

Team Note of **hibye1217**

SolvedAC: [hibye1217](#), CodeForces: [hibye1217](#), AtCoder: [hibye1217](#)

Compiled on December 5, 2025

Contents

1 Have You Tried...	2	6 Graph Theory - Connectivity	9
1.1 • Soft Techniques	2	6.1 • Strongly Connected Component	9
1.2 • Hard Techniques	2	6.2 • Solution of the 2-SAT	9
2 Mathematics - Number Theory	2	6.3 • Biconnected Component	9
2.1 • Extended Euclidean Algorithm	2	6.4 • Articulation Point & Edge	10
2.2 • Chinese Remainder Theorem	2		
2.3 • Möbius Function	2		
2.4 • Miller-Rabin	2		
2.5 • Pollard's Rho	3		
2.6 • Power Tower	3		
3 Mathematics - Combinatorics	3	7 Graph Theory - Network Flow & Matching	10
3.1 • Lucas Theorem	3	7.1 • Hall's Marriage Theorem	10
3.2 • Well-known Sequences	3	7.2 • Dinitz	10
3.3 • Lindström-Gessel-Viennot Lemma	3	7.3 • Modeling - Flow with Demands	10
3.4 • Generating Function	3	7.4 • Minimum Cost Maximum Flow w/ SPFA	10
4 Mathematics - Algebra	3	7.5 • Stable Marriage Problem	10
4.1 • Modeling - 2-SAT - Operations	3	7.6 • Hungarian Algorithm	11
4.2 • Modeling - 2-SAT - At most 1	3	7.7 • General Matching	11
4.3 • Modeling - Minimizing Quadratic Pseudo-Boolean Function	4		
4.4 • Fast Fourier Transform & Number Theoretic Transform	4		
4.5 • Fast Welsh-Hadamard Transform	4		
4.6 • Gauss-Jordan Elimination	4		
4.7 • Floor Sum of Arithmetic Progression	4		
4.8 • Linear Programming	4		
5 Mathematics - Geometry	5	8 Graph Theory - Miscellaneous	12
5.1 • Line Intersection	5	8.1 • Eulerian Path	12
5.2 • Convex Hull	5	8.2 • Tree Isomorphism	12
5.3 • Point in Convex Polygon	5		
5.4 • Point in Non-convex Polygon	5		
5.5 • Rotating Calipers	5		
5.6 • Minimum Enclosing Circle	5		
5.7 • Bulldozer Trick	6		
5.8 • Halfplane Intersection	6		
5.9 • Shamos-Hoey	6		
5.10 • Faces of the Planar Graph	7	9 Dynamic Programming	12
		9.1 • Rerooting	12
		9.2 • Rerooting - All Directions	12
		9.3 • Sum over Subset	12
		9.4 • Convex Hull Trick	13
		9.5 • Divide and Conquer Optimization	13
		9.6 • WQS Binary Search (Aliens Trick)	13
		9.7 • Monotone Queue Optimization	13
		9.8 • Slope Trick	13
		9.9 • Knuth Optimization	13
		9.10 • Kitamasa	13
		9.11 • Hirschberg	14
		9.12 • Connection Profile	15
		9.13 • Permutation DP	15
		9.14 • Berlekamp-Massey	15
		10 String	16
		10.1 • Knuth-Morris-Pratt	16
		10.2 • Z Algorithm	16
		10.3 • Aho-Corasick	16
		10.4 • Suffix Array & Longest Common Prefix Array	16
		10.5 • Suffix Array - Traversing	16
		10.6 • Manacher's	17
		10.7 • Rolling Hash & Rabin-Karp	17

11 Data Structures

11.1 ● Li-Chao Tree	17
11.2 ● Multi-Dimensional Segment Tree	17
11.3 ● Persistent Segment Tree	17
11.4 ● Segment Tree Beats	18
11.5 ● Splay Tree	18
11.6 ● Link-Cut Tree	19

12 Query & Decomposition

12.1 ● Heavy-Light Decomposition	19
12.2 ● Centroid Decomposition	19
12.3 ● Tree Compression	20
12.4 ● Parallel Binary Search	20
12.5 ● Offline Dynamic Query	20

13 Greedy

13.1 ● Job Scheduling w/ Deadline & Duration	21
13.2 ● Bounded Scheduling	21

14 Heuristics

14.1 ● Simulated Annealing	21
14.2 ● Diversified Late Acceptance Search	21

15 Code Snippet - C++

15.1 ● Default Setting	21
15.2 ● Line Input	21
15.3 ● Vector Manipulation	21
15.4 ● Bitwise Function	21
15.5 ● Randomization	21
15.6 ● Custom Hash	21
15.7 ● Policy Based Data Structure	21
15.8 ● Custom Comparison	21

16 Code Snippet - Python

16.1 ● Default Setting	22
16.2 ● List Manipulation	22
16.3 ● Arbitrary Precision	22
16.4 ● Data Structure	22
16.5 ● Custom Hashing	22
16.6 ● Custom Comparison	22

17 Miscellaneous

17.1 ● List of Primes	22
17.2 ● List of Highly Composite Numbers	22
17.3 ● ASCII Table	22

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 auxiliary 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?
- 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 Euclidean Algorithm**

