      int p2 = mxp.sc;
      solve(i1, m1, i2, p2, 0); solve(m1+1+1, j1, p2+1, j2, 0);
    }
    else{
      int p2 = mxp.sc;
      solve(i1, m1, i2, p2-1, 0); cout << s2[p2] << ENDL; solve(m1+1+1, j1, p2+1, j2, 0);
    }
  }
}

void Main(){
  cin >> s1 >> s2; int l1 = s1.size(), l2 = s2.size();
  s1 = " " + s1; s2 = " " + s2;
  solve(1, l1, 1, l2);
}
```

9.11 ● Connection Profile

Single Connected Region, with DP condition.
Time Complexity: $\mathcal{O}(NMB_M)$, $B_9 = 21\,147$. Actual state is about $2\,000$?

```
int n, m; int arr[10][10];
inline int f(string& bit){
  int cvt[10] = {}; int val = 0;
  for (char& c : bit){
    int x = c-'0'; if (x == 0){ continue; }
    if (cvt[x] == 0){ cvt[x] = ++val; }
    c = cvt[x]+'0';
  } return val;
}
const ll INF = 1e18;
map<string, ll> dp[10][10];
ll dpf(int y, int x, string bit){ int cnt = f(bit);
  if (x == m){ y += 1; x = 0; }
  if (y == n){ return cnt <= 1 ? 0 : INF; }
  if (dp[y][x].count(bit)){ return dp[y][x][bit]; }
  dp[y][x][bit] = INF;
  bool flg = bit.front() == '0';
  for (int b = 1; b < m; b++){ flg |= bit[0] == bit[b]; }
  if (flg){
    string nxt = bit.substr(1, m-1) + '0';
    dp[y][x][bit] = min(dp[y][x][bit], dpf(y, x+1, nxt));
  }
  char u = bit.front(), l = bit.back(); if (x == 0){ l = '0'; }
  if (u == '0' && l == '0'){
    string nxt = bit.substr(1, m-1) + (char)(cnt+1+'0');
    dp[y][x][bit] = min(dp[y][x][bit], dpf(y, x+1, nxt) + arr[y][x]);
  }
  else if (u == '0' || l == '0'){
    string nxt = bit.substr(1, m-1) + (u=='0' ? l : u);
    dp[y][x][bit] = min(dp[y][x][bit], dpf(y, x+1, nxt) + arr[y][x]);
  }
  else{
    string nxt = bit.substr(1, m-1) + u;
    for (char& c : nxt){ if (c == l){ c = u; } }
    dp[y][x][bit] = min(dp[y][x][bit], dpf(y, x+1, nxt) + arr[y][x]);
  }
  if (cnt <= 1){ dp[y][x][bit] = min<ll>(dp[y][x][bit], 0); }
  return dp[y][x][bit];
```

```
}
dpf(0, 0, string(m, '0')); // minimize the sum of arr[i][j], connected.
```

9.12 ● Permutation DP

Let $D_{i,j}$ be the number of pre-permutations that uses the value from 1 to i, and has j components. The transition is to:
- Create new component: $(i,j) \to (i+1, j+1)$.
- Append $i+1$ to one of the component: $(i,j) \to (i+1, j)$.
- Connect two components using $i+1$: $(i,j) \to (i+1, j-1)$.

When the starting/ending point is specified, you need to be careful about the number of *open*-ended points.

9.13 ● Berlekamp-Massey

Use with *Kitamasa*.

```
vector<ll> mulp(vector<ll> arr, vector<ll> brr){ // A \times B
  normp(arr); normp(brr);
  int al = arr.size(), bl = brr.size();
  vector<ll> res(al+bl-1);
  for (int i = 0; i < al; i++){
    for (int j = 0; j < bl; j++){
      res[i+j] += arr[i]*brr[j] % mod; res[i+j] %= mod;
    }
  } return res;
}
vector<ll> solve(vector<ll> arr){
  int n = arr.size();
  vector<ll> res, val; int pos = -1;
  for (int i = 0; i < n; i++){
    int rl = res.size();
    ll d = arr[i]; for (int j = 0; j < rl; j++){
      d -= res[j]*arr[i-j-1] % mod; d = (d+mod)%mod;
    } if (d == 0){ continue; }
    if (pos == -1){
      res.resize(i+1); for (int j = 0; j <= i; j++){ res[j] = 1; }
      pos = i;
    } else{
      vector<ll> v{1}; for (ll x : val){ v.push_back((mod-x)%mod); }
      int vl = v.size();
      ll c = 0; for (int j = 0; j < vl; j++){
        c += v[j]*arr[pos-j] % mod; c %= mod;
      } c = d*finv(c) % mod;
      for (ll& x : v){ x = x*c % mod; }
      vector<ll> tmp(i-pos-1, 0); for (ll x : v){ tmp.push_back(x); }
      if (i - res.size() >= pos - val.size()){ val = res; pos = i; }
      int rl = max(res.size(), tmp.size()); res.resize(rl);
      for (int i = 0; i < rl; i++){ res[i] = (res[i]+tmp[i]) % mod; }
    }
  } return res;
}
```

10 String

10.1 ● Knuth-Morris-Pratt

```
int j = 0; for (int i = 1; i < m; i++){
  while (j != 0){
    if (t[i] == t[j]){ break; } else{ j = jmp[j-1]; }
  } if (t[i] == t[j]){ j += 1; jmp[i] = j; }
```

