

## Содержание

1	final/template/vimrc.txt
2	final/template/template.cpp
3	final/template/fastIO.cpp
4	final/template/hashTable.cpp
5	final/template/optimizations.cpp
6	final/template/std-rb-tree.cpp
7	final/numeric/fft.cpp
8	final/numeric/fftint.cpp
9	final/numeric/blackbox.cpp
10	final/numeric/crt.cpp
11	final/numeric/modReverse.cpp
12	final/numeric/pollard.cpp
13	final/numeric/poly.cpp
14	final/numeric/simplex.cpp
15	final/geom/commonTangents.cpp
16	final/geom/halfplaneIntersection.cpp
17	final/geom/minDisc.cpp
18	final/geom/plane3DInt.cpp
19	final/geom/polygonArcCut.cpp
20	final/strings/eertree.cpp
21	final/strings/sufAutomaton.cpp
22	final/graphs/centroid.cpp
23	final/graphs/dinica.cpp
24	final/graphs/dominatorTree.cpp
25	final/graphs/fenwick-min.cpp
26	final/graphs/generalMatching.cpp
27	final/graphs/heavyLight.cpp
28	final/graphs/hungary.cpp
29	final/graphs/max-flow-min-cost.cpp
30	final/graphs/retro.cpp
31	final/graphs/smith.cpp
32	final/graphs/twoChinese.cpp

## final/template/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
2  9  set rnu
10 syntax on
11
12 map <c-t> :tabnew <CR>
13 map <c-l> :tabn <CR>
3  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

```

## final/template/template.cpp

```

7  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()
8  #define pw(x) (1LL<<(x))
12
13 using namespace std;
8  14
15 typedef long long ll;
16 typedef double dbl;
8  17 typedef vector<int> vi;
18 typedef pair<int, int> pi;
19
9  20 const int INF = 1.01e9;
21 const dbl eps = 1e-9;
9  22
23 /* — main part — */
24
10 25
26
27
10 28
29
30
11 31 int main()
32 {
11 33 #define TASK ""
34 #ifdef home
35     assert(freopen(TASK".in", "r", stdin));
12 36     //assert(freopen(TASK".out", "w", stdout));
37 #endif
38
12 39
40
41
13 42
43 #ifdef home
13 44     eprintf("time = %d ms\n", (int)(clock() * 1000. / ↵
        CLOCKS_PER_SEC));
45 #endif
46     return 0;
14 47 }

```

## final/template/fastIO.cpp

```

1 #include <stdio>
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 // 10M int [0..1e9]
47 // cin 3.02
48 // scanf 1.2
49 // cin_sync_with_stdio(false) 0.71
50 // fastRead getchar 0.53
51 // fastRead fread 0.15
52

```

## final/template/hashTable.cpp

```

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     }
24 }

```

```

23         return f[i];
24     }
25 };
26
27 hashTable <13, int, int, 0> h;

```

## final/template/optimizations.cpp

```

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 fasterLLDdivMod(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 // have no idea what sse flags are really cool; list of some of them
26 // --- very good with bitsets
27 #pragma GCC optimize("O3")
28 #pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx")
29

```

## final/template/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 elements in set/map strictly less than 10
9 // --- *find_by_order(10) returns 10-th smallest element in set/map (0-based)

```

## final/numeric/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(←
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            for (i, N) rev[i] = (rev[i >> 1] >> 1) + ((i & ←
51                1) << (base - 1));
52        }
53
54        void fft(num *a, num *f)
55        {
56            for (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) for (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                for (i, N)
71                {
72                    int x = A[i] % mod;
73                    a[i] = num(x & (pw(15) - 1), x >> 15);
74                }
75                for (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                for (i, N)
83                {
84                    int j = (N - i) & (N - 1);

```

