

vimrc

```

1 map <F9> :wall! <CR> :!g++ -Wall -Wextra -Wshadow ↵
   Wno-unused-result -o %:r % -std=c++14 -DHOME ↵
   D_GLIBCXX_DEBUG -fsanitize=address <CR>
2 map <F7> :wall! <CR> :!g++ -Wall -Wextra -Wshadow ↵
   Wno-unused-result -o %:r % -std=c++14 -DHOME ↵
   02 <CR>
3 map <F8> :wall! <CR> :!./%:r <CR>
4
5 inoremap {<CR> {<CR>}<ESC>O
6 map <c-a> ggVG
7
8 set nu
9 set rnu
10 syntax on
11
12 map <c-t> :tabnew <CR>
13 map <c-l> :tabn <CR>
14 map <c-h> :tabp <CR>
15
16 set cin
17 set sw=4
18 set so=99
19 set bs=2
20 set et
21 set sts=4

```

template

```

1 // team : SPb ITMO University 1
2 #include <bits/stdc++.h>
3
4 #define F first
5 #define S second
6 #define pb push_back
7 #define forn(i, n) for(int i = 0 ; (i) < (n) ; ++i)
8 #define eprintf(...) fprintf(stderr, __VA_ARGS__) ↵
   fflush(stderr)
9 #define sz(a) ((int)(a).size())
10 #define all(a) (a).begin(), a.end()
11 #define pw(x) (1LL<<(x))
12
13 using namespace std;
14
15 typedef long long ll;
16 typedef double dbl;
17 typedef vector<int> vi;
18 typedef pair<int, int> pi;
19
20 const int INF = 1.01e9;
21 const dbl eps = 1e-9;
22
23 /* — main part — */
24
25
26
27
28
29
30
31 int main()
32 {
33 #define TASK ""
34 #ifdef home
35     assert(freopen(TASK".in", "r", stdin));
36     //assert(freopen(TASK".out", "w", stdout));
37 #endif
38
39
40
41
42
43 #ifdef home
44     eprintf("time = %d ms\n", (int)(clock() * 1000. / ↵
        CLOCKS_PER_SEC));
45 #endif
46     return 0;
47 }

```

crt

```

1 int CRT(int a1, int m1, int a2, int m2)
2 {
3     return (a1 - a2 % m1 + m1) * (1ll)rev(m2, m1) % m1 ↵
        * m2 + a2;
4 }

```

fastIO

```

1 #include <cstdio>
2 #include <algorithm>
3
4 /** Interface */
5
6 inline int readInt();
7 inline int readUInt();
8 inline bool isEof();
9
10 /** Read */
11
12 static const int buf_size = 100000;
13 static char buf[buf_size];
14 static int buf_len = 0, pos = 0;
15
16 inline bool isEof() {
17     if (pos == buf_len) {
18         pos = 0, buf_len = fread(buf, 1, buf_size, stdin)↵
            );
19         if (pos == buf_len) return 1;
20     }
21     return 0;
22 }
23
24 inline int getChar() { return isEof() ? -1 : buf[pos↵
    ++]; }
25
26 inline int readChar() {
27     int c = getChar();
28     while (c != -1 && c <= 32) c = getChar();
29     return c;
30 }
31
32 inline int readUInt() {
33     int c = readChar(), x = 0;
34     while ('0' <= c && c <= '9') x = x * 10 + c - '0',↵
        c = getChar();
35     return x;
36 }
37
38 inline int readInt() {
39     int s = 1, c = readChar();
40     int x = 0;
41     if (c == '-') s = -1, c = getChar();
42     while ('0' <= c && c <= '9') x = x * 10 + c - '0',↵
        c = getChar();
43     return s == 1 ? x : -x;
44 }
45
46
47 // 10M int [0..1e9)
48 // cin 3.02
49 // scanf 1.2
50 // cin_sync_with_stdio(false) 0.71
51 // fastRead getchar 0.53
52 // fastRead fread 0.15

```

fft

```

1 namespace fft
2 {
3     const int maxBase = 21;
4     const int maxN = 1 << maxBase;
5
6     struct num
7     {
8         dbl x, y;
9         num() {}