```
}
j = 0; for (int i = 0; i < n; i++){
  while (j != 0){
    if (s[i] == t[j]){ break; } else{ j = jmp[j-1]; }
  } if (s[i] == t[j]){
    if (j+1 == m){ /* matched on (i-m, i] */ j = jmp[j]; }
    else{ j += 1; }
  }
}
```

10.2   ● Z Algorithm

```
int l = 0, r = 0; for (int i = 1; i < sl; i++){
  if (i < r){ z[i] = min(r-i, z[i-l]); }
  while (i+z[i] < sl){
    if (s[z[i]] == s[i+z[i]]){ z[i] += 1; }
    else{ break; }
  } if (r < i+z[i]){ l = i; r = i+z[i]; }
} z[0] = sl;
```

10.3   ● Aho-Corasick

```
int q; cin >> q; while (q--){
  string s; cin >> s; /* Trie */ psh(s);
}
queue<int> q; q.push(0); /* Root */ while (!q.empty()){
  int now = q.front(); q.pop();
  for (int nxi : nxtSet){
    int nxt = trie[now].nxt[nxi]; if (nxt == 0){ continue; }
    if (now == 0){ trie[nxt].jmp = now; }
    else{
      int ptr = trie[now].jmp; while (ptr != 0){
        if (trie[ptr].nxt[nxi] != 0){ break; } else{ ptr = trie[ptr].jmp; }
      } if (trie[ptr].nxt[nxi] != 0){ ptr = trie[ptr].nxt[nxi]; }
      trie[nxt].jmp = ptr;
    } q.push(nxt);
    trie[nxt].chk |= trie[trie[nxt].jmp].chk;
  }
}
int ptr = 0; for (int nxi : s){
  while (ptr != 0){
    if (trie[ptr].nxt[nxi] != 0){ break; } else{ ptr = trie[ptr].jmp; }
  } if (trie[ptr].nxt[nxi] != 0){ ptr = trie[ptr].nxt[nxi]; }
  if (trie[ptr].chk){ ans += 1; }
}
```

10.4   ● Suffix Array & Longest Common Prefix Array

```
int sa[1000020], pos[1000020], lcp[1000020];
int tmp[1000020], cnt[1000020], res[1000020];
void init(string s){
  s.push_back('#'); int sl = s.size();
  for (int i = 0; i < sl; i++){ sa[i] = i; }
  sort(sa, sa+sl, [&s](int s1, int s2){ return s[s1] < s[s2]; });
  res[sa[0]] = 0; for (int i = 1; i < sl; i++){
    int s1 = sa[i-1], s2 = sa[i]; res[s2] = res[s1] + (s[s1] != s[s2]);
  }
  for (int k = 1; ; k *= 2){
    for (int i = 0; i < sl; i++){ cnt[i] = 0; tmp[i] = (sa[i]-k%sl+sl)%sl; }
    for (int i = 0; i < sl; i++){ cnt[res[i]] += 1; }
```

```
    for (int i = 1; i < sl; i++){ cnt[i] += cnt[i-1]; }
    for (int i = sl-1; i >= 0; i--){
      int sp = tmp[i]; int r = res[sp];
      int p = cnt[r]; sa[p-1] = sp; cnt[r] -= 1;
    }
    tmp[sa[0]] = 0; for (int i = 1; i < sl; i++){
      int s1 = sa[i-1], s2 = sa[i];
      int t1 = (s1+k)%sl, t2 = (s2+k)%sl;
      tmp[s2] = tmp[s1] + (res[s1] != res[s2] || res[t1] != res[t2]);
    } memcpy(res, tmp, sizeof(res)); if (res[sa[sl-1]] == sl-1){ break; }
  } s.pop_back(); sl -= 1;
  for (int i = 0; i < sl; i++){ sa[i] = sa[i+1]; }
  for (int i = 0; i < sl; i++){ pos[sa[i]] = i; }
  int l = 0; for (int i = 0; i < sl; i++){
    l = max(l-1, 0); int p1 = pos[i], p2 = pos[i]-1; if (p2 < 0){ continue; }
    int s1 = sa[p1], s2 = sa[p2]; while (s1+l < sl && s2+l < sl){
      if (s[s1+l] != s[s2+l]){ break; } else{ l += 1; }
    } lcp[p2] = l;
  }
}
```

10.5   ● Suffix Array - Traversing

```
int idx[1000020]; // idx[i] = NULL position of sa[i]. Useful when S is multi-string.
// qry(st, ed) returns pair(min lcp[i], argmin) for [st, ed].
// solve on sa[st..ed][i..j].
ll solve(int st, int ed, int i){
  if (st == ed){ int j = idx[st]; return chk[st].fr <= ed && ed < chk[st].sc ? j-i : 0; }
  int j = qry(st, ed).fr; vector<int> mid{st};
  while (mid.back() < ed){
    pi2 p = qry(mid.back(), ed); if (p.fr != j){ break; }
    mid.push_back(p.sc+1);
  } mid.push_back(ed+1);
  ll ans = (chk[st].fr <= ed && ed < chk[st].sc ? j-i : 0);
  int len = mid.size(); for (int p = 1; p < len; p++){
    ans += solve(mid[p-1], mid[p]-1, j);
  } return ans;
}
```

10.6   ● Manacher's