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

```
p13 egcd(l1 a, l1 b){
    if (b == 0){ return {{1, 0}, a}; }
    p13 p = egcd(b, a%b);
    l1 x = p.fr.fr, y = p.fr.sc, g = p.sc;
    l1 xx = y, yy = x - a/b*y;
    return {{xx, yy}, g};
} inline l1 finv(l1 a, l1 b){ p13 p = egcd(a, n); return (p.fr.fr%n+n)%n; }
```

2.2 ● Chinese Remainder Theorem

```
p12 crt(p12 f1, p12 f2){
    if (f1.sc < f2.sc){ swap(f1, f2); }
    l1 a1 = f1.fr, m1 = f1.sc; l1 a2 = f2.fr, m2 = f2.sc;
    l1 g = gcd(m1, m2); l1 l = lcm(m1, m2);
    if ((a2-a1)%g != 0){ return {-1, -1}; }
    l1 mg1 = m1/g, mg2 = m2/g, ag = (a2-a1)/g;
    l1 y = ag *finv(mg1, mg2) %mg2;
    l1 x = m1*y + a1; return {(x%l+1)%l, 1};
}
```

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<l1> prr = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
bool prime(l1 n){
    if (n <= 40){ if (n == p){ if (n == p){ return 1; } }
    return 0;
} if (n%2 == 0){ return 0; }
int s = 0; l1 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

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

```

int n; int arr[N+20];
const int CUT = 100, EXP = 10; // >= log2(mod) // >= log2(CUT)
const int LEN = 4; // (tower with length of LEN+1) > CUT
int f(int idx){
    int val = arr[n]; for (int i = n-1; i >= idx; i--){
        if (val >= EXP){ return CUT+1; }
        int res = 1; while (val--){
            res = min(res*arr[i], CUT+1);
        } val = res;
    } return val;
}
int f(int idx, int mod){
    if (mod == 1){ return 0; }
    int cnt = n-idx+1; if (cnt <= LEN){
        if (f(idx+1) <= CUT){ return fpow(arr[idx], f(idx+1), mod); }
    } return fpow(arr[idx], f(idx+1, phi[mod]) + CUT*mod, 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 $i \rightarrow j$.

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

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 $\langle b_0, b_1, \dots, b_{k-1}, b_k, b_{k+1}, \dots \rangle = \langle 0, 0, \dots, 0, a_0, a_1, \dots \rangle$.
 - If A and B are both OGF, then $B(x) = x^k A(x)$.
 - If A and B are both EGF, then $B(x) = \int^{(k)} A(x) dx^k$.
- Let $\langle b_0, b_1, \dots \rangle = \langle a_k, a_{k+1}, \dots \rangle$.
 - If A and B are both OGF, then $B(x) = (A(x) - a_0 - a_1x - \dots - a_{k-1}x^{k-1})/x^k$.
 - If A and B are both EGF, then $B(x) = \frac{a^k}{d^k x^k} A(x)$.
- 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 $b_n = \sum_{i_1+i_2+\dots+i_k=n} a_{i_1} a_{i_2} \dots a_{i_k}$.
 - If A and B are both EGF, then $b_n = \sum_{i_1+i_2+\dots+i_k=n} \frac{n!}{i_1! i_2! \dots i_k!} a_{i_1} a_{i_2} \dots a_{i_k}$.
- If A and B are both OGF with $B(x) = \frac{A(x)}{1-x}$, then $b_i = a_0 + a_1 + \dots + a_i$.
- $(1+x)^k = \sum_{n=0}^{\infty} \binom{k}{n} x^n$.
- $\frac{1}{(1-x)^k} = \sum_{n=0}^{\infty} \binom{n+k-1}{n} x^n = \sum_{n=0}^{\infty} {}_k H_n x^n$.

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, \dots, x_N) can be done with the following method. Let y_i be true if one of x_1, x_2, \dots, 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, \dots, x_N , minimize the cost $f(x_1, x_2, \dots, x_N) = c + \sum_i b_i(x_i) + \sum_{i < j} a_{i,j}(x_i, x_j)$. 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, \dots, 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 + 1] * wp;
      arr[i+j] = a+b; arr[i+j + 1] = 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 $i \neq r$.

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){
  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

5.1 • Line Intersection

```

int Intersection(p14 l1, p14 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){
        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 */
}

```

5.2 • Convex Hull

```

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

```

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

```

int j = 0; for (int i = 0; i < sp; i++){
    while (j < sp){
        int ip = (i+1) % sp, jp = (j+1) % sp;
        pl2 l1 = { arr[ip].fr - arr[i].fr, arr[ip].sc - arr[i].sc };
        pl2 r1 = { arr[jp].fr - arr[j].fr, arr[jp].sc - arr[j].sc };
        if (ccw({0, 0}, l1, r1) < 0){ break; }
        maxDist(i, j); j += 1;
    } if (j < sp){ ans = max(ans, dist(i, j)); }
}

```

5.6 • Minimum Enclosing Circle

```

inline ld dis(const pd2& p1, const pd2& p2){
    ld y = p1.fr-p2.fr, x = p1.sc-p2.sc;
    return sqrtl(y*y + x*x);
}
inline bool inside(const pd2& o, const ld& r, const pd2& p){
    return dis(o, p) <= r+eps;
}
inline int ccw(const pl2& p1, const pl2& p2, const pl2& 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 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.

```

5.7 • Bulldozer Trick

```

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;
}

```

5.8 • Halfplane Intersection

