Contents

Mobius Inversion:

$$g(n) = \sum_{d|n} f(d) \Leftrightarrow f(n) = \sum_{d|n} \mu(d) g(n/d)$$

Other useful formulas/forms:

$\sum_{d|n} \mu(d) = [n = 1]$ (very useful)

$g(n) = \sum_{n|d} f(d) \Leftrightarrow f(n) = \sum_{n|d} \mu(d/n) g(d)$

$g(n) = \sum_{1 \le m \le n} f(\lfloor \frac{n}{m} \rfloor) \Leftrightarrow f(n) = \sum_{1 \le m \le n} \mu(m) g(\lfloor \frac{n}{m} \rfloor)$

Number of ways of writing n as a sum of positive integers, disregarding the order of the summands.

$$p(0) = 1,\ p(n) = \sum_{k \in Z \setminus \{0\}} (-1)^{k+1} p(n - k(3k - 1)/2)$$

$$p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$$

n	0	1	2	3	4	5	6	7	8	9	20	50	100
$p(n)$	1	1	2	3	5	7	11	15	22	30	627	∼2e5	∼2e8

on n vertices: n^{n-2}

on k existing trees of size n_i: $n_1 n_2 \cdots n_k n^{k-2}$

with degrees d_i: $(n-2)!/((d_1-1)!\cdots(d_n-1)!)$

$$B(p^m+n) \equiv mB(n)+B(n+1) \pmod{p}$$

1 Data Structure

1.1 centroid

```
struct Graph {
  vector<vector<int>> adj;
  Graph(int n) : adj(n + 1) {}
  void add_edge(int a, int b, bool
      directed = false) {
    adj[a].pb(b);
    if (!directed) adj[b].pb(a);
  }
};

struct Centroid {
  vector<int> stree, parent;
  void _dfs(vector<vector<int>> &adj,
      ll x, ll par = -1) {
    stree[x] = 1, parent[x] = par;
    for (auto &p : adj[x]) {
      if (p != par) {
        _dfs(adj, p, x);
        stree[x] += stree[p];
      }
    }
  }
  int decompose(Graph &G, Graph &cd,
      ll root = 1) {
    int n = G.adj.size() - 1;
    stree.resize(n + 1);
    parent.resize(n + 1);
    _dfs(G.adj, root);
    vector<bool> done(n + 1);
    return construct(G, cd, done, root)
        ;
  }
  int construct(Graph &G, Graph &cd,
      vector<bool> &done, ll root) {
    while (true) {
      ll maxm = 0, ind = -1;
      for (auto &x : G.adj[root]) {
        if (!done[x] && stree[x] > maxm)
            {
          maxm = stree[x];
          ind = x;
        }
      }
      if (maxm <= stree[root] / 2) {
        done[root] = true;
        for (auto &p : G.adj[root]) {
          if (!done[p]) {
            ll x = construct(G, cd, done,
                p);
            cd.add_edge(x, root);
            // root is parent of x is
                centroid tree
            // cd.parent[x] = root;
          }
        }
        return root;
      } else {
        ll temp = stree[root];
        stree[root] -= stree[ind];
        stree[ind] = temp;
        root = ind;
      }
    }
  }
};
```

1.2 DSU on tree

```
int cnt[maxn];
void dfs(int v, int p, bool keep) {
  int mx = -1, bigChild = -1;
  for (auto u : g[v])
    if (u != p && sz[u] > mx) mx = sz[u
        ], bigChild = u;
  for (auto u : g[v])
    if (u != p && u != bigChild)
      dfs(u, v, 0);
  if (bigChild != -1)
    dfs(bigChild, v, 1);
  for (auto u : g[v])
    if (u != p && u != bigChild)
      for (int p = st[u]; p < ft[u]; p
          ++) cnt[col[ver[p]]]++;
  cnt[col[v]]++;
  if (keep == 0)
    for (int p = st[v]; p < ft[v]; p++)
        cnt[col[ver[p]]]--;
}
```

1.3 hld

```
struct HLD {
  vector<int> sz, tin, tout, nxt,
      order, level, pars;
  int timer;
  // SegTree ST;
  void dfs(vector<vector<int>> &adj,
      int x, int par = -1) {
    sz[x] = 1;
    pars[x] = par;
    for (auto &p : adj[x])
      if (p != par) {
        level[p] = level[x] + 1;
        dfs(adj, p, x);
        if (adj[x][0] == par || sz[p] >
            sz[adj[x][0]]) swap(p, adj[x
```

```
        ][0]);
    }
}
void dfs2(vector<vector<int>> &adj,
    int x) {
  tin[x] = timer++;
  order.push_back(x);
  for (auto &p : adj[x]) {
    if (p == pars[x]) continue;
    nxt[p] = (p == adj[x][0] ? nxt[x]
         : p);
    dfs2(adj, p);
  }
  tout[x] = timer;
}
HLD(vector<vector<int>> &adj, int N,
    int root = 1)
    : sz(N + 5),
      tin(N + 5),
      tout(N + 5),
      nxt(N + 5),
      level(N + 5),
      pars(N + 5),
      timer(0) {
  int n = adj.size() - 1;
  level[root] = 0;
  dfs(adj, root);
  dfs2(adj, root);
  // build segment tree on "order"
      here
  // ST.resize(order.size());
  // ST.build(0, 0, order.size()-1,
      order);
}
int path_query(int a, int b) {
  int N = order.size();
  // int answer = -INFINT;
  while (nxt[a] != nxt[b]) {
    if (level[nxt[a]] < level[nxt[b
        ]]) swap(a, b);
    // answer = max(answer, ST.
        range_query(0, 0, N-1, tin[nxt
        [a]], tin[a]));
    a = pars[nxt[a]];
  }
  if (tin[a] > tin[b]) swap(a, b);
  // answer = max(answer, ST.
      range_query(0, 0, N-1, tin[a],
      tin[b]));
  return answer;
}
void point_update(int x, int val) {
  // ST.point_update(0, 0, order.size
      ()-1, tin[x], val);
}
};
```

1.4 Roll back

```
/**If undo is not needed, skip st,
   time() and rollback().
 * Usage: int t = uf.time(); ...; uf.
    rollback(t);
 * Time: $O(\log(N))$*/
struct RollbackUF {
  vi e;
  vector<pii> st;
  RollbackUF(int n) : e(n, -1) {}
  int size(int x) { return -e[find(x)
      ]; }
  int find(int x) { return e[x] < 0 ?
      x : find(e[x]); }
  int time() { return sz(st); }
  void rollback(int t) {
    for (int i = time(); i-- > t;) e[st
        [i].first] = st[i].second;
    st.resize(t);
  }
  bool join(int a, int b) {
    a = find(a), b = find(b);
    if (a == b) return false;
    if (e[a] > e[b]) swap(a, b);
    st.push_back({a, e[a]});
    st.push_back({b, e[b]});
    e[a] += e[b];
    e[b] = a;
    return true;
  }
};
```