```
string s = "#"; for (char c : inp){ s += c; s += '#'; }
int sl = s.size(); int ptr = -1, mid = -1;
for (int i = 0; i < sl; i++){
  if (ptr >= i){ pos[i] = min(ptr-i, pos[mid + mid-i]); }
  else{ pos[i] = 0; }
  int i1 = i-pos[i], i2 = i+pos[i];
  while (0 <= i1-1 && i2+1 < sl){
    if (s[i1-1] == s[i2+1]){ i1--; i2++; pos[i] += 1; }
    else{ break; }
  } if (i2 > ptr){ ptr = i2; mid = i; }
} // length = *max_element(pos, pos+sl);
```

10.7   ● Rolling Hash & Rabin-Karp

```
const int H = 2;
typedef array<ll, H> alH;
const ll mod[H] = {993244853, 998244853};
const ll mul[H] = {31, 37};
ll ppw[H][N+20], prf[H][N+20];
```

```cpp
void init(int hi, const string& s){ int sl = s.size();
  ppw[hi][0] = 1; for (int i = 1; i <= N; i++){ ppw[hi][i] = ppw[hi][i-1]*mul[hi] % mod[hi];
  }
  for (int i = 1; i <= sl; i++){ prf[hi][i] = (prf[hi][i-1]*mul[hi] + s[i]) % mod[hi]; }
}
inline alH hsh(int st, int ed){
  alH res; for (int hi = 0; hi < H; hi++){
    res[hi] = (prf[hi][ed] - prf[hi][st-1]*ppw[hi][ed-st+1] % mod[hi] + mod[hi]) % mod[hi];
  } return res;
}
```

11   Data Structures

11.1   ● Li-Chao Tree

Find max $f_i(x) = a_i x + b_i$, with line insertion update.

Time Complexity: $\mathcal{O}(\log N)$ per update/query.

```cpp
// Max version. if Min then use the commentated version.
inline ll f(pl2 l, ll x){ return l.fr*x + l.sc; }
struct Node{ pl2 l = {0, -INF /* INF */}; pi2 nxt = {0, 0}; };
vector<Node> seg;
void upd(int ni, ll ns, ll ne, pl2 l1){
  pl2 l2 = seg[ni].l;
  if (f(l1, ns) < /* > */ f(l2, ns)){ swap(l1, l2); }
  if (f(l1, ne) >= /* <= */ f(l2, ne)){ seg[ni].l = l1; return; }
  ll nm = ns+ne >> 1;
  if (f(l1, nm) >= /* <= */ f(l2, nm)){
    seg[ni].l = l1; if (seg[ni].nxt.sc == 0){
      seg[ni].nxt.sc = seg.size(); seg.emplace_back();
    } upd(seg[ni].nxt.sc, nm+1, ne, l2);
  }
  else{
    seg[ni].l = l2; if (seg[ni].nxt.fr == 0){
      seg[ni].nxt.fr = seg.size(); seg.emplace_back();
    } upd(seg[ni].nxt.fr, ns, nm, l1);
  }
} inline void upd(pl2 l){ return upd(0, -X, X, l); }
ll qry(int ni, ll ns, ll ne, ll x){
  pl2 l = seg[ni].l; ll nm = ns+ne >> 1;
  int nxt = (x <= nm ? seg[ni].nxt.fr : seg[ni].nxt.sc);
  if (nxt == 0){ return f(l, x); }
  else{
    if (x <= nm){ return max /* min */(f(l, x), qry(nxt, ns, nm, x)); }
    else{ return max /* min */(f(l, x), qry(nxt, nm+1, ne, x)); }
  }
} inline ll qry(ll x){ return qry(0, -X, X, x); }
```

11.2   ● Multi-Dimensional Segment Tree

```cpp
const int N = 1024;
int seg[2050][2050];
void updx(int yp, int x, int val){ int xp = x+N-1;
  if (yp >= N){ seg[yp][xp] = val; }
  else{ seg[yp][xp] = max(seg[yp<<1][xp], seg[yp<<1|1][xp]); }
  xp >>= 1; while (xp){ seg[yp][xp] = max(seg[yp][xp<<1], seg[yp][xp<<1|1]); xp >>= 1; }
}
void upd(int y, int x, int val){ int yp = y+N-1;
  updx(yp, x, val); yp >>= 1;
  while (yp){ updx(yp, x, val); yp >>= 1; }
}
```

```cpp
int qryx(int yp, int x1, int x2){ x1 += N-1; x2 += N-1;
  int res = 0; while (x1 <= x2){
    if (x1 & 1){ res = max(seg[yp][x1], res); x1 += 1; }
    if (~x2 & 1){ res = max(seg[yp][x2], res); x2 -= 1; }
    if (x1 > x2){ break; } x1 >>= 1; x2 >>= 1;
  } return res;
}
int qry(int y1, int y2, int x1, int x2){ y1 += N-1; y2 += N-1;
  int res = 0; while (y1 <= y2){
    if (y1 & 1){ res = max(qryx(y1, x1, x2), res); y1 += 1; }
    if (~y2 & 1){ res = max(qryx(y2, x1, x2), res); y2 -= 1; }
    if (y1 > y2){ break; } y1 >>= 1; y2 >>= 1;
  } return res;
}
```

11.3   ● Persistent Segment Tree