```

inline pl2 ltoi(const pl4& l){ return pl2{l.sc.fr-l.fr.fr, l.sc.sc-l.fr.sc}; }
pair<bool, pd2> itx(const pl4& l1, const pl4& l2){
    pl2 v1 = ltoi(l1), v2 = ltoi(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)crs(dp, v2) / crs(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 = itx(l1, l2); if (!res.fr){ return 0; }
        return out(res.sc, l);
    }
    bool hpi(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 = ltoi(l1), v2 = ltoi(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 = ltoi(l), v2 = ltoi(dq[dql-1]);
                if (ccw(0, v1, v2) == 0){
                    if ((v1 > 0) == (v2 > 0)){
                        pl2 p1 = l1.fr;
                        if (!out(p1, dq[dql-1])){ dq.pop_back(); dq.push_back(l); }
                        continue;
                    } else{ return 0; }
                }
                if (dql < 2 || !out(dq[dql-1], l, dq[0])){ dq.push_back(l); }
            }
            return dq.size() >= 3;
        }
        // Actual Point Reconstruction is done by
        // 0. res = []
        // 1. for (int i = 0; i < dql; i++)
        // 2.     l1, l2 = dq[i], dq[i+1]
        // 3.     if (itx(l1, l2)) res.push(itx(l1, l2).point)
    }
}

```

5.9 • Shamos-Hoey

This assumes that **none of the line** is parallel to the y -axis. You can just rotate the plane to ensure this.

Apparently I just ran the Shamos-Hoey thrice with (x, y) , (y, x) , $(x+y, x-y)$.

```

#define x first
#define y second

```

```

inline int sgn(ll x){ return (x > 0) - (x < 0); }

```

```

pl2 operator+(const pl2& p1, const pl2& p2){ return { p1.fr+p2.fr, p1.sc+p2.sc }; }
pl2 operator-(const pl2& p1, const pl2& p2){ return { p1.fr-p2.fr, p1.sc-p2.sc }; }
pl2 operator*(const pl2& p, const ll& a){ return { p.fr*a, p.sc*a }; }
pl2 operator/(const pl2& p, const ll& a){ return { p.fr/a, p.sc/a }; }
pl2& operator+=(pl2& p1, const pl2& p2){ return p1 = p1+p2; }
pl2& operator-=(pl2& p1, const pl2& p2){ return p1 = p1-p2; }
pl2& operator*=(pl2& p, const ll& a){ return p = p*a; }
pl2& operator/=(pl2& p, const ll& a){ return p = p/a; }

ll operator*(const pl2& p1, const pl2& p2){ return p1.x*p2.x + p1.y*p2.y; }
ll operator/(const pl2& p1, const pl2& p2){ return p1.x*p2.y - p1.y*p2.x; }

inline int ccw(const pl2& p1, const pl2& p2){ return sgn(p1/p2); }
inline int ccw(const pl2& p1, const pl2& p2, const pl2& p3){ return ccw(p2-p1, p3-p1); }

inline ll dis(const pl2& p1, const pl2& p2){ pl2 r(p1-p2); return r*r; }

ll ptr;
class Lin{ public:
    pl2 p1, p2;
    Lin(){ p1 = {0, 0}; p2 = {0, 0}; }
    Lin(pl2 a, pl2 b){ p1 = a; p2 = b; }
};

inline ld f(const Lin& l, ll x){
    return (ld)(l.p2.y-l.p1.y) / (l.p2.x-l.p1.x) * (x-l.p1.x) + l.p1.y;
}

bool operator<(const Lin& l1, const Lin& l2){
    ld p1 = f(l1, ptr), p2 = f(l2, ptr);
    return p1 < p2;
}

bool crs(const Lin& l1, const Lin& l2){
    int c11 = ccw(l1.p1, l1.p2, l2.p1), c12 = ccw(l1.p1, l1.p2, l2.p2);
    int c21 = ccw(l2.p1, l2.p2, l1.p1), c22 = ccw(l2.p1, l2.p2, l1.p2);
    //cout << "CCW " << c11 << ' ' << c12 << ' ' << c21 << ' ' << c22 << endl << flush;
    if (c11 == 0 && c12 == 0){
        ll p11, p12, p21, p22;
        if (l1.p1.x == l1.p2.x){ p11 = l1.p1.y; p12 = l1.p2.y; p21 = l2.p1.y; p22 = l2.p2.y; }
        else{ p11 = l1.p1.x; p12 = l1.p2.x; p21 = l2.p1.x; p22 = l2.p2.x; }
        if (p11 > p21){ swap(p11, p21); swap(p12, p22); }
        //cout << p11 << ' ' << p12 << " | " << p21 << ' ' << p22 << endl << flush;
        return p12-p11 + p22-p21 >= p22-p11;
    }
    return c11 != c12 && c21 != c22;
}

int n;
pl4 arr[200020], pnt[200020]; pl4 pos[400020];
multiset<Lin> lin;

bool smh(){
    sort(pos+1, pos+n+n+1);
    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;

```