2 dp opti

2.1 1D-1D(convex)

```
// Monge condition : a<bc<d,
// Convex Monge condition : f(a,c)+f(b
    ,d)f(a,d)+f(b,c)
// Concave Monge condition : f(a,c)+f(
    b,d)f(a,d)+f(b,c)
// Totally monotone :  a<bc<d,
// Convex totally monotone : f(a,c)f(b
    ,c)f(a,d)f(b,d)
// Concave totally monotone : f(a,c)f(
    b,c)f(a,d)f(b,d)
// Usually f(i,j) is something like
    dpi+cost(i+1,j) or cost(i,j).

struct Node {
  ll p, l, r; // p is the best
      transition point for dp[l], dp[l
      +1], ..., dp[r]
};
```

```
deque<Node> dq;
dp[0] = 0;
dq.push_back({0, 1, n});
for (int i = 1; i <= n; ++i) {
  dp[i] = f(dq.front().p, i)
        // r == i implies that this
           Node is useless later, so
           pop it
        if (dq.front().r == i) dq.
           pop_front();
  // else update l
  else dq.front().l++;

  // find l, r for i
  // f(i, dq.back().l) < f(dq.back().p
     , dq.back().l) implies the last
     Node in
  // deque is useless
  while (!dq.empty() && f(i, dq.back()
     .l) < f(dq.back().p, dq.back().l)
     )
    dq.pop_back();
  // we know that r=n, now we need to
     find l
  // l=i+1 as deque is empty
  if (dq.empty()) dq.push_back({i, i +
      1, n});
  // find l by binary search
  else {
    int l = dq.back().l, r = dq.back().
       r;
    while (l < r) {
      int mid = r - (r - l) / 2;
      if (f(i, mid) < f(dq.back().p,
         mid))
        r = mid - 1;
      else
        l = mid;
    }
    dq.back().r = l;
    // l == n means that i is useless
    if (l != n) dq.push_back({i, l + 1,
        n});
  }
}
```

2.2 CHT dynamic

```
// * Description: Container where you
   can add lines of the form kx+m,
   and query
// maximum values at points x.
#pragma once

struct Line {
  mutable ll k, m, p;
  bool operator<(const Line& o) const
     { return k < o.k; }
  bool operator<(ll x) const { return
     p < x; }
};

struct LineContainer : multiset<Line,
    less<>> {
  // (for doubles, use inf = 1/.0, div
     (a,b) = a/b)
  static const ll inf = LLONG_MAX;
  ll div(ll a, ll b) { // floored
     division
    return a / b - ((a ^ b) < 0 && a %
       b);
  }
  bool isect(iterator x, iterator y) {
    if (y == end()) return x->p = inf,
       0;
    if (x->k == y->k)
      x->p = x->m > y->m ? inf : -inf;
    else
      x->p = div(y->m - x->m, x->k - y
         ->k);
    return x->p >= y->p;
  }
  void add(ll k, ll m) {
    auto z = insert({k, m, 0}), y = z
       ++, x = y;
    while (isect(y, z)) z = erase(z);
    if (x != begin() && isect(--x, y))
       isect(x, y = erase(y));
    while ((y = x) != begin() && (--x)
       ->p >= y->p) isect(x, erase(y))
       ;
  }
  ll query(ll x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
  }
};
```

2.3 CHT Normal

```
vector<point> hull, vecs;

void add_line(ftype k, ftype b) {
  point nw = {k, b};
  while (!vecs.empty() && dot(vecs.
     back(), nw - hull.back()) < 0) {
    hull.pop_back();
    vecs.pop_back();
  }
  if (!hull.empty()) {
    vecs.push_back(1i * (nw - hull.back
       ()));
  }
  hull.push_back(nw);
```

```
}
int get(ftype x) {
  point query = {x, 1};
  auto it = lower_bound(vecs.begin(),
      vecs.end(), query,
                    [](point a, point b)
                        { return cross(
                       a, b) > 0; });
  return dot(query, hull[it - vecs.
      begin()]);
}
```

2.4 Li Chao

```
typedef long long ftype;
typedef complex<ftype> point;
#define x real
#define y imag

ftype dot(point a, point b) { return (
    conj(a) * b).x(); }

ftype f(point a, ftype x) { return dot
    (a, {x, 1}); }
const int maxn = 2e5;

point line[4 * maxn];

void add_line(point nw, int v = 1, int
     l = 0, int r = maxn) {
  int m = (l + r) / 2;
  bool lef = f(nw, l) < f(line[v], l);
  bool mid = f(nw, m) < f(line[v], m);
  if (mid) {
    swap(line[v], nw);
  }
  if (r - l == 1) {
    return;
  } else if (lef != mid) {
    add_line(nw, 2 * v, l, m);
  } else {
    add_line(nw, 2 * v + 1, m, r);
  }
}

ftype get(int x, int v = 1, int l = 0,
     int r = maxn) {
  int m = (l + r) / 2;
  if (r - l == 1) {
    return f(line[v], x);
  } else if (x < m) {
    return min(f(line[v], x), get(x, 2
        * v, l, m));
  } else {
    return min(f(line[v], x), get(x, 2
        * v + 1, m, r));
  }
}
```

2.5 sos-dp

```
for (int i = 0; i < (1 << N); ++i) F[i
    ] = A[i];
for (int i = 0; i < N; ++i)
  for (int mask = 0; mask < (1 << N);
      ++mask) {
    if (mask & (1 << i)) F[mask] += F[
        mask ^ (1 << i)];}
```

3 FFT

3.1 fft

```
using cd = complex<double>;
const double PI = acos(-1);
void fft(vector<cd> &a, bool invert) {
  int n = a.size();
  for (int i = 1, j = 0; i < n; i++) {
    int bit = n >> 1;
    for (; j & bit; bit >>= 1) j ^= bit
        ;
    j ^= bit;
    if (i < j) swap(a[i], a[j]);
  }

  for (int len = 2; len <= n; len <<=
      1) {
    double ang = 2 * PI / len * (invert
        ? -1 : 1);
    cd wlen(cos(ang), sin(ang));
    for (int i = 0; i < n; i += len) {
      cd w(1);
      for (int j = 0; j < len / 2; j++)
          {
        cd u = a[i + j], v = a[i + j +
            len / 2] * w;
        a[i + j] = u + v;
        a[i + j + len / 2] = u - v;
        w *= wlen;
      }
    }
  }
  if (invert) {
    for (cd &x : a) x /= n;
  }
}

vector<int> multiply(vector<int> const
    &a, vector<int> const &b) {
  vector<cd> fa(a.begin(), a.end()),
      fb(b.begin(), b.end());
  int n = 1;
  while (n < a.size() + b.size()) n
      <<= 1;
```

```
  fa.resize(n);
  fb.resize(n);
  fft(fa, false);
  fft(fb, false);
  for (int i = 0; i < n; i++) fa[i] *=
      fb[i];
  fft(fa, true);
  vector<int> result(n);
  for (int i = 0; i < n; i++) result[i
     ] = round(fa[i].real());
  return result;}
```

3.2 ntt

```
const int mod = 7340033;
const int root = 5;
const int root_1 = 4404020;
const int root_pw = 1 << 20;

const int mod = 998244353;
const int root = 3;
const int root_1 = 332748118;
const int root_pw = 1 << 23;
const int root = generator(mod);
const int root_1 = mod_inv(root, mod);

void fft(vector<int>& a, bool invert)
    {
  for (int len = 2; len <= n; len <<=
      1) {
    int wlen = invert ? root_1 : root;
    for (int i = len; i < root_pw; i
        <<= 1)
      wlen = (int)(1LL * wlen * wlen %
          mod);

    for (int i = 0; i < n; i += len) {
      int w = 1;
      for (int j = 0; j < len / 2; j++)
          {
        int u = a[i + j], v = (int)(1LL
            * a[i + j + len / 2] * w %
            mod);
        a[i + j] = u + v < mod ? u + v :
            u + v - mod;
        a[i + j + len / 2] = u - v >= 0
            ? u - v : u - v + mod;
        w = (int)(1LL * w * wlen % mod);
      }
    }
  }

  if (invert) {
    int n_1 = inverse(n, mod);
    for (int& x : a) x = (int)(1LL * x
        * n_1 % mod);
  }
}
```

