

crt

```
1 int CRT(int a1, int m1, int a2, int m2)
2 {
3     return (a1 - a2 % m1 + m1) * (1ll)rev(m2, m1) % m1
4         * m2 + a2;
5 }
```

fastIO

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

fft

```
1 namespace fft
2 {
3     const int maxBase = 21;
4     const int maxN = 1 << maxBase;
5
6     struct num
7     {
8         dbl x, y;
9         num() {}
10        num(dbl xx, dbl yy): x(xx), y(yy) {}
11        num(dbl alp): x(cos(alp)), y(sin(alp)) {}
12    };
13
14    inline num operator + (num a, num b) { return num(a.x + b.x, a.y + b.y); }
15    inline num operator - (num a, num b) { return num(a.x - b.x, a.y - b.y); }
16    inline num operator * (num a, num b) { return num(a.x * b.x - a.y * b.y, a.x * b.y + a.y * b.x); }
17    inline num conj(num a) { return num(a.x, -a.y); }
18
19    const dbl PI = acos(-1);
20
21    num root[maxN];
22    int rev[maxN];
23    bool rootsPrepared = false;
24
25    void prepRoots()
26    {
27        if (rootsPrepared) return;
28        rootsPrepared = true;
29        root[1] = num(1, 0);
30        for (int k = 1; k < maxBase; ++k)
31        {
32            num x(2 * PI / pw(k + 1));
33            for (int i = pw(k - 1); i < pw(k); ++i)
34            {
35                root[2 * i] = root[i];
36                root[2 * i + 1] = root[i] * x;
37            }
38        }
39    }
40
41    int base, N;
42
43    int lastRevN = -1;
44    void prepRev()
45    {
46        if (lastRevN == N) return;
47        lastRevN = N;
48        forn(i, N) rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (base - 1));
49    }
50
51    void fft(num *a, num *f)
52    {
53        forn(i, N) f[i] = a[rev[i]];
54        for (int k = 1; k < N; k <= 1) for (int i = 0; i < N; i += 2 * k) forn(j, k)
55        {
56            num z = f[i + j + k] * root[j + k];
57            f[i + j + k] = f[i + j] - z;
58            f[i + j] = f[i + j] + z;
59        }
60    }
61
62    num a[maxN], b[maxN], f[maxN], g[maxN];
63    ll A[maxN], B[maxN], C[maxN];
64
65    void _multMod(int mod)
66    {
```

```

67     forn(i, N)
68     {
69         int x = A[i] % mod;
70         a[i] = num(x & (pw(15) - 1), x >> 15);
71     }
72     forn(i, N)
73     {
74         int x = B[i] % mod;
75         b[i] = num(x & (pw(15) - 1), x >> 15);
76     }
77     fft(a, f);
78     fft(b, g);
79
80     forn(i, N)
81     {
82         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         num b2 = (g[i] - conj(g[j])) * num(0, -0.5 / N);
87         a[j] = a1 * b1 + a2 * b2 * num(0, 1);
88         b[j] = a1 * b2 + a2 * b1;
89     }
90
91     fft(a, f);
92     fft(b, g);
93
94     forn(i, N)
95     {
96         ll aa = f[i].x + 0.5;
97         ll bb = g[i].x + 0.5;
98         ll cc = f[i].y + 0.5;
99         C[i] = (aa + bb % mod * pw(15) + cc % mod * pw
100 (30)) % mod;
101     }
102
103 void prepAB(int n1, int n2)
104 {
105     base = 1;
106     N = 2;
107     while (N < n1 + n2) base++, N <= 1;
108
109     for (int i = n1; i < N; ++i) A[i] = 0;
110     for (int i = n2; i < N; ++i) B[i] = 0;
111
112     prepRoots();
113     prepRev();
114 }
115
116 void mult(int n1, int n2)
117 {
118     prepAB(n1, n2);
119     forn(i, N) a[i] = num(A[i], B[i]);
120     fft(a, f);
121     forn(i, N)
122     {
123         int j = (N - i) & (N - 1);
124         a[i] = (f[j] * f[j] - conj(f[i] * f[i])) * num
125 (0, -0.25 / N);
126     }
127     fft(a, f);
128     forn(i, N) C[i] = (ll)round(f[i].x);
129 }
130
131 void multMod(int n1, int n2, int mod)
132 {
133     prepAB(n1, n2);
134     _multMod(mod);
135 }
136
137 int D[maxN];
138
139 void multLL(int n1, int n2)
140 {
141     prepAB(n1, n2);
142
143     int mod1 = 1.5e9;
144     int mod2 = mod1 + 1;
145
146     _multMod(mod1);
147
148     forn(i, N) D[i] = C[i];
149     _multMod(mod2);
150
151     forn(i, N)
152     {
153         C[i] = D[i] + (C[i] - D[i] + (ll)mod2) * (ll)
154 mod1 % mod2 * mod1;
155     }
156 }
157 // HOW TO USE ::
158 // -- set correct maxBase
159 // -- use mult(n1, n2), multMod(n1, n2, mod) and
160 multLL(n1, n2)
161 // -- input : A[], B[]
162 // -- output : C[]
162}