```

83        num a1 = (f[i] + conj(f[j])) * num(0.5, 0);
84        num a2 = (f[i] - conj(f[j])) * num(0, -0.5);
85        num b1 = (g[i] + conj(g[j])) * num(0.5 / N, 0) ←
86        ;
87        num b2 = (g[i] - conj(g[j])) * num(0, -0.5 / N) ←
88        );
89        a[j] = a1 * b1 + a2 * b2 * num(0, 1);
90        b[j] = a1 * b2 + a2 * b1;
91    }
92
93    fft(a, f);
94    fft(b, g);
95
96    for (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    void prepAB(int n1, int n2)
106    {
107        base = 1;
108        N = 2;
109        while (N < n1 + n2) base++, N <<= 1;
110
111        for (int i = n1; i < N; ++i) A[i] = 0;
112        for (int i = n2; i < N; ++i) B[i] = 0;
113
114        prepRoots();
115        prepRev();
116    }
117
118    void mult(int n1, int n2)
119    {
120        prepAB(n1, n2);
121        for (i, N) a[i] = num(A[i], B[i]);
122        fft(a, f);
123        for (i, N)
124        {
125            int j = (N - i) & (N - 1);
126            a[i] = (f[j] * f[j] - conj(f[i] * f[i])) * num ←
127                (0, -0.25 / N);
128        }
129        fft(a, f);
130        for (i, N) C[i] = (ll)round(f[i].x);
131    }
132
133    void multMod(int n1, int n2, int mod)
134    {
135        prepAB(n1, n2);
136        _multMod(mod);
137    }
138
139    int D[maxN];
140
141    void multLL(int n1, int n2)
142    {
143        prepAB(n1, n2);
144
145        int mod1 = 1.5e9;
146        int mod2 = mod1 + 1;
147
148        _multMod(mod1);
149
150        for (i, N) D[i] = C[i];
151
152        _multMod(mod2);
153
154        for (i, N)
155        {
156            C[i] = D[i] + (C[i] - D[i] + (ll)mod2) * (ll) ←
157                mod1 % mod2 * mod1;
158        }
159    }
160
161    // HOW TO USE ::
162    // — set correct maxBase
163    // — use mult(n1, n2), multMod(n1, n2, mod) and ←
164    // multLL(n1, n2)
165    // — input : A[], B[]
166    // — output : C[]

```

## final/numeric/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 * (11)ROOT % mod;
21         }
22         for (int i = NN - 1; i > 0; --i) root[i] = root←
23             [2 * i];
24     }
25
26     void fft(int *a, int *f)
27     {
28         forn(i, N) f[i] = a[rev[i]];
29         for (int k = 1; k < N; k <<= 1) for (int i = 0; ←
30             i < N; i += 2 * k) forn(j, k)
31         {
32             int z = f[i + j + k] * (11)root[j + k] % mod;
33             f[i + j + k] = (f[i + j] - z + mod) % mod;
34             f[i + j] = (f[i + j] + z) % mod;
35         }
36     }
37
38     int A[N], B[N], C[N];
39     int F[N], G[N];
40
41     void _mult(int eq)
42     {
43         fft(A, F);
44         if (eq) forn(i, N) G[i] = F[i];
45         else fft(B, G);
46         int invN = inv(N);
47         forn(i, N) A[i] = F[i] * (11)G[i] % mod * invN %←
48             mod;
49         reverse(A + 1, A + N);
50         fft(A, C);
51     }
52
53     void mult(int n1, int n2, int eq = 0)
54     {
55         for (int i = n1; i < N; ++i) A[i] = 0;
56         for (int i = n2; i < N; ++i) B[i] = 0;
57
58         _mult(eq);
59
60         //forn(i, n1 + n2) C[i] = 0;
61         //forn(i, n1) forn(j, n2) C[i + j] = (C[i + j] +←
62             A[i] * (11)B[j]) % mod;
63     }
64 }

```

## final/numeric/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] * (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        }
22    }
23 }

```

```

20         z <<= 1;
21     }
22     return C[k];
23 }
24 // A — constant array
25 // magic(k, x) :: B[k] = x, returns C[k]
26 // !! WARNING !! better to set N twice the size ←
27 // needed
28 }

```

## final/numeric/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;
5 }