```

```

10     num(dbl xx, dbl yy): x(xx), y(yy) {}
11     num(dbl alp): x(cos(alp)), y(sin(alp)) {}
12 };
13
14 inline num operator + (num a, num b) { return num(←
15     a.x + b.x, a.y + b.y); }
16 inline num operator - (num a, num b) { return num(←
17     a.x - b.x, a.y - b.y); }
18 inline num operator * (num a, num b) { return num(←
19     a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x); }
20 inline num conj(num a) { return num(a.x, -a.y); }
21
22 const dbl PI = acos(-1);
23
24 num root[maxN];
25 int rev[maxN];
26 bool rootsPrepared = false;
27
28 void prepRoots()
29 {
30     if (rootsPrepared) return;
31     rootsPrepared = true;
32     root[1] = num(1, 0);
33     for (int k = 1; k < maxBase; ++k)
34     {
35         num x(2 * PI / pw(k + 1));
36         for (int i = pw(k - 1); i < pw(k); ++i)
37         {
38             root[2 * i] = root[i];
39             root[2 * i + 1] = root[i] * x;
40         }
41     }
42
43     int base, N;
44
45     int lastRevN = -1;
46     void prepRev()
47     {
48         if (lastRevN == N) return;
49         lastRevN = N;
50         forn(i, N) rev[i] = (rev[i >> 1] >> 1) + ((i & ←
51             1) << (base - 1));
52     }
53
54     void fft(num *a, num *f)
55     {
56         forn(i, N) f[i] = a[rev[i]];
57         for (int k = 1; k < N; k <= 1) for (int i = 0; ←
58             i < N; i += 2 * k) forn(j, k)
59         {
60             num z = f[i + j + k] * root[j + k];
61             f[i + j + k] = f[i + j] - z;
62             f[i + j] = f[i + j] + z;
63         }
64
65         num a[maxN], b[maxN], f[maxN], g[maxN];
66         ll A[maxN], B[maxN], C[maxN];
67
68         void _multMod(int mod)
69         {
70             forn(i, N)
71             {
72                 int x = A[i] % mod;
73                 a[i] = num(x & (pw(15) - 1), x >> 15);
74             }
75             forn(i, N)
76             {
77                 int x = B[i] % mod;
78                 b[i] = num(x & (pw(15) - 1), x >> 15);
79             }
80             fft(a, f);
81             fft(b, g);
82
83             forn(i, N)
84             {
85                 int j = (N - i) & (N - 1);
86                 num a1 = (f[i] + conj(f[j])) * num(0.5, 0);
87                 num a2 = (f[i] - conj(f[j])) * num(0, -0.5);
88                 num b1 = (g[i] + conj(g[j])) * num(0.5 / N, 0) ←
89                 num b2 = (g[i] - conj(g[j])) * num(0, -0.5 / N ←
90                 a[j] = a1 * b1 + a2 * b2 * num(0, 1);
91                 b[j] = a1 * b2 + a2 * b1;
92             }
93
94             fft(a, f);
95             fft(b, g);
96             forn(i, N)
97             {
98                 ll aa = f[i].x + 0.5;
99                 ll bb = g[i].x + 0.5;
100                 ll cc = f[i].y + 0.5;
101                 C[i] = (aa + bb % mod * pw(15) + cc % mod * pw ←
102                 (30)) % mod;
103             }
104         }
105
106         void prepAB(int n1, int n2)
107         {
108             base = 1;
109             N = 2;
110             while (N < n1 + n2) base++, N <= 1;
111
112             for (int i = n1; i < N; ++i) A[i] = 0;
113             for (int i = n2; i < N; ++i) B[i] = 0;
114
115             prepRoots();
116             prepRev();
117         }
118
119         void mult(int n1, int n2)
120         {
121             prepAB(n1, n2);
122             forn(i, N) a[i] = num(A[i], B[i]);
123             fft(a, f);
124             forn(i, N)
125             {
126                 int j = (N - i) & (N - 1);
127                 a[i] = (f[j] * f[j] - conj(f[i] * f[i])) * num ←
128                 (0, -0.25 / N);
129             }
130             fft(a, f);
131             forn(i, N) C[i] = (ll)round(f[i].x);
132         }
133
134         void multMod(int n1, int n2, int mod)
135         {
136             prepAB(n1, n2);
137             _multMod(mod);
138         }
139
140         int D[maxN];
141
142         void multLL(int n1, int n2)
143         {
144             prepAB(n1, n2);
145
146             int mod1 = 1.5e9;
147             int mod2 = mod1 + 1;
148
149             _multMod(mod1);
150
151             forn(i, N) D[i] = C[i];
152
153             _multMod(mod2);
154
155             forn(i, N)
156             {
157                 C[i] = D[i] + (C[i] - D[i] + (ll)mod2) * (ll) ←
158                 mod1 % mod2 * mod1;
159             }
160         }
161
162         // HOW TO USE ::
163         // — set correct maxBase
164         // — use mult(n1, n2), multMod(n1, n2, mod) and ←
165         multLL(n1, n2)
166         // — input : A[], B[]
167         // — output : C[]
168     }