```cpp
const int N = 1000000;
struct Node{ int sum; int nxt[2]; };
vector<Node> seg; vector<int> root;

void upd(int ni, int ns, int ne, int qi, int qx){
  if (ns == ne){ seg[ni].sum += qx; return; }
  int nm = ns+ne >> 1; int nxi = (qi <= nm ? 0 : 1);
  int p = seg[ni].nxt[nxi]; seg[ni].nxt[nxi] = seg.size();
  seg.push_back(seg[p]);
  if (qi <= nm){ upd(seg[ni].nxt[nxi], ns, nm, qi, qx); }
  else{ upd(seg[ni].nxt[nxi], nm+1, ne, qi, qx); }
  seg[ni].sum += qx;
}
inline void upd(int idx, int pos, int val){
  root.push_back(seg.size()); seg.push_back(seg[root[idx]]);
  upd(root.back(), 0, N, pos, val);
}

int qry(int ni, int ns, int ne, int qs, int qe){
  if (qs <= ns && ne <= qe){ return seg[ni].sum; }
  int nm = ns+ne >> 1;
  int res = 0;
  if (qs <= nm && seg[ni].nxt[0]){ res += qry(seg[ni].nxt[0], ns, nm, qs, qe); }
  if (nm+1 <= qe && seg[ni].nxt[1]){ res += qry(seg[ni].nxt[1], nm+1, ne, qs, qe); }
  return res;
}
inline int qry(int idx, int st, int ed){
  return qry(root[idx], 0, N, st, ed);
}
```

11.4   ● Segment Tree Beats

Lazy Segtree, but less lazy and more planning.

Time Complexity: $\mathcal{O}(Q \log N)$, amortized.

```cpp
struct Node{ int mx1, mx2, mxc; int sum; int laz; }; // update on a = min(a, x)
inline Node mrg(const Node& l, const Node& r){ Node p;
  p.mx1 = max(l.mx1, r.mx1);
  p.mx2 = max((l.mx1 == p.mx1 ? l.mx2 : l.mx1), (r.mx1 == p.mx1 ? r.mx2 : r.mx1));
  p.mxc = (l.mx1 == p.mx1 ? l.mxc : 0) + (r.mx1 == p.mx1 ? r.mxc : 0);
  p.sum = l.sum + r.sum;
  p.laz = INF; return p;
}
```

```
Node seg[2097152];
inline void pro(int ni, int ns, int ne){
  if (ns != ne){
    seg[ni<<1].laz = min(seg[ni<<1].laz, seg[ni].laz);
    seg[ni<<1|1].laz = min(seg[ni<<1|1].laz, seg[ni].laz);
  }
  ll dif = seg[ni].laz - seg[ni].mx1; if (dif <= 0){
    seg[ni].sum += dif * seg[ni].mxc;
    seg[ni].mx1 += dif;
  } seg[ni].laz = INF;
}
void upd(int ni, int ns, int ne, int qs, int qe, int qx){
  pro(ni, ns, ne);
  if (qe < ns || ne < qs || seg[ni].mx1 <= qx){ return; }
  if (qs <= ns && ne <= qe && seg[ni].mx2 < qx){ seg[ni].laz = qx; return pro(ni, ns, ne); }
  int nm = ns+ne >> 1;
  upd(ni<<1, ns, nm, qs, qe, qx); upd(ni<<1|1, nm+1, ne, qs, qe, qx);
  seg[ni] = mrg(seg[ni<<1], seg[ni<<1|1]);
} inline void upd(int st, int ed, int val){ return upd(1, 1, N, st, ed, val); }
Node qry(int ni, int ns, int ne, int qs, int qe){
  pro(ni, ns, ne);
  if (qe < ns || ne < qs){ return {-1, -1, 0, 0, INF}; }
  if (qs <= ns && ne <= qe){ return seg[ni]; }
  int nm = ns+ne >> 1;
  return mrg(qry(ni<<1, ns, nm, qs, qe), qry(ni<<1|1, nm+1, ne, qs, qe));
} inline Node qry(int st, int ed){ return qry(1, 1, N, st, ed); }
```

11.5 ● Splay Tree

k is 1-indexed, assuming dummy on both end. insert inserts after k-th vertex (front if $k = 0$).
Time Complexity: $\mathcal{O}(Q \log N)$, amortized.

```
const int INF = 1e9;
struct Node{
  int p = -1, l = -1, r = -1; bool flp = 0; int val = 0;
  int cnt = 1; ll sum = 0; int mn = INF, mx = -INF;
}; vector<Node> tree; int root = -1;
Node init(int x){
  Node p; p.cnt = 1; p.val = x;
  p.sum = p.mn = p.mx = x; return p;
} void put(int ni, const Node p){
  tree[ni].cnt = p.cnt; tree[ni].sum = p.sum; tree[ni].mn = p.mn; tree[ni].mx = p.mx;
}
Node mrg(int li, int ri){
  const Node &l = tree[li], &r = tree[ri];
  Node p; p.cnt = l.cnt + r.cnt;
  p.sum = l.sum + r.sum; p.mn = min(l.mn, r.mn); p.mx = max(l.mx, r.mx);
  return p;
}
void propagate(int ni){
  if (ni == -1){ return; } if (tree[ni].flp == 0){ return; }
  swap(tree[ni].l, tree[ni].r);
  if (tree[ni].l != -1){ tree[tree[ni].l].flp ^= 1; }
  if (tree[ni].r != -1){ tree[tree[ni].r].flp ^= 1; }
  tree[ni].flp = 0;
}
void update(int ni){
  put(ni, init(tree[ni].val));
  if (tree[ni].l != -1){ propagate(tree[ni].l); put(ni, mrg(tree[ni].l, ni)); }
```