3.3 polynomial

```
namespace algebra {
const int inf = 1e9;
const int magic = 500; // threshold
    for sizes to run the naive algo
namespace fft {
const int maxn = 1 << 18;
typedef double ftype;
typedef complex<ftype> point;
const ftype pi = acos(-1);
template <typename T>
void mul(vector<T> &a, const vector<T>
    &b) {
  static const int shift = 15, mask =
      (1 << shift) - 1;
  size_t n = a.size() + b.size() - 1;
  while (__builtin_popcount(n) != 1) {
    n++;
  }
  a.resize(n);
  for (size_t i = 0; i < n; i++) {
    A[i] = point(a[i] & mask, a[i] >>
        shift);
    if (i < b.size()) {
      B[i] = point(b[i] & mask, b[i] >>
          shift);
    } else {
      B[i] = 0;
    }
  }
  fft(A, C, n);
  fft(B, D, n);
  for (size_t i = 0; i < n; i++) {
    point c0 = C[i] + conj(C[(n - i) %
        n]);
    point c1 = C[i] - conj(C[(n - i) %
        n]);
    point d0 = D[i] + conj(D[(n - i) %
        n]);
    point d1 = D[i] - conj(D[(n - i) %
        n]);
    A[i] = c0 * d0 - point(0, 1) * c1 *
        d1;
    B[i] = c0 * d1 + d0 * c1;
  }
  fft(A, C, n);
  fft(B, D, n);
  reverse(C + 1, C + n);
  reverse(D + 1, D + n);
  int t = 4 * n;
  for (size_t i = 0; i < n; i++) {
    int64_t A0 = llround(real(C[i]) / t
        );
    T A1 = llround(imag(D[i]) / t);
```

```
    T A2 = llround(imag(C[i]) / t);
    a[i] = A0 + (A1 << shift) + (A2 <<
        2 * shift);
  }
  return;
}
} // namespace fft
template <typename T>
struct poly {
  poly inv(size_t n) const { // get
      inverse series mod x^n
    assert(!is_zero());
    poly ans = a[0].inv();
    size_t a = 1;
    while (a < n) {
      poly C = (ans * mod_xk(2 * a)).
          substr(a, 2 * a);
      ans -= (ans * C).mod_xk(a).mul_xk
          (a);
      a *= 2;
    }
    return ans.mod_xk(n);
  }
  pair<poly, poly> divmod_slow(
      const poly &b) const { // when
          divisor or quotient is small
    vector<T> A(a);
    vector<T> res;
    while (A.size() >= b.a.size()) {
      res.push_back(A.back() / b.a.back
          ());
      if (res.back() != T(0)) {
        for (size_t i = 0; i < b.a.size
            (); i++) {
          A[A.size() - i - 1] -= res.
              back() * b.a[b.a.size() - i
              - 1];
        }
      }
      A.pop_back();
    }
    std::reverse(begin(res), end(res));
    return {res, A};
  }
  pair<poly, poly> divmod(
      const poly &b) const { // returns
          quotiend and remainder of a
          mod b
    if (deg() < b.deg()) {
      return {poly{0}, *this};
    }
    int d = deg() - b.deg();
    if (min(d, b.deg()) < magic) {
      return divmod_slow(b);
    }
    poly D = (reverse(d + 1) * b.
        reverse(d + 1).inv(d + 1))
            .mod_xk(d + 1)
            .reverse(d + 1, 1);
    return {D, *this - D * b};
  }
  poly log(size_t n) { // calculate
      log p(x) mod x^n
    assert(a[0] == T(1));
    return (deriv().mod_xk(n) * inv(n))
        .integr().mod_xk(n);
  }
  poly exp(size_t n) { // calculate
      exp p(x) mod x^n
    if (is_zero()) {
      return T(1);
    }
    assert(a[0] == T(0));
    poly ans = T(1);
    size_t a = 1;
    while (a < n) {
      poly C = ans.log(2 * a).div_xk(a)
          - substr(a, 2 * a);
      ans -= (ans * C).mod_xk(a).mul_xk
          (a);
      a *= 2;
    }
    return ans.mod_xk(n);
  }
  poly pow(size_t k, size_t n) { //
      calculate p^k(n) mod x^n
    if (is_zero()) {
      return *this;
    }
    if (k < magic) {
      return pow_slow(k, n);
    }
    int i = leading_xk();
    T j = a[i];
    poly t = div_xk(i) / j;
    return bpow(j, k) * (t.log(n) * T(k
        )).exp(n).mul_xk(i * k).mod_xk(
        n);
  }
  vector<T> chirpz_even(T z, int n) {
      // P(1), P(z^2), P(z^4), ..., P(z
      ^2(n-1))
    int m = deg();
    if (is_zero()) return vector<T>(n,
        0);
    vector<T> vv(m + n);
    T zi = z.inv();T zz = zi * zi;
    T cur = zi;T total = 1;
    for (int i = 0; i <= max(n - 1, m);
        i++) {
      if (i <= m) vv[m - i] = total;
```

```
    if (i < n) vv[m + i] = total;
    total *= cur;cur *= zz;
  }
  poly w = (mulx_sq(z) * vv).substr(m
      , m + n).mulx_sq(z);
  vector<T> res(n);
  for (int i = 0; i < n; i++) res[i]
      = w[i];
  return res;
}
vector<T> chirpz(T z, int n) { // P
    (1), P(z), P(z^2), ..., P(z^(n-1)
    )
  auto even = chirpz_even(z, (n + 1)
      / 2);
  auto odd = mulx(z).chirpz_even(z, n
      / 2);
  vector<T> ans(n);
  for (int i = 0; i < n / 2; i++) {
    ans[2 * i] = even[i];ans[2 * i +
        1] = odd[i];}
  if (n % 2 == 1) ans[n - 1] = even.
      back();
  return ans;
}
template <typename iter>
vector<T> eval(vector<poly> &tree,
    int v, iter l,
           iter r) { // auxiliary
              evaluation function
  if (r - l == 1) return {eval(*l)};
  else {
    auto m = l + (r - l) / 2;
    auto A = (*this % tree[2 * v]).
        eval(tree, 2 * v, l, m);
    auto B = (*this % tree[2 * v +
        1]).eval(tree, 2 * v + 1, m, r
        );
    A.insert(end(A), begin(B), end(B)
        );
    return A; }
}
vector<T> eval(vector<T> x) { //
    evaluate polynomial in (x1, ...,
    xn)
  int n = x.size();
  if (is_zero())return vector<T>(n, T
      (0));
  vector<poly> tree(4 * n);
  build(tree, 1, begin(x), end(x));
  return eval(tree, 1, begin(x), end(
      x));
}
template <typename iter>
poly inter(vector<poly> &tree, int v
    , iter l, iter r, iter ly,
         iter ry) { // auxiliary
            interpolation function
  if (r - l == 1) {
    return {*ly / a[0]};
  } else {
    auto m = l + (r - l) / 2;
    auto my = ly + (ry - ly) / 2;
    auto A = (*this % tree[2 * v]).
        inter(tree, 2 * v, l, m, ly,
        my);
    auto B = (*this % tree[2 * v +
        1]).inter(tree, 2 * v + 1, m,
        r, my, ry);
    return A * tree[2 * v + 1] + B *
        tree[2 * v];
  }
  }
};
template <typename T, typename iter>
poly<T> build(vector<poly<T>> &res,
    int v, iter L,
           iter R) { // builds
              evaluation tree for (x-
              a1)(x-a2)...(x-an)
  if (R - L == 1) {
    return res[v] = vector<T>{-*L, 1};
  } else {
    iter M = L + (R - L) / 2;
    return res[v] = build(res, 2 * v, L
        , M) * build(res, 2 * v + 1, M,
         R);
  }
}
template <typename T>
poly<T> inter(
    vector<T> x,
    vector<T> y) { // interpolates
       minimum polynomial from (xi, yi
       ) pairs
  int n = x.size();
  vector<poly<T>> tree(4 * n);
  return build(tree, 1, begin(x), end(
      x))
      .deriv()
      .inter(tree, 1, begin(x), end(x),
          begin(y), end(y));
}
}; // namespace algebra
using namespace algebra;
typedef poly<base> polyn;
```