```

        //cout << "I " << i << endl << x << ' ' << y << " / " << ed << ' ' << idx << endl << flush;
        if (pnt[idx].fr.x == pnt[idx].sc.x){ continue; }
        Lin l(pnt[idx].fr, pnt[idx].sc);
        //cout << "L " << l.p1.x << ' ' << l.p1.y << " . " << l.p2.x << ' ' << l.p2.y << endl << flush;
        ptr = x;
        if (!ed){
            auto it = lin.insert(l);
            if (it != lin.begin())&gt;&gt; if (crs(*it, *prev(it))){ return 1; } }
            if (next(it) != lin.end())&gt;&gt; if (crs(*it, *next(it))){ return 1; } }
        }
        else{
            auto it = lin.lower_bound(l);
            //cout << (*it).p1.x << ' ' << (*it).p1.y << " / "
            // << (*it).p2.x << ' ' << (*it).p2.y << endl << flush;
            if (it != lin.begin() && next(it) != lin.end()){
                if (crs(*prev(it), *next(it))){ return 1; }
            }
            lin.erase(it);
        }
    }
    return 0;
}



## 5.10 • Faces of the Planar Graph



This code is from BOJ 17442 - Tripartite Graph.



```

#include <bits/stdc++.h>
#define endl '\n'
using namespace std;
using ll = long long;
using ld = long double;
using pi2 = pair<int, int>;
using pl2 = pair<ll, ll>;
using pi3 = pair<pi2, int>;
#define fr first
#define sc second
const int PRECISION = 6;
template <typename T> using priority_stack = priority_queue< T, vector<T>, greater<T> >;
inline int sgn(ll x){ return (x > 0) - (x < 0); }
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(const vector<T>& v, const T& x){
 return lower_bound(v.begin(), v.end(), x) - v.begin() + 1;
}

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);
}

pi2 arr[100020]; vector<int> xp;
vector<int> adj[100020];

vector<int> gph[100020];

```


```

```

map<int, int> mp_gph[100020];
pi3 qrr[200020]; int ans[200020];
int deg[100020];
int mn, mx;
void dfs(int v, int u){
    //cout << "dfs " << u << ' ' << v << endl;
    mn = min({mn, arr[v].sc, arr[u].sc}); mx = max({mx, arr[v].sc, arr[u].sc});
    int ui = mp_gph[u][v]; gph[u][ui] *= -1;
    int vi = mp_gph[v][u]; vi += 1; if (vi == gph[v].size()) vi = 0;
    if (gph[v][vi] <= 0){ return; }
    else{ return dfs(gph[v][vi], v); }
}
bool chk[100020];
void dfs2(int v){
    chk[v] = 1;
    mn = min(mn, arr[v].sc); mx = max(mx, arr[v].sc);
    for (int w : gph[v]){
        if (chk[w]){ continue; }
        dfs2(w);
    }
}
const int N = 131072;
int fen1[131080], fen2[131080];
void upd(int* fen, int pos, int val){
    for (int i = pos; i < N; i+=i&-i){ fen[i] += val; }
}
int qry(int* fen, int pos){
    int res = 0;
    for (int i = pos; i > 0; i-=i&-i){ res += fen[i]; }
    return res;
}
const int INF = 1e9;
void Main(){
    int n, m, q; cin >> n >> m >> q; int val = n-m;
    for (int i = 1; i <= n; i++){
        cin >> arr[i].sc >> arr[i].fr;
        xp.push_back(arr[i].sc);
    } unq(xp);
    while (m--){
        int v, w; cin >> v >> w;
        adj[v].push_back(w); adj[w].push_back(v);
    }
    for (int i = 1; i <= q; i++){
        cin >> qrr[i].fr.fr >> qrr[i].fr.sc;
        qrr[i].sc = i;
    }
    {
        queue<int> q;
        for (int i = 1; i <= n; i++){
            deg[i] = adj[i].size();
            if (deg[i] <= 1){ deg[i] = 0; q.push(i); }
        }
        while (!q.empty()){
            int v = q.front(); q.pop();
            for (int w : adj[v]){
                deg[w] -= 1; if (deg[w] == 1){
                    deg[w] = 0; q.push(w);
                }
            }
        }
        for (int v = 1; v <= n; v++){
            if (deg[v] > 0){
                for (int w : adj[v]){
                    if (deg[w] > 0){ gph[v].push_back(w); }
                }
            }
        }
        sort(gph[v].begin(), gph[v].end(), [v](int w1, int w2){
            pl2 p1 = {arr[w1].fr-arr[v].fr, arr[w1].sc-arr[v].sc};
            pl2 p2 = {arr[w2].fr-arr[v].fr, arr[w2].sc-arr[v].sc};
            const pl2& p0 = {0, 0};
            if ((p1 > p0) != (p2 > p0)){ return p1 > p2; }
            return ccw(p0, p1, p2) > 0;
        });
        int l = gph[v].size(); for (int i = 0; i < l; i++){
            int w = gph[v][i]; mp_gph[v][w] = i;
        }
    }
    vector<pl2> faces;
    for (int v = 1; v <= n; v++){
        for (int w : gph[v]){
            if (w <= 0){ continue; }
            //cout << "face " << v << ' ' << w << endl;
            mn = INF; mx = -INF; dfs(w, v);
            faces.push_back({mn, mx}); val += 1;
            //cout << "faces " << mn << ' ' << mx << endl;
        }
    }
    vector<pl2> rmv;
    for (int v = 1; v <= n; v++){
        for (int &w : gph[v]){ w = -w; }
    }
    for (int v = 1; v <= n; v++){
        if (!chk[v]){
            mn = INF; mx = -INF; dfs2(v);
            if (mn == mx){ continue; }
            rmv.push_back({mn, mx}); val -= 1;
            //cout << "nofaces " << mn << ' ' << mx << endl;
        }
    }
    for (int i = 1; i <= q; i++){ ans[i] = val; }
    //cout << "init " << val << ' ' << faces.size() << ' ' << rmv.size() << endl;
    sort(qrr+1, qrr+q+1);
    for (int v = 1; v <= n; v++){
        for (int w : adj[v]){
            for (int w : adj[v]){
                if (v < w){ continue; }
                int x1 = arr[v].sc, x2 = arr[w].sc;
                if (x1 > x2){ swap(x1, x2); }
                int i1 = cvt(xp, x1), i2 = cvt(xp, x2);
            }
        }
    }
}

```