```

fftint

```

1 namespace fft
2 {
3     const int mod = 998244353;
4     const int base = 20;
5     const int N = 1 << base;
6     const int ROOT = 646;
7
8     int root[N];
9     int rev[N];
10
11     void init()
12     {

```

```

13     forn(i, N) rev[i] = (rev[i >> 1] >> 1) + ((i & ←
14     1) << (base - 1));
15     int NN = N >> 1;
16     int z = 1;
17     forn(i, NN)
18     {
19         root[i + NN] = z;
20         z = z * (11)ROOT % mod;
21     }
22     for (int i = NN - 1; i > 0; --i) root[i] = root←
23     [2 * i];
24 }
25 void fft(int *a, int *f)
26 {
27     forn(i, N) f[i] = a[rev[i]];
28     for (int k = 1; k < N; k <= 1) for (int i = 0; ←
29     i < N; i += 2 * k) forn(j, k)
30     {
31         int z = f[i + j + k] * (11)root[j + k] % mod;
32         f[i + j + k] = (f[i + j] - z + mod) % mod;
33         f[i + j] = (f[i + j] + z) % mod;
34     }
35 }
36 int A[N], B[N], C[N];
37 int F[N], G[N];
38 void _mult(int eq)
39 {
40     fft(A, F);
41     if (eq) forn(i, N) G[i] = F[i];
42     else fft(B, G);
43     int invN = inv(N);
44     forn(i, N) A[i] = F[i] * (11)G[i] % mod * invN %←
45     mod;
46     reverse(A + 1, A + N);
47     fft(A, C);
48 }
49 void mult(int n1, int n2, int eq = 0)
50 {
51     for (int i = n1; i < N; ++i) A[i] = 0;
52     for (int i = n2; i < N; ++i) B[i] = 0;
53
54     _mult(eq);
55
56     //forn(i, n1 + n2) C[i] = 0;
57     //forn(i, n1) forn(j, n2) C[i + j] = (C[i + j] ←
58     A[i] * (11)B[j]) % mod;
59 }

```

blackbox

```

1 namespace blackbox
2 {
3     int A[N];
4     int B[N];
5     int C[N];
6
7     int magic(int k, int x)
8     {
9         B[k] = x;
10        C[k] = (C[k] + A[0] * (11)B[k]) % mod;
11        int z = 1;
12        if (k == N - 1) return C[k];
13        while ((k & (z - 1)) == (z - 1))
14        {
15            //mult B[k - z + 1 ... k] x A[z .. 2 * z - 1]
16            forn(i, z) fft::A[i] = A[z + i];
17            forn(i, z) fft::B[i] = B[k - z + 1 + i];
18            fft::multMod(z, z, mod);
19            forn(i, 2 * z - 1) C[k + 1 + i] = (C[k + 1 + i] ←
20            + fft::C[i]) % mod;
21            z <= 1;
22        }
23        return C[k];
24    }
25    // A — constant array
26    // magic(k, x) :: B[k] = x, returns C[k]
27    // !! WARNING !! better to set N twice the size ←
28    needed

```

halfplaneIntersection

```

1 int getPart(pt v) {
2     return less(0, v.y) || (equal(0, v.y) && less(v.x, ←
3     0));
4 }
5 int cmpV(pt a, pt b) {
6     int partA = getPart(a);
7     int partB = getPart(b);
8     if (partA < partB) return -1;
9     if (partA > partB) return 1;
10    if (equal(0, a * b)) return 0;
11    if (0 < a * b) return -1;
12    return 1;
13 }
14 double planeInt(vector<Line> l) {
15     int n = l.size();
16     sort(all(l), [](Line a, Line b) {
17         int r = cmpV(a.v, b.v);
18         if (r != 0) return r < 0;
19         return a.0 % a.v.rotate() < b.0 % a.v.rotate() ←
20         );
21     });
22     int cur = 0;
23     for (int i = 0; i < n; i++) {
24         int j = i;
25         for (; i < n && cmpV(l[j].v, l[i].v) == 0 && ←
26         cmpV(l[i].v, l[j].v) == 0; i++);
27         l[cur++] = l[i - 1];
28     }
29     n = cur;
30
31     for (int i = 0; i < n; i++)
32         l[i].id = i;
33
34     int flagUp = 0;
35     int flagDown = 0;
36     for (int i = 0; i < n; i++) {
37         int part = getPart(l[i].v);
38         if (part == 1) flagUp = 1;
39         if (part == 0) flagDown = 1;
40     }
41     if (!flagUp || !flagDown) return -1;
42
43     for (int i = 0; i < n; i++) {
44         pt v = l[i].v;
45         pt u = l[(i + 1) % n].v;
46         if (equal(0, v * u) && less(v % u, 0)) {
47             pt dir = l[i].v.rotate();
48             if (lessE(l[(i + 1) % n].0 % dir, l[i].0 % dir ←
49             )) return 0;
50             return -1;
51         }
52         if (less(v * u, 0))
53             return -1;
54     }
55
56     cur = 0;
57     vector<Line> st(n * 2);
58     for (int tt = 0; tt < 2; tt++) {
59         for (int i = 0; i < n; i++) {
60             for (; cur >= 2; cur--) {
61                 pt G = st[cur - 1] * l[i];
62                 if (!lessE(st[cur - 2].v * (G - st[cur - 2]. ←
63                 0), 0)) break;
64             }
65             st[cur++] = l[i];
66             if (cur >= 2 && lessE(st[cur - 2].v * st[cur ←
67             1].v, 0)) return 0;
68         }
69     }
70     vector<int> use(n, -1);
71     int left = -1, right = -1;
72     for (int i = 0; i < n; i++) {
73         if (use[st[i].id] == -1) {
74             use[st[i].id] = i;
75         }
76         else {
77             left = use[st[i].id];
78             right = i;
79             break;
80         }
81     }
82     vector<Line> tmp;
83     for (int i = left; i < right; i++)
84         tmp.pb(st[i]);