4 flow

4.1 dinic

```
struct FlowEdge {
  int v, u;
  long long cap, flow = 0;
  FlowEdge(int v, int u, long long cap
      ) : v(v), u(u), cap(cap) {}
};

struct Dinic {
  const long long flow_inf = 1e18;
  vector<FlowEdge> edges;
  vector<vector<int>> adj;
  int n, m = 0;
  int s, t;
  vector<int> level, ptr;
  queue<int> q;

  Dinic(int n, int s, int t) : n(n), s
      (s), t(t) {
    adj.resize(n);
    level.resize(n);
    ptr.resize(n);
  }

  void add_edge(int v, int u, long
      long cap) {
    edges.emplace_back(v, u, cap);
    edges.emplace_back(u, v, 0);
    adj[v].push_back(m);
    adj[u].push_back(m + 1);
    m += 2;
  }

  bool bfs() {
    while (!q.empty()) {
      int v = q.front();
      q.pop();
      for (int id : adj[v]) {
        if (edges[id].cap - edges[id].
            flow < 1) continue;
        if (level[edges[id].u] != -1)
            continue;
        level[edges[id].u] = level[v] +
            1;
        q.push(edges[id].u);
      }
    }
    return level[t] != -1;
  }

  long long dfs(int v, long long
      pushed) {
    if (pushed == 0) return 0;
    if (v == t) return pushed;
    for (int& cid = ptr[v]; cid < (int)
        adj[v].size(); cid++) {
      int id = adj[v][cid];
      int u = edges[id].u;
      if (level[v] + 1 != level[u] ||
          edges[id].cap - edges[id].flow
           < 1)
        continue;
      long long tr = dfs(u, min(pushed,
          edges[id].cap - edges[id].
          flow));
      if (tr == 0) continue;
      edges[id].flow += tr;
      edges[id ^ 1].flow -= tr;
      return tr;
    }
    return 0;
  }

  long long flow() {
    long long f = 0;
    while (true) {
      fill(level.begin(), level.end(),
          -1);
      level[s] = 0;
      q.push(s);
      if (!bfs()) break;
      fill(ptr.begin(), ptr.end(), 0);
      while (long long pushed = dfs(s,
          flow_inf)) {
        f += pushed;
      }
    }
    return f;
  }
};
```

4.2 global min cut

```
/* Description: Find a global minimum
    cut in an undirected graph, as
    represented
 * by an adjacency matrix. Time: O(V
    ^3) */
pair<int, vi> globalMinCut(vector<vi>
    mat) {
  pair<int, vi> best = {INT_MAX, {}};
  int n = sz(mat);
  vector<vi> co(n);
  rep(i, 0, n) co[i] = {i};
  rep(ph, 1, n) {
    vi w = mat[0];
    size_t s = 0, t = 0;
    rep(it, 0, n - ph) { // O(V^2) -> O
        (E log V) with prio. queue
      w[t] = INT_MIN;
```

```
    s = t, t = max_element(all(w)) -
        w.begin();
    rep(i, 0, n) w[i] += mat[t][i];
   }
   best = min(best, {w[t] - mat[t][t],
       co[t]});
   co[s].insert(co[s].end(), all(co[t
       ]));
   rep(i, 0, n) mat[s][i] += mat[t][i
       ];
   rep(i, 0, n) mat[i][s] = mat[s][i];
   mat[0][t] = INT_MIN;
  }
  return best;
}
```

4.3 hungarian emaxx

```
// a[1....n][1.....m] -> cost function
// n<=m with n people having to assign
    m jobs

vector<int> u(n + 1), v(m + 1), p(m +
   1), way(m + 1);
for (int i = 1; i <= n; ++i) {
  p[0] = i;
  int j0 = 0;
  vector<int> minv(m + 1, INF);
  vector<char> used(m + 1, false);
  do {
   used[j0] = true;
   int i0 = p[j0], delta = INF, j1;
   for (int j = 1; j <= m; ++j)
    if (!used[j]) {
     int cur = a[i0][j] - u[i0] - v[j
         ];
     if (cur < minv[j]) minv[j] = cur
         , way[j] = j0;
     if (minv[j] < delta) delta =
         minv[j], j1 = j;
    }
   for (int j = 0; j <= m; ++j)
    if (used[j])
     u[p[j]] += delta, v[j] -= delta;
    else
     minv[j] -= delta;
   j0 = j1;
  } while (p[j0] != 0);
  do {
   int j1 = way[j0];
   p[j0] = p[j1];
   j0 = j1;
  } while (j0);
}

vector<int> ans(n + 1);
for (int j = 1; j <= m; ++j) ans[p[j]]
    = j;

int cost = -v[0];
```

4.4 kuhn

```
int n, k;
vector<vector<int>> g;
vector<int> mt;
vector<bool> used;
bool try_kuhn(int v) {
   if (used[v])
      return false;
   used[v] = true;
   for (int to : g[v]) {
      if (mt[to] == -1 || try_kuhn(mt[
          to])) {
         mt[to] = v;
         return true;
      }
   }
   return false;
}
int main() {
   mt.assign(k, -1);
   for (int v = 0; v < n; ++v) {
      used.assign(n, false);
      try_kuhn(v);
   }
   for (int i = 0; i < k; ++i)
      if (mt[i] != -1)
         printf("%d %d\n", mt[i] + 1,
             i + 1);
}
```

