

Содержание

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

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 ↵
  02 <CR>
3 map <F8> :wall! <CR> :!ulimit -s 500000 && ./%:r <CR↵
  >
4
5 inoremap {<CR> {<CR>}<ESC>0
6 map <c-a> ggVG
7
8 set nu
9 set rnu
10 syntax on
11
12 map <c-t> :tabnew <CR>
13 map <c-l> :tabn <CR>
14 map <c-h> :tabp <CR>
15
16 set cin
17 set sw=4
18 set so=99
19 set bs=2
20 set et
21 set sts=4

```

musthave/template.cpp

```

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

```

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 ↵
  * m2 + a2;
4 }

```

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 = getChar();
35     return x;
36 }
37
38 inline int readInt() {
39     int s = 1, c = readChar();
40     int x = 0;
41     if (c == '-') s = -1, c = getChar();
42     while ('0' <= c && c <= '9') x = x * 10 + c - '0',↵
  c = getChar();
43     return s == 1 ? x : -x;
44 }
45
46
47 // 10M int [0..1e9)
48 // cin 3.02
49 // scanf 1.2
50 // cin_sync_with_stdio(false) 0.71
51 // fastRead getchar 0.53
52 // fastRead fread 0.15

```

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(←
15     a.x + b.x, a.y + b.y); }
16 inline num operator - (num a, num b) { return num(←
17     a.x - b.x, a.y - b.y); }
18 inline num operator * (num a, num b) { return num(←
19     a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x); }
20 inline num conj(num a) { return num(a.x, -a.y); }
21
22 const dbl PI = acos(-1);
23
24 num root[maxN];
25 int rev[maxN];
26 bool rootsPrepared = false;
27
28 void prepRoots()
29 {
30     if (rootsPrepared) return;
31     rootsPrepared = true;
32     root[1] = num(1, 0);
33     for (int k = 1; k < maxBase; ++k)
34     {
35         num x(2 * PI / pw(k + 1));
36         for (int i = pw(k - 1); i < pw(k); ++i)
37         {
38             root[2 * i] = root[i];
39             root[2 * i + 1] = root[i] * x;
40         }
41     }
42
43     int base, N;
44
45     int lastRevN = -1;
46     void prepRev()
47     {
48         if (lastRevN == N) return;
49         lastRevN = N;
50         forn(i, N) rev[i] = (rev[i >> 1] >> 1) + ((i & ←
51             1) << (base - 1));
52     }
53
54     void fft(num *a, num *f)
55     {
56         forn(i, N) f[i] = a[rev[i]];
57         for (int k = 1; k < N; k <= 1) for (int i = 0; ←
58             i < N; i += 2 * k) forn(j, k)
59         {
60             num z = f[i + j + k] * root[j + k];
61             f[i + j + k] = f[i + j] - z;
62             f[i + j] = f[i + j] + z;
63         }
64
65         num a[maxN], b[maxN], f[maxN], g[maxN];
66         ll A[maxN], B[maxN], C[maxN];
67
68         void _multMod(int mod)
69         {
70             forn(i, N)
71             {
72                 int x = A[i] % mod;
73                 a[i] = num(x & (pw(15) - 1), x >> 15);
74             }
75             forn(i, N)
76             {
77                 int x = B[i] % mod;
78                 b[i] = num(x & (pw(15) - 1), x >> 15);
79             }
80             fft(a, f);
81             fft(b, g);
82
83             forn(i, N)
84             {
85                 int j = (N - i) & (N - 1);
86                 num a1 = (f[i] + conj(f[j])) * num(0.5, 0);
87                 num a2 = (f[i] - conj(f[j])) * num(0, -0.5);
88                 num b1 = (g[i] + conj(g[j])) * num(0.5 / N, 0) ←
89                 num b2 = (g[i] - conj(g[j])) * num(0, -0.5 / N ←
90                 a[j] = a1 * b1 + a2 * b2 * num(0, 1);
91                 b[j] = a1 * b2 + a2 * b1;
92             }
93
94             fft(a, f);
95             fft(b, g);
96             forn(i, N)
97             {
98                 ll aa = f[i].x + 0.5;
99                 ll bb = g[i].x + 0.5;
100                 ll cc = f[i].y + 0.5;
101                 C[i] = (aa + bb % mod * pw(15) + cc % mod * pw ←
102                 (30)) % mod;
103             }
104         }
105
106         void prepAB(int n1, int n2)
107         {
108             base = 1;
109             N = 2;
110             while (N < n1 + n2) base++, N <= 1;
111
112             for (int i = n1; i < N; ++i) A[i] = 0;
113             for (int i = n2; i < N; ++i) B[i] = 0;
114
115             prepRoots();
116             prepRev();
117         }
118
119         void mult(int n1, int n2)
120         {
121             prepAB(n1, n2);
122             forn(i, N) a[i] = num(A[i], B[i]);
123             fft(a, f);
124             forn(i, N)
125             {
126                 int j = (N - i) & (N - 1);
127                 a[i] = (f[j] * f[j] - conj(f[i] * f[i])) * num ←
128                 (0, -0.25 / N);
129             }
130             fft(a, f);
131             forn(i, N) C[i] = (ll)round(f[i].x);
132         }
133
134         void multMod(int n1, int n2, int mod)
135         {
136             prepAB(n1, n2);
137             _multMod(mod);
138         }
139
140         int D[maxN];
141
142         void multLL(int n1, int n2)
143         {
144             prepAB(n1, n2);
145
146             int mod1 = 1.5e9;
147             int mod2 = mod1 + 1;
148
149             _multMod(mod1);
150
151             forn(i, N) D[i] = C[i];
152
153             _multMod(mod2);
154
155             forn(i, N)
156             {
157                 C[i] = D[i] + (C[i] - D[i] + (ll)mod2) * (ll) ←
158                 mod1 % mod2 * mod1;
159             }
160         }
161
162         // HOW TO USE ::
163         // — set correct maxBase
164         // — use mult(n1, n2), multMod(n1, n2, mod) and ←
165         multLL(n1, n2)
166         // — input : A[], B[]
167         // — output : C[]
168     }
169 }