```

fftint

```

1 namespace fft
2 {
3     const int mod = 998244353;
4     const int base = 20;
5     const int N = 1 << base;
6     const int ROOT = 646;
7
8     int root[N];
9     int rev[N];
10
11 void init()
12 {
13     forn(i, N) rev[i] = (rev[i >> 1] >> 1) + ((i &
14 1) << (base - 1));
15     int NN = N >> 1;
16     int z = 1;
17     forn(i, NN)
18     {
19         root[i + NN] = z;
20         z = z * (ll)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] * (ll)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 }

```

```

35 int A[N], B[N], C[N];
36 int F[N], G[N];
37
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] * (ll)G[i] % mod * invN % mod;
45     reverse(A + 1, A + N);
46     fft(A, C);
47 }
48
49 void mult(int n1, int n2, int eq = 0)
50 {
51     for (int i = n1; i < N; ++i) A[i] = 0;
52     for (int i = n2; i < N; ++i) B[i] = 0;
53
54     _mult(eq);
55
56     //forn(i, n1 + n2) C[i] = 0;
57     //forn(i, n1) forn(j, n2) C[i + j] = (C[i + j] +
58         A[i] * (ll)B[j]) % mod;
59 }

```

halfplaneIntersection

```

1int getPart(pt v) {
2    return less(0, v.y) || (equal(0, v.y) && less(v.x, 0));
3}
4
5int 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
15double planeInt(vector<Line> l) {
16     int n = l.size();
17     sort(all(l), [](Line a, Line b) {
18         int r = cmpV(a.v, b.v);
19         if (r != 0) return r < 0;
20         return a.0 % a.v.rotate() < b.0 % a.v.rotate();
21     });
22
23     int cur = 0;
24     for (int i = 0; i < n; ) {
25         int j = i;
26         for (; i < n && cmpV(l[j].v, l[i].v) == 0 &&
27             cmpV(l[i].v, l[j].v) == 0; i++);
28         l[cur++] = l[i - 1];
29     }
30     n = cur;
31
32     for (int i = 0; i < n; i++)
33         l[i].id = i;
34
35     int flagUp = 0;
36     int flagDown = 0;
37     for (int i = 0; i < n; i++) {
38         int part = getPart(l[i].v);
39         if (part == 1) flagUp = 1;
40         if (part == 0) flagDown = 1;
41     }
42     if (!flagUp || !flagDown) return -1;
43
44     for (int i = 0; i < n; i++) {
45         pt v = l[i].v;
46         pt u = l[(i + 1) % n].v;
47         if (equal(0, v * u) && less(v % u, 0)) {
48             pt dir = l[i].v.rotate();
49             if (lessE(l[(i + 1) % n].0 % dir, l[i].0 % dir))
50                 return 0;
51             return -1;
52         }
53     }
54
55     cur = 0;
56     vector<Line> st(n * 2);
57     for (int tt = 0; tt < 2; tt++) {
58         for (int i = 0; i < n; i++) {
59             for (; cur >= 2; cur--) {
60                 pt G = st[cur - 1] * l[i];
61                 if (!lessE(st[cur - 2].v * (G - st[cur - 2].0), 0))
62                     break;
63             }
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 }

```

hash_table

```

1template <const int max_size, class HashType, class
2    Data, const Data default_value>
3struct hashTable {
4    HashType hash[max_size];
5    Data f[max_size];
6
7    int position(HashType H) const {
8        int i = H % max_size;
9        while (hash[i] && hash[i] != H)
10            if (++i == max_size)
11                i = 0;
12        return i;

```

```

13 }
14
15 Data & operator [] (HashType H ) {
16     assert(H != 0);
17     int i = position(H);
18     if (!hash[i]) {
19         hash[i] = H;
20         f[i] = default_value;
21         size++;
22     }
23     return f[i];
24 }
25};
26
27hashTable<13, int, int, 0> h;