```

## final/numeric/modReverse.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 }

```

## final/numeric/pollard.cpp

```

1 namespace pollard
2 {
3     using math::p;
4
5     vector<pair<ll, int>> getFactors(ll N)
6     {
7         vector<ll> primes;
8
9         const int MX = 1e5;
10        const ll MX2 = MX * (11)MX;
11
12        assert(MX <= math::maxP && math::pc > 0);
13
14        function<void(ll)> go = [&go, &primes](ll n)
15        {
16            for (ll x : primes) while (n % x == 0) n /= x;
17            if (n == 1) return;
18            if (n > MX2)
19            {
20                auto F = [&](ll x) {
21                    ll k = ((long double)x * x) / n;
22                    ll r = (x * x - k * n + 3) % n;
23                    return r < 0 ? r + n : r;
24                };
25                ll x = mt19937_64()() % n, y = x;
26                const int C = 3 * pow(n, 0.25);
27
28                ll val = 1;
29                forn(it, C)
30                {
31                    x = F(x), y = F(F(y));
32                    if (x == y) continue;
33                    ll delta = abs(x - y);
34                    ll k = ((long double)val * delta) / n;
35                    val = (val * delta - k * n) % n;
36                    if (val < 0) val += n;
37                    if (val == 0)
38                    {
39                        ll g = __gcd(delta, n);
40                        go(g), go(n / g);
41                        return;
42                    }
43                }
44                if ((it & 255) == 0)
45                {
46                    ll g = __gcd(val, n);
47                    if (g != 1)
48                    {
49                        go(g), go(n / g);
50                    }
51                }
52            }
53        }
54    }
55 }

```

```

48         go(g), go(n / g);
49         return;
50     }
51 }
52 }
53 }
54 primes.pb(n);
55 };
56
57 ll n = N;
58
59 for (int i = 0; i < math::pc && p[i] < MX; ++i) ←
60 if (n % p[i] == 0)
61 {
62     primes.pb(p[i]);
63     while (n % p[i] == 0) n /= p[i];
64 }
65 go(n);
66
67 sort(primes.begin(), primes.end());
68
69 vector<pair<ll, int>> res;
70 for (ll x : primes)
71 {
72     int cnt = 0;
73     while (N % x == 0)
74     {
75         cnt++;
76         N /= x;
77     }
78     res.push_back({x, cnt});
79 }
80 return res;
81 }
82 }

```

## final/numeric/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 - l);
104        for (int i = l; i < r; ++i)
105        {
106            if (i < sz(from)) R[i - l] = 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) * (ll)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     };

```

```

140 poly q = rev((inv(rev(B), sz(A) - sz(B) + 1) * rev←
141 (A)).cut(sz(A) - sz(B) + 1));
142 poly r = A - B * q;
143
144 return {q, r};
145 }

```

## final/numeric/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];
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    void solve() {
56        for (int i = 0; i < n; i++)
57            up[i] = i;
58        for (int i = 0; i < m; i++)
59            left[i] = i + n;
60
61        while (1) {
62            int x = -1;
63            for (int i = 0; i < m; i++)
64                if (ls(b[i], 0) && (x == -1 || b[i] < b[x])) ←
65                {
66                    x = i;
67                }
68            if (x == -1) break;
69            int y = -1;
70            for (int j = 0; j < n; j++)
71                if (ls(a[x][j], 0)) {
72                    y = j;
73                    break;
74                }
75            if (y == -1) {
76                assert(false); // no solution

```

```

77            }
78            pivot(x, y);
79        }
80        while (1) {
81            int y = -1;
82            for (int i = 0; i < n; i++)
83                if (ls(0, c[i]) && (y == -1 || (c[i] > c[y]) ←
84                )) {
85                    y = i;
86                }
87            if (y == -1) break;
88            int x = -1;
89            for (int i = 0; i < m; i++) {
90                if (ls(0, a[i][y])) {
91                    if (x == -1 || (b[i] / a[i][y] < b[x] / a[←
92                    x][y])) {
93                        x = i;
94                    }
95                }
96            }
97            if (y == -1) {
98                assert(false); // infinite solution
99            }
100            pivot(x, y);
101        }
102        memset(res, 0, sizeof(res));
103
104        for (int i = 0; i < m; i++) {
105            if (left[i] < n) {
106                res[left[i]] = b[i];
107            }
108        }
109    }
110    // HOW TO USE ::
111    // — call init(n, m)
112    // — call solve()
113    // — variables in "up" equals to zero
114    // — variables in "left" equals to b
115    // — max: c * x
116    // — b[i] >= a[i] * x
117    // — answer in "v"
118    // — sertificate in "res"
119 }

```

## final/geom/commonTangents.cpp