```

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

```

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] * (11)B[k]) % mod;
11        int z = 1;
12        if (k == N - 1) return C[k];
13        while ((k & (z - 1)) == (z - 1))
14        {
15            // mult B[k - z + 1 ... k] x A[z .. 2 * z - 1]
16            forn(i, z) fft::A[i] = A[z + i];
17            forn(i, z) fft::B[i] = B[k - z + 1 + i];
18            fft::multMod(z, z, mod);
19            forn(i, 2 * z - 1) C[k + 1 + i] = (C[k + 1 + i] ←
20            + fft::C[i]) % mod;
21            z <= 1;
22        }
23        return C[k];
24    }
25    // A — constant array
26    // magic(k, x) :: B[k] = x, returns C[k]
27    // !! WARNING !! better to set N twice the size ←
28    needed
29 }

```

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

```

```

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 rB) {
4     vector<Line> res;
5     pt C = B - A;
6     dbl z = C.len2();
7     for (int i = -1; i <= 1; i += 2) {
8         for (int j = -1; j <= 1; j += 2) {
9             dbl r = rB * j - rA * i;
10            dbl d = z - r * r;
11            if (ls(d, 0)) continue;
12            d = sqrt(max(0.01, d));
13            pt magic = pt(r, d) / z;
14            pt v(magic % C, magic * C);
15            dbl CC = (rA * i - v % A) / v.len2();
16            pt O = v * -CC;
17            res.pb(Line(O, O + v.rotate()));
18        }
19    }
20    return res;
21 }
22
23 // HOW TO USE ::
24 // --- *D*-----*F*
25 // --- *...* - - - *...*
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
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,
11     Line l) {
12     vector<pair<pt, Meta>> res;
13     int n = p.size();
14     for (int i = 0; i < n; i++) {
15         pt A = p[i].F;
16         pt B = p[(i + 1) % n].F;
17         if (le(0, l.v * (A - l.O))) {
18             if (eq(0, l.v * (A - l.O)) && p[i].S.type == 1)
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.O))
25                 == -1) {
26                 pt FF = Line(A, B) * l;
27                 res.pb(make_pair(FF, SEG));
28             }
29         }
30         else {
31             pt E, F;
32             if (intCL(p[i].S.O, p[i].S.R, l, E, F)) {
33                 if (onArc(p[i].S.O, A, E, B))
34                     res.pb({E, SEG});
35                 if (onArc(p[i].S.O, A, F, B))
36                     res.pb({F, p[i].S});
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            i += 1;
12        return i;
13    }
14
15    Data & operator [] (HashType H) {
16        assert(H != 0);
17        int i = position(H);
18        if (!hash[i]) {
19            hash[i] = H;
20            f[i] = default_value;
21            size++;
22        }
23        return f[i];
24    }
25 };
26
27 hashTable<13, int, int, 0> h;

```

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