4.5 mcmf with negative cycle

```
// Push-Relabel implementation of the
    cost-scaling algorithm
// Runs in O( <max_flow> * log(V *
    max_edge_cost)) = O( V^3 * log(V *
     C))
// 3e4 edges are fine.
// Operates on integers, costs are
    multiplied by N!!

#include <bits/stdc++.h>
using namespace std;

template <typename flow_t = int,
    typename cost_t = int>
struct mcSFlow {
  struct Edge {
   cost_t c;
   flow_t f;
   int to, rev;
   Edge(int _to, cost_t _c, flow_t _f,
       int _rev)
```

```
    : c(_c), f(_f), to(_to), rev(
      _rev) {}
};
static constexpr cost_t INFCOST =
    numeric_limits<cost_t>::max() /
    2;
cost_t eps;
int N, S, T;
vector<vector<Edge> > G;
vector<unsigned int> isq, cur;
vector<flow_t> ex;
vector<cost_t> h;
mcSFlow(int _N, int _S, int _T) :
    eps(0), N(_N), S(_S), T(_T), G(_N
    ) {}
void add_edge(int a, int b, cost_t
    cost, flow_t cap) {
  assert(cap >= 0);
  assert(a >= 0 && a < N && b >= 0 &&
      b < N);
  if (a == b) {
    assert(cost >= 0);
    return;
  }
  cost *= N;
  eps = max(eps, abs(cost));
  G[a].emplace_back(b, cost, cap, G[b
      ].size());
  G[b].emplace_back(a, -cost, 0, G[a
      ].size() - 1);
}
void add_flow(Edge &e, flow_t f) {
  Edge &back = G[e.to][e.rev];
  if (!ex[e.to] && f) hs[h[e.to]].
      push_back(e.to);
  e.f -= f;
  ex[e.to] += f;
  back.f += f;
  ex[back.to] -= f;
}
vector<vector<int> > hs;
vector<int> co;
flow_t max_flow() {
  ex.assign(N, 0);
  h.assign(N, 0);
  hs.resize(2 * N);
  co.assign(2 * N, 0);
  cur.assign(N, 0);
  h[S] = N;
  ex[T] = 1;
  co[0] = N - 1;
  for (auto &e : G[S]) add_flow(e, e.
      f);
  if (hs[0].size())
    for (int hi = 0; hi >= 0;) {
      int u = hs[hi].back();
      hs[hi].pop_back();
      while (ex[u] > 0) { // discharge
          u
        if (cur[u] == G[u].size()) {
          h[u] = 1e9;
          for (unsigned int i = 0; i <
              G[u].size(); ++i) {
            auto &e = G[u][i];
            if (e.f && h[u] > h[e.to] +
                1) {
              h[u] = h[e.to] + 1, cur[u]
                  = i;
            }
          }
          if (++co[h[u]], !--co[hi] &&
              hi < N)
            for (int i = 0; i < N; ++i)
              if (hi < h[i] && h[i] < N)
                  {
                --co[h[i]];
                h[i] = N + 1;
              }
          hi = h[u];
        } else if (G[u][cur[u]].f && h
            [u] == h[G[u][cur[u]].to] +
            1)
          add_flow(G[u][cur[u]], min(ex
              [u], G[u][cur[u]].f));
        else
          ++cur[u];
      }
      while (hi >= 0 && hs[hi].empty()
          ) --hi;
    }
  return -ex[S];
}
void push(Edge &e, flow_t amt) {
  if (e.f < amt) amt = e.f;
  e.f -= amt;
  ex[e.to] += amt;
  G[e.to][e.rev].f += amt;
  ex[G[e.to][e.rev].to] -= amt;
}
void relabel(int vertex) {
  cost_t newHeight = -INFCOST;
  for (unsigned int i = 0; i < G[
      vertex].size(); ++i) {
    Edge const &e = G[vertex][i];
    if (e.f && newHeight < h[e.to] -
        e.c) {
      newHeight = h[e.to] - e.c;
      cur[vertex] = i;
    }
  }
```

```
  h[vertex] = newHeight - eps;
}
static constexpr int scale = 2;
pair<flow_t, cost_t> minCostMaxFlow
    () {
  cost_t retCost = 0;
  for (int i = 0; i < N; ++i)
    for (Edge &e : G[i]) retCost += e
        .c * (e.f);
  // find max-flow
  flow_t retFlow = max_flow();
  h.assign(N, 0);
  ex.assign(N, 0);
  isq.assign(N, 0);
  cur.assign(N, 0);
  queue<int> q;
  for (; eps; eps >>= scale) {
    // refine
    fill(cur.begin(), cur.end(), 0);
    for (int i = 0; i < N; ++i)
      for (auto &e : G[i])
        if (h[i] + e.c - h[e.to] < 0
            && e.f) push(e, e.f);
    for (int i = 0; i < N; ++i) {
      if (ex[i] > 0) {
        q.push(i);
        isq[i] = 1;
      }
    }
    // make flow feasible
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      isq[u] = 0;
      while (ex[u] > 0) {
        if (cur[u] == G[u].size())
            relabel(u);
        for (unsigned int &i = cur[u],
            max_i = G[u].size(); i <
            max_i; ++i) {
          Edge &e = G[u][i];
          if (h[u] + e.c - h[e.to] < 0)
              {
            push(e, ex[u]);
            if (ex[e.to] > 0 && isq[e.
                to] == 0) {
              q.push(e.to);
              isq[e.to] = 1;
            }
            if (ex[u] == 0) break;
          }
        }
      }
    }
    if (eps > 1 && eps >> scale == 0)
        {
      eps = 1 << scale;
    }
  }
  for (int i = 0; i < N; ++i) {
    for (Edge &e : G[i]) {
      retCost -= e.c * (e.f);
    }
  }
  return make_pair(retFlow, retCost /
      2 / N);
}
flow_t getFlow(Edge const &e) {
    return G[e.to][e.rev].f; }
};
```

5 Geometry

5.1 Convex Hull

```
struct pt {
  double x, y;
};

int orientation(pt a, pt b, pt c) {
  double v = a.x * (b.y - c.y) + b.x *
      (c.y - a.y) + c.x * (a.y - b.y);
  if (v < 0) return -1; // clockwise
  if (v > 0) return +1; // counter-
     clockwise
  return 0;
}

bool cw(pt a, pt b, pt c, bool
    include_collinear) {
  int o = orientation(a, b, c);
  return o < 0 || (include_collinear
      && o == 0);
}
bool ccw(pt a, pt b, pt c, bool
    include_collinear) {
  int o = orientation(a, b, c);
  return o > 0 || (include_collinear
      && o == 0);
}

void convex_hull(vector<pt>& a, bool
    include_collinear = false) {
  if (a.size() == 1) return;

  sort(a.begin(), a.end(),
      [](pt a, pt b) { return
          make_pair(a.x, a.y) <
          make_pair(b.x, b.y); });
```

```
pt p1 = a[0], p2 = a.back();
vector<pt> up, down;
up.push_back(p1);
down.push_back(p1);
for (int i = 1; i < (int)a.size(); i
    ++) {
  if (i == a.size() - 1 || cw(p1, a[i
      ], p2, include_collinear)) {
    while (up.size() >= 2 &&
          !cw(up[up.size() - 2], up[up
              .size() - 1], a[i],
              include_collinear))
      up.pop_back();
    up.push_back(a[i]);
  }
  if (i == a.size() - 1 || ccw(p1, a[
      i], p2, include_collinear)) {
    while (down.size() >= 2 &&
          !ccw(down[down.size() - 2],
              down[down.size() - 1], a[
              i],
               include_collinear))
      down.pop_back();
    down.push_back(a[i]);
  }
}

if (include_collinear && up.size()
    == a.size()) {
  reverse(a.begin(), a.end());
  return;
}
a.clear();
for (int i = 0; i < (int)up.size();
    i++) a.push_back(up[i]);
for (int i = down.size() - 2; i > 0;
     i--) a.push_back(down[i]);
}
```