```

```

83 vector < pt > res;
84 for (int i = 0; i < (int)tmp.size(); i++)
85     res.pb(tmp[i] * tmp[(i + 1) % tmp.size()]);
86 double area = 0;
87 for (int i = 0; i < (int)res.size(); i++)
88     area += res[i] * res[(i + 1) % res.size()];
89 return area / 2;
90 }

```

hash_table

```

1  template <const int max_size, class HashType, class ←
      Data, const Data default_value>
2  struct hashTable {
3      HashType hash[max_size];
4      Data f[max_size];
5      int size;
6
7      int position(HashType H) const {
8          int i = H % max_size;
9          while (hash[i] && hash[i] != H)
10             if (++i == max_size)
11                 i = 0;
12             return i;
13     }
14
15     Data & operator [] (HashType H) {
16         assert(H != 0);
17         int i = position(H);
18         if (!hash[i]) {
19             hash[i] = H;
20             f[i] = default_value;
21             size++;
22         }
23         return f[i];
24     }
25 };
26
27 hashTable<13, int, int, 0> h;

```

hungary

```

1  namespace hungary
2  {
3      const int N = 210;
4
5      int a[N][N];
6      int ans[N];
7
8      int calc(int n, int m)
9      {
10         ++n, ++m;
11         vi u(n), v(m), p(m), prev(m);
12         for (int i = 1; i < n; ++i)
13         {
14             p[0] = i;
15             int x = 0;
16             vi mn(m, inf);
17             vi was(m, 0);
18             while (p[x])
19             {
20                 was[x] = 1;
21                 int ii = p[x], dd = inf, y = 0;
22                 for (int j = 1; j < m; ++j) if (!was[j])
23                 {
24                     int cur = a[ii][j] - u[ii] - v[j];
25                     if (cur < mn[j]) mn[j] = cur, prev[j] = x;
26                     if (mn[j] < dd) dd = mn[j], y = j;
27                 }
28                 forn(j, m)
29                 {
30                     if (was[j]) u[p[j]] += dd, v[j] -= dd;
31                     else mn[j] -= dd;
32                 }
33                 x = y;
34             }
35             while (x)
36             {
37                 int y = prev[x];
38                 p[x] = p[y];
39                 x = y;

```

```

40     }
41 }
42 for (int j = 1; j < m; ++j)
43 {
44     ans[p[j]] = j;
45 }
46 return -v[0];
47 }
48 // HOW TO USE ::
49 // -- set values to a[1..n][1..m] (n <= m)
50 // -- run calc(n, m) to find MINIMUM
51 // -- to restore permutation use ans[]
52 // -- everything works on negative numbers
53 //
54 // !! i don't understand this code, it's ←
55 // copped from e-maxx (and rewritten by enot110←

```

modReverseOneLine

```

1  int rev(int x, int m)
2  {
3      if (x == 1) return 1;
4      return (1 - rev(m % x, x) * (11)m) / x + m;
5  }