```

musthave/modReverseOneLine.cpp

```

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

```

musthave/optimizations.cpp

```

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

```

```

11         : "d" (xh), "a" (x1), "r" (y)
12     );
13 #else
14     __asm {
15         mov edx, dword ptr[xh];
16         mov eax, dword ptr[x1];
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,←
30     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/centroid.cpp

```

1 // original author: burunduk1, rewritten by me (←
2 // enot110)
3 // !!! warning !!! this code is not tested well
4 const int N = 1e5, K = 17;
5
6 int pivot, level[N], parent[N];
7 vector<int> v[N];
8
9 int get_pivot(int x, int xx, int n) {
10     int size = 1;
11     for (int y : v[x])
12     {
13         if (y != xx && level[y] == -1) size += get_pivot←
14         (y, x, n);
15     }
16     if (pivot == -1 && (size * 2 >= n || xx == -1)) ←
17         pivot = x;
18     return size;
19 }
20
21 void build(int x, int xx, int dep, int size) {
22     assert(dep < K);
23     pivot = -1;
24     get_pivot(x, -1, size);
25     x = pivot;
26     level[x] = dep, parent[x] = xx;
27     for (int y : v[x]) if (level[y] == -1)
28     {
29         build(y, x, dep + 1, size / 2);
30     }
31 }

```

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;

```

```

9  ll n, m;
10 int left[MAX_M];
11 int up[MAX_N];
12 int pos[MAX_N];
13 dbl res[MAX_N];
14
15 void init(int nn, int mm) {
16     n = nn;
17     m = mm;
18     v = 0;
19     for (int i = 0; i < m; i++)
20         for (int j = 0; j < n; j++)
21             a[i][j] = 0;
22     for (int i = 0; i < m; i++)
23         b[i] = 0;
24     for (int i = 0; i < n; i++)
25         c[i] = 0;
26 }
27
28 void pivot(int x, int y) {
29     swap(left[x], up[y]);
30     dbl k = a[x][y];
31     a[x][y] = 1;
32     b[x] /= k;
33     int cur = 0;
34     for (int i = 0; i < n; i++) {
35         a[x][i] = a[x][i] / k;
36         if (!eq(a[x][i], 0))
37             pos[cur++] = i;
38     }
39
40     for (int i = 0; i < m; i++) {
41         if (i == x || eq(a[i][y], 0)) continue;
42         dbl cof = a[i][y];
43         b[i] -= cof * b[x];
44         a[i][y] = 0;
45         for (int j = 0; j < cur; j++)
46             a[i][pos[j]] -= cof * a[x][pos[j]];
47     }
48     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])) x = i;
66
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
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]))) y = i;
84         if (y == -1) break;
85
86         int x = -1;
87         for (int i = 0; i < m; i++) {
88             if (ls(0, a[i][y])) {
89                 if (x == -1 || (b[i] / a[i][y] < b[x] / a[x][y])) x = i;
90             }
91         }
92         if (y == -1) {
93             assert(false); // infinite solution
94         }
95     }
96 }