```

hungary

```

1 namespace hungary
2 {
3     const int N = 210;
4
5     int a[N][N];
6     int ans[N];
7
8     int calc(int n, int m)
9     {
10         ++n, ++m;
11         vi u(n), v(m), p(m), prev(m);
12         for (int i = 1; i < n; ++i)
13         {
14             p[0] = i;
15             int x = 0;
16             vi mn(m, inf);
17             vi was(m, 0);
18             while (p[x])
19             {
20                 was[x] = 1;
21                 int ii = p[x], dd = inf, y = 0;
22                 for (int j = 1; j < m; ++j) if (!was[j])
23                 {
24                     int cur = a[ii][j] - u[ii] - v[j];
25                     if (cur < mn[j]) mn[j] = cur, prev[j] = x;
26                     if (mn[j] < dd) dd = mn[j], y = j;
27                 }
28                 forn(j, m)
29                 {
30                     if (was[j]) u[p[j]] += dd, v[j] -= dd;
31                     else mn[j] -= dd;
32                 }
33                 x = y;
34             }
35             while (x)
36             {
37                 int y = prev[x];
38                 p[x] = p[y];
39                 x = y;
40             }
41         }
42         for (int j = 1; j < m; ++j)
43         {
44             ans[p[j]] = j;
45         }
46         return -v[0];
47     }
48     // HOW TO USE ::
49     // -- set values to a[1..n][1..m] (n <= m)
50     // -- run calc(n, m) to find MINIMUM
51     // -- to restore permutation use ans[]
52     // -- everything works on negative numbers
53     //

```

```

54 // !! i don't understand this code, it's
55 // copyasted from e-maxx (and rewritten by enot110)
56 }

```

modReverseOneLine

```

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

```

optimizations

```

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

```

plane3DInt

```

1 // (A, v) * (B, u) -> (0, n)
2
3 pt n = v * u;
4 pt m = v * n;
5 double t = (B - A) % u / (u % m);
6 pt 0 = A - m * t;

```

simplex

```

1
2 struct Simplex {
3     ll a[MAX_M][MAX_N];
4     ll b[MAX_M];
5     ll c[MAX_N];
6     ll v;
7     ll n, m;
8     ll left[MAX_M];

```

```

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

```

```

80     }
81     while (1) {
82         int y = -1;
83         for (int i = 0; i < n; i++)
84             if (c[i] > 0 && (y == -1 || (c[i] > c[y])))
85                 y = i;
86     }
87     if (y == -1) break;
88
89     int x = -1;
90     for (int i = 0; i < m; i++) {
91         if (a[i][y] > 0) {
92             if (x == -1 || (b[i] / a[i][y] < b[x] / a[
93                 x][y])) {
94                 x = i;
95             }
96         }
97     }
98     if (y == -1) {
99         assert(false); // infinite solution
100     }
101     pivot(x, y);
102 }
103 memset(res, 0, sizeof(res));
104
105 for (int i = 0; i < m; i++) {
106     if (left[i] < n) {
107         res[left[i]] = b[i];
108     }
109 }
110 }
111};

```

std_rb_tree

```

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

```

sufAutomaton

```

1namespace 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
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

Суффиксный автомат

```

1 namespace SA {
2     const int MAXN = 1 << 18;
3     const int SIGMA = 26;
4
5     int sz, last;
6     int nxt[MAXN][SIGMA];
7     int link[MAXN], len[MAXN], pos[MAXN];
8
9     void init() {
10         memset(nxt, -1, sizeof(nxt));
11         memset(link, -1, sizeof(link));
12         memset(len, 0, sizeof(len));
13         last = 0;
14         sz = 1;
15     }
16
17     void add(int c) {
18         int cur = sz++;
19         len[cur] = len[last] + 1;
20
21         pos[cur] = len[cur];
22         int p = last;
23         last = cur;
24         for (; p != -1 && nxt[p][c] == -1; p = link[p])
25             nxt[p][c] = cur;
26         if (p == -1) {
27             link[cur] = 0;
28             return;
29         }
30         int q = nxt[p][c];
31         if (len[p] + 1 == len[q]) {
32             link[cur] = q;
33             return;
34         }
35         int clone = sz++;
36         memcpy(nxt[clone], nxt[q], sizeof(nxt[q]));
37         len[clone] = len[p] + 1;
38         pos[clone] = pos[q];
39         link[clone] = link[q];
40         link[q] = link[cur] = clone;
41         for (; p != -1 && nxt[p][c] == q; p = link[p])
42             nxt[p][c] = clone;
43     }
44 }
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

