Содержание

musthave/vimrc.txt

```
map <F9> :wall! <CR> :!g++ -Wall -Wextra -Wshadow -Wno-unused-result -o %:r % -std=c++14 -DHOME -D_GLIBCXX_DEBUG -fsanitize=address <CR>
map <F7> :wall! <CR> :!g++ -Wall -Wextra -Wshadow -Wno-unused-result -o %:r % -std=c++14 -DHOME -O2 <CR>
map <F8> :wall! <CR> :!ulimit -s 500000 && ./%:r <CR>

inoremap {<CR> {<CR>}<ESC>O
map <c-a> ggVG

set nu
set rnu
syntax on

map <c-t> :tabnew <CR>
map <c-l> :tabn <CR>
map <c-h> :tabp <CR>

set cin
set sw=4
set so=99
set bs=2
set et
set sts=4
```

musthave/template.cpp

```cpp
// team : SPb ITMO University 1
#include <bits/stdc++.h>

#define F first
#define S second
#define pb push_back
#define forn(i, n) for(int i = 0 ; (i) < (n) ; ++i)
#define eprintf(...) fprintf(stderr, __VA_ARGS__),fflush(stderr)
#define sz(a) ((int)(a).size())
#define all(a) (a).begin(),a.end()
#define pw(x) (1LL<<(x))

using namespace std;

typedef long long ll;
typedef double dbl;
typedef vector<int> vi;
typedef pair<int, int> pi;

const int INF = 1.01e9;
const dbl eps = 1e-9;

/* --- main part --- */







int main()
{
#define TASK ""
#ifdef home
  assert(freopen(TASK".in", "r", stdin));
  //assert(freopen(TASK".out", "w", stdout));
#endif





#ifdef home
  eprintf("time = %d ms\n", (int)(clock() * 1000. / CLOCKS_PER_SEC));
#endif
  return 0;
}
```

musthave/crt.cpp

```cpp
int CRT(int a1, int m1, int a2, int m2)
{
  return (a1 - a2 % m1 + m1) * (ll)rev(m2, m1) % m1 * m2 + a2;
}
```

musthave/fastIO.cpp

```cpp
#include <cstdio>
#include <algorithm>

/** Interface */

inline int readInt();
inline int readUInt();
inline bool isEof();

/** Read */

static const int buf_size = 100000;
static char buf[buf_size];
static int buf_len = 0, pos = 0;

inline bool isEof() {
  if (pos == buf_len) {
    pos = 0, buf_len = fread(buf, 1, buf_size, stdin);
    if (pos == buf_len) return 1;
  }
  return 0;
}

inline int getChar() { return isEof() ? -1 : buf[pos++]; }

inline int readChar() {
  int c = getChar();
  while (c != -1 && c <= 32) c = getChar();
  return c;
}

inline int readUInt() {
  int c = readChar(), x = 0;
  while ('0' <= c && c <= '9') x = x * 10 + c - '0', c = getChar();
  return x;
}

inline int readInt() {
  int s = 1, c = readChar();
  int x = 0;
  if (c == '-') s = -1, c = getChar();
  while ('0' <= c && c <= '9') x = x * 10 + c - '0', c = getChar();
  return s == 1 ? x : -x;
}


// 10M int [0..1e9)
// cin 3.02
// scanf 1.2
// cin sync_with_stdio(false) 0.71
// fastRead getchar 0.53
// fastRead fread 0.15
```