```

//cout << "edge1 " << x1 << ' ' << x2 << endl;
upd(fen1, i1, +1); upd(fen1, i2, -1);
}
}
for (pl2 p : faces){
    int x1 = p.fr, x2 = p.sc;
    int i1 = cvt(xp, x1), i2 = cvt(xp, x2);
    //cout << "face1 " << x1 << ' ' << x2 << endl;
    upd(fen1, i1, -1); upd(fen1, i2, +1);
}
for (pl2 p : rmv){
    int x1 = p.fr, x2 = p.sc;
    int i1 = cvt(xp, x1), i2 = cvt(xp, x2);
    //cout << "noface1 " << x1 << ' ' << x2 << endl;
    upd(fen1, i1, +1); upd(fen1, i2, -1);
}
sort(faces.begin(), faces.end()); sort(rmv.begin(), rmv.end());
int fi = 0, ri = 0; int fl = faces.size(), rl = rmv.size();
for (int i = 1; i <= q; i++){
    int st = qrr[i].fr.fr, ed = qrr[i].fr.sc; int qi = qrr[i].sc;
    ans[qi] += qry(fen1, cvt(xp, st)-1) + qry(fen1, cvt(xp, ed)-1);
    while (fi < fl){
        int l = faces[fi].fr, r = faces[fi].sc;
        if (l > st){ break; }
        int p = cvt(xp, r); upd(fen2, p, +1); fi += 1;
    }
    while (ri < rl){
        int l = rmv[ri].fr, r = rmv[ri].sc;
        if (l > st){ break; }
        int p = cvt(xp, r); upd(fen2, p, -1); ri += 1;
    }
    ans[qi] += qry(fen2, N-1) - qry(fen2, cvt(xp, ed)-1);
    //cout << "query " << st << ' ' << ed << " / " << val << ' ' << qry(fen1, cvt(xp, st)-1)
    << ' ' << qry(fen1, cvt(xp, ed)-1) << ' ' << qry(fen2, N-1) << ' ' << qry(fen2, cvt(xp, ed)-1) << endl;
}
for (int i = 1; i <= q; i++){ cout << ans[i] << endl; }
}

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

```

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] == -1){ 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

This code is old, and are not recommended to directly copy this.
for (int i = 1; i <= 2*n; i++){
    for (int j : adj[i]){
        int pi = scc[i], pj = scc[j];
        if (pi == pj){ continue; }
        dag[pi].push_back(pj); cnt[pj] += 1;
    }
}
queue<int> q; for (int i = 1; i <= 2*n; i++){
    if (scc[i] != i){ continue; }
    if (cnt[i] == 0){ q.push(i); }
}
memset(ans, -1, sizeof(ans));
while (!q.empty()){
    int now = q.front(); q.pop();
    for (int v : vtx[now]){
        if (ans[v] != -1){ continue; }
        ans[v] = 0; ans[idx(-inv(v))] = 1;
    }
    for (int nxt : dag[now]){
        pc[nxt] -= 1; if (pc[nxt] == 0){ q.push(nxt); }
    }
}

```

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[320];
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 i th vertex need to have
 - $\text{req}_i < 0$: Need to get it from source
 - $\text{req}_i > 0$: Need to send it to sink
- $v \rightarrow w$ with lower/upper bound of l and u .
 - $\text{req}_v = l$
 - $\text{req}_w = -l$
 - Capacity of $v \rightarrow w = u - l$
- Create new source and sink, and connect the v with source or sink with capacity req_v .
- From the old sink to old source, connect an edge with the capacity ∞ .
- 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+1] = 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;
    }
}

int num[50020];
vector<int> adj[50020]; map<int, int> rnk[10020];

int ptr[50020]; priority_queue<pi2> res[10020];
// pq: rank, person
```

7.5 • Stable Marriage Problem

Try matching greedily. 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 num[50020];
vector<int> adj[50020]; map<int, int> rnk[10020];

int ptr[50020]; priority_queue<pi2> res[10020];
// pq: rank, person
```

```

bool f(int i, int j, int idx){
    //cout << "F " << i << ' ' << j << ' ' << idx << endl;
    if (res[j].size() != num[j]) { res[j].push({rnk[j][i], i}); ptr[i] = idx; return 1; }
    pi2 p = res[j].top();
    if (p.fr > rnk[j][i]){
        res[j].pop(); res[j].push({rnk[j][i], i});
        ptr[p.sc] += 1; for (int& k = ptr[p.sc]; k < adj[p.sc].size(); k++){
            assert(1 <= adj[p.sc][k] && adj[p.sc][k] <= 10000);
            if (f(p.sc, adj[p.sc][k], k)){ break; }
        } return 1;
    } else{ return 0; }
}

for (int i = 1; i <= n; i++){
    for (int& idx = ptr[i]; idx < adj[i].size(); idx++){
        int j = adj[i][idx];
        if (f(i, j, idx)){ 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] = 1; 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

```

vector<int> adj[1020]; int ptr[1020];
int ind[1020], oud[1020];

int cnt = 0;
void dfs(int now){
    for (int& p = ptr[now]; p < adj[now].size();){
        int nxt = adj[now][p++];
        cnt += 1; dfs(nxt);
    }
}

```

8.2 • Tree Isomorphism

Only has the hash part, as apparently that was the only thing I needed back then??