5.2 geometry 2d

```
namespace geometry_2d {
typedef double T;
typedef complex<T> pt;
int sgn(T x) { return (T(0) < x) - (x
    < T(0)); }
#define x real()
#define y imag()
T sq(pt p) { return p.x * p.x + p.y *
    p.y; }

pt translate(pt v, pt p) { return p +
    v; }
pt scale(pt c, double factor, pt p) {
    return c + (p - c) * factor; }
pt rot(pt p, double a) { return p *
    polar(1.0, a); }
pt perp(pt p) { return {-p.y, p.x}; }
pt linearTransfo(pt p, pt q, pt r, pt
    fp, pt fq) {
  return fp + (r - p) * (fq - fp) / (q
      - p);
}
T dot(pt v, pt w) { return (conj(v) *
    w).x; }
T cross(pt v, pt w) { return (conj(v)
    * w).y; }
bool isPerp(pt v, pt w) { return dot(v
    , w) == 0; }
double angle(pt v, pt w) {
  return acos(clamp(dot(v, w) / abs(v)
      / abs(w), -1.0, 1.0));
}
T orient(pt a, pt b, pt c) { return
    cross(b - a, c - a); }
bool inAngle(pt a, pt b, pt c, pt p) {
  assert(orient(a, b, c) != 0);
  if (orient(a, b, c) < 0) swap(b, c);
  return orient(a, b, p) >= 0 &&
      orient(a, c, p) <= 0;
}
double orientedAngle(pt a, pt b, pt c)
    {
  if (orient(a, b, c) >= 0)
    return angle(b - a, c - a);
  else
    return 2 * M_PI - angle(b - a, c -
        a);
}
bool isConvex(vector<pt> p) {
  bool hasPos = false, hasNeg = false;
  for (int i = 0, n = p.size(); i < n;
      i++) {
    int o = orient(p[i], p[(i + 1) % n
        ], p[(i + 2) % n]);
    if (o > 0) hasPos = true;
    if (o < 0) hasNeg = true;
  }
  return !(hasPos && hasNeg);
}
bool half(pt p) {
  // true if in blue half
  assert(p.x != 0 || p.y != 0); // the
      argument of (0,0) isundefined
  return p.y > 0 || (p.y == 0 && p.x <
      0);
}
void polarSort(vector<pt> &v) {
  sort(v.begin(), v.end(), [](pt v, pt
      w) {
    return make_tuple(half(v), 0, sq(v)
        ) <
```

```
        make_tuple(half(w), cross(v,
            w), sq(w));
  });
}
void polarSortAround(pt o, vector<pt>
    &v) {
  sort(v.begin(), v.end(), [=](pt v,
      pt w) {
    return make_tuple(half(v - o), 0) <
        make_tuple(half(w - o), cross
            (v - o, w - o));
  });
}
struct line {
  pt v;
  T c;
  // From direction vector v and
     offset c
  line(pt v, T c) : v(v), c(c) {}
  // From equation ax+by=c
  line(T a, T b, T c) : v({b, -a}), c(
      c) {}
  // From points P and Q
  line(pt p, pt q) : v(q - p), c(cross
      (v, p)) {}
  // Will be defined later:
  // - these work with T = int
  T side(pt p) { return cross(v, p) -
      c; }
  double dist(pt p) { return abs(side(
      p)) / abs(v); }
  double sqDist(pt p) { return side(p)
      * side(p) / (double)sq(v); }
  line perpThrough(pt p) { return {p,
      p + perp(v)}; }
  bool cmpProj(pt p, pt q) { return
      dot(v, p) < dot(v, q); }
  line translate(pt t) { return {v, c
      + cross(v, t)}; }
  line shiftLeft(double dist) { return
      {v, c + dist * abs(v)}; }
  bool inter(line l1, line l2, pt &out
      ) {
    T d = cross(l1.v, l2.v);
    if (d == 0) return false;
    out =
        (l2.v * l1.c - l1.v * l2.c) / d;
            // requires floating-point
           coordinates
    return true;
  }
  pt proj(pt p) { return p - perp(v) *
      side(p) / sq(v); }
  pt refl(pt p) { return p - perp(v) *
      T(2) * side(p) / sq(v); }
};

line bisector(line l1, line l2, bool
    interior) {
  assert(cross(l1.v, l2.v) != 0); //
      l1 and l2 cannot be parallel!
  double sign = interior ? 1 : -1;
  return {l2.v / abs(l2.v) + l1.v /
      abs(l1.v) * sign,
          l2.c / abs(l2.v) + l1.c / abs(
              l1.v) * sign};
}
bool inDisk(pt a, pt b, pt p) { return
    dot(a - p, b - p) <= 0; }
bool onSegment(pt a, pt b, pt p) {
  return orient(a, b, p) == 0 &&
      inDisk(a, b, p);
}
bool properInter(pt a, pt b, pt c, pt
    d, pt &out) {
  double oa = orient(c, d, a), ob =
      orient(c, d, b), oc = orient(a, b
      , c),
          od = orient(a, b, d);
  // Proper intersection exists iff
      opposite signs
  if (oa * ob < 0 && oc * od < 0) {
    out = (a * ob - b * oa) / (ob - oa)
        ;
    return true;
  }
  return false;
}
struct cmpX {
  bool operator()(pt a, pt b) const {
    return make_pair(a.x, a.y) <
        make_pair(b.x, b.y);
  }
};
set<pt, cmpX> inters(pt a, pt b, pt c,
    pt d) {
  pt out;
  if (properInter(a, b, c, d, out))
      return {out};
  set<pt, cmpX> s;
  if (onSegment(c, d, a)) s.insert(a);
  if (onSegment(c, d, b)) s.insert(b);
  if (onSegment(a, b, c)) s.insert(c);
  if (onSegment(a, b, d)) s.insert(d);
  return s;
}
double segPoint(pt a, pt b, pt p) {
  if (a != b) {
    line l(a, b);
```

```
    if (l.cmpProj(a, p) && l.cmpProj(p,
        b)) // if closest toprojection
      return l.dist(p);
    // output distance toline
  }
  return min(abs(p - a), abs(p - b));
      // otherwise distance to A or B
}
double segSeg(pt a, pt b, pt c, pt d)
    {
  pt dummy;
  if (properInter(a, b, c, d, dummy))
      return 0;
  return min({segPoint(a, b, c),
      segPoint(a, b, d), segPoint(c, d,
       a),
           segPoint(c, d, b)});
}
double areaTriangle(pt a, pt b, pt c)
    { return abs(cross(b - a, c - a))
    / 2.0; }
double areaPolygon(vector<pt> p) {
  double area = 0.0;
  for (int i = 0, n = p.size(); i < n;
       i++) {
    area += cross(p[i], p[(i + 1) % n])
        ; // wrap back to 0 if i == n-1
  }
  return abs(area) / 2.0;
}

// true if P at least as high as A (
    blue part)
bool above(pt a, pt p) { return p.y >=
     a.y; }
// check if [PQ] crosses ray from A
bool crossesRay(pt a, pt p, pt q) {
  return (above(a, q) - above(a, p)) *
      orient(a, p, q) > 0;
}
// if strict, returns false when A is
    on the boundary
bool inPolygon(vector<pt> p, pt a,
    bool strict = true) {
  int numCrossings = 0;
  for (int i = 0, n = p.size(); i < n;
       i++) {
    if (onSegment(p[i], p[(i + 1) % n],
        a)) return !strict;
    numCrossings += crossesRay(a, p[i],
        p[(i + 1) % n]);
  }
  return numCrossings & 1; // inside
      if odd number of crossings
}
double angleTravelled(pt a, pt p, pt q
    ) {
  // remainder ensures the value is in
       [-pi,pi]
  return remainder(arg(q - a) - arg(p
      - a), 2 * M_PI);
}
int windingNumber(vector<pt> p, pt a)
    {
  double ampli = 0;
  for (int i = 0, n = p.size(); i < n;
       i++)
    ampli += angleTravelled(a, p[i], p
        [(i + 1) % n]);
  return round(ampli / (2 * M_PI));
}

pt circumCenter(pt a, pt b, pt c) {
  b = b - a, c = c - a; // consider
      coordinates relative to A
  assert(cross(b, c) != 0); // no
      circumcircle if A,B,C aligned
  return a + perp(b * sq(c) - c * sq(b
      )) / cross(b, c) / T(2);
}
int circleLine(pt o, double r, line l,
     pair<pt, pt> &out) {
  double h2 = r * r - l.sqDist(o);
  if (h2 >= 0) {
    // the line touches the circle
    pt p = l.proj(o);          // point
        P
    pt h = l.v * sqrt(h2) / abs(l.v);
        // vector parallel to l,
        oflength h
    out = {p - h, p + h};
  }
  return 1 + sgn(h2);
}
int circleCircle(pt o1, double r1, pt
    o2, double r2, pair<pt, pt> &out)
    {
  pt d = o2 - o1;
  double d2 = sq(d);
  if (d2 == 0) {
    assert(r1 != r2);
    return 0;
  }                                  //
      concentric circles
  double pd = (d2 + r1 * r1 - r2 * r2)
      / 2; // = |O_1P| * d
  double h2 = r1 * r1 - pd * pd / d2;
      // = h2
  if (h2 >= 0) {
```

```
    pt p = o1 + d * pd / d2, h = perp(d
        ) * sqrt(h2 / d2);
    out = {p - h, p + h};
  }
  return 1 + sgn(h2);
}
int tangents(pt o1, double r1, pt o2,
    double r2, bool inner,
          vector<pair<pt, pt>> &out) {
  if (inner) r2 = -r2;
  pt d = o2 - o1;
  double dr = r1 - r2, d2 = sq(d), h2
      = d2 - dr * dr;
  if (d2 == 0 || h2 < 0) {
    assert(h2 != 0);
    return 0;
  }
  for (double sign : {-1, 1}) {
    pt v = (d * dr + perp(d) * sqrt(h2)
        * sign) / d2;
    out.push_back({o1 + v * r1, o2 + v
        * r2});
  }
  return 1 + (h2 > 0);
}
} // namespace geometry_2d
```