musthave/fft.cpp

```cpp
namespace fft
{
  const int maxBase = 21;
  const int maxN = 1 << maxBase;

  struct num
  {
    dbl x, y;
    num(){}
    num(dbl xx, dbl yy): x(xx), y(yy) {}
    num(dbl alp): x(cos(alp)), y(sin(alp)) {}
  };

  inline num operator + (num a, num b) { return num(a.x + b.x, a.y + b.y); }
  inline num operator - (num a, num b) { return num(a.x - b.x, a.y - b.y); }
  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); }
  inline num conj(num a) { return num(a.x, -a.y); }

  const dbl PI = acos(-1);

  num root[maxN];
  int rev[maxN];
  bool rootsPrepared = false;

  void prepRoots()
  {
    if (rootsPrepared) return;
    rootsPrepared = true;
    root[1] = num(1, 0);
    for (int k = 1; k < maxBase; ++k)
    {
      num x(2 * PI / pw(k + 1));
      for (int i = pw(k - 1); i < pw(k); ++i)
      {
        root[2 * i] = root[i];
        root[2 * i + 1] = root[i] * x;
      }
    }
  }

  int base, N;

  int lastRevN = -1;
  void prepRev()
  {
    if (lastRevN == N) return;
    lastRevN = N;
    forn(i, N) rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (base - 1));
  }

  void fft(num *a, num *f)
  {
    forn(i, N) f[i] = a[rev[i]];
    for (int k = 1; k < N; k <<= 1) for (int i = 0; i < N; i += 2 * k) forn(j, k)
    {
      num z = f[i + j + k] * root[j + k];
      f[i + j + k] = f[i + j] - z;
      f[i + j] = f[i + j] + z;
    }
  }

  num a[maxN], b[maxN], f[maxN], g[maxN];
  ll A[maxN], B[maxN], C[maxN];

  void _multMod(int mod)
  {
    forn(i, N)
    {
      int x = A[i] % mod;
      a[i] = num(x & (pw(15) - 1), x >> 15);
    }
    forn(i, N)
    {
      int x = B[i] % mod;
      b[i] = num(x & (pw(15) - 1), x >> 15);
    }
    fft(a, f);
    fft(b, g);

    forn(i, N)
    {
      int j = (N - i) & (N - 1);
      num a1 = (f[i] + conj(f[j])) * num(0.5, 0);
      num a2 = (f[i] - conj(f[j])) * num(0, -0.5);
      num b1 = (g[i] + conj(g[j])) * num(0.5 / N, 0);
      num b2 = (g[i] - conj(g[j])) * num(0, -0.5 / N);
      a[j] = a1 * b1 + a2 * b2 * num(0, 1);
      b[j] = a1 * b2 + a2 * b1;
    }

    fft(a, f);
    fft(b, g);

    forn(i, N)
    {
      ll aa = f[i].x + 0.5;
      ll bb = g[i].x + 0.5;
      ll cc = f[i].y + 0.5;
      C[i] = (aa + bb % mod * pw(15) + cc % mod * pw(30)) % mod;
    }
  }

  void prepAB(int n1, int n2)
  {
    base = 1;
    N = 2;
    while (N < n1 + n2) base++, N <<= 1;

    for (int i = n1; i < N; ++i) A[i] = 0;
    for (int i = n2; i < N; ++i) B[i] = 0;

    prepRoots();
    prepRev();
  }

  void mult(int n1, int n2)
  {
    prepAB(n1, n2);
    forn(i, N) a[i] = num(A[i], B[i]);
    fft(a, f);
    forn(i, N)
    {
      int j = (N - i) & (N - 1);
      a[i] = (f[j] * f[j] - conj(f[i] * f[i])) * num(0, -0.25 / N);
    }
    fft(a, f);
    forn(i, N) C[i] = (ll)round(f[i].x);
  }


  void multMod(int n1, int n2, int mod)
  {
    prepAB(n1, n2);
    _multMod(mod);
  }

  int D[maxN];

  void multLL(int n1, int n2)
  {
    prepAB(n1, n2);

    int mod1 = 1.5e9;
    int mod2 = mod1 + 1;

    _multMod(mod1);

    forn(i, N) D[i] = C[i];

    _multMod(mod2);

    forn(i, N)
    {
      C[i] = D[i] + (C[i] - D[i] + (ll)mod2) * (ll)mod1 % mod2 * mod1;
    }
  }
  // HOW TO USE ::
  // -- set correct maxBase
  // -- use mult(n1, n2), multMod(n1, n2, mod) and multLL(n1, n2)
  // -- input : A[], B[]
  // -- output : C[]
}
```

musthave/fftint.cpp

```
namespace fft
{
  const int mod = 998244353;
  const int base = 20;
  const int N = 1 << base;
  const int ROOT = 646;

  int root[N];
  int rev[N];

  void init()
  {
    forn(i, N) rev[i] = (rev[i >> 1] >> 1) + ((i & 1) << (base - 1));
    int NN = N >> 1;
    int z = 1;
    forn(i, NN)
    {
      root[i + NN] = z;
      z = z * (ll)ROOT % mod;
    }
    for (int i = NN - 1; i > 0; --i) root[i] = root[2 * i];
  }

  void fft(int *a, int *f)
  {
    forn(i, N) f[i] = a[rev[i]];
    for (int k = 1; k < N; k <<= 1) for (int i = 0; i < N; i += 2 * k) forn(j, k)
    {
      int z = f[i + j + k] * (ll)root[j + k] % mod;
      f[i + j + k] = (f[i + j] - z + mod) % mod;
      f[i + j] = (f[i + j] + z) % mod;
    }
  }

  int A[N], B[N], C[N];
  int F[N], G[N];

  void _mult(int eq)
  {
    fft(A, F);
    if (eq) forn(i, N) G[i] = F[i];
    else fft(B, G);
    int invN = inv(N);
    forn(i, N) A[i] = F[i] * (ll)G[i] % mod * invN % mod;
    reverse(A + 1, A + N);
    fft(A, C);
  }

  void mult(int n1, int n2, int eq = 0)
  {
    for (int i = n1; i < N; ++i) A[i] = 0;
    for (int i = n2; i < N; ++i) B[i] = 0;

    _mult(eq);

    //forn(i, n1 + n2) C[i] = 0;
    //forn(i, n1) forn(j, n2) C[i + j] = (C[i + j] + A[i] * (ll)B[j]) % mod;
  }
}
```

musthave/blackbox.cpp

```
namespace blackbox
{
  int A[N];
  int B[N];
  int C[N];

  int magic(int k, int x)
  {
    B[k] = x;
    C[k] = (C[k] + A[0] * (ll)B[k]) % mod;
    int z = 1;
    if (k == N - 1) return C[k];
    while ((k & (z - 1)) == (z - 1))
    {
      //mult B[k - z + 1 ... k] x A[z .. 2 * z - 1]
      forn(i, z) fft::A[i] = A[z + i];
      forn(i, z) fft::B[i] = B[k - z + 1 + i];
      fft::multMod(z, z, mod);
      forn(i, 2 * z - 1) C[k + 1 + i] = (C[k + 1 + i] + fft::C[i]) % mod;
      z <<= 1;
    }
    return C[k];
  }
  // A -- constant array
  // magic(k, x):: B[k] = x, returns C[k]
  // !! WARNING !! better to set N twice the size needed
}
```

musthave/halfplaneIntersection.cpp

```
int getPart(pt v) {
  return less(0, v.y) || (equal(0, v.y) && less(v.x, 0));
}

int cmpV(pt a, pt b) {
  int partA = getPart(a);
  int partB = getPart(b);
  if (partA < partB) return -1;
  if (partA > partB) return 1;
  if (equal(0, a * b)) return 0;
  if (0 < a * b) return -1;
  return 1;
}

double planeInt(vector<Line> l) {
  int n = l.size();
  sort(all(l), [](Line a, Line b) {
      int r = cmpV(a.v, b.v);
      if (r != 0) return r < 0;
      return a.O % a.v.rotate() < b.O % a.v.rotate();
    });

  int cur = 0;
  for (int i = 0; i < n; ) {
    int j = i;
    for (; i < n && cmpV(l[j].v, l[i].v) == 0 && cmpV(l[i].v, l[j].v) == 0; i++);
    l[cur++] = l[i - 1];
  }
  n = cur;

  for (int i = 0; i < n; i++)
    l[i].id = i;

  int flagUp = 0;
  int flagDown = 0;
  for (int i = 0; i < n; i++) {
    int part = getPart(l[i].v);
    if (part == 1) flagUp = 1;
    if (part == 0) flagDown = 1;
  }
  if (!flagUp || !flagDown) return -1;

  for (int i = 0; i < n; i++) {
    pt v = l[i].v;
    pt u = l[(i + 1) % n].v;
    if (equal(0, v * u) && less(v % u, 0)) {
      pt dir = l[i].v.rotate();
      if (lessE(l[(i + 1) % n].O % dir, l[i].O % dir)) return 0;
      return -1;
    }
    if (less(v * u, 0))
      return -1;
  }

  cur = 0;
  vector<Line> st(n * 2);
  for (int tt = 0; tt < 2; tt++) {
    for (int i = 0; i < n; i++) {
      for (; cur >= 2; cur--) {
        pt G = st[cur - 1] * l[i];
        if (!lessE(st[cur - 2].v * (G - st[cur - 2].O), 0))
          break;
      }
      st[cur++] = l[i];
      if (cur >= 2 && lessE(st[cur - 2].v * st[cur - 1].v, 0)) return 0;
    }
  }
  vector < int > use(n, -1);
  int left = -1, right = -1;
  for (int i = 0; i < cur; i++) {
    if (use[st[i].id] == -1) {
      use[st[i].id] = i;
    }
    else {
      left = use[st[i].id];
      right = i;
      break;
    }
  }
  vector < Line > tmp;
  for (int i = left; i < right; i++)
    tmp.pb(st[i]);
  vector < pt > res;
  for (int i = 0; i < (int)tmp.size(); i++)
    res.pb(tmp[i] * tmp[(i + 1) % tmp.size()]);
  double area = 0;
  for (int i = 0; i < (int)res.size(); i++)
    area += res[i] * res[(i + 1) % res.size()];
  return area / 2;
}
```

musthave/commonTangents.cpp

```


vector<Line> commonTangents(pt A, dbl rA, pt B, dbl rB) {
  vector<Line> res;
  pt C = B - A;
  dbl z = C.len2();
  for (int i = -1; i <= 1; i += 2) {
    for (int j = -1; j <= 1; j += 2) {
      dbl r = rB * j - rA * i;
      dbl d = z - r * r;
      if (ls(d, 0)) continue;
      d = sqrt(max(0.01, d));
      pt magic = pt(r, d) / z;
      pt v(magic % C, magic * C);
      dbl CC = (rA * i - v % A) / v.len2();
      pt O = v * -CC;
      res.pb(Line(O, O + v.rotate()));
    }
  }
  return res;
}

// HOW TO USE ::
// --    *D*----------------*F*
// --    *...* -        - *...*
// --   *.....* -      - *.....*
// --   *.......* -  - *.......*
// --   *...A...*  --  *...B...*
// --   *.......* -  - *.......*
// --   *.....* -      - *.....*
// --    *...* -        - *...*
// --    *C*----------------*E*
// -- res = {CE, CF, DE, DF}
```

musthave/minDisc.cpp

```

pair<pt, dbl> minDisc(vector<pt> p) {
  int n = p.size();
  pt O = pt(0, 0);
  dbl R = 0;
  random_shuffle(all(p));
  for (int i = 0; i < n; i++) {
    if (ls(R, (O - p[i]).len())) {
      O = p[i];
      R = 0;
      for (int j = 0; j < i; j++) {
        if (ls(R, (O - p[j]).len())) {
          O = (p[i] + p[j]) / 2;
          R = (p[i] - p[j]).len() / 2;
          for (int k = 0; k < j; k++) {
            if (ls(R, (O - p[k]).len())) {
              Line l1((p[i] + p[j]) / 2, (p[i] + p[j]) / 2 + (p[i] - p[j]).rotate());
              Line l2((p[k] + p[j]) / 2, (p[k] + p[j]) / 2 + (p[k] - p[j]).rotate());
              O = l1 * l2;
              R = (p[i] - O).len();
            }
          }
        }
      }
    }
  }
  return {O, R};
}
```

musthave/polygonArcCut.cpp

```

struct Meta {
  int type; // 0 - seg, 1 - circle
  pt O;
  dbl R;
};

const Meta SEG = {0, pt(0, 0), 0};

vector<pair<pt, Meta>> cut(vector<pair<pt, Meta>> p, Line l) {
  vector<pair<pt,Meta>> res;
  int n = p.size();
  for (int i = 0; i < n; i++) {
    pt A = p[i].F;
    pt B = p[(i + 1) % n].F;
    if (le(0, l.v * (A - l.O))) {
      if (eq(0, l.v * (A - l.O)) && p[i].S.type == 1 && ls(0, l.v % (p[i].S.O - A)))
        res.pb({A, SEG});
      else
        res.pb(p[i]);
    }
    if (p[i].S.type == 0) {
      if (sign(l.v * (A - l.O)) * sign(l.v * (B - l.O)) == -1) {
        pt FF = Line(A, B) * l;
        res.pb(make_pair(FF, SEG));
      }
    }
    else {
      pt E, F;
      if (intCL(p[i].S.O, p[i].S.R, l, E, F)) {
        if (onArc(p[i].S.O, A, E, B))
          res.pb({E, SEG});
        if (onArc(p[i].S.O, A, F, B))
          res.pb({F, p[i].S});
      }
    }
  }
  return res;
}
```

musthave/hashTable.cpp

```
template <const int max_size, class HashType, class Data, const Data default_value>
struct hashTable {
  HashType hash[max_size];
  Data f[max_size];
  int size;

  int position(HashType H ) const {
    int i = H % max_size;
    while (hash[i] && hash[i] != H)
      if (++i == max_size)
        i = 0;
    return i;
  }

  Data & operator [] (HashType H ) {
    assert(H != 0);
    int i = position(H);
    if (!hash[i]) {
      hash[i] = H;
      f[i] = default_value;
      size++;
    }
    return f[i];
  }
};

hashTable<13, int, int, 0> h;
```

musthave/hungary.cpp

```
namespace hungary
{
  const int N = 210;

  int a[N][N];
  int ans[N];

  int calc(int n, int m)
  {
    ++n, ++m;
    vi u(n), v(m), p(m), prev(m);
    for (int i = 1; i < n; ++i)
    {
      p[0] = i;
      int x = 0;
      vi mn(m, inf);
      vi was(m, 0);
      while (p[x])
      {
        was[x] = 1;
        int ii = p[x], dd = inf, y = 0;
        for (int j = 1; j < m; ++j) if (!was[j])
        {
          int cur = a[ii][j] - u[ii] - v[j];
          if (cur < mn[j]) mn[j] = cur, prev[j] = x;
          if (mn[j] < dd) dd = mn[j], y = j;
        }
        forn(j, m)
        {
          if (was[j]) u[p[j]] += dd, v[j] -= dd;
          else mn[j] -= dd;
        }
        x = y;
      }
      while (x)
      {
        int y = prev[x];
        p[x] = p[y];
        x = y;
      }
    }
    for (int j = 1; j < m; ++j)
    {
      ans[p[j]] = j;
    }
    return -v[0];
  }
  // HOW TO USE ::
  // -- set values to a[1..n][1..m] (n <= m)
  // -- run calc(n, m) to find MINIMUM
  // -- to restore permutation use ans[]
  // -- everything works on negative numbers
  //
  // !! i don't understand this code, it's copypasted from e-maxx (and rewritten by enot110)
}
```

musthave/modReverseOneLine.cpp

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

musthave/optimizations.cpp

```
// from anta code http://codeforces.com/contest/755/submission/23864531

#pragma GCC optimize ("O3")
#pragma GCC target ("sse4")
inline void fasterLLDivMod(unsigned long long x, unsigned y, unsigned &out_d, unsigned &out_m) {
  unsigned xh = (unsigned)(x >> 32), xl = (unsigned)x, d, m;
#ifdef __GNUC__
  asm(
    "divl %4; \n\t"
    : "=a" (d), "=d" (m)
    : "d" (xh), "a" (xl), "r" (y)
  );
#else
  __asm {
    mov edx, dword ptr[xh];
    mov eax, dword ptr[xl];
    div dword ptr[y];
    mov dword ptr[d], eax;
    mov dword ptr[m], edx;
  };
#endif
  out_d = d; out_m = m;
}


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

musthave/plane3DInt.cpp

```
//(A, v) * (B, u) -> (O, n)

pt n = v * u;
pt m = v * n;
double t = (B - A) % u / (u % m);
pt O = A - m * t;
```

musthave/simplex.cpp

```

namespace simplex {
  const int MAX_N = -1; // number of variables
  const int MAX_M = -1; // number of inequalities
  dbl a[MAX_M][MAX_N];
  dbl b[MAX_M];
  dbl c[MAX_N];
  dbl v;
  ll n, m;
  int left[MAX_M];
  int up[MAX_N];
  int pos[MAX_N];
  dbl res[MAX_N];

  void init(int nn, int mm) {
    n = nn;
    m = mm;
    v = 0;
    for (int i = 0; i < m; i++)
      for (int j = 0; j < n; j++)
        a[i][j] = 0;
    for (int i = 0; i < m; i++)
      b[i] = 0;
    for (int i = 0; i < n; i++)
      c[i] = 0;
  }

  void pivot(int x, int y) {
    swap(left[x], up[y]);
    dbl k = a[x][y];
    a[x][y] = 1;
    b[x] /= k;
    int cur = 0;
    for (int i = 0; i < n; i++) {
      a[x][i] = a[x][i] / k;
      if (!eq(a[x][i], 0))
        pos[cur++] = i;
    }

    for (int i = 0; i < m; i++) {
      if (i == x || eq(a[i][y], 0)) continue;
      dbl cof = a[i][y];
      b[i] -= cof * b[x];
      a[i][y] = 0;
      for (int j = 0; j < cur; j++)
        a[i][pos[j]] -= cof * a[x][pos[j]];
    }
    dbl cof = c[y];
    v += cof * b[x];
    c[y] = 0;
    for (int i = 0; i < cur; i++) {
      c[pos[i]] -= cof * a[x][pos[i]];
    }
  }

  void solve() {
    for (int i = 0; i < n; i++)
      up[i] = i;
    for (int i = 0; i < m; i++)
      left[i] = i + n;

    while (1) {
      int x = -1;
      for (int i = 0; i < m; i++)
        if (ls(b[i], 0) && (x == -1 || b[i] < b[x])) {
          x = i;
        }
      if (x == -1) break;
      int y = -1;
      for (int j = 0; j < n; j++)
        if (ls(a[x][j], 0)) {
          y = j;
          break;
        }
      if (y == -1) {
        assert(false); // no solution
      }
      pivot(x, y);
    }
    while (1) {
      int y = -1;
      for (int i = 0; i < n; i++)
        if (ls(0, c[i]) && (y == -1 || (c[i] > c[y]))) {
          y = i;
        }
      if (y == -1) break;

      int x = -1;
      for (int i = 0; i < m; i++) {
        if (ls(0, a[i][y])) {
          if (x == -1 || (b[i] / a[i][y] < b[x] / a[x][y])) {
            x = i;
          }
        }
      }
      if (y == -1) {
        assert(false); // infinite solution
      }
      pivot(x, y);
    }

    memset(res, 0, sizeof(res));

    for (int i = 0; i < m; i++) {
      if (left[i] < n) {
        res[left[i]] = b[i];
      }
    }
  }
  // HOW TO USE ::
  // -- call init(n, m)
  // -- call solve()
  // -- variables in "up" equals to zero
  // -- variables in "left" equals to b
  // -- max: c * x
  // -- b[i] >= a[i] * x
  // -- answer in "v"
  // -- sertificate in "res"
};
```

musthave/std-rb-tree.cpp

```
#include "ext/pb_ds/assoc_container.hpp"
using namespace __gnu_pbds;

template <typename T> using ordered_set = tree<T, null_type, less<T>, rb_tree_tag, tree_order_statistics_node_update>;
template <typename K, typename V> using ordered_map = tree<K, V, less<K>, rb_tree_tag, tree_order_statistics_node_update>;

```

```
// HOW TO USE ::
// -- order_of_key(10) returns the number of elements in set/map strictly less than 10
// -- *find_by_order(10) returns 10-th smallest element in set/map (0-based)
```

musthave/sufAutomaton.cpp

```
namespace SA {
  const int MAXN = 1 << 18;
  const int SIGMA = 26;

  int sz, last;
  int nxt[MAXN][SIGMA];
  int link[MAXN], len[MAXN], pos[MAXN];

  void init() {
    memset(nxt, -1, sizeof(nxt));
    memset(link, -1, sizeof(link));
    memset(len, 0, sizeof(len));
    last = 0;
    sz = 1;
  }

  void add(int c) {
    int cur = sz++;
    len[cur] = len[last] + 1;
    pos[cur] = len[cur];
    int p = last;
    last = cur;
    for (; p != -1 && nxt[p][c] == -1; p = link[p]) nxt[p][c] = cur;
    if (p == -1) {
      link[cur] = 0;
      return;
    }
    int q = nxt[p][c];
    if (len[p] + 1 == len[q]) {
      link[cur] = q;
      return;
    }
    int clone = sz++;
    memcpy(nxt[clone], nxt[q], sizeof(nxt[q]));
    len[clone] = len[p] + 1;
    pos[clone] = pos[q];
    link[clone] = link[q];
    link[q] = link[cur] = clone;
    for (; p != -1 && nxt[p][c] == q; p = link[p]) nxt[p][c] = clone;
  }

  int n;
  string s;
  int l[MAXN], r[MAXN];
  int e[MAXN][SIGMA];

  void getSufTree(string _s) {
    memset(e, -1, sizeof(e));
    s = _s;
    n = s.length();
    reverse(s.begin(), s.end());
    init();
    for (int i = 0; i < n; i++) add(s[i] - 'a');
    reverse(s.begin(), s.end());
    for (int i = 1; i < sz; i++) {
      int j = link[i];
      l[i] = n - pos[i] + len[j];
      r[i] = n - pos[i] + len[i];
      e[j][s[l[i]] - 'a'] = i;
    }
  }
}
```

musthave/generalMatching.cpp

```
//COPYPASTED FROM E-MAXX
namespace GeneralMatching {
  const int MAXN = 256;
  int n;
  vector<int> g[MAXN];
  int match[MAXN], p[MAXN], base[MAXN], q[MAXN];
  bool used[MAXN], blossom[MAXN];

  int lca (int a, int b) {
    bool used[MAXN] = { 0 };
    for (;;) {
      a = base[a];
      used[a] = true;
      if (match[a] == -1) break;
      a = p[match[a]];
    }
    for (;;) {
      b = base[b];
      if (used[b]) return b;
      b = p[match[b]];
    }
  }

  void mark_path (int v, int b, int children) {
    while (base[v] != b) {
      blossom[base[v]] = blossom[base[match[v]]] = true;
      p[v] = children;
      children = match[v];
      v = p[match[v]];
    }
  }

  int find_path (int root) {
    memset (used, 0, sizeof used);
    memset (p, -1, sizeof p);
    for (int i=0; i<n; ++i)
      base[i] = i;

    used[root] = true;
    int qh=0, qt=0;
    q[qt++] = root;
    while (qh < qt) {
      int v = q[qh++];
      for (size_t i=0; i<g[v].size(); ++i) {
        int to = g[v][i];
        if (base[v] == base[to] || match[v] == to) continue;
        if (to == root || (match[to] != -1 && p[match[to]] != -1)) {
          int curbase = lca (v, to);
          memset (blossom, 0, sizeof blossom);
          mark_path (v, curbase, to);
          mark_path (to, curbase, v);
          for (int i=0; i<n; ++i)
            if (blossom[base[i]]) {
              base[i] = curbase;
              if (!used[i]) {
                used[i] = true;
                q[qt++] = i;
              }
            }
        }
        else if (p[to] == -1) {
          p[to] = v;
          if (match[to] == -1)
            return to;
          to = match[to];
          used[to] = true;
          q[qt++] = to;
        }
      }
    }
    return -1;
  }

  vector<pair<int, int> > solve(int _n, vector<pair<int, int> > edges) {
    n = _n;
    for (int i = 0; i < n; i++) g[i].clear();
    for (auto o : edges) {
      g[o.first].push_back(o.second);
      g[o.second].push_back(o.first);
    }
    memset (match, -1, sizeof match);
    for (int i=0; i<n; ++i) {
      if (match[i] == -1) {
        int v = find_path (i);
        while (v != -1) {
          int pv = p[v], ppv = match[pv];
          match[v] = pv, match[pv] = v;
          v = ppv;
        }
      }
    }
    vector<pair<int, int> > ans;
    for (int i = 0; i < n; i++) {
      if (match[i] > i) {
        ans.push_back(make_pair(i, match[i]));
      }
    }
    return ans;
  }
}
```

useful/dinica.cpp

```
namespace flow
{
  const int maxn = 1e5 + 10;
  const int maxe = 2 * maxn;

  int head[maxn], next[maxe], to[maxe], f[maxe], ec = 1;
  int ST, EN, N = maxn;

  inline void setN(int n)
  {
    ST = n;
    EN = n + 1;
    N = n + 2;
  }

  inline void _add(int x, int y, int ff)
  {
    ++ec;
    to[ec] = y;
    next[ec] = head[x];
    head[x] = ec;
    f[ec] = ff;
  }

  inline int add(int x, int y, int ff)
  {
    _add(x, y, ff);
    _add(y, x, 0);
    return ec - 1;
  }

  void clear()
  {
    forn(i, N) head[i] = 0;
    ec = 1;
  }

  int d[maxn];
  int q[maxn], st = 0, en = 0;

  int bfs()
  {
    forn(i, N) d[i] = 1e9;
    st = 0, en = 0;
    d[ST] = 0;
    q[en++] = ST;
    while (st < en)
    {
      int x = q[st++];
      if (x == EN) return 1;
      for (int e = head[x]; e; e = next[e])
      {
        int y = to[e];
        if (d[y] == 1e9 && f[e])
        {
          d[y] = d[x] + 1;
          q[en++] = y;
        }
      }
    }
    return 0;
  }

  int pushed;
  int fst[maxn];

  int dfs(int x, int flow = 1e9)
  {
    if (x == EN)
    {
      pushed = flow;
      return 1;
    }
    for (; fst[x]; fst[x] = next[fst[x]])
    {
      int e = fst[x];
      int y = to[e];
      if (d[y] == d[x] + 1 && f[e] && dfs(y, min(f[e], flow)))
      {
        f[e] -= pushed;
        f[e ^ 1] += pushed;
        return 1;
      }
    }
    return 0;
  }


  ll calcFlow()
  {
    ll res = 0;
    while (bfs())
    {
      forn(i, N) fst[i] = head[i];
      while (dfs(ST))
      {
        res += pushed;
      }
    }
    return res;
  }

  // HOW TO USE ::
  // -- set maxn and maxe (special for izban)
  // -- add adges using add(x, y, f), call setN(n)
  // -- run calcFlow
}
```

useful/max-flow-min-cost.cpp

```
namespace flow
{
  const int maxn = 2e5 + 10;
  const int maxe = 2 * maxn;

  int head[maxn], next[maxe], to[maxe], flow[maxe], cost[maxe], ec = 1;
  int ST, EN, N = maxn;

  inline void setN(int n)
  {
    ST = n;
    EN = n + 1;
    N = n + 2;
  }

  inline void _add(int x, int y, int f, int c)
  {
    ++ec;
    to[ec] = y;
    next[ec] = head[x];
    head[x] = ec;
    flow[ec] = f;
    cost[ec] = c;
  }

  inline int add(int x, int y, int f, int c)
  {
    _add(x, y, f, c);
    _add(y, x, 0, -c);
    return ec - 1;
  }

  void clear()
  {
    forn(i, N) head[i] = 0;
    ec = 1;
  }

  ll d[maxn], p[maxn];
  int last[maxn];
  int used[maxn];

  pair<ll, ll> _calc(int flag)
  {
    const ll INF = 1e12;
    forn(i, N) p[i] = INF;
    p[ST] = 0;
    forn(_, N) forn(x, N) for (int e = head[x]; e; e = next[e]) if (flow[e] > 0)
    {
      int y = to[e];
      if (p[y] > p[x] + cost[e])
      {
        p[y] = p[x] + cost[e];
      }
    }

    ll resFlow = 0, resCost = 0;
    while (1)
    {
      forn(i, N) d[i] = INF, used[i] = 0;
      d[ST] = 0;
      forn(_, N)
      {
        int x = -1;
        forn(i, N) if (!used[i] && (x == -1 || d[x] > d[i])) x = i;
        used[x] = 1;
        if (d[x] == INF) break;
        for (int e = head[x]; e; e = next[e]) if (flow[e] > 0)
        {
          int y = to[e];
          ll len = cost[e] + p[x] - p[y];
          if (d[y] > d[x] + len)
          {
            d[y] = d[x] + len;
            last[y] = e;
          }
        }
      }

      if (d[EN] == INF) break;

      ll realCost = d[EN] + p[EN] - p[ST];
      if (flag && realCost > 0) break;

      int pushed = inf;
      int x = EN;
      while (x != ST)
      {
        int e = last[x];
        pushed = min(pushed, flow[e]);
        x = to[e ^ 1];
      }

      resCost += realCost * pushed;
      resFlow += pushed;

      x = EN;
      while (x != ST)
      {
        int e = last[x];
        flow[e] -= pushed;
        flow[e ^ 1] += pushed;
        x = to[e ^ 1];
      }

      forn(i, N) p[i] += d[i];
    }
    return mp(resFlow, resCost);
  }

  pair<ll, ll> maxFlow()
  {
    return _calc(0);
  }

  pair<ll, ll> minCost()
  {
    return _calc(1);
  }

  // HOW TO USE::
  // -- add adges using add(x, y, f, c), call setN(n)
  // -- run maxFlow/minCost, returns pair(flow, cost)
}
```

useful/poly.cpp

```
struct poly
{
  vi v;
  poly() {}
  poly(vi vv)
  {
    v = vv;
  }
  int size()
  {
    return (int)v.size();
  }
  poly cut(int maxLen)
  {
    if (maxLen < sz(v)) v.resize(maxLen);
    return *this;
  }
  poly norm()
  {
    while (sz(v) > 1 && v.back() == 0) v.pop_back();
    return *this;
  }
  inline int& operator [] (int i)
  {
    return v[i];
  }
  void out(string name="")
  {
    stringstream ss;
    if (sz(name)) ss << name << "=";
    int fst = 1;
    forn(i, sz(v)) if (v[i])
    {
      int x = v[i];
      int sgn = 1;
      if (x > mod / 2) x = mod-x, sgn = -1;
      if (sgn == -1) ss << "-";
      else if (!fst) ss << "+";
      fst = 0;
      if (!i || x != 1)
      {
        ss << x;
        if (i > 0) ss << "*x";
        if (i > 1) ss << "^" << i;
      }
      else
      {
        ss << "x";
        if (i > 1) ss << "^" << i;
      }
    }
    if (fst) ss <<"0";
    string s;
    ss >> s;
    eprintf("%s\n", s.data());
  }
};

poly operator + (poly A, poly B)
{
  poly C;
  C.v = vi(max(sz(A), sz(B)));
  forn(i, sz(C))
  {
    if (i < sz(A)) C[i] = (C[i] + A[i]) % mod;
    if (i < sz(B)) C[i] = (C[i] + B[i]) % mod;
  }
  return C.norm();
}

poly operator - (poly A, poly B)
{
  poly C;
  C.v = vi(max(sz(A), sz(B)));
  forn(i, sz(C))
  {
    if (i < sz(A)) C[i] = (C[i] + A[i]) % mod;
    if (i < sz(B)) C[i] = (C[i] + mod - B[i]) % mod;
  }
  return C.norm();
}

poly operator * (poly A, poly B)
{
  poly C;
  C.v = vi(sz(A) + sz(B) - 1);

  forn(i, sz(A)) fft::A[i] = A[i];
  forn(i, sz(B)) fft::B[i] = B[i];
  fft::multMod(sz(A), sz(B), mod);
  forn(i, sz(C)) C[i] = fft::C[i];
  return C.norm();
}

poly inv(poly A, int n) // returns A^-1 mod x^n
{
  assert(sz(A) && A[0] != 0);
  A.cut(n);

  auto cutPoly = [](poly &from, int l, int r)
```

```cpp
  {
    poly R;
    R.v.resize(r - 1);
    for (int i = 1; i < r; ++i)
    {
      if (i < sz(from)) R[i - 1] = from[i];
    }
    return R;
  };

  function<int(int, int)> rev = [&rev](int x, int m)->int
  {
    if (x == 1) return 1;
    return (1 - rev(m % x, x) * (1ll)m) / x + m;
  };

  poly R({rev(A[0], mod)});
  for (int k = 1; k < n; k <<= 1)
  {
    poly A0 = cutPoly(A, 0, k);
    poly A1 = cutPoly(A, k, 2 * k);
    poly H = A0 * R;
    H = cutPoly(H, k, 2 * k);
    poly R1 = (((A1 * R).cut(k) + H) * (poly({0}) - R)).cut(k);
    R.v.resize(2 * k);
    forn(i, k) R[i + k] = R1[i];
  }
  return R.cut(n).norm();
}

pair<poly, poly> divide(poly A, poly B)
{
  if (sz(A) < sz(B)) return {poly({0}), A};

  auto rev = [](poly f)
  {
    reverse(all(f.v));
    return f;
  };

  poly q = rev((inv(rev(B), sz(A) - sz(B) + 1) * rev(A)).cut(sz(A) - sz(B) + 1));
  poly r = A - B * q;

  return {q, r};
}
```

useful/primes.cpp

```cpp
namespace math
{
  const int maxP = 1e6;
  int pp[maxP];
  int p[maxP / 10 + 1000], pc = 0;

  void gen_primes()
  {
    pp[0] = pp[1] = -1;
    for (int i = 2; i < maxP; ++i) pp[i] = i;
    for (int i = 2; i < maxP; ++i)
    {
      if (pp[i] == i) p[pc++] = i;
      for (int j = 0; j < pc && p[j] <= pp[i] && i * p[j] < maxP; ++j) pp[i * p[j]] = p[j];
    }
  }
  bool is_prime(int x)
  {
    if (x < maxP) return pp[x] == x;
    for (int i = 0; p[i] * p[i] <= x; ++i) if (x % p[i] == 0) return false;
    return true;
  }
  // HOW TO USE ::
  // pp[x] <-- smallest prime divisor {x} (or -1 for {x < 2})
  // p[0 .. pc - 1] <-- list of primes < maxP
}
```

retro

```cpp
namespace retro
{
  const int N = 4e5 + 10;

  vi v[N];
  vi vrev[N];

  void add(int x, int y)
  {
    v[x].pb(y);
    vrev[y].pb(x);
  }

  const int UD = 0;
  const int WIN = 1;
  const int LOSE = 2;

  int res[N];
  int moves[N];
  int deg[N];
  int q[N], st, en;

  void calc(int n)
  {
    forn(i, n) deg[i] = sz(v[i]);
    st = en = 0;
    forn(i, n) if (!deg[i])
    {
      q[en++] = i;
      res[i] = LOSE;
    }
    while (st < en)
    {
      int x = q[st++];
      for (int y : vrev[x])
      {
        if (res[y] == UD && (res[x] == LOSE || (--deg[y] == 0 && res[x] == WIN)))
        {
          res[y] = 3 - res[x];
          moves[y] = moves[x] + 1;
          q[en++] = y;
        }
      }
    }
  }
}
```