```

pi2 res = {-1, -1};
int cent(int now, int pre){
    int siz = 1; bool flg = 1; for (int nxt : adj[now]){
        if (nxt == pre){ continue; }
        int cnt = cent(nxt, now);
        if (cnt*2 > n){ flg = 0; } siz += cnt;
    }
    if (flg && (n-siz)*2 <= n){ (res.fr == -1 ? res.fr : res.sc) = now; }
    return siz;
}
void dfs(int now, int pre){

```

```

    dep[now] = dep[pre]+1; dpt = max(dpt, dep[now]);
    dpv[dep[now]].push_back(now);
    for (int nxt : adj[now]){
        if (nxt == pre){ continue; } dfs(nxt, now);
    }

    int val[N+20];
    cent(1, 1); if (res.sc != -1){
        erase(adj[res.fr], res.sc); erase(adj[res.sc], res.fr);
        n++; adj[res.fr].push_back(n); adj[res.sc].push_back(n);
        adj[n].push_back(res.fr); adj[n].push_back(res.sc); res = {n, -1};
    } dfs(res.fr, res.fr); // unrooted. for rooted ignore above, just dfs(root, root).
    for (int d = dpt; d > 1; d--){
        vector<pvi> res;
        for (int v : dpv[d-1]){
            res.emplace_back(); res.back().sc = v;
            for (int w : adj[v]){
                if (dep[w] != d){ continue; }
                res.back().fr.push_back(val[w]);
            }
        }
        for (pvi& p : res){ sort(p.fr.begin(), p.fr.end()); }
        sort(res.begin(), res.end()); int len = res.size();
        int x = 0; for (int i = 0; i < len; i++){
            if (i > 0 && res[i].fr == res[i-1].fr){ val[res[i].sc] = x; }
            else{ val[res[i].sc] = ++x; }
        }
    }
}

```

9 Dynamic Programming

9.1 • Rerooting

Tree DP, with changing root.

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

```

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 • Rerooting - All Directions

9.3 • Sum over Subset

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

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

```

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]; }
    }
}

```

9.4 • 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.

```
inline ll crs(p12 l1, p12 l2){ return (l1)(l2.sc-l1.sc)/(l1.fr-l2.fr); }
pair<p12, ll> stk[1000020]; int sp = 0;
inline void psh(p12 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; }
} p12 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);
```

9.5 • Divide and Conquer Optimization

$$D_{k,i} = \max_{j < i} D_{k-1,j} + A_{j,i} \cdot \text{opt}_i < \text{opt}_{i+1}$$

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

```
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);
}
```

9.6 • WQS Binary Search (Aliens Trick)

$$D_{k,i} = \min_{j < i} D_{k-1,j} + A_{j,i}. D_{k,N} \text{ must be convex.}$$

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

```
// 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;
```

9.7 • 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.

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

```
deque<int> dq;
p12 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){
    //cout << "solve " << 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;
}
```

9.8 • 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 duplicates, and adding the linear function.

9.9 • Knuth Optimization

$$D_{i,j} = D_{i,k} + D_{k+1,j} + A_{i,j} \text{ where } \text{opt}_{i,j-1} \leq \text{opt}_{i,j} \leq \text{opt}_{i+1,j}. \text{ This is satisfied if } C \text{ is monge.}$$

Time Complexity: $\mathcal{O}(N^2)$.

```
for (int d = 0; d < n; d++){
    for (int s=1, e=e+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;
```

9.10 • Kitamasa

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

```
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;
}
```

```

    } 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;
}



### 9.11 • 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, j1, p2+1, j2, 0);
}
else {
    int p2 = mxp.sc;
    solve(i1, m1, i2, p2-1, 0); cout << s2[p2] << endl; solve(m1+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);
}

```

9.12 • Connection Profile

Single Connected Region, with DP condition.

Time Complexity: $\mathcal{O}(NMB_M)$, $B_9 = 21147$. Actual state is about 2000?

```

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.

```

9.13 • 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: $(i, j) \rightarrow (i+1, j+1)$.
- Append $i+1$ to one of the component: $(i, j) \rightarrow (i+1, j)$.
- Connect two components using $i+1$: $(i, j) \rightarrow (i+1, j-1)$.

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

9.14 • 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-1]); }
    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 sa1[100020], int pos1[100020], lcp1[100020];
int tmp[100020], cnt[100020], res[100020];
void init(string s, int* sa, int* pos, int* lcp){
    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){
        /*cout << k << endl << flush;
        cout << "SA "; for (int i = 0; i < sl; i++){ cout << sa[i] << ' '; } cout << endl;
        cout << "RES "; for (int i = 0; i < sl; i++){ cout << res[i] << ' '; } cout << endl;
        if (k > 10000000){ break; }*/
        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+1 < sl && s2+1 < sl){
            if (s[s1+1] != s[s2+1]){ break; } else{ l += 1; }
        } lcp[p1] = l;
    }
}

```

10.5 • Suffix Array - Traversing