```

optimizations

```

1  // from anta code http://codeforces.com/contest/755/submit/23864531
2
3  #pragma GCC optimize ("O3")
4  #pragma GCC target ("sse4")
5  inline void fasterLLDivMod(unsigned long long x, ←
      unsigned y, unsigned &out_d, unsigned &out_m) {
6      unsigned xh = (unsigned)(x >> 32), xl = (unsigned)←
          x, d, m;
7  #ifdef __GNUC__
8      asm(
9          "divl %4; \n\t"
10         : "=a" (d), "=d" (m)
11         : "d" (xh), "a" (xl), "r" (y)
12         );
13 #else
14     __asm {
15         mov edx, dword ptr[xh];
16         mov eax, dword ptr[xl];
17         div dword ptr[y];
18         mov dword ptr[d], eax;
19         mov dword ptr[m], edx;
20     };
21 #endif
22     out_d = d; out_m = m;
23 }
24
25
26 // have no idea what sse flags are really cool; list←
    of some of them
27 // -- very good with bitsets
28 #pragma GCC optimize("O3")
29 #pragma GCC target ("sse, sse2, sse3, ssse3, sse4, popcnt, ←
    abm, mmx")

```

plane3DInt

```

1  //(A, v) * (B, u) -> (O, n)
2
3  pt n = v * u;
4  pt m = v * n;
5  double t = (B - A) % u / (u % m);
6  pt O = A - m * t;

```

simplex

```

1
2 struct Simplex {
3     ll a[MAX_M][MAX_N];
4     ll b[MAX_M];
5     ll c[MAX_N];
6     ll v;
7     ll n, m;
8     ll left[MAX_M];
9     ll up[MAX_N];
10
11 void init(ll nn, ll mm) {
12     n = nn;
13     m = mm;
14     v = 0;
15     for (int i = 0; i < m; i++)
16         for (int j = 0; j < n; j++)
17             a[i][j] = 0;
18     for (int i = 0; i < m; i++)
19         b[i] = 0;
20     for (int i = 0; i < n; i++)
21         c[i] = 0;
22 }
23
24 int pos[MAX_N];
25
26 void pivot(ll x, ll y) {
27     swap(left[x], up[y]);
28     ll k = a[x][y];
29     assert(abs(k) == 1);
30     a[x][y] = 1;
31     b[x] /= k;
32     int cur = 0;
33     for (int i = 0; i < n; i++) {
34         a[x][i] = a[x][i] / k;
35         if (a[x][i] != 0)
36             pos[cur++] = i;
37     }
38
39     for (int i = 0; i < m; i++) {
40         if (i == x || a[i][y] == 0) continue;
41         ll cof = a[i][y];
42         b[i] -= cof * b[x];
43         a[i][y] = 0;
44         for (int j = 0; j < cur; j++)
45             a[i][pos[j]] -= cof * a[x][pos[j]];
46     }
47     ll cof = c[y];
48     v += cof * b[x];
49     c[y] = 0;
50     for (int i = 0; i < cur; i++) {
51         c[pos[i]] -= cof * a[x][pos[i]];
52     }
53 }
54
55 ll res[MAX_N];
56 void solve() {
57     for (int i = 0; i < n; i++)
58         up[i] = i;
59     for (int i = 0; i < m; i++)
60         left[i] = i + n;
61
62     while (1) {
63         int x = -1;
64         for (int i = 0; i < m; i++)
65             if (b[i] < 0 && (x == -1 || b[i] < b[x])) {
66                 x = i;
67             }
68         if (x == -1) break;
69
70         int y = -1;
71         for (int j = 0; j < n; j++)
72             if (a[x][j] < 0) {
73                 y = j;
74                 break;
75             }
76         if (y == -1) {
77             assert(false); // no solution
78         }
79         pivot(x, y);
80     }
81     while (1) {
82         int y = -1;
83         for (int i = 0; i < n; i++)
84             if (c[i] > 0 && (y == -1 || (c[i] > c[y]))) ←
85                 y = i;
86         if (y == -1) break;
87     }

```

```

88
89     int x = -1;
90     for (int i = 0; i < m; i++) {
91         if (a[i][y] > 0) {
92             if (x == -1 || (b[i] / a[i][y] < b[x] / a[←
93                 x = i;
94             }
95         }
96     }
97     if (y == -1) {
98         assert(false); // infinite solution
99     }
100    pivot(x, y);
101 }
102
103 memset(res, 0, sizeof(res));
104
105 for (int i = 0; i < m; i++) {
106     if (left[i] < n) {
107         res[left[i]] = b[i];
108     }
109 }
110 }
111 };

```

std_rb_tree

```

1 #include "ext/pb_ds/assoc_container.hpp"
2 using namespace __gnu_pbds;
3
4 template <typename T> using ordered_set = tree<T, ←
5     null_type, less<T>, rb_tree_tag, ←
6     tree_order_statistics_node_update>;
7
8 template <typename K, typename V> using ordered_map ←
9     = tree<K, V, less<K>, rb_tree_tag, ←
10     tree_order_statistics_node_update>;
11
12 // HOW TO USE ::
13 // -- order_of_key(10) returns the number of ←
14 //    elements in set/map strictly less than 10
15 // -- *find_by_order(10) returns 10-th smallest ←
16 //    element in set/map (0-based)

```

Суффиксный автомат

```

1 namespace SA {
2     const int MAXN = 1 << 18;
3     const int SIGMA = 26;
4
5     int sz, last;
6     int nxt[MAXN][SIGMA];
7     int link[MAXN], len[MAXN], pos[MAXN];
8
9     void init() {
10         memset(nxt, -1, sizeof(nxt));
11         memset(link, -1, sizeof(link));
12         memset(len, 0, sizeof(len));
13         last = 0;
14         sz = 1;
15     }
16
17     void add(int c) {
18         int cur = sz++;
19         len[cur] = len[last] + 1;
20         pos[cur] = len[cur];
21         int p = last;
22         last = cur;
23         for (; p != -1 && nxt[p][c] == -1; p = link[p]) ←
24             nxt[p][c] = cur;
25         if (p == -1) {
26             link[cur] = 0;
27             return;
28         }
29         int q = nxt[p][c];
30         if (len[p] + 1 == len[q]) {
31             link[cur] = q;
32             return;
33         }
34         int clone = sz++;
35         memcpy(nxt[clone], nxt[q], sizeof(nxt[q]));

```

```

35     len[clone] = len[p] + 1;
36     pos[clone] = pos[q];
37     link[clone] = link[q];
38     link[q] = link[cur] = clone;
39     for (; p != -1 && nxt[p][c] == q; p = link[p]) ←
40         nxt[p][c] = clone;
41 }
42
43 int n;
44 string s;
45 int l[MAXN], r[MAXN];
46 int e[MAXN][SIGMA];
47
48 void getSufTree(string _s) {
49     memset(e, -1, sizeof(e));
50     s = _s;
51     n = s.length();
52     reverse(s.begin(), s.end());
53     init();
54     for (int i = 0; i < n; i++) add(s[i] - 'a');
55     reverse(s.begin(), s.end());
56     for (int i = 1; i < sz; i++) {
57         int j = link[i];
58         l[i] = n - pos[i] + len[j];
59         r[i] = n - pos[i] + len[i];
60         e[j][s[l[i]] - 'a'] = i;
61     }
62 }

```

dinica

```

1 namespace flow
2 {
3     const int maxn = 1e5 + 10;
4     const int maxe = 2 * maxn;
5
6     int head[maxn], next[maxe], to[maxe], f[maxe], ec ←
7         = 1;
8     int ST, EN, N = maxn;
9
10    inline void setN(int n)
11    {
12        ST = n;
13        EN = n + 1;
14        N = n + 2;
15    }
16
17    inline void _add(int x, int y, int ff)
18    {
19        ++ec;
20        to[ec] = y;
21        next[ec] = head[x];
22        head[x] = ec;
23        f[ec] = ff;
24    }
25
26    inline int add(int x, int y, int ff)
27    {
28        _add(x, y, ff);
29        _add(y, x, 0);
30        return ec - 1;
31    }
32
33    void clear()
34    {
35        forn(i, N) head[i] = 0;
36        ec = 1;
37    }
38
39    int d[maxn];
40    int q[maxn], st = 0, en = 0;
41
42    int bfs()
43    {
44        forn(i, N) d[i] = 1e9;
45        st = 0, en = 0;
46        d[ST] = 0;
47        q[en++] = ST;
48        while (st < en)
49        {
50            int x = q[st++];
51            if (x == EN) return 1;
52            for (int e = head[x]; e; e = next[e])
53            {
54                int y = to[e];
55                if (d[y] == 1e9 && f[e])

```

```

56                {
57                    d[y] = d[x] + 1;
58                    q[en++] = y;
59                }
60            }
61            return 0;
62        }
63
64        int pushed;
65        int fst[maxn];
66
67        int dfs(int x, int flow = 1e9)
68        {
69            if (x == EN)
70            {
71                pushed = flow;
72                return 1;
73            }
74            for (; fst[x]; fst[x] = next[fst[x]])
75            {
76                int e = fst[x];
77                int y = to[e];
78                if (d[y] == d[x] + 1 && f[e] && dfs(y, min(f[e] ←
79                    ], flow)))
80                {
81                    f[e] -= pushed;
82                    f[e ^ 1] += pushed;
83                    return 1;
84                }
85            }
86            return 0;
87        }
88
89        ll calcFlow()
90        {
91            ll res = 0;
92            while (bfs())
93            {
94                forn(i, N) fst[i] = head[i];
95                while (dfs(ST))
96                {
97                    res += pushed;
98                }
99            }
100            return res;
101        }
102
103        // HOW TO USE ::
104        // — set maxn and maxe (special for izban)
105        // — add adges using add(x, y, f), call setN(n)
106        // — run calcFlow
107    }

```

max-flow-min-cost

```

1 namespace flow
2 {
3     const int maxn = 2e5 + 10;
4     const int maxe = 2 * maxn;
5
6     int head[maxn], next[maxe], to[maxe], flow[maxe], ←
7         cost[maxe], ec = 1;
8     int ST, EN, N = maxn;
9
10    inline void setN(int n)
11    {
12        ST = n;
13        EN = n + 1;
14        N = n + 2;
15    }
16
17    inline void _add(int x, int y, int f, int c)
18    {
19        ++ec;
20        to[ec] = y;
21        next[ec] = head[x];
22        head[x] = ec;
23        flow[ec] = f;
24        cost[ec] = c;
25    }
26
27    inline int add(int x, int y, int f, int c)
28    {
29        _add(x, y, f, c);
30        _add(y, x, 0, -c);
31    }

```

```

30     return ec - 1;
31 }
32
33 void clear()
34 {
35     forn(i, N) head[i] = 0;
36     ec = 1;
37 }
38
39 ll d[maxn], p[maxn];
40 int last[maxn];
41 int used[maxn];
42
43 pair<ll, ll> _calc(int flag)
44 {
45     const ll INF = 1e12;
46     forn(i, N) p[i] = INF;
47     p[ST] = 0;
48     forn(_, N) forn(x, N) for (int e = head[x]; e; e ←
49         = next[e]) if (flow[e] > 0)
50     {
51         int y = to[e];
52         if (p[y] > p[x] + cost[e])
53         {
54             p[y] = p[x] + cost[e];
55         }
56     }
57     ll resFlow = 0, resCost = 0;
58     while (1)
59     {
60         forn(i, N) d[i] = INF, used[i] = 0;
61         d[ST] = 0;
62         forn(_, N)
63         {
64             int x = -1;
65             forn(i, N) if (!used[i] && (x == -1 || d[x] ←
66 > d[i])) x = i;
67             used[x] = 1;
68             if (d[x] == INF) break;
69             for (int e = head[x]; e; e = next[e]) if (←
70 flow[e] > 0)
71             {
72                 int y = to[e];
73                 ll len = cost[e] + p[x] - p[y];
74                 if (d[y] > d[x] + len)
75                 {
76                     d[y] = d[x] + len;
77                     last[y] = e;
78                 }
79             }
80             if (d[EN] == INF) break;
81
82             ll realCost = d[EN] + p[EN] - p[ST];
83             if (flag && realCost > 0) break;
84
85             int pushed = inf;
86             int x = EN;
87             while (x != ST)
88             {
89                 int e = last[x];
90                 pushed = min(pushed, flow[e]);
91                 x = to[e ^ 1];
92             }
93
94             resCost += realCost * pushed;
95             resFlow += pushed;
96
97             x = EN;
98             while (x != ST)
99             {
100                 int e = last[x];
101                 flow[e] -= pushed;
102                 flow[e ^ 1] += pushed;
103                 x = to[e ^ 1];
104             }
105
106             forn(i, N) p[i] += d[i];
107         }
108         return mp(resFlow, resCost);
109     }
110
111 pair<ll, ll> maxFlow()
112 {
113     return _calc(0);
114 }
115
116 pair<ll, ll> minCost()
117 {
118     return _calc(1);
119 }

```

```

120
121 // HOW TO USE::
122 // — add adges using add(x, y, f, c), call setN(n←
123 // — run maxFlow/minCost, returns pair(flow, cost←
124 }