template

```

1 // team : SPb ITMO University 1
2 #include <bits/stdc++.h>
3
4 #define F first
5 #define S second
6 #define pb push_back
7 #define forn(i, n) for(int i = 0 ; (i) < (n) ; ++i)
8 #define eprintf(...) fprintf(stderr, __VA_ARGS__),
9         fflush(stderr)
10 #define sz(a) ((int)(a).size())
11 #define all(a) (a).begin(), a.end()
12 #define pw(x) (1LL<<(x))
13
14 using namespace std;
15
16 typedef long long ll;
17 typedef double dbl;
18 typedef vector<int> vi;
19 typedef pair<int, int> pi;
20
21 const int INF = 1.01e9;
22 const dbl eps = 1e-9;
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

```

23/* --- main part --- */
24
25
26
27
28
29
30
31int 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=%dms\n", (int)(clock() * 1000. /
45        CLOCKS_PER_SEC));
46    return 0;
47}

```

vimrc

```

1map <F9> :wall! <CR> :!g++ -Wall -Wextra -std=c++14
    -o %:r % -D_GLIBCXX_DEBUG -fsanitize=address -
    DHOME -Wshadow -Wno-unused-result <CR>
2map <F7> :wall! <CR> :!g++ -Wall -Wextra -std=c++14
    -o %:r % -O2 -DHOME -Wshadow -Wno-unused-result
    <CR>
3map <F8> :wall! <CR> :!./%:r <CR>
4
5inoremap {<CR> {<CR>}<ESC>O
6inoremap <c-p> {
7map <c-a> ggVG
8
9set nu
10set rnu
11syntax on
12
13map <c-t> :tabnew <CR>
14map <c-l> :tabn <CR>
15map <c-h> :tabp <CR>
16
17set cin
18set sw=4
19set so=99
20set bs=2
21set et
22set sts=4

```

dinica

```

1namespace 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;

```

```

82     return 1;
83 }
84 }
85 return 0;
86 }
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}

```

fastcnk

```

1 namespace math
2 {
3     const int N = 3e5 + 10;
4
5     int fact[N];
6     int revfact[N];
7
8     int revx[N];
9
10    void init()
11    {
12        fact[0] = 1;
13        for (int i = 1; i < N; ++i) fact[i] = fact[i - 1] * (1ll)i % mod;
14        revfact[0] = revfact[1] = 1;
15        revx[1] = 1;
16        for (int i = 2; i < N; ++i)
17        {
18            int c = mod / i + 1;
19            revx[i] = revx[c * i - mod] * (1ll)c % mod;
20            revfact[i] = revfact[i - 1] * (1ll)revx[i] % mod;
21        }
22    }
23
24    int cnk(int n, int k)
25    {
26        if (k < 0 || k > n) return 0;
27        return fact[n] * (1ll)revfact[n - k] % mod * revfact[k] % mod;
28    }
29}

```

max-flow-min-cost

```

1 namespace flow
2 {
3     const int maxn = 2e5 + 10;
4     const int maxe = 2 * maxn;
5

```

```

6     int head[maxn], next[maxe], to[maxe], flow[maxe],
7     cost[maxe], ec = 1;
8     int ST, EN, N = maxn;
9     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 f, int c)
17    {
18        ++ec;
19        to[ec] = y;
20        next[ec] = head[x];
21        head[x] = ec;
22        flow[ec] = f;
23        cost[ec] = c;
24    }
25
26    inline int add(int x, int y, int f, int c)
27    {
28        _add(x, y, f, c);
29        _add(y, x, 0, -c);
30        return ec - 1;
31    }
32
33    void clear()
34    {
35        forn(i, N) head[i] = 0;
36        ec = 1;
37    }
38
39    ll d[maxn], p[maxn];
40    int last[maxn];
41    int used[maxn];
42
43    pair<ll, ll> _calc(int flag)
44    {
45        const ll INF = 1e12;
46        forn(i, N) p[i] = INF;
47        p[ST] = 0;
48        forn(_, N) forn(x, N) for (int e = head[x]; e; e = next[e]) if (flow[e] > 0)
49        {
50            int y = to[e];
51            if (p[y] > p[x] + cost[e])
52            {
53                p[y] = p[x] + cost[e];
54            }
55        }
56
57        ll resFlow = 0, resCost = 0;
58        while (1)
59        {
60            forn(i, N) d[i] = INF, used[i] = 0;
61            d[ST] = 0;
62            forn(_, N)
63            {
64                int x = -1;
65                forn(i, N) if (!used[i] && (x == -1 || d[x] > d[i])) x = i;
66                used[x] = 1;
67                if (d[x] == INF) break;
68                for (int e = head[x]; e; e = next[e]) if (flow[e] > 0)
69                {
70                    int y = to[e];
71                    ll len = cost[e] + p[x] - p[y];
72                    if (d[y] > d[x] + len)

```