```

qry(st, ed) denotes the minimum value of LCP[st, ed].
// sa[st..ed][i..j) -> t more eat, res jumped
void solve(int st, int ed, int i, int t, int res){
    res += 1; pi2 p = qry(st, ed); int mid = p.sc, j = min(p.fr, n);
    int len = j-i; int cnt = prf[sa[st] + j-1] - prf[sa[st] + i];
    //cout << "solve " << st << ' ' << ed << ' ' << i << ' ' << t << ' ' << res << " / " <<
    mid << ' ' << j << ' ' << len << ' ' << cnt << endl << flush;
}

```

```

if (cnt < t){
    if (j >= n){ cout << -1; exit(0); }
    solve(st, mid, j, t-cnt, res + len-1 - cnt);
    solve(mid+1, ed, j, t-cnt, res + len-1 - cnt);
}
else{
    int d = prf[sa[st] + i] + t;
    auto p = lower_bound(prf, prf+n+n+1, d) - prf; p -= sa[st];
    int l = p-i; res += l-t; ans = max(ans, res);
    return;
}

```

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];
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.

```

// Max version. If Min then use the commented 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

```

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; }
}
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

```

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.

```

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);
}
11 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



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



```

struct Node{ Node *p, *l, *r; int sum; bool flp; }; Node *root;
void propagate(Node *nod){
if (nod->flp == 0){ return; }
swap(nod->l, nod->r);
if (nod->l){ nod->l->flp ^= 1; } if (nod->r){ nod->r->flp ^= 1; }
nod->flp = 0;
}
void update(Node *nod){
if (nod->l){ /* Left Value, swap if l->flp */ }
if (nod->r){ /* Right Value, swap if r->flp */ }
/* update nod */
}
void rotate(Node *now){
Node *par = now->p; if (!par){ return; }
Node *pre = par->p; if (pre){
if (pre->l == par){ pre->l = now; } else{ pre->r = now; }
now->p = pre;
} else{ root = now; now->p = nullptr; }
Node *nxt;
if (par->l == now){ par->l = nxt = now->r; now->r = par; }
else{ par->r = nxt = now->l; now->l = par; }
par->p = now; if (nxt){ nxt->p = par; }
update(par); update(now);
}
void splay(Node *now, Node *top = nullptr){
while (now->p != top){
Node *par = now->p; Node *pre = par->p;
if (pre != top){
if ((pre->l==par) == (par->l==now)){ rotate(par); } else{ rotate(now); }
} rotate(now);
}
int num = 1;
void insert(int val){ // Insert to Rightmost Only
if (!root){ root = new Node; return; }
Node *ptr = root; while (ptr->r){ ptr = ptr->r; }
ptr->r = new Node; ptr->r->p = ptr;
ptr = ptr->r; ptr->val = ptr->mx1 = ptr->mxr = ptr->ans = val;
ptr->index = num; num += 1;
splay(ptr);
}
void kth(int k){ k += 1;
Node *ptr = root; while (1){
propagate(ptr);
if (ptr->l){
if (ptr->l->cnt >= k){ ptr = ptr->l; continue; }
if (ptr->l->cnt+1 < k){ k -= ptr->l->cnt+1; ptr = ptr->r; continue; }
break;
}
else{
if (k != 1){ k -= 1; ptr = ptr->r; } else{ break; }
}
}

```


```

```

    }
    splay(ptr);
}

void segmentate(int st, int ed){ // Desired Node in root->r->l.
    kth(ed+1); Node* nod = root; kth(st-1); splay(nod, root);
}
void reverse(int st, int ed){
    segmentate(st, ed); Node* nod = root->r->l; nod->flp ^= 1; propagate(nod);
}

11.6 • Link-Cut Tree

Use with Splay Tree.

struct Node{ Node *l, *r, *p; int idx; };
inline bool isRoot(Node* now){
    if (!now->p){ return 1; }
    return now->p->l != now && now->p->r != now;
}
void rotate(Node *now){
    if (isRoot(now)){ return; }
    Node *par = now->p; Node *pre = par->p;
    if (isRoot(par)){ now->p = pre; }
    else{
        if (pre->l == par){ pre->l = now; } else{ pre->r = now; }
        now->p = pre;
    }
    Node *nxt;
    if (par->l == now){ par->l = nxt = now->r; now->r = par; }
    else{ par->r = nxt = now->l; now->l = par; }
    par->p = now; if (nxt){ nxt->p = par; }
}
void splay(Node *now){
    while (!isRoot(now)){
        Node *par = now->p; Node *pre = par->p;
        if (!isRoot(par)){
            if ((par->l==now) == (pre->l==par)){ rotate(par); } else{ rotate(now); }
            rotate(now);
        }
    }
}
void access(Node *now){
    splay(now); now->r = nullptr;
    while (now->p){
        splay(now->p); now->p->r = now; splay(now);
    }
}
void link(Node *now, Node *par){
    access(now); access(par);
    now->l = par; par->p = now;
}
void cut(Node *now){
    access(now); now->l->p = nullptr; now->l = nullptr;
}
Node* lca(Node *v, Node *w){
    access(v); access(w); spl(v);
    if (v->p){ return v->p; } else{ return v; }
}
Node* vtx[N+20]; vtx[i]->idx = i; // might be a good idea to directly access the node