```

```

99     pivot(x, y);
100 }
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 };

```

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, ←
5     null_type, less<T>, rb_tree_tag, ←
6     tree_order_statistics_node_update>;
7
8 template <typename K, typename V> using ordered_map ←
9     = tree<K, V, less<K>, rb_tree_tag, ←
10     tree_order_statistics_node_update>;
11
12 // HOW TO USE ::
13 // — order_of_key(10) returns the number of ←
14 //   elements in set/map strictly less than 10
15 // — *find_by_order(10) returns 10-th smallest ←
16 //   element in set/map (0-based)

```

musthave/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 }

```

musthave/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)) >= ll) {
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)
48 }

```

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/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[←
14             pos]) {
15             u = suf[u];
16         }
17     }
18
19     int add(int pos) {
20         go(last, pos);
21         int u = suf[last];
22         go(u, pos);
23         int c = s[pos] - 'a';
24         int res = 0;
25         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 }

```

musthave/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[←
14             pos]) {
15             u = suf[u];
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     }

```

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     return ans;
103 }

```

musthave/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)]] ← 1) nedges.push_back(ee);
79         return nedges;
80     }
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 }

```

musthave/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], sdом[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 (sdом[cmn[p[u]]] < sdом[cmn[u]]) cmn[u] = cmn ← [p[u]];
41         p[u] = v;
42         return x ? v : cmn[u];
43     }
44
45     void uni(int u, int v) {
46         p[v] = u;
47     }
48
49     void calc() {
50         for (int i = 0; i < n; i++) {
51             in[i] = -1;
52             adom[i] = -1;
53             dom[i] = sdом[i] = p[i] = cmn[i] = i;
54             vct[i].clear();
55             g[i].clear();
56         }
57         tmr = 0;
58         dfs(0);
59         for (int i = tmr - 1; i >= 0; i--) {
60             for (int to : g[i]) sdом[i] = min(sdом[i], ← sdом[get(to)]);
61             if (i > 0) vct[sdом[i]].push_back(i);
62             for (int w : vct[i]) {
63                 int v = get(w);
64                 if (sdом[v] == sdом[w]) dom[w] = sdом[w];
65                 else dom[w] = v;
66             }
67             if (i > 0) uni(par[i], i);
68         }
69         for (int i = 1; i < tmr; i++) {
70             if (dom[i] != sdом[i]) dom[i] = dom[dom[i]];
71             adom[rin[i]] = rin[dom[i]];
72         }
73     }
74 }

```

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];
79            if (d[y] == d[x] + 1 && f[e] && dfs(y, min(f[e] ←
80            , flow)))
81            {
82                f[e] -= pushed;
83                f[e ^ 1] += pushed;
84                return 1;
85            }
86        }
87        return 0;
88    }

```

```

87
88
89    ll calcFlow()
90    {
91        ll res = 0;
92        while (bfs())
93        {
94            forn(i, N) fst[i] = head[i];
95            while (dfs(ST))
96            {
97                res += pushed;
98            }
99        }
100        return res;
101    }
102
103    // HOW TO USE ::
104    // — set maxn and maxe (special for izban)
105    // — add adges using add(x, y, f), call setN(n)
106    // — run calcFlow
107 }

```

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])
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;

```

```

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     ll realCost = d[EN] + p[EN] - p[ST];
81     if (flag && realCost > 0) break;
82     int pushed = inf;
83     int x = EN;
84     while (x != ST)
85     {
86         int e = last[x];
87         pushed = min(pushed, flow[e]);
88         x = to[e ^ 1];
89     }
90     resCost += realCost * pushed;
91     resFlow += pushed;
92     x = EN;
93     while (x != ST)
94     {
95         int e = last[x];
96         flow[e] -= pushed;
97         flow[e ^ 1] += pushed;
98         x = to[e ^ 1];
99     }
100     forn(i, N) p[i] += d[i];
101     return mp(resFlow, resCost);
102 }
103
104 pair<ll, ll> maxFlow()
105 {
106     return _calc(0);
107 }
108
109 pair<ll, ll> minCost()
110 {
111     return _calc(1);
112 }
113
114 // HOW TO USE::
115 // — add adges using add(x, y, f, c), call setN(n)
116 // — run maxFlow/minCost, returns pair(flow, cost)
117 }