5.3 Point in convex polygon

```
struct pt {
  long long x, y;
  pt() {}
  pt(long long _x, long long _y) : x(
      _x), y(_y) {}
  pt operator+(const pt &p) const {
      return pt(x + p.x, y + p.y); }
  pt operator-(const pt &p) const {
      return pt(x - p.x, y - p.y); }
  long long cross(const pt &p) const {
      return x * p.y - y * p.x; }
  long long dot(const pt &p) const {
      return x * p.x + y * p.y; }
  long long cross(const pt &a, const
      pt &b) const {
    return (a - *this).cross(b - *this)
        ;
  }
  long long dot(const pt &a, const pt
      &b) const {
    return (a - *this).dot(b - *this);
  }
  long long sqrLen() const { return
      this->dot(*this); }
};

bool lexComp(const pt &l, const pt &r)
    {
  return l.x < r.x || (l.x == r.x && l
      .y < r.y);
}

int sgn(long long val) { return val >
    0 ? 1 : (val == 0 ? 0 : -1); }

vector<pt> seq;
pt translation;
int n;

bool pointInTriangle(pt a, pt b, pt c,
    pt point) {
  long long s1 = abs(a.cross(b, c));
  long long s2 =
      abs(point.cross(a, b)) + abs(
          point.cross(b, c)) + abs(point
          .cross(c, a));
  return s1 == s2;
}

void prepare(vector<pt> &points) {
  n = points.size();
  int pos = 0;
  for (int i = 1; i < n; i++) {
    if (lexComp(points[i], points[pos])
        ) pos = i;
  }
  rotate(points.begin(), points.begin
      () + pos, points.end());

  n--;
  seq.resize(n);
  for (int i = 0; i < n; i++) seq[i] =
      points[i + 1] - points[0];
  translation = points[0];
}

bool pointInConvexPolygon(pt point) {
  point = point - translation;
  if (seq[0].cross(point) != 1 &&
      sgn(seq[0].cross(point)) != sgn(
          seq[0].cross(seq[n - 1])))
    return false;
  if (seq[n - 1].cross(point) != 0 &&
      sgn(seq[n - 1].cross(point)) !=
          sgn(seq[n - 1].cross(seq[0])))
    return false;

  if (seq[0].cross(point) == 0) return
      seq[0].sqrLen() >= point.sqrLen
      ();

  int l = 0, r = n - 1;
  while (r - l > 1) {
```

```
    int mid = (l + r) / 2;
    int pos = mid;
    if (seq[pos].cross(point) >= 0)
      l = mid;
    else
      r = mid;
  }
  int pos = l;
  return pointInTriangle(seq[pos], seq
      [pos + 1], pt(0, 0), point);
}
```

5.4 Shortest Distance between two points

```
vector<pt> t;

void rec(int l, int r) {
  if (r - l <= 3) {
    for (int i = l; i < r; ++i) {
      for (int j = i + 1; j < r; ++j) {
        upd_ans(a[i], a[j]);
      }
    }
    sort(a.begin() + l, a.begin() + r,
        cmp_y());
    return;
  }

  int m = (l + r) >> 1;
  int midx = a[m].x;
  rec(l, m);
  rec(m, r);

  merge(a.begin() + l, a.begin() + m,
      a.begin() + m, a.begin() + r, t.
      begin(),
       cmp_y());
  copy(t.begin(), t.begin() + r - l, a
      .begin() + l);

  int tsz = 0;
  for (int i = l; i < r; ++i) {
    if (abs(a[i].x - midx) < mindist) {
      for (int j = tsz - 1; j >= 0 && a
          [i].y - t[j].y < mindist; --j)
        upd_ans(a[i], t[j]);
      t[tsz++] = a[i];
    }
  }
}
```

6 Graph

6.1 articulation$_{p}oint$

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency
    list of graph
vector<bool> visited;
vector<int> tin, low;
int timer;
void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    int children=0;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to])
                ;
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to])
                ;
            if (low[to] >= tin[v] && p
                !=-1)
                IS_CUTPOINT(v);
            ++children;
        }
    }
    if(p == -1 && children > 1)
        IS_CUTPOINT(v); // v is
            articulation point so print
            it
}
void find_cutpoints() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs (i);
    }
}
```

6.2 bridges

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency
    list of graph
vector<bool> visited;
vector<int> tin, low;
int timer;
void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to])
                ;
        } else {
            dfs(to, v);
```

```
            low[v] = min(low[v], low[to])
                ;
            if (low[to] > tin[v])
                IS_BRIDGE(v, to);
        }
    }
}
void find_bridges() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
}
```