```

73     {
74         d[y] = d[x] + len;
75         last[y] = e;
76     }
77 }
78 }
79
80 if (d[EN] == INF) break;
81
82 ll realCost = d[EN] + p[EN] - p[ST];
83 if (flag && realCost > 0) break;
84
85 int pushed = inf;
86 int x = EN;
87 while (x != ST)
88 {
89     int e = last[x];
90     pushed = min(pushed, flow[e]);
91     x = to[e ^ 1];
92 }
93
94 resCost += realCost * pushed;
95 resFlow += pushed;
96
97 x = EN;
98 while (x != ST)
99 {
100     int e = last[x];
101     flow[e] -= pushed;
102     flow[e ^ 1] += pushed;
103     x = to[e ^ 1];
104 }
105
106 forn(i, N) p[i] += d[i];
107 }
108 return mp(resFlow, resCost);
109 }
110
111 pair<ll, ll> maxFlow()
112 {
113     return _calc(0);
114 }
115
116 pair<ll, ll> minCost()
117 {
118     return _calc(1);
119 }
120
121 // HOW TO USE::
122 // -- add edges using add(x, y, f, c), call setN(n)
123 // -- run maxFlow/minCost, returns pair(flow, cost)
124 }

```

poly

```

1 struct poly
2 {
3     vi v;
4     poly() {}
5     poly(vi vv)
6     {
7         v = vv;
8     }
9     int size()
10    {
11        return (int)v.size();
12    }
13    poly cut(int maxLen)
14    {
15        if (maxLen < sz(v)) v.resize(maxLen);
16        return *this;
17    }
18    poly norm()
19    {
20        while (sz(v) > 1 && v.back() == 0) v.pop_back();
21        return *this;
22    }
23    inline int& operator [] (int i)
24    {
25        return v[i];
26    }
27    void out(string name="")
28    {
29        stringstream ss;
30        if (sz(name)) ss << name << "=";
31        int fst = 1;
32        forn(i, sz(v)) if (v[i])
33        {
34            int x = v[i];
35            int sgn = 1;
36            if (x > mod / 2) x = mod - x, sgn = -1;
37            if (sgn == -1) ss << "-";
38            else if (!fst) ss << "+";
39            fst = 0;
40            if (!i || x != 1)
41            {
42                ss << x;
43                if (i > 0) ss << "*x";
44                if (i > 1) ss << "^" << i;
45            }
46            else
47            {
48                ss << "x";
49                if (i > 1) ss << "^" << i;
50            }
51        }
52        if (fst) ss << "0";
53        string s;
54        ss >> s;
55        eprintf("%s\n", s.data());
56    }
57};
58
59 poly operator + (poly A, poly B)
60 {
61     poly C;
62     C.v = vi(max(sz(A), sz(B)));
63     forn(i, sz(C))
64     {
65         if (i < sz(A)) C[i] = (C[i] + A[i]) % mod;
66         if (i < sz(B)) C[i] = (C[i] + B[i]) % mod;
67     }
68     return C.norm();
69 }
70
71 poly operator - (poly A, poly B)
72 {
73     poly C;
74     C.v = vi(max(sz(A), sz(B)));
75     forn(i, sz(C))
76     {
77         if (i < sz(A)) C[i] = (C[i] + A[i]) % mod;
78         if (i < sz(B)) C[i] = (C[i] + mod - B[i]) % mod;
79     }
80     return C.norm();
81 }
82
83 poly operator * (poly A, poly B)
84 {
85     poly C;

```

```

86 C.v = vi(sz(A) + sz(B) - 1);
87
88 forn(i, sz(A)) fft::A[i] = A[i];
89 forn(i, sz(B)) fft::B[i] = B[i];
90 fft::multMod(sz(A), sz(B), mod);
91 forn(i, sz(C)) C[i] = fft::C[i];
92 return C.norm();
93}
94
95poly inv(poly A, int n) // returns  $A^{-1} \bmod x^n$ 
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  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
132pair<poly, poly> divide(poly A, poly B)
133{
134  if (sz(A) < sz(B)) return {poly({0}), A};
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}

```

```

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

```

primes

```

1 namespace math
2 {
3   const int maxP = 1e6;
4   int pp[maxP];
5   int p[maxP / 10 + 1000], pc = 0;

```