```

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
19    poly norm()
20    {
21        while (sz(v) > 1 && v.back() == 0) v.pop_back();
22        return *this;
23    }
24    inline int& operator [] (int i)
25    {
26        return v[i];
27    }
28    void out(string name="")
29    {
30        stringstream ss;
31        if (sz(name)) ss << name << " = ";
32        int fst = 1;
33        forn(i, sz(v)) if (v[i])
34        {
35            int x = v[i];
36            int sgn = 1;
37            if (x > mod / 2) x = mod - x, sgn = -1;
38            if (sgn == -1) ss << "-";
39            else if (!fst) ss << "+";
40            fst = 0;
41            if (!i || x != 1)
42            {
43                ss << x;
44                if (i > 0) ss << "*x";
45                if (i > 1) ss << "^" << i;
46            }
47            else
48            {
49                ss << "x";
50                if (i > 1) ss << "^" << i;
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) * (11)m) / x + m;
116     };
117 poly R({rev(A[0], mod)});
118 for (int k = 1; k < n; k <= 1)
119 {
120     poly A0 = cutPoly(A, 0, k);
121     poly A1 = cutPoly(A, k, 2 * k);
122     poly H = A0 * R;
123     H = cutPoly(H, k, 2 * k);
124     poly R1 = (((A1 * R).cut(k) + H) * (poly({0}) - ↔
125         R)).cut(k);
126     R.v.resize(2 * k);
127     forn(i, k) R[i + k] = R1[i];
128 }
129 return R.cut(n).norm();
130 }
131 pair<poly, poly> divide(poly A, poly B)
132 {
133     if (sz(A) < sz(B)) return {poly({0}), A};
134
135     auto rev = [] (poly f)
136     {
137         reverse(all(f.v));
138         return f;
139     };
140
141     poly q = rev((inv(rev(B), sz(A) - sz(B) + 1) * rev↔
142         (A)).cut(sz(A) - sz(B) + 1));
143     poly r = A - B * q;
144     return {q, r};
145 }

```

```

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 }

```

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        bool is_prime(int x)
18        {
19            if (x < maxP) return pp[x] == x;
20            for (int i = 0; p[i] * p[i] <= x; ++i) if (x % p↔
21                [i] == 0) return false;
22            return true;
23        }
24        // HOW TO USE ::
25        // pp[x] <= smallest prime divisor {x} (or -1 for↔
26        // {x < 2})
27        // p[0] .. pc - 1 <= list of primes < maxP
28    }

```

musthave/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)

```

musthave/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", &m);
14        forn(i, m)
15        {
16            int x, y;
17            scanf("%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]↔
43                == val)
44            {
45                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     val++;
70 }
71 forn(x, n) eprintf("%d%c", f[x], " \n"[x + 1 == n]);
72 forn(x, n) if (f[x] == -1)
73 {
74     for (int y : v[x]) if (f[y] != -1) s[x].insert(f[y]);
75 }
76 }
77 } g1, g2;
78
79 string get(int x, int y)
80 {
81     int f1 = g1.f[x], f2 = g2.f[y];
82     if (f1 == -1 && f2 == -1) return "draw";
83     if (f1 == -1) {
84         if (g1.s[x].count(f2)) return "first";
85         return "draw";
86     }
87     if (f2 == -1) {
88         if (g2.s[y].count(f1)) return "first";
89         return "draw";
90     }
91     if (f1 ^ f2) return "first";
92     return "second";
93 }
94

```

musthave/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
83 // root is 0
84 vector<Edge> getMst(int _n, vector<Edge> _edges) {
85     n = _n;
86     edges = _edges;
87     p.resize(n);
88     for (int i = 0; i < n; i++) p[i] = i;
89     return solve();
90 }
91

```