6.3 min vertex cover

```
/**
 * Description: Simple bipartite
    matching algorithm. Graph $g$
    should be a list
 * of neighbors of the left partition,
     and $btoa$ should be a vector
    full of
 * -1's of the same size as the right
    partition. Returns the size of the
 * matching. $btoa[i]$ will be the
    match for vertex $i$ on the right
    side, or
 * $-1$ if it's not matched. Time: O(
    VE) Usage: vi btoa(m, -1);
    dfsMatching(g,
 * btoa); Description: Finds a minimum
     vertex cover in a bipartite graph
    . The
 * size is the same as the size of a
    maximum matching, and the
    complement is a
 * maximum independent set*/
bool find(int j, vector<vi>& g, vi&
    btoa, vi& vis) {
  if (btoa[j] == -1) return 1;
  vis[j] = 1;
  int di = btoa[j];
  for (int e : g[di])
    if (!vis[e] && find(e, g, btoa, vis
        )) {
      btoa[e] = di;
      return 1;
    }
  return 0;
}
int dfsMatching(vector<vi>& g, vi&
    btoa) {
  vi vis;
  rep(i, 0, sz(g)) {
    vis.assign(sz(btoa), 0);
    for (int j : g[i])
      if (find(j, g, btoa, vis)) {
        btoa[j] = i;
        break;
      }
  }
  return sz(btoa) - (int)count(all(
      btoa), -1);
}
vi cover(vector<vi>& g, int n, int m)
    {
  vi match(m, -1);
  int res = dfsMatching(g, match);
  vector<bool> lfound(n, true), seen(m
      );
  for (int it : match)
    if (it != -1) lfound[it] = false;
  vi q, cover;
  rep(i, 0, n) if (lfound[i]) q.
      push_back(i);
  while (!q.empty()) {
    int i = q.back();
    q.pop_back();
    lfound[i] = 1;
    for (int e : g[i])
      if (!seen[e] && match[e] != -1) {
        seen[e] = true;
        q.push_back(match[e]);
      }
  }
  rep(i, 0, n) if (!lfound[i]) cover.
      push_back(i);
  rep(i, 0, m) if (seen[i]) cover.
      push_back(n + i);
  assert(sz(cover) == res);
  return cover;
}
```

6.4 virtual tree

```
vector<int> vg[MAXN];
int mstack[MAXN], msize;
// st start time in dfs order
bool cmp(int u, int v) { return st[u]
    < st[v]; }
void build_tree(vector<int> vec){
 // vec nodes given in query
 if(vec.empty()) { vg[0].clear();
     return; }
 sort(vec.begin(), vec.end(), cmp);
 vec.erase(unique(vec.begin(), vec.end
     ()), vec.end());
 int st_sz = vec.size();
 for(int i = 0; i + 1 < st_sz; i++)
```

```
 vec.push_back(lca(vec[i], vec[i +
     1]));
sort(vec.begin(), vec.end(), cmp);
vec.erase(unique(vec.begin(), vec.end
    ()), vec.end());
for(int vertex: vec)
 vg[vertex].clear();
msize = 0;
mstack[msize++] = vec[0];
for(int i = 1; i < (int)vec.size(); i
    ++){
 int anc = lca(mstack[msize - 1], vec
     [i]);
 if(anc != mstack[msize - 1]){
  while(msize > 1 && d[mstack[msize -
      2]] >= d[anc]){
   vg[mstack[msize - 2]].push_back(
       mstack[msize - 1]);
   msize--;
  }
  if(anc != mstack[msize - 1]){
   vg[anc].push_back(mstack[msize -
       1]);
   msize--; mstack[msize++] = anc;
  }
 }

 mstack[msize++] = vec[i];
}
for(int i = 0; i < msize - 1; i++)
 vg[mstack[i]].push_back(mstack[i +
     1]);
vg[0].clear(); vg[0].push_back(mstack
    [0]);
}
```

7 Math

7.1 CRT

```
for (int i = 0; i < k; ++i) {
   x[i] = a[i];
   for (int j = 0; j < i; ++j) {
      x[i] = r[j][i] * (x[i] - x[j]);

      x[i] = x[i] % p[i];
      if (x[i] < 0)
         x[i] += p[i];
   }
}
```

7.2 discrete log

```
// Returns minimum x for which a ^ x %
    m = b % m.
int solve(int a, int b, int m) {
   a %= m, b %= m;
   int k = 1, add = 0, g;
   while ((g = gcd(a, m)) > 1) {
      if (b == k)
         return add;
      if (b % g)
         return -1;
      b /= g, m /= g, ++add;
      k = (k * 1ll * a / g) % m;
   }

   int n = sqrt(m) + 1;
   int an = 1;
   for (int i = 0; i < n; ++i)
      an = (an * 1ll * a) % m;

   map<int, int> vals;
   for (int q = 0, cur = b; q <= n; ++
      q) {
      vals[cur] = q;
      cur = (cur * 1ll * a) % m;
   }

   for (int p = 1, cur = k; p <= n; ++
      p) {
      cur = (cur * 1ll * an) % m;
      if (vals.find(cur) != vals.end()
          ) {
         int ans = n * p - vals[cur] +
             add;
         return ans;
      }
   }
   return -1;
}
```

7.3 euclid gcd

```
int gcd(int a, int b, int& x, int& y)
    {
  x = 1, y = 0;
  int x1 = 0, y1 = 1, a1 = a, b1 = b;
  while (b1) {
    int q = a1 / b1;
    tie(x, x1) = make_tuple(x1, x - q *
        x1);
    tie(y, y1) = make_tuple(y1, y - q *
        y1);
    tie(a1, b1) = make_tuple(b1, a1 - q
        * b1);
  }
  return a1;
}
```

7.4 integer factorization polard rho brent

```
long long f(long long x, long long c,
    long long mod) {
  return (mult(x, x, mod) + c) % mod;
```

```
}
long long brent(long long n, long long
    x0 = 2, long long c = 1) {
  long long x = x0;
  long long g = 1;
  long long q = 1;
  long long xs, y;

  int m = 128;
  int l = 1;
  while (g == 1) {
    y = x;
    for (int i = 1; i < l; i++) x = f(x
        , c, n);
    int k = 0;
    while (k < l && g == 1) {
      xs = x;
      for (int i = 0; i < m && i < l -
          k; i++) {
        x = f(x, c, n);
        q = mult(q, abs(y - x), n);
      }
      g = gcd(q, n);
      k += m;
    }
    l *= 2;
  }
  if (g == n) {
    do {
      xs = f(xs, c, n);
      g = gcd(abs(xs - y), n);
    } while (g == 1);
  }
  return g;
}
```

7.5 prime list

```
999999937
NTT Prime: 998244353 = 119 * 2^23 +
    1. Primitive root: 3. 985661441 =
     235 * 2^22 + 1. Primitive root:
    3. 1012924417 = 483 * 2^21 + 1.
    Primitive root: 5.
```

7.6 prime test miller rabin

```
using u64 = uint64_t;
using u128 = __uint128_t;

bool check_composite(u64 n, u64 a, u64
    d, int s) {
  u64 x = binpower(a, d, n);
  if (x == