

## Содержание

|    |                                    |
|----|------------------------------------|
| 1  | musthave/vimrc.txt                 |
| 2  | musthave/template.cpp              |
| 3  | musthave/crt.cpp                   |
| 4  | musthave/fastIO.cpp                |
| 5  | musthave/fft.cpp                   |
| 6  | musthave/fftint.cpp                |
| 7  | musthave/blackbox.cpp              |
| 8  | musthave/halfplaneIntersection.cpp |
| 9  | musthave/commonTangents.cpp        |
| 10 | musthave/minDisc.cpp               |
| 11 | musthave/polygonArcCut.cpp         |
| 12 | musthave/hashTable.cpp             |
| 13 | musthave/hungary.cpp               |
| 14 | musthave/modReverseOneLine.cpp     |
| 15 | musthave/optimizations.cpp         |
| 16 | musthave/plane3DInt.cpp            |
| 17 | musthave/simplex.cpp               |
| 18 | musthave/std-rb-tree.cpp           |
| 19 | musthave/sufAutomaton.cpp          |
| 20 | musthave/generalMatching.cpp       |
| 21 | useful/dinica.cpp                  |
| 22 | useful/max-flow-min-cost.cpp       |
| 23 | useful/poly.cpp                    |
| 24 | useful/primes.cpp                  |
| 25 | retro                              |

## musthave/vimrc.txt

```

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 -↵
    O2 <CR>
3  map <F8> :wall! <CR> :!ulimit -s 500000 && ./%:r <CR>↵
    >
4
5  inoremap {<CR> {<CR>}<ESC>O
2  6  map <c-a> ggVG
7
8  set nu
9  set rnu
10 syntax on
3  11
12 map <c-t> :tabnew <CR>
13 map <c-l> :tabn <CR>
4  14 map <c-h> :tabp <CR>
15
16 set cin
4  17 set sw=4
18 set so=99
19 set bs=2
4  20 set et
21 set sts=4

```

## musthave/template.cpp

```

5  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
6  7  #define forn(i, n) for(int i = 0 ; (i) < (n) ; ++i)
8  #define eprintf(...) fprintf(stderr, __VA_ARGS__) ,↵
    fflush(stderr)
6  9  #define sz(a) ((int)(a).size())
10 #define all(a) (a).begin(), a.end()
7  11 #define pw(x) (1LL<<(x))
12
13 using namespace std;
7  14
15 typedef long long ll;
16 typedef double dbl;
8  17 typedef vector<int> vi;
18 typedef pair<int, int> pi;
19
8  20 const int INF = 1.01e9;
21 const dbl eps = 1e-9;
9  22
23 /* — main part — */
24
10 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 }

```

## musthave/crt.cpp

```

1 int CRT(int a1, int m1, int a2, int m2)
2 {
3     return (a1 - a2 % m1 + m1) * (11)rev(m2, m1) % m1 ←
4     * m2 + a2;

```

## musthave/fastIO.cpp

```

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 = readChar();
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 = readChar();
42     while ('0' <= c && c <= '9') x = x * 10 + c - '0', c = readChar();
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