```

12 Query & Decomposition

12.1 • Heavy-Light Decomposition

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

```

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[200020];
int siz[100020];
void szf(int now, int pre){
    siz[now] = 1;
    for (int nxt : adj[now]){
        if (chk[nxt]){ continue; }
        if (nxt == pre){ continue; }
        szf(nxt, now); siz[now] += siz[nxt];
    }
}
int cen(int now){
    szf(now, -1); int pre = -1, tar = siz[now]/2;
    while (1){

```

```

bool flg = 1;
for (int nxt : adj[now]) { if (chk[nxt]) { continue; }
    if (nxt == pre) { continue; }
    if (siz[nxt] <= tar) { continue; }
    pre = now; now = nxt; flg = 0; break;
} if (flg) { return now; }
}

int ans[100020];
void dnc(int now, int pre){
    now = cen(now);
    /* Conquer Part */
    chk[now] = 1; for (int nxt : adj[now]){
        if (chk[nxt]) { continue; }
        dnc(nxt, now);
    }
}

```

12.3 • 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++){
    l1 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]});
}

```

12.4 • 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
}

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

```

12.5 • Offline Dynamic Query

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

// BOJ 24272: 루트 노트가 많은 트리일수록 좋은 트리이다

```

inline pi3 f(int& v, int& w, string& s){
    if (v > w){ swap(v, w); swap(s[0], s[1]); }
    return {{v, w}, s=="--" ? 0 : s[0]=="-' ? +1 : -1};
}

map<pi2, pi2> mp;

int N = 131072;
vector<pi3> qrr[262150];
void upd(int ni, int ns, int ne, int qs, int qe, pi3 p){
    if (qe < ns || ne < qs){ return; }
    if (qs <= ns && ne <= qe){ qrr[ni].push_back(p); return; }
    int nm = ns+ne >> 1;
    upd(ni<<1, ns, nm, qs, qe, p); upd(ni<<1|1, nm+1, ne, qs, qe, p);
} inline void upd(int st, int ed, pi3 p){ return upd(1, 1, N, st, ed, p); }

int par[100020]; int siz[100020]; int ind[100020], oud[100020]; int ans;
struct State{ int qi, typ; int v, w; int ans; };
stack<State> stk;
void init(int n){
    for (int i = 1; i <= n; i++){ par[i] = i; siz[i] = 1; ind[i] = oud[i] = 0; }
    ans = n;
}
void undo(int qi){
    while (!stk.empty()){
        State p = stk.top(); if (p.qi != qi){ break; } stk.pop();
        if (p.typ == 1){
            oud[p.w] -= oud[p.v]; ind[p.w] -= ind[p.v];
            siz[p.w] -= siz[p.v]; par[p.v] = p.v; ans = p.ans;
        }
        if (p.typ == 2){
            ind[p.w] -= 1; oud[p.v] -= 1; ans = p.ans;
        }
    }
}
int fnd(int v){ return par[v]==v ? v : fnd(par[v]); }
void uni(int qi, int v, int w){ // v -- w
    v = fnd(v); w = fnd(w); if (siz[v] > siz[w]){ swap(v, w); }
    stk.push({qi, 1, v, w, ans});
    if (ind[w] && ind[v]){ ans = 0; }
    if (ans > 0 && ind[w] == 0 && ind[v] > 0){ ans -= siz[w]; }
    if (ans > 0 && ind[v] == 0 && ind[w] > 0){ ans -= siz[v]; }
    par[v] = w; siz[w] += siz[v];
    ind[w] += ind[v]; oud[w] += oud[v];
}
void dir(int qi, int v, int w){ // v -> w
    v = fnd(v); w = fnd(w);
}

```

```

    stk.push({qi, 2, v, w, ans});
    if (ind[w]) { ans = 0; }
    if (ans > 0 && ind[w] == 0) { ans -= siz[w]; }
    oud[v] += 1; ind[w] += 1;
}

void dnc(int ni, int ns, int ne){
    for (pi3 p : qrr[ni]){
        int v = p.fr.fr, w = p.fr.sc, d = p.sc;
        if (d == -1){ swap(v, w); d = +1; }
        if (d == 0){ uni(ni, v, w); }
        if (d == +1){ dir(ni, v, w); }
    }
    if (ns != ne){
        int nm = ns+ne >> 1;
        dnc(ni<<1, ns, nm); dnc(ni<<1|1, nm+1, ne);
    }
    else{ cout << ans << endl; }
    undo(ni);
} void dnc(){ return dnc(1, 1, N); }

void Main(){
    int n; cin >> n;
    for (int i = 1; i < n; i++){
        int v, w; string d; cin >> v >> d >> w; pi3 p = f(v, w, d);
        mp[p.fr] = {0, p.sc};
    }
    int q; cin >> q; N = q;
    for (int i = 1; i <= q; i++){
        int v, w; string d; cin >> v >> d >> w; pi3 p = f(v, w, d);
        upd(mp[p.fr].fr, i-1, {p.fr, mp[p.fr].sc}); mp[p.fr] = {i, p.sc};
    }
    for (pair<pi2, pi2> kv : mp){
        upd(kv.sc.fr, q, {kv.fr, kv.sc.sc});
    }
    init(n); dnc();
}

```

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)) * 0xb58476d1ce4e5b9;
        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

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

```
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

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

16.3 • Arbitrary Precision

```
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

```
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

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

16.6 • Custom Comparison

```
from functools import cmp_to_key
def cmp(a, b): return sgn(a-b) # a < b then -, a > b then +, otherwise 0
1.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	/	?	0	_	o	DEL