```
  if (tree[ni].r != -1){ propagate(tree[ni].r); put(ni, mrg(ni, tree[ni].r)); }
} inline void update(int ni, ll val){ tree[ni].val = val; return update(ni); }
void rotate(int ni){
  propagate(tree[ni].p); propagate(ni);
  int p1 = tree[ni].p; if (p1 == -1){ return; }
  int p2 = tree[p1].p; if (p2 != -1){
    (tree[p2].l == p1 ? tree[p2].l : tree[p2].r) = ni;
    tree[ni].p = p2;
  } else{ root = ni; tree[ni].p = -1; }
  int p3 = -1; if (tree[p1].l == ni){
    tree[p1].l = p3 = tree[ni].r; tree[ni].r = p1;
  } else{
    tree[p1].r = p3 = tree[ni].l; tree[ni].l = p1;
  } tree[p1].p = ni; if (p3 != -1){ tree[p3].p = p1; }
  update(p1); update(ni);
}
int splay(int ni, int top = -1){
  while (tree[ni].p != top){
    propagate(tree[tree[ni].p].p); propagate(tree[ni].p); propagate(ni);
    int p1 = tree[ni].p; int p2 = tree[p1].p;
    if (p2 != top){ rotate((tree[p2].l==p1) == (tree[p1].l==ni) ? p1 : ni); }
    rotate(ni);
  } propagate(ni); return ni;
}
int kth(int k){
  int ni = root; while (1){
    propagate(ni);
    int cnt = (tree[ni].l == -1 ? 0 : tree[tree[ni].l].cnt);
    if (cnt == k){ break; }
    if (k < cnt){ ni = tree[ni].l; }
    else{ k -= cnt+1; ni = tree[ni].r; }
  } return splay(ni);
}
int segment(int st, int ed){
  kth(ed+1); int r = root; kth(st-1); splay(r, root); return tree[tree[root].r].l;
}
void insert(int k, int val){
  if (root == -1){ root = tree.size(); tree.push_back(init(val)); return; }
  int ni = root; while (1){
    int cnt = (tree[ni].l == -1 ? 0 : tree[tree[ni].l].cnt);
    if (k < cnt){
      if (tree[ni].l == -1){
        tree[ni].l = tree.size(); tree.push_back(init(val));
        tree[tree[ni].l].p = ni; ni = tree[ni].l; break;
      } else{ ni = tree[ni].l; }
    } else{
      if (tree[ni].r == -1){
        tree[ni].r = tree.size(); tree.push_back(init(val));
        tree[tree[ni].r].p = ni; ni = tree[ni].r; break;
      } else{ k -= cnt+1; ni = tree[ni].r; }
    }
  } splay(ni);
}
void erase(int k){
  segment(k, k); tree[tree[root].r].l = -1; splay(tree[root].r);
}
```

```cpp
void rev(int st, int ed){
  if (st > ed){ return; }
  int ni = segment(st, ed); tree[ni].flp ^= 1; propagate(ni);
}
```

11.6 • Link-Cut Tree

Use with *Splay Tree*. Path Update not tested.

```cpp
bool isroot(int ni){
  if (tree[ni].p == -1){ return 1; }
  return tree[tree[ni].p].l != ni && tree[tree[ni].p].r != ni;
}
void rotate(int ni){
  propagate(tree[ni].p); propagate(ni);
  int p1 = tree[ni].p; if (isroot(ni)){ return; }
  int p2 = tree[p1].p; if (!isroot(p1)){
    (tree[p2].l == p1 ? tree[p2].l : tree[p2].r) = ni;
    tree[ni].p = p2;
  } else{ tree[ni].p = p2; }
  int p3 = -1; if (tree[p1].l == ni){
    tree[p1].l = p3 = tree[ni].r; tree[ni].r = p1;
  } else{
    tree[p1].r = p3 = tree[ni].l; tree[ni].l = p1;
  } tree[p1].p = ni; if (p3 != -1){ tree[p3].p = p1; }
  update(p1); update(ni);
}
int splay(int ni, int top = -1){
  while (!isroot(ni)){
    propagate(tree[tree[ni].p].p); propagate(tree[ni].p); propagate(ni);
    int p1 = tree[ni].p; int p2 = tree[p1].p;
    if (!isroot(p1)){ rotate((tree[p2].l==p1) == (tree[p1].l==ni) ? p1 : ni); }
    rotate(ni);
  } propagate(ni); return ni;
}
void access(int ni){
  splay(ni); tree[ni].r = -1;
  while (tree[ni].p != -1){
    splay(tree[ni].p); tree[tree[ni].p].r = ni; splay(ni);
  }
}
void reroot(int ni){
  access(ni); splay(ni); tree[ni].flp ^= 1;
}
int parent(int ni){
  access(ni); propagate(ni);
  if (tree[ni].l == -1){ return -1; }
  ni = tree[ni].l; propagate(ni);
  while (tree[ni].r != -1){ ni = tree[ni].r; propagate(ni); }
  return splay(ni);
}
int find(int ni){
  access(ni); propagate(ni);
  if (0){
    int n = tree.size();
    for (int i = 0; i < n; i++){
      cout << "Node " << i << ": " << tree[i].p << ", " << tree[i].l << ' ' << tree[i].r <<
      endl;
      cout << tree[i].flp << ' ' << tree[i].val << ' ' << tree[i].sum << endl;
```

```cpp
    } cout << endl << flush;
  }
  while (tree[ni].l != -1){ ni = tree[ni].l; propagate(ni); }
  return splay(ni);
}
void link(int ni, int pi){
  reroot(ni); access(ni); access(pi); tree[ni].l = pi; tree[pi].p = ni; update(ni);
}
void cut(int ni){
  access(ni); tree[tree[ni].l].p = -1; tree[ni].l = -1; update(ni);
}
int lca(int vi, int wi){
  access(vi); access(wi); splay(vi);
  if (tree[vi].p != -1){ return tree[vi].p; } else{ return vi; }
}

// Connected? find(vi) == find(wi)
void upd(int vi, ll x){
  splay(vi); update(vi, x);
}
void upd(int vi, int wi, ll x){
  int r = find(vi);
  reroot(vi); access(wi); splay(vi); tree[vi].laz += x; propagate(vi);
  reroot(r);
  // Edge Update: v - l - w excluding l.
  int li = lca(vi, wi);
  access(li); splay(li); propagate(li); update(li, tree[li].val-x);
}
ll qry(int vi, int wi){
  int li = lca(vi, wi); ll res = tree[li].val;
  access(vi); splay(li); if (tree[li].r != -1){ res += tree[tree[li].r].sum; }
  access(wi); splay(li); if (tree[li].r != -1){ res += tree[tree[li].r].sum; }
  return res;
}
```

12 Query & Decomposition

12.1 • Heavy-Light Decomposition

Time Complexity: $\mathcal{O}(T(N)\log N)$ per update/query.

```cpp
vector<int> adj[500020]; int par[500020];
int siz[500020], dep[500020];
void dfs1(int now, int pre){
  siz[now] = 1; par[now] = pre; erase(adj[now], pre);
  for (int& nxt : adj[now]){
    dep[nxt] = dep[now]+1; dfs1(nxt, now);
    siz[now] += siz[nxt];
    if (siz[nxt] >= siz[adj[now][0]]){ swap(nxt, adj[now][0]); }
  }
}
pi2 ord[500020]; int ont;
int chn[500020];
void dfs2(int now, int pre){
  ord[now].fr = ++ont;
  for (int nxt : adj[now]){
    if (nxt == adj[now][0]){ chn[nxt] = chn[now]; }
    else{ chn[nxt] = nxt; } dfs2(nxt, now);
  }
  ord[now].sc = ont;
```

```
}
// dfs1(root, root); chn[root] = root; dfs2(root, root);
void upd_pth(int v, int w, int x){
  while (chn[v] != chn[w]){
    if (dep[chn[v]] > dep[chn[w]]){ swap(v, w); }
    upd(ord[chn[w]].fr, ord[w].fr, x); w = par[chn[w]];
  }
  if (dep[v] > dep[w]){ swap(v, w); } upd(ord[v].fr, ord[w].fr, p);
  // if Edge Weight: if v == w then no update. otherwise v = chl[v][0] before update
}
u32 qry_pth(int v, int w){
  u32 res = 0; while (chn[v] != chn[w]){
    if (dep[chn[v]] > dep[chn[w]]){ swap(v, w); }
    res += qry(ord[chn[w]].fr, ord[w].fr); w = par[chn[w]];
  }
  if (dep[v] > dep[w]){ swap(v, w); }
  return res+qry(ord[v].fr, ord[w].fr);
  // if Edge Weight: if v == w then no query. otherwise v = chl[v][0] before query
}
```

12.2   • Centroid Decomposition

Time Complexity: $\mathcal{O}(N \log N)$.

```
bool chk[100020]; int siz[100020];
void sizf(int v, int p){
  siz[v] = 1; for (pi2 e : adj[v]){
    int w = e.fr; if (chk[w]){ continue; } if (w == p){ continue; }
    sizf(w, v); siz[v] += siz[w];
  }
}
pi2 cent = {-1, -1}; void centroid(int v, int p, int n){
  bool flg = 1; for (pi2 e : adj[v]){
    int w = e.fr; if (chk[w]){ continue; } if (w == p){ continue; }
    centroid(w, v, n); if (siz[w]*2 > n){ flg = 0; }
  } if (flg && (n-siz[v])*2 <= n){ (cent.fr == -1 ? cent.fr : cent.sc) = v; }
} inline void centroid(int v){ cent = {-1, -1}; sizf(v, v); centroid(v, v, siz[v]); }

ll dnc(int v){
  centroid(v); v = cent.fr;
  // solve on path that passes v. divide it by subtree, and merge subtree-wise. use
  top-down.
  chk[v] = 1; for (pi2 e : adj[v]){
    int w = e.fr; if (chk[w]){ continue; }
    ans += dnc(w);
  } return ans;
}
```

12.3   • Centroid Tree

Repeatedly apply centroid to the subtrees, and make the tree. The height of the centroid tree is at most $\lfloor \log_2 N \rfloor + 1$.

Also, the path $v \Rightarrow w$ from original tree always pass through LCA of v and w in centroid tree.

```
vector<int> tree[100020]; int par[100020];
int dfs(int v){
  centroid(v); v = cent.fr; chk[v] = 1;
  for (pi2 e : adj[v]){
    int w = e.fr; if (chk[w]){ continue; }
    int p = dfs(w); tree[v].push_back(p); par[p] = v;
  } return v;
}
```

12.4   • Tree Compression

Time Complexity: $\mathcal{O}(K \log N)$ per query.

```
vector<int> arr;
for (int i = 1; i <= n; i++){ int v; cin >> v; arr.push_back(v); }
sort(arr.begin(), arr.end(), [](int v, int w){ return ord[v] < ord[w]; });
for (int i = 1; i < n; i++){
  int v = arr[i-1], w = arr[i]; arr.push_back(lca(v, w));
} sort(arr.begin(), arr.end(), [](int v, int w){ return ord[v] < ord[w]; });
arr.erase(unique(arr.begin(), arr.end()), arr.end());
int al = arr.size(); for (int i = 1; i < al; i++){
  ll l = lca(arr[i-1], arr[i]);
  adj[cvt(arr, l)].push_back({i, dis[arr[i]]-dis[l]});
  adj[i].push_back({cvt(arr, l), dis[arr[i]]-dis[l]});
}
```

12.5   • Parallel Binary Search

Time Complexity: $\mathcal{O}(Q \log X)$.

```
// Binary Search: path available after ans-th event.
// 0-th event: impossible as nothing is active
// m-th event: possible as everything is connected
struct Query{ int st, ed; int idx; int v, w; };
Query qrr[100020];
while (1){
  sort(qrr+1, qrr+q+1, [](Query& q1, Query& q2){
    return (q1.st+q1.ed)/2 < (q2.st+q2.ed)/2;
  });
  for (int i = 1; i <= n; i++){ par[i] = -1; }
  bool flg = 0;
  int qi = 1; for (int ei = 0; ei <= n; ei++){
    if (ei > 0){ // Update
      int v = evt[ei].fr;
      par[v] = v; for (pi2 p : adj[v]){
        int w = p.fr; if (par[w] != -1){ uni(v, w); }
      }
    }
    while (qi <= q){ // Try answering the queries
      if (qrr[qi].st+1 > qrr[qi].ed-1){ qi++; continue; }
      int mid = qrr[qi].st+qrr[qi].ed >> 1; flg = 1;
      if (mid != ei){ break; }
      int v = qrr[qi].v, w = qrr[qi].w;
      if (par[v] != -1 && par[w] != -1 && fnd(v) == fnd(w)){ qrr[qi].ed = mid; }
      else{ qrr[qi].st = mid; } qi++;
    }
  } if (!flg){ break; }
} sort(qrr+1, qrr+q+1, [](Query& q1, Query& q2){ return q1.idx < q2.idx; });
for (int i = 1; i <= q; i++){ cout << evt[qrr[i].ed].sc << endl; } // ed = first possible
```

13   Greedy

13.1   • Job Scheduling w/ Deadline & Duration

```
pl2 arr[250020]; // {deadline, duration}
sort(arr+1, arr+n+1, [](pl2 p1, pl2 p2){ return p1.fr > p2.fr; });
priority_stack<ll> pq; int ans = 0;
int i = 1; while (i <= n){
  int j = i; while (j <= n){
    if (arr[i].fr == arr[j].fr){ j += 1; } else{ break; }
  }
```

```
  for (int p = i; p < j; p++){ pq.push(arr[p].sc); }
  ll tim = arr[i].fr - arr[j].fr; while (tim > 0){
    if (pq.empty()){ break; } ll t = pq.top(); pq.pop();
    ll d = min(t, tim); if (t-d == 0){ tim -= d; ans += 1; }
    else{ tim -= d; t -= d; pq.push(t); }
  } i = j;
} cout << ans;
```

13.2 ● Bounded Scheduling

If $x \geq B_i$, then $x \leftarrow x + A_i$. $B_i + A_i \geq 0$.

```
pl2 arr[100020]; // (A[i], B[i])
sort(arr+1, arr+n+1, [](pl2 p1, pl2 p2){
  ll a1 = p1.fr, b1 = p1.sc;
  ll a2 = p2.fr, b2 = p2.sc;
  if ((a1 >= 0) != (a2 >= 0)){ return (a1 >= 0) > (a2 >= 0); }
  if (a1 >= 0){ return b1 < b2; } else{ return a1+b1 > a2+b2; }
});
ll x = 0; for (int i = 1; i <= n; i++){
  if (x < arr[i].sc){ /* Impossible; Quit */ }
  x += arr[i].fr;
} /* Possible */
```

14 Heuristics

14.1 ● Simulated Annealing

14.2 ● Diversified Late Acceptance Search

15 Code Snippet - C++

15.1 ● Default Setting

```
#include <bits/stdc++.h>
#define endl '\n'
const int PRECISION = 0;
using namespace std;

void Main(){

}

int main(){
  ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
  cout.setf(ios::fixed); cout.precision(PRECISION); Main();
}
```

15.2 ● Line Input

```
inline void icl(istream& in){ in.ignore(998244353, '\n'); } // Between cin and getline
string s; getline(cin, s); // No newline at the end
```

15.3 ● Vector Manipulation

```
template <typename T> inline void unq(vector<T>& v){
  sort(v.begin(), v.end()); v.erase(unique(v.begin(), v.end()), v.end());
}
template <typename T> inline int cvt(vector<T>& v, T x){
  return lower_bound(v.begin(), v.end(), x) - v.begin();
}
template <typename T> inline void erase(vector<T>& v, const T& x){
  v.erase(remove(v.begin(), v.end(), x), v.end());
} // removes every x in v
```

15.4 ● Bitwise Function

```
inline int bit1(int x){ return __builtin_popcount(x); } // # of 1
inline int bit2(int x){ return x==0 ? 32 : __builtin_ctz(x); } // max k s.t. n | 2^k
inline int bitl(int x){ return x==0 ? 0 : 32 - __builtin_clz(x); } // # of bits
inline int bith(int x){ return x==0 ? 0 : 1 << bitl(x)-1; } // max 2^k s.t. 2^k <= n
inline int bitp(int x){
  int y = bith(x); return x==0 ? 0 : y << (x!=y);
} // min 2^k s.t. n <= 2^k
```

15.5 ● Randomization

```
const time_t TIME = chrono::high_resolution_clock::now().time_since_epoch().count();
mt19937 gen(TIME);
uniform_int_distribution<int> rng(a, b); // [a, b] range
int value = rng(gen); shuffle(begin, end, gen);
```

15.6 ● Custom Hash

```
struct custom_hash {
  static uint64_t splitmix64(uint64_t x) {
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
    x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
    return x ^ (x >> 31);
  }

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

unordered_map<ll, int, custom_hash> mp;
```

15.7 ● Policy Based Data Structure

```
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.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 K, typename V> using ordered_map = tree<K, V, less<K>, rb_tree_tag,
tree_order_statistics_node_update>;

ordered_set<int> s; s.insert(value);
s.find_by_order(index); // 0-based, returns iterator
s.order_of_key(value); // 0-based, lower_bound

ordered_map<int, int> mp;
mp.insert({1, 4}); mp[4] = 7;
mp.find_by_order(1)->first // key, 4
mp.find_by_order(1)->second // value, 7
```

15.8 ● Custom Comparison

```
bool cmp(int a, int b){ return a < b; }
sort(begin, end, cmp); // a -> b
set<int, decltype(&cmp)> s(cmp); // a -> b (inorder)
priority_queue<int, vector<int>, decltype(&cmp)> pq(cmp); // b on top
```

```cpp
sort(v.begin(), v.end(), [](pi2 a, pi2 b){ return a.fr+a.sc < b.fr+b.sc; });
auto st = lower_bound(v.begin(), v.end(), x, [](pi2 p, int x){ return p.fr+p.sc < x; });
auto ed = upper_bound(v.begin(), v.end(), x, [](int x, pi2 p){ return x < p.fr+p.sc; });
// [st, ed) is where p.fr+p.sc == x

struct cmp{
  bool operator()(const pi2& p1, const pi2& p2){ return p1.fr+p1.sc < p2.fr+p2.sc; }
};
tree<pi2, null_type, cmp, rb_tree_tag, tree_order_statistics_node_update> s;
```

16　Code Snippet - Python

16.1　● Default Setting

```python
import sys; input = lambda: sys.stdin.readline().rstrip('\n')
inputs = lambda t, l=tuple: l(map(t, input().split()))
sys.setrecursionlimit(100000)
```

16.2　● List Manipulation

```python
l.reverse(); l = list(reversed(l))
l.sort(); l = sorted(l)
```

16.3　● Arbitrary Pricision

```python
from fractions import * # Fraction, no limit
x = Fraction(10)

from decimal import * # Decimal, limited
getcontext().prec = 28 # Number of precision = digits
x = Decimal(20)
```

16.4　● Data Structure

```python
from collections import deque
q = deque() # Deque
q.append(x); q.appendleft(x)
x = q.pop(); x = q.popleft()

import heapq # Heap
pq = []
heapq.heappush(pq) # push
x = heapq.heappop(pq) # pop
```

16.5　● Custom Hashing

```python
import time; TIME = int(time.time() * 1000)
dict[(x, TIME)] # anti-hash
```

16.6　● Custom Comparison

```python
from functools import cmp_to_key
def cmp(a, b): return sgn(a-b) # a < b then -, a > b then +, otherwise 0
l.sort(key=cmp_to_key(cmp)); sorted(l, key=cmp_to_key(cmp))
```

17　Miscellaneous

17.1　● List of Primes

17.2　● List of Highly Composite Numbers

- 840: 32 divisors
- 720 720: 240 divisors
- 735 134 400: 1 344 divisors
- 963 761 198 400: 6 720 divisors
- 897 612 484 786 617 600: 103 680 divisors

17.3　● ASCII Table

```
  |  0   1   2   3   4   5   6   7
--+--------------------------------
0 | NUL DLE SP  0   @   P   `   p
1 | SOH DC1 !   1   A   Q   a   q
2 | STX DC2 "   2   B   R   b   r
3 | ETX DC3 #   3   C   S   c   s
4 | EOT DC4 $   4   D   T   d   t
5 | ENQ NAK %   5   E   U   e   u
6 | ACK SYN &   6   F   V   f   v
7 | BEL ETB '   7   G   W   g   w
8 |  BS CAN (   8   H   X   h   x
9 | TAB  EM )   9   I   Y   i   y
A |  LF SUB *   :   J   Z   j   z
B |  VT ESC +   ;   K   [   k   {
C |  FF  FS ,   <   L   \   l   |
D |  CR  GS -   =   M   ]   m   }
E |  SO  RS .   >   N   ^   n   ~
F |  SI  US /   ?   O   _   o  DEL
```