```

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

```

## final/geom/halfplaneIntersection.cpp

```

1  int getPart(pt v) {
2      return less(0, v.y) || (equal(0, v.y) && less(v.x, 0));
3  }
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.O % a.v.rotate() < b.O % a.v.rotate();
21     });
22
23     int cur = 0;
24     for (int i = 0; i < n; i++) {
25         int j = i;
26         for (; i < n && cmpV(l[j].v, l[i].v) == 0 && 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].O % dir, l[i].O % 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].v), 0))
63                     break;
64                 st[cur++] = l[i];
65                 if (cur >= 2 && lessE(st[cur - 2].v * st[cur - 1].v, 0)) return 0;
66             }
67         }
68         vector<int> use(n, -1);
69         int left = -1, right = -1;
70         for (int i = 0; i < cur; i++) {
71             if (use[st[i].id] == -1) {
72                 use[st[i].id] = i;
73             }
74             else {
75                 left = use[st[i].id];
76                 right = i;
77                 break;
78             }
79         }
80         vector<Line> tmp;
81         for (int i = left; i < right; i++)
82             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     }

```

## final/geom/minDisc.cpp

```

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

```

## final/geom/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;

```

## final/geom/polygonArcCut.cpp

```

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

```

## final/strings/eertree.cpp

```

1 namespace eertree {
2     const int INF = 1e9;
3     const int N = 5e6 + 10;
4     char _s[N];
5     char *s = _s + 1;
6     int to[N][2];
7     int suf[N], len[N];
8     int sz, last;
9
10     const int odd = 1, even = 2, blank = 3;
11
12     void go(int &u, int pos) {
13         while (u != blank && s[pos - len[u] - 1] != s[←
            pos]) {
14             u = suf[u];
15         }
16     }
17
18     int add(int pos) {
19         go(last, pos);
20         int u = suf[last];
21         go(u, pos);
22         int c = s[pos] - 'a';
23         int res = 0;
24         if (!to[last][c]) {
25             res = 1;
26             to[last][c] = sz;
27             len[sz] = len[last] + 2;
28             suf[sz] = to[u][c];
29             sz++;
30         }
31         last = to[last][c];
32         return res;
33     }
34
35     void init() {
36         to[blank][0] = to[blank][1] = even;
37         len[blank] = suf[blank] = INF;
38         len[even] = 0, suf[even] = odd;
39         len[odd] = -1, suf[odd] = blank;
40         last = even;
41         sz = 4;
42     }
43 }

```

## final/strings/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]) ←
            nxt[p][c] = cur;
24         if (p == -1) {
25             link[cur] = 0;
26             return;
27         }
28         int q = nxt[p][c];
29         if (len[p] + 1 == len[q]) {
30             link[cur] = q;
31             return;
32         }
33         int clone = sz++;
34         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]) ←
        nxt[p][c] = clone;
40 }
41
42 int n;
43 string s;
44 int l[MAXN], r[MAXN];
45 int e[MAXN][SIGMA];
46
47 void getSufTree(string _s) {
48     memset(e, -1, sizeof(e));
49     s = _s;
50     n = s.length();
51     reverse(s.begin(), s.end());
52     init();
53     for (int i = 0; i < n; i++) add(s[i] - 'a');
54     reverse(s.begin(), s.end());
55     for (int i = 1; i < sz; i++) {
56         int j = link[i];
57         l[i] = n - pos[i] + len[j];
58         r[i] = n - pos[i] + len[i];
59         e[j][s[l[i]] - 'a'] = i;
60     }
61 }
62 }

```

## final/graphs/centroid.cpp