```

## musthave/fft.cpp

```

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(a.x + b.x, a.y + b.y); }
15     inline num operator - (num a, num b) { return num(a.x - b.x, a.y - b.y); }
16     inline num operator * (num a, num b) { return num(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x); }
17     inline num conj(num a) { return num(a.x, -a.y); }
18
19     const dbl PI = acos(-1);
20
21     num root[maxN];
22     int rev[maxN];
23     bool rootsPrepared = false;
24
25     void prepRoots()
26     {
27         if (rootsPrepared) return;
28         rootsPrepared = true;
29         root[1] = num(1, 0);
30         for (int k = 1; k < maxBase; ++k)
31         {
32             num x(2 * PI / pw(k + 1));
33             for (int i = pw(k - 1); i < pw(k); ++i)
34             {
35                 root[2 * i] = root[i];
36                 root[2 * i + 1] = root[i] * x;
37             }
38         }
39     }
40
41     int base, N;
42
43     int lastRevN = -1;
44     void prepRev()
45     {
46         if (lastRevN == N) return;
47         lastRevN = N;
48         for (int i = 1; i < N; ++i) rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (base - 1));
49     }
50
51     void fft(num *a, num *f)
52     {
53         for (int i = 1; i < N; i += 2 * k) for (int j = 0; j < N; j += 2 * k)
54         {
55             num z = f[i + j + k] * root[j + k];
56             f[i + j + k] = f[i + j] - z;
57             f[i + j] = f[i + j] + z;
58         }
59     }
60
61     num a[maxN], b[maxN], f[maxN], g[maxN];
62     ll A[maxN], B[maxN], C[maxN];
63
64     void _multMod(int mod)
65     {
66         for (int i = 1; i < N; ++i)
67         {
68             int x = A[i] % mod;
69             a[i] = num(x & (pw(15) - 1), x >> 15);
70         }
71         for (int i = 1; i < N; ++i)
72         {
73             int x = B[i] % mod;
74             b[i] = num(x & (pw(15) - 1), x >> 15);
75         }
76         fft(a, f);
77         fft(b, g);
78
79         for (int i = 1; i < N; ++i)
80         {
81             int j = (N - i) & (N - 1);
82             num a1 = (f[i] + conj(f[j])) * num(0.5, 0);
83             num a2 = (f[i] - conj(f[j])) * num(0, -0.5);
84             num b1 = (g[i] + conj(g[j])) * num(0.5 / N, 0);
85             num b2 = (g[i] - conj(g[j])) * num(0, -0.5 / N);
86             a[j] = a1 * b1 + a2 * b2 * num(0, 1);
87             b[j] = a1 * b2 + a2 * b1;
88         }
89     }
90
91     fft(a, f);
92     fft(b, g);
93
94     for (int i = 1; i < N; ++i)

```

```

95     {
96         ll aa = f[i].x + 0.5;
97         ll bb = g[i].x + 0.5;
98         ll cc = f[i].y + 0.5;
99         C[i] = (aa + bb % mod * pw(15) + cc % mod * pw(
100             30)) % mod;
101     }
102
103 void prepAB(int n1, int n2)
104 {
105     base = 1;
106     N = 2;
107     while (N < n1 + n2) base++, N <= 1;
108
109     for (int i = n1; i < N; ++i) A[i] = 0;
110     for (int i = n2; i < N; ++i) B[i] = 0;
111
112     prepRoots();
113     prepRev();
114 }
115
116 void mult(int n1, int n2)
117 {
118     prepAB(n1, n2);
119     forn(i, N) a[i] = num(A[i], B[i]);
120     fft(a, f);
121     forn(i, N)
122     {
123         int j = (N - i) & (N - 1);
124         a[i] = (f[j] * f[j] - conj(f[i] * f[i])) * num(
125             0, -0.25 / N);
126     }
127     fft(a, f);
128     forn(i, N) C[i] = (ll)round(f[i].x);
129 }
130
131 void multMod(int n1, int n2, int mod)
132 {
133     prepAB(n1, n2);
134     _multMod(mod);
135 }
136
137 int D[maxN];
138
139 void multLL(int n1, int n2)
140 {
141     prepAB(n1, n2);
142
143     int mod1 = 1.5e9;
144     int mod2 = mod1 + 1;
145
146     _multMod(mod1);
147
148     forn(i, N) D[i] = C[i];
149
150     _multMod(mod2);
151
152     forn(i, N)
153     {
154         C[i] = D[i] + (C[i] - D[i] + (ll)mod2) * (ll)←
155         mod1 % mod2 * mod1;
156     }
157     // HOW TO USE ::
158     // — set correct maxBase
159     // — use mult(n1, n2), multMod(n1, n2, mod) and ←
160     multLL(n1, n2)
161     // — input : A[], B[]
162     // — output : C[]

```

## musthave/fftint.cpp

```

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 * (ll)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] * (ll)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
39 void _mult(int eq)
40 {
41     fft(A, F);
42     if (eq) forn(i, N) G[i] = F[i];
43     else fft(B, G);
44     int invN = inv(N);
45     forn(i, N) A[i] = F[i] * (ll)G[i] % mod * invN %←
46         mod;
47     reverse(A + 1, A + N);
48     fft(A, C);
49 }
50
51 void mult(int n1, int n2, int eq = 0)
52 {
53     for (int i = n1; i < N; ++i) A[i] = 0;
54     for (int i = n2; i < N; ++i) B[i] = 0;
55
56     _mult(eq);
57
58     //forn(i, n1 + n2) C[i] = 0;
59     //forn(i, n1) forn(j, n2) C[i + j] = (C[i + j] +←
60     A[i] * (ll)B[j]) % mod;

```

## musthave/blackbox.cpp

```

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] * (ll)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

```

## musthave/halfplaneIntersection.cpp

```

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
15 double planeInt(vector<Line> l) {
16     int n = l.size();
17     sort(all(l), [(Line a, Line b) {
18         int r = cmpV(a.v, b.v);
19         if (r != 0) return r < 0;
20         return a.0 % a.v.rotate() < b.0 % a.v.rotate() ←
21     });
22
23     int cur = 0;
24     for (int i = 0; i < n; ) {
25         int j = i;
26         for (; i < n && cmpV(l[j].v, l[i].v) == 0 && ←
27             cmpV(l[i].v, l[j].v) == 0; i++);
28         l[cur++] = l[i - 1];
29     }
30     n = cur;
31
32     for (int i = 0; i < n; i++)
33         l[i].id = i;
34
35     int flagUp = 0;
36     int flagDown = 0;
37     for (int i = 0; i < n; i++) {
38         int part = getPart(l[i].v);
39         if (part == 1) flagUp = 1;
40         if (part == 0) flagDown = 1;
41     }
42     if (!flagUp || !flagDown) return -1;
43
44     for (int i = 0; i < n; i++) {
45         pt v = l[i].v;
46         pt u = l[(i + 1) % n].v;
47         if (equal(0, v * u) && less(v % u, 0)) {
48             pt dir = l[i].v.rotate();
49             if (lessE(l[(i + 1) % n].0 % dir, l[i].0 % dir ←
50                 )) return 0;
51             return -1;
52         }
53         if (less(v * u, 0))
54             return -1;
55     }
56
57     cur = 0;
58     vector<Line> st(n * 2);
59     for (int tt = 0; tt < 2; tt++) {
60         for (int i = 0; i < n; i++) {
61             for (; cur >= 2; cur--) {
62                 pt G = st[cur - 1] * l[i];
63                 if (!lessE(st[cur - 2].v * (G - st[cur - 2]. ←
64                     0), 0)) break;
65             }
66             st[cur++] = l[i];
67             if (cur >= 2 && lessE(st[cur - 2].v * st[cur ←
68                 1].v, 0)) return 0;
69         }
70     }
71     vector<int> use(n, -1);
72     int left = -1, right = -1;
73     for (int i = 0; i < n; i++) {
74         if (use[st[i].id] == -1) {
75             use[st[i].id] = i;
76         }
77         else {
78             left = use[st[i].id];
79             right = i;
80             break;
81         }
82     }
83     vector<Line> tmp;
84     for (int i = left; i < right; i++)
85         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 }

```

## musthave/commonTangents.cpp

```

1
2
3  vector<Line> commonTangents(pt A, dbl rA, pt B, dbl ←
4  rB) {
5      vector<Line> res;
6      pt C = B - A;
7      dbl z = C.len2();
8      for (int i = -1; i <= 1; i += 2) {
9          for (int j = -1; j <= 1; j += 2) {
10             dbl r = rB * j - rA * i;
11             dbl d = z - r * r;
12             if (ls(d, 0)) continue;
13             d = sqrt(max(0.01, d));
14             pt magic = pt(r, d) / z;
15             pt v(magic % C, magic * C);
16             dbl CC = (rA * i - v % A) / v.len2();
17             pt O = v * -CC;
18             res.pb(Line(0, 0 + v.rotate()));
19         }
20     }
21     return res;
22 }
23
24 // HOW TO USE ::
25 // --- *D*-----*F*
26 // --- *...* - - - *...*
27 // --- *...* - - - *...*
28 // --- *...A...* - - *...B...*
29 // --- *...* - - - *...*
30 // --- *...* - - - *...*
31 // --- *...* - - - *...*
32 // --- *C*-----*E*
33 // --- res = {CE, CF, DE, DF}

```

## musthave/minDisc.cpp

```

1
2  pair<pt, dbl> minDisc(vector<pt> p) {
3      int n = p.size();
4      pt O = pt(0, 0);
5      dbl R = 0;
6      random_shuffle(all(p));
7      for (int i = 0; i < n; i++) {
8          if (ls(R, (O - p[i]).len())) {
9              O = p[i];
10             R = 0;
11             for (int j = 0; j < i; j++) {
12                 if (ls(R, (O - p[j]).len())) {
13                     O = (p[i] + p[j]) / 2;
14                     R = (p[i] - p[j]).len() / 2;
15                     for (int k = 0; k < j; k++) {
16                         if (ls(R, (O - p[k]).len())) {
17                             Line l1((p[i] + p[j]) / 2, (p[i] + p[j] ←
18                                 )) / 2 + (p[i] - p[j]).rotate();
19                             Line l2((p[k] + p[j]) / 2, (p[k] + p[j] ←
20                                 )) / 2 + (p[k] - p[j]).rotate();
21                             O = l1 * l2;
22                             R = (p[i] - O).len();
23                         }
24                     }
25                 }
26             }
27         }
28     }
29     return {O, R};
30 }

```

## musthave/polygonArcCut.cpp

```

1 struct Meta {
2     int type; // 0 — seg, 1 — circle
3     pt O;
4     dbl R;
5 };
6
7 const Meta SEG = {0, pt(0, 0), 0};
8
9 vector<pair<pt, Meta>> cut(vector<pair<pt, Meta>> p, ←
10     Line l) {
11     vector<pair<pt, Meta>> res;
12     int n = p.size();
13     for (int i = 0; i < n; i++) {
14         pt A = p[i].F;
15         pt B = p[(i + 1) % n].F;
16         if (le(0, l.v * (A - l.O))) {
17             if (eq(0, l.v * (A - l.O)) && p[i].S.type == 1 ←
18                 && ls(0, l.v * (p[i].S.O - A)))
19                 res.pb({A, SEG});
20             else
21                 res.pb(p[i]);
22         }
23         if (p[i].S.type == 0) {
24             if (sign(l.v * (A - l.O)) * sign(l.v * (B - l. ←
25                 0)) == -1) {
26                 pt FF = Line(A, B) * l;
27                 res.pb(make_pair(FF, SEG));
28             }
29             else {
30                 pt E, F;
31                 if (intCL(p[i].S.O, p[i].S.R, l, E, F)) {
32                     if (onArc(p[i].S.O, A, E, B))
33                         res.pb({E, SEG});
34                     if (onArc(p[i].S.O, A, F, B))
35                         res.pb({F, p[i].S});
36                 }
37             }
38         }
39     }
40     return res;
41 }

```

## musthave/hashTable.cpp

```

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

```

## musthave/hungary.cpp

```

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
49     // HOW TO USE ::
50     // — set values to a[1..n][1..m] (n <= m)
51     // — run calc(n, m) to find MINIMUM
52     // — to restore permutation use ans[]
53     // — everything works on negative numbers
54     // !! i don't understand this code, it's ←
55     // copped from e-maxx (and rewritten by enot110 ←
56     // )
57 }

```

## musthave/modReverseOneLine.cpp

```

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 }

```

## musthave/optimizations.cpp

```

1 // from anta code http://codeforces.com/contest/755/←
2 // submission/23864531
3
4 #pragma GCC optimize ("O3")
5 #pragma GCC target ("sse4")
6 inline void fasterLLDivMod(unsigned long long x, ←
7     unsigned y, unsigned &out_d, unsigned &out_m) {
8     unsigned xh = (unsigned)(x >> 32), x1 = (unsigned)←
9     x, d, m;
10 #ifdef __GNUC__
11     asm(
12         "divl %4; \n\t"
13         : "=a" (d), "=d" (m)
14     );
15 #endif
16 }

```

```

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 // have no idea what sse flags are really cool; list
26 // 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")

```

## musthave/plane3DInt.cpp

```

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;

```

## musthave/simplex.cpp

```

1 namespace simplex {
2     const int MAX_N = -1; // number of variables
3     const int MAX_M = -1; // number of inequalities
4     dbl a[MAX_M][MAX_N];
5     dbl b[MAX_M];
6     dbl c[MAX_N];
7     dbl v;
8     ll n, m;
9     int left[MAX_M];
10    int up[MAX_N];
11    int pos[MAX_N];
12    dbl res[MAX_N];
13
14    void init(int nn, int mm) {
15        n = nn;
16        m = mm;
17        v = 0;
18        for (int i = 0; i < m; i++)
19            for (int j = 0; j < n; j++)
20                a[i][j] = 0;
21        for (int i = 0; i < m; i++)
22            b[i] = 0;
23        for (int i = 0; i < n; i++)
24            c[i] = 0;
25    }
26
27    void pivot(int x, int y) {
28        swap(left[x], up[y]);
29        dbl k = a[x][y];
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 (!eq(a[x][i], 0))
36                pos[cur++] = i;
37        }
38
39        for (int i = 0; i < m; i++) {
40            if (i == x || eq(a[i][y], 0)) continue;
41            dbl 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        dbl cof = c[y];

```

```

49    v += cof * b[x];
50    c[y] = 0;
51    for (int i = 0; i < cur; i++) {
52        c[pos[i]] -= cof * a[x][pos[i]];
53    }
54 }
55
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 (ls(b[i], 0) && (x == -1 || b[i] < b[x]))
66                 x = i;
67         if (x == -1) break;
68         int y = -1;
69         for (int j = 0; j < n; j++)
70             if (ls(a[x][j], 0)) {
71                 y = j;
72                 break;
73             }
74         if (y == -1) {
75             assert(false); // no solution
76         }
77         pivot(x, y);
78     }
79     while (1) {
80         int y = -1;
81         for (int i = 0; i < n; i++)
82             if (ls(0, c[i]) && (y == -1 || (c[i] > c[y])))
83                 y = i;
84         if (y == -1) break;
85         int x = -1;
86         for (int i = 0; i < m; i++) {
87             if (ls(0, a[i][y])) {
88                 if (x == -1 || (b[i] / a[i][y] < b[x] / a[x][y]))
89                     x = i;
90             }
91         }
92         if (y == -1) {
93             assert(false); // infinite solution
94         }
95         pivot(x, y);
96     }
97     memset(res, 0, sizeof(res));
98     for (int i = 0; i < m; i++) {
99         if (left[i] < n) {
100             res[left[i]] = b[i];
101         }
102     }
103     // HOW TO USE ::
104     // — call init(n, m)
105     // — call solve()
106     // — variables in "up" equals to zero
107     // — variables in "left" equals to b
108     // — max: c * x
109     // — b[i] >= a[i] * x
110     // — answer in "v"
111     // — certificate in "res"
112 }

```

## musthave/std-rb-tree.cpp

```

1 #include "ext/pb_ds/assoc_container.hpp"
2 using namespace __gnu_pbds;
3
4 template <typename T> using ordered_set = tree<T, ←
    null_type, less<T>, rb_tree_tag, ←
    tree_order_statistics_node_update>;
5 template <typename K, typename V> using ordered_map ←
    = tree<K, V, less<K>, rb_tree_tag, ←
    tree_order_statistics_node_update>;
6

```

```

7 // HOW TO USE ::
8 // — order_of_key(10) returns the number of ←
9 // — *find_by_order(10) returns 10-th smallest ←
  // — element in set/map (0-based)

```

## musthave/sufAutomaton.cpp

```

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]));
36         len[clone] = len[p] + 1;
37         pos[clone] = pos[q];
38         link[clone] = link[q];
39         link[q] = link[cur] = clone;
40         for (; p != -1 && nxt[p][c] == q; p = link[p]) ←
41             nxt[p][c] = clone;
42     }
43
44     int n;
45     string s;
46     int l[MAXN], r[MAXN];
47     int e[MAXN][SIGMA];
48
49     void getSufTree(string _s) {
50         memset(e, -1, sizeof(e));
51         s = _s;
52         n = s.length();
53         reverse(s.begin(), s.end());
54         init();
55         for (int i = 0; i < n; i++) add(s[i] - 'a');
56         reverse(s.begin(), s.end());
57         for (int i = 1; i < sz; i++) {
58             int j = link[i];
59             l[i] = n - pos[i] + len[j];
60             r[i] = n - pos[i] + len[i];
61             e[j][s[l[i]] - 'a'] = i;
62         }
63     }
64 }

```

## musthave/generalMatching.cpp

```

1 //COPYPASTED FROM E-MAXX
2 namespace GeneralMatching {
3     const int MAXN = 256;
4     int n;
5     vector<int> g[MAXN];
6     int match[MAXN], p[MAXN], base[MAXN], q[MAXN];

```

```

7     bool used[MAXN], blossom[MAXN];
8
9     int lca(int a, int b) {
10         bool used[MAXN] = { 0 };
11         for (;) {
12             a = base[a];
13             used[a] = true;
14             if (match[a] == -1) break;
15             a = p[match[a]];
16         }
17         for (;) {
18             b = base[b];
19             if (used[b]) return b;
20             b = p[match[b]];
21         }
22     }
23
24     void mark_path(int v, int b, int children) {
25         while (base[v] != b) {
26             blossom[base[v]] = blossom[base[match[v]]] = ←
27                 true;
28             p[v] = children;
29             children = match[v];
30             v = p[match[v]];
31         }
32     }
33
34     int find_path(int root) {
35         memset(used, 0, sizeof(used));
36         memset(p, -1, sizeof(p));
37         for (int i = 0; i < n; ++i)
38             base[i] = i;
39
40         used[root] = true;
41         int qh = 0, qt = 0;
42         q[qt++] = root;
43         while (qh < qt) {
44             int v = q[qh++];
45             for (size_t i = 0; i < g[v].size(); ++i) {
46                 int to = g[v][i];
47                 if (base[v] == base[to] || match[v] == to) ←
48                     continue;
49                 if (to == root || (match[to] != -1 && p[←
50                     match[to]] != -1)) {
51                     int curbase = lca(v, to);
52                     memset(blossom, 0, sizeof(blossom));
53                     mark_path(v, curbase, to);
54                     mark_path(to, curbase, v);
55                     for (int i = 0; i < n; ++i)
56                         if (blossom[base[i]]) {
57                             base[i] = curbase;
58                             if (!used[i]) {
59                                 used[i] = true;
60                                 q[qt++] = i;
61                             }
62                         }
63                 }
64                 else if (p[to] == -1) {
65                     p[to] = v;
66                     if (match[to] == -1)
67                         return to;
68                     to = match[to];
69                     used[to] = true;
70                     q[qt++] = to;
71                 }
72             }
73         }
74         return -1;
75     }
76
77     vector<pair<int, int>> solve(int _n, vector<pair<←
78         int, int>> edges) {
79         n = _n;
80         for (int i = 0; i < n; i++) g[i].clear();
81         for (auto o : edges) {
82             g[o.first].push_back(o.second);
83             g[o.second].push_back(o.first);
84         }
85         memset(match, -1, sizeof(match));
86         for (int i = 0; i < n; ++i) {
87             if (match[i] == -1) {
88                 int v = find_path(i);
89                 while (v != -1) {
90                     int pv = p[v], ppv = match[pv];
91                     match[v] = pv, match[pv] = v;
92                     v = ppv;
93                 }
94             }
95         }
96         vector<pair<int, int>> ans;
97         for (int i = 0; i < n; i++) {
98             if (match[i] > i) {
99                 ans.push_back(make_pair(i, match[i]));
100             }
101         }
102     }
103 }

```

```

96     }
97   }
98   return ans;
99 }
100 }
```

## useful/dinica.cpp

```

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    }
62    return 0;
63  }
64
65  int pushed;
66  int fst[maxn];
67
68  int dfs(int x, int flow = 1e9)
69  {
70    if (x == EN)
71    {
72      pushed = flow;
73      return 1;
74    }
75    for (; fst[x]; fst[x] = next[fst[x]])
76    {
77      int e = fst[x];
78      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 edges using add(x, y, f), call setN(n)
106  // — run calcFlow
107 }
```

## useful/max-flow-min-cost.cpp

```

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    return ec - 1;
32  }
33
34  void clear()
35  {
36    forn(i, N) head[i] = 0;
37    ec = 1;
38  }
39
40  ll d[maxn], p[maxn];
41  int last[maxn];
42  int used[maxn];
43
44  pair<ll, ll> _calc(int flag)
45  {
46    const ll INF = 1e12;
47    forn(i, N) p[i] = INF;
48    p[ST] = 0;
49    forn(_, N) forn(x, N) for (int e = head[x]; e; e ←
50    = next[e]) if (flow[e] > 0)
51    {
52      int y = to[e];
53      if (p[y] > p[x] + cost[e])
```



```

52     {
53         p[y] = p[x] + cost[e];
54     }
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] > d[i])) x = i;
66         used[x] = 1;
67         if (d[x] == INF) break;
68         for (int e = head[x]; e; e = next[e]) if (flow[e] < 0)
69         {
70             int y = to[e];
71             ll len = cost[e] + p[x] - p[y];
72             if (d[y] > d[x] + len)
73             {
74                 d[y] = d[x] + len;
75                 last[y] = e;
76             }
77         }
78     }
79     if (d[EN] == INF) break;
80
81     ll realCost = d[EN] + p[EN] - p[ST];
82     if (flag && realCost > 0) break;
83
84     int pushed = inf;
85     int x = EN;
86     while (x != ST)
87     {
88         int e = last[x];
89         pushed = min(pushed, flow[e]);
90         x = to[e ^ 1];
91     }
92     resCost += realCost * pushed;
93     resFlow += pushed;
94
95     x = EN;
96     while (x != ST)
97     {
98         int e = last[x];
99         flow[e] -= pushed;
100        flow[e ^ 1] += pushed;
101        x = to[e ^ 1];
102    }
103    forn(i, N) p[i] += d[i];
104    return mp(resFlow, resCost);
105 }
106
107 pair<ll, ll> maxFlow()
108 {
109     return _calc(0);
110 }
111
112 pair<ll, ll> minCost()
113 {
114     return _calc(1);
115 }
116
117 // HOW TO USE::
118 // — add adges using add(x, y, f, c), call setN(n)
119 // — run maxFlow/minCost, returns pair(flow, cost)
120
121 }

```

## useful/poly.cpp

```

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 - 1);
104     for (int i = 1; i < r; ++i)
105     {
106         if (i < sz(from)) R[i - 1] = from[i];
107     }
108     return R;
109 };
110
111 function<int(int, int)> rev = [&rev](int x, int m)↔
112     →int
113 {
114     if (x == 1) return 1;
115     return (1 - rev(m % x, x) * (11)m) / x + m;
116 };
117
118 poly R({rev(A[0], mod)});
119 for (int k = 1; k < n; k <= 1)
120 {
121     poly A0 = cutPoly(A, 0, k);
122     poly A1 = cutPoly(A, k, 2 * k);
123     poly H = A0 * R;
124     H = cutPoly(H, k, 2 * k);
125     poly R1 = (((A1 * R).cut(k) + H) * (poly({0}) - ↔
126     R)).cut(k);
127     R.v.resize(2 * k);
128     forn(i, k) R[i + k] = R1[i];
129 }
130 return R.cut(n).norm();
131 }
132
133 pair<poly, poly> divide(poly A, poly B)
134 {
135     if (sz(A) < sz(B)) return {poly({0}), A};
136
137     auto rev = [(poly f)
138     {
139         reverse(all(f.v));
140         return f;
141     }];
142
143     poly q = rev((inv(rev(B), sz(A) - sz(B) + 1) * rev↔
144     (A)).cut(sz(A) - sz(B) + 1));
145     poly r = A - B * q;
146
147     return {q, r};
148 }

```

```

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
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     }
47 }

```

## useful/primes.cpp

```

1 namespace math
2 {
3     const int maxP = 1e6;
4     int pp[maxP];
5     int p[maxP / 10 + 1000], pc = 0;
6
7     void gen_primes()
8     {
9         pp[0] = pp[1] = -1;
10         for (int i = 2; i < maxP; ++i) pp[i] = i;
11         for (int i = 2; i < maxP; ++i)
12         {
13             if (pp[i] == i) p[pc++] = i;
14             for (int j = 0; j < pc && p[j] <= pp[i] && i * ↔
15             p[j] < maxP; ++j) pp[i * p[j]] = p[j];
16         }
17     }
18     bool is_prime(int x)
19     {
20         if (x < maxP) return pp[x] == x;
21         for (int i = 0; p[i] * p[i] <= x; ++i) if (x % p↔
22         [i] == 0) return false;
23         return true;
24     }
25     // HOW TO USE ::
26     // pp[x] <= smallest prime divisor {x} (or -1 for↔
27     // {x < 2})
28     // p[0] .. pc - 1] <= list of primes < maxP
29 }

```

## retro