```

poly

```

1 struct poly
2 {
3     vi v;
4     poly() {}
5     poly(vi vv)
6     {
7         v = vv;
8     }
9     int size()
10    {
11        return (int)v.size();
12    }
13    poly cut(int maxLen)
14    {
15        if (maxLen < sz(v)) v.resize(maxLen);
16        return *this;
17    }
18    poly norm()
19    {
20        while (sz(v) > 1 && v.back() == 0) v.pop_back();
21        return *this;
22    }
23    inline int& operator [] (int i)
24    {
25        return v[i];
26    }
27    void out(string name="")
28    {
29        stringstream ss;
30        if (sz(name)) ss << name << "=";
31        int fst = 1;
32        forn(i, sz(v)) if (v[i])
33        {
34            int x = v[i];
35            int sgn = 1;
36            if (x > mod / 2) x = mod - x, sgn = -1;
37            if (sgn == -1) ss << "-";
38            else if (!fst) ss << "+";
39            fst = 0;
40            if (!i || x != 1)
41            {
42                ss << x;
43                if (i > 0) ss << "*x";
44                if (i > 1) ss << "^" << i;
45            }
46            else
47            {
48                ss << "x";
49                if (i > 1) ss << "^" << i;
50            }
51        }
52        if (fst) ss << "0";
53        string s;
54        ss >> s;
55        eprintf("%s\n", s.data());
56    }
57 };
58
59 poly operator + (poly A, poly B)
60 {
61     poly C;
62     C.v = vi(max(sz(A), sz(B)));
63     forn(i, sz(C))
64     {
65         if (i < sz(A)) C[i] = (C[i] + A[i]) % mod;
66         if (i < sz(B)) C[i] = (C[i] + B[i]) % mod;
67     }
68     return C.norm();
69 }
70
71 poly operator - (poly A, poly B)
72 {
73     poly C;
74     C.v = vi(max(sz(A), sz(B)));
75     forn(i, sz(C))
76     {
77         if (i < sz(A)) C[i] = (C[i] + A[i]) % mod;

```

```

78     if (i < sz(B)) C[i] = (C[i] + mod - B[i]) % mod;
79 }
80 return C.norm();
81 }
82
83 poly operator * (poly A, poly B)
84 {
85     poly C;
86     C.v = vi(sz(A) + sz(B) - 1);
87
88     forn(i, sz(A)) fft::A[i] = A[i];
89     forn(i, sz(B)) fft::B[i] = B[i];
90     fft::multMod(sz(A), sz(B), mod);
91     forn(i, sz(C)) C[i] = fft::C[i];
92     return C.norm();
93 }
94
95 poly inv(poly A, int n) // returns A^{-1} mod x^n
96 {
97     assert(sz(A) && A[0] != 0);
98     A.cut(n);
99
100     auto cutPoly = [](poly &from, int l, int r)
101     {
102         poly R;
103         R.v.resize(r - l);
104         for (int i = l; i < r; ++i)
105             if (i < sz(from)) R[i - l] = from[i];
106         return R;
107     };
108
109     function<int(int, int)> rev = [&rev](int x, int m) ←
110     → int
111     {
112         if (x == 1) return 1;
113         return (1 - rev(m % x, x) * (ll)m) / x + m;
114     };
115
116     poly R({rev(A[0], mod)});
117     for (int k = 1; k < n; k <= 1)
118     {
119         poly A0 = cutPoly(A, 0, k);
120         poly A1 = cutPoly(A, k, 2 * k);
121         poly H = A0 * R;
122         H = cutPoly(H, k, 2 * k);
123         poly R1 = (((A1 * R).cut(k) + H) * (poly({0}) - ←
124         R)).cut(k);
125         R.v.resize(2 * k);
126         forn(i, k) R[i + k] = R1[i];
127     }
128     return R.cut(n).norm();
129 }
130
131 pair<poly, poly> divide(poly A, poly B)
132 {
133     if (sz(A) < sz(B)) return {poly({0}), A};
134
135     auto rev = [](poly f)
136     {
137         reverse(all(f.v));
138         return f;
139     };
140
141     poly q = rev((inv(rev(B), sz(A) - sz(B) + 1) * rev ←
142     (A)).cut(sz(A) - sz(B) + 1));
143     poly r = A - B * q;
144     return {q, r};
145 }
14
14 const int UD = 0;
15 const int WIN = 1;
16 const int LOSE = 2;
17
18 int res[N];
19 int moves[N];
20 int deg[N];
21 int q[N], st, en;
22
23 void calc(int n)
24 {
25     forn(i, n) deg[i] = sz(v[i]);
26     st = en = 0;
27     forn(i, n) if (!deg[i])
28     {
29         q[en++] = i;
30         res[i] = LOSE;
31     }
32     while (st < en)
33     {
34         int x = q[st++];
35         for (int y : vrev[x])
36         {
37             if (res[y] == UD && (res[x] == LOSE || (←
38             deg[y] == 0 && res[x] == WIN)))
39             {
40                 res[y] = 3 - res[x];
41                 moves[y] = moves[x] + 1;
42                 q[en++] = y;
43             }
44         }
45     }
46 }

```

retro

```

1 namespace retro
2 {
3     const int N = 4e5 + 10;
4
5     vi v[N];
6     vi vrev[N];
7
8     void add(int x, int y)
9     {
10         v[x].pb(y);
11         vrev[y].pb(x);
12     }
13 }

```