```

1 // original author: burunduk1, rewritten by me (←
    enot110)
2 // !!! warning !!! this code is not tested well
3 const int N = 1e5, K = 17;
4
5 int pivot, level[N], parent[N];
6 vector<int> v[N];
7
8 int get_pivot(int x, int xx, int n) {
9     int size = 1;
10    for (int y : v[x])
11    {
12        if (y != xx && level[y] == -1) size += get_pivot←
            (y, x, n);
13    }
14    if (pivot == -1 && (size * 2 >= n || xx == -1)) ←
        pivot = x;
15    return size;
16 }
17
18 void build(int x, int xx, int dep, int size) {
19     assert(dep < K);
20     pivot = -1;
21     get_pivot(x, -1, size);
22     x = pivot;
23     level[x] = dep, parent[x] = xx;
24     for (int y : v[x]) if (level[y] == -1)
25     {
26         build(y, x, dep + 1, size / 2);
27     }
28 }

```

## final/graphs/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 ←
        = 1;
7     int ST, EN, N = maxn;
8
9     inline void setN(int n)
10    {
11        ST = n;
12        EN = n + 1;
13        N = n + 2;
14    }
15
16    inline void _add(int x, int y, int ff)
17    {
18        ++ec;
19        to[ec] = y;
20        next[ec] = head[x];
21        head[x] = ec;
22        f[ec] = ff;
23    }
24
25    inline int add(int x, int y, int ff)
26    {
27        _add(x, y, ff);
28        _add(y, x, 0);
29        return ec - 1;
30    }
31
32    void clear()
33    {
34        forn(i, N) head[i] = 0;
35        ec = 1;
36    }
37
38    int d[maxn];
39    int q[maxn], st = 0, en = 0;
40
41    int bfs()
42    {
43        forn(i, N) d[i] = 1e9;
44        st = 0, en = 0;
45        d[ST] = 0;
46        q[en++] = ST;
47        while (st < en)

```

```

48 {
49     int x = q[st++];
50     if (x == EN) return 1;
51     for (int e = head[x]; e; e = next[e])
52     {
53         int y = to[e];
54         if (d[y] == 1e9 && f[e])
55         {
56             d[y] = d[x] + 1;
57             q[en++] = y;
58         }
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         for (i = 1; i <= n; i++) f[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 getN(n)
106 // — run calcFlow
107 }

```

## final/graphs/dominatorTree.cpp

```

1 namespace domtree {
2     const int MAXN = 300100;
3     int n;
4     vector<int> e[MAXN];
5     vector<int> g[MAXN];
6     int par[MAXN];
7     int in[MAXN], rin[MAXN], tmr;
8     int dom[MAXN], sdom[MAXN], cmn[MAXN];
9     int p[MAXN];
10    int adom[MAXN];
11    vector<int> vct[MAXN];
12
13    void init(int _n) {
14        n = _n;
15        for (int i = 0; i < n; i++) {
16            e[i].clear();
17        }
18    }
19
20    void addEdge(int from, int to) {
21        e[from].push_back(to);
22    }
23 }

```

```

24 void dfs(int v) {
25     in[v] = tmr++;
26     rin[in[v]] = v;
27     for (int to : e[v]) {
28         if (in[to] == -1) {
29             dfs(to);
30             par[in[to]] = in[v];
31         }
32         g[in[to]].push_back(in[v]);
33     }
34 }
35
36 int get(int u, int x = 0) {
37     if (u == p[u]) return x ? -1 : u;
38     int v = get(p[u], x + 1);
39     if (v < 0) return u;
40     if (sdom[cmn[p[u]]] < sdom[cmn[u]]) cmn[u] = cmn ←
41         [p[u]];
42     p[u] = v;
43     return x ? v : cmn[u];
44 }
45
46 void uni(int u, int v) {
47     p[v] = u;
48 }
49
50 void calc() {
51     for (int i = 0; i < n; i++) {
52         in[i] = -1;
53         adom[i] = -1;
54         dom[i] = sdom[i] = p[i] = cmn[i] = i;
55         vct[i].clear();
56         g[i].clear();
57     }
58     tmr = 0;
59     dfs(0);
60     for (int i = tmr - 1; i >= 0; i--) {
61         for (int to : g[i]) sdom[i] = min(sdom[i], ←
62             sdom[get(to)]);
63         if (i > 0) vct[sdom[i]].push_back(i);
64         for (int w : vct[i]) {
65             int v = get(w);
66             if (sdom[v] == sdom[w]) dom[w] = sdom[w];
67             else dom[w] = v;
68         }
69         if (i > 0) uni(par[i], i);
70     }
71     for (int i = 1; i < tmr; i++) {
72         if (dom[i] != sdom[i]) dom[i] = dom[dom[i]];
73         adom[rin[i]] = rin[dom[i]];
74     }
75 }

```

## final/graphs/fenwick-min.cpp

```

1 const int inf = 1.01e9;
2 const int maxn = 1e5;
3
4 namespace fenwick
5 {
6     const int N = maxn + 1;
7
8     int a[N], l[N], r[N];
9
10    void modify(int q, int v) {
11        q++;
12        a[q] = min(a[q], v);
13        int x = q;
14        while (x < N) {
15            l[x] = min(l[x], v);
16            x = (x | (x - 1)) + 1;
17        }
18        x = q;
19        while (x > 0) {
20            r[x] = min(r[x], v);
21            x &= x - 1;
22        }
23    }
24
25    int find_min(int ll, int rr) {
26        ll++;
27        rr++;
28        int res = inf;
29        int x = ll;
30        while ((x | (x - 1)) + 1 <= rr) {
31            res = min(res, r[x]);

```

```

32     x = (x | (x - 1)) + 1;
33 }
34 res = min(res, a[x]);
35 x = rr;
36 while ((x & (x - 1)) >= 1) {
37     res = min(res, l[x]);
38     x &= x - 1;
39 }
40 return res;
41 }
42
43 // indexes 0 .. maxn-1
44 // (!) to init fill (a, l, r) with INF
45 // (!) modify supports only decreasing of the ←
46 // value
47 // find_min [l, r] (both inclusive)

```

## final/graphs/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                     else if (p[to] == -1) {
64                         p[to] = v;
65                         if (match[to] == -1)
66                             return to;
67                     }
68                 }
69             }
70         }
71         return -1;
72     }
73
74     vector<pair<int, int>> solve (int _n, vector<pair<←
75         int, int>> edges) {
76         n = _n;
77         for (int i = 0; i < n; ++i) g[i].clear();
78         for (auto o : edges) {
79             g[o.first].push_back(o.second);
80             g[o.second].push_back(o.first);
81         }
82         memset (match, -1, sizeof match);
83         for (int i=0; i<n; ++i) {
84             if (match[i] == -1) {
85                 int v = find_path (i);
86                 while (v != -1) {
87                     int pv = p[v], ppv = match[pv];
88                     match[v] = pv, match[pv] = v;
89                     v = ppv;
90                 }
91             }
92         }
93         vector<pair<int, int>> ans;
94         for (int i = 0; i < n; ++i) {
95             if (match[i] > i) {
96                 ans.push_back(make_pair(i, match[i]));
97             }
98         }
99         return ans;
100     }

```

```

65         to = match[to];
66         used[to] = true;
67         q[qt++] = to;
68     }
69 }
70 }
71 return -1;
72 }
73
74 vector<pair<int, int>> solve (int _n, vector<pair<←
75     int, int>> edges) {
76     n = _n;
77     for (int i = 0; i < n; ++i) g[i].clear();
78     for (auto o : edges) {
79         g[o.first].push_back(o.second);
80         g[o.second].push_back(o.first);
81     }
82     memset (match, -1, sizeof match);
83     for (int i=0; i<n; ++i) {
84         if (match[i] == -1) {
85             int v = find_path (i);
86             while (v != -1) {
87                 int pv = p[v], ppv = match[pv];
88                 match[v] = pv, match[pv] = v;
89                 v = ppv;
90             }
91         }
92     }
93     vector<pair<int, int>> ans;
94     for (int i = 0; i < n; ++i) {
95         if (match[i] > i) {
96             ans.push_back(make_pair(i, match[i]));
97         }
98     }
99     return ans;
100 }

```

## final/graphs/heavyLight.cpp

```

1 namespace hld {
2     const int N = 1 << 17;
3     int par[N], heavy[N], h[N];
4     int root[N], pos[N];
5     int n;
6     vector<vector<int>> e;
7     segtree tree;
8
9     int dfs (int v) {
10         int sz = 1, mx = 0;
11         for (int to : e[v]) {
12             if (to == par[v]) continue;
13             par[to] = v;
14             h[to] = h[v] + 1;
15             int cur = dfs(to);
16             if (cur > mx) heavy[v] = to, mx = cur;
17             sz += cur;
18         }
19         return sz;
20     }
21
22     template <typename T>
23     void path (int u, int v, T op) {
24         for (; root[u] != root[v]; v = par[root[v]]) {
25             if (h[root[u]] > h[root[v]]) swap(u, v);
26             op(pos[root[v]], pos[v] + 1);
27         }
28         if (h[u] > h[v]) swap(u, v);
29         op(pos[u], pos[v] + 1);
30     }
31
32     void init (vector<vector<int>> _e) {
33         e = _e;
34         n = e.size();
35         tree = segtree(n);
36         memset (heavy, -1, sizeof (heavy[0]) * n);
37         par[0] = -1;
38         h[0] = 0;
39         dfs(0);
40         for (int i = 0, cpos = 0; i < n; ++i) {
41             if (par[i] == -1 || heavy[par[i]] != i) {
42                 for (int j = i; j != -1; j = heavy[j]) {
43                     root[j] = i;
44                     pos[j] = cpos++;
45                 }
46             }
47         }

```

```

48 }
49
50 void add(int v, int x) {
51     tree.add(pos[v], x);
52 }
53
54 int get(int u, int v) {
55     int res = 0;
56     path(u, v, [&](int l, int r) {
57         res = max(res, tree.get(l, r));
58     });
59     return res;
60 }
61 }

```

## final/graphs/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 rewrited by enot110←
56 }

```

## final/graphs/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])
54            {
55                p[y] = p[x] + cost[e];
56            }
57        }
58
59        ll resFlow = 0, resCost = 0;
60        while (1)
61        {
62            forn(i, N) d[i] = INF, used[i] = 0;
63            d[ST] = 0;
64            forn(_, N)
65            {
66                int x = -1;
67                forn(i, N) if (!used[i] && (x == -1 || d[x] ←
68                    > d[i])) x = i;
69                used[x] = 1;
70                if (d[x] == INF) break;
71                for (int e = head[x]; e; e = next[e]) if (←
72                    flow[e] > 0)
73                {
74                    int y = to[e];
75                    ll len = cost[e] + p[x] - p[y];
76                    if (d[y] > d[x] + len)
77                    {
78                        d[y] = d[x] + len;
79                        last[y] = e;
80                    }
81                }
82            }
83
84            if (d[EN] == INF) break;
85
86            ll realCost = d[EN] + p[EN] - p[ST];
87            if (flag && realCost > 0) break;
88
89            int pushed = inf;
90            int x = EN;
91            while (x != ST)
92            {
93                int e = last[x];
94                pushed = min(pushed, flow[e]);
95                x = to[e ^ 1];
96            }

```

```

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 }

```

## final/graphs/retro.cpp

```

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 || (--deg[y] == 0 && res[x] == WIN)))
38                 {
39                     res[y] = 3 - res[x];
40                     moves[y] = moves[x] + 1;
41                     q[en++] = y;
42                 }
43             }
44         }
45     }
46 }

```

## final/graphs/smith.cpp

```

1 const int N = 1e5 + 10;
2
3 struct graph
4 {
5     int n;
6
7     vi v[N];
8     vi vrev[N];
9
10    void read()
11    {
12        int m;
13        scanf("%d%d", &n, &m);
14        forn(i, m)
15        {
16            int x, y;
17            scanf("%d%d", &x, &y);
18            --x, --y;
19            v[x].pb(y);
20            vrev[y].pb(x);
21        }
22    }
23
24    int deg[N], cnt[N], used[N], f[N];
25    int q[N], st, en;
26
27    set<int> s[N];
28
29    void calc()
30    {
31        forn(x, n) f[x] = -1, cnt[x] = 0;
32        int val = 0;
33        while (1)
34        {
35            st = en = 0;
36            forn(x, n)
37            {
38                deg[x] = 0;
39                used[x] = 0;
40                for (int y : v[x]) if (f[y] == -1) deg[x]++;
41            }
42            forn(x, n) if (!deg[x] && f[x] == -1 && cnt[x] <= val)
43            {
44                q[en++] = x;
45                f[x] = val;
46            }
47            if (!en) break;
48            while (st < en)
49            {
50                int x = q[st];
51                st++;
52                for (int y : vrev[x])
53                {
54                    if (used[y] == 0 && f[y] == -1)
55                    {
56                        used[y] = 1;
57                        cnt[y]++;
58                        for (int z : vrev[y])
59                        {
60                            deg[z]--;
61                            if (f[z] == -1 && deg[z] == 0 && cnt[z] <= val)
62                            {
63                                f[z] = val;
64                                q[en++] = z;
65                            }
66                        }
67                    }
68                }
69            }
70            val++;
71        }
72        forn(x, n) eprintf("%d%c", f[x], " \n"[x + 1 == n]);
73        forn(x, n) if (f[x] == -1)
74        {
75            for (int y : v[x]) if (f[y] != -1) s[x].insert(f[y]);
76        }
77    }
78    } g1, g2;
79
80    string get(int x, int y)
81    {
82        int f1 = g1.f[x], f2 = g2.f[y];
83        if (f1 == -1 && f2 == -1) return "draw";
84        if (f1 == -1) {

```

```

85     if (g1.s[x].count(f2)) return "first";
86     return "draw";
87 }
88 if (f2 == -1) {
89     if (g2.s[y].count(f1)) return "first";
90     return "draw";
91 }
92 if (f1 ^ f2) return "first";
93 return "second";
94 }

```

## final/graphs/twoChinese.cpp

```

1  const int INF = 1e9;
2  struct Edge {
3      int from, to, w, id;
4  };
5  namespace dmst {
6      int n;
7      vector<int> p;
8      vector<Edge> edges;
9
10     int get(int x) {
11         if (x == p[x]) return x;
12         return p[x] = get(p[x]);
13     }
14
15     void uni(int u, int v) {
16         p[get(v)] = get(u);
17     }
18
19     vector<Edge> solve() {
20         vector<int> id(n, -1);
21         vector<int> vert;
22         int cn = 0;
23         for (int i = 0; i < n; i++) if (get(i) == i) {
24             vert.push_back(i);
25             id[i] = cn++;
26         }
27         if (cn == 1) return vector<Edge>();
28
29         vector<vector<int>> e(cn);
30         for (int i = 0; i < (int)edges.size(); i++) {
31             if (get(edges[i].to) != get(edges[i].from)) {
32                 e[id[get(edges[i].to)]].push_back(i);
33             }
34         }
35
36         vector<int> nxtId(cn, -1);
37         for (int i = 0; i < cn; i++) {
38             int mn = INF;
39             for (int id : e[i]) mn = min(mn, edges[id].w);
40             for (int id : e[i]) {
41                 edges[id].w -= mn;
42                 if (edges[id].w == 0) nxtId[i] = id;
43             }
44         }
45
46         vector<char> vis(cn);
47         vis[0] = 1;
48         int cur = 1;
49         while (!vis[cur]) {
50             vis[cur] = 1;
51             cur = id[get(edges[nxtId[cur]].from)];
52         }
53         vector<Edge> ans;
54         if (cur == 0) {
55             for (int i = 0; i < cn; i++) {
56                 if (vis[i] && i != 0) {
57                     ans.push_back(edges[nxtId[i]]);
58                     uni(0, vert[i]);
59                 }
60             }
61             auto nans = solve();
62             for (auto ee : nans) ans.push_back(ee);
63             return ans;
64         }
65         vector<int> cp = p;
66         int o = cur;
67         while (1) {
68             uni(vert[o], vert[cur]);
69             ans.push_back(edges[nxtId[cur]]);
70             int to = id[get(edges[nxtId[cur]].from)];
71             if (to == o) break;
72             cur = to;
73         }
74         vector<Edge> nedges = solve();

```

```

75     p = cp;
76     vector<char> covered(cn);
77     for (auto ee : nedges) covered[id[get(ee.to)]] = 1;
78     for (auto ee : ans) if (!covered[id[get(ee.to)]]
79                             nedges.push_back(ee);
80     return nedges;
81 }
82 // root is 0
83 vector<Edge> getMst(int _n, vector<Edge> _edges) {
84     n = _n;
85     edges = _edges;
86     p.resize(n);
87     for (int i = 0; i < n; i++) p[i] = i;
88
89     return solve();
90 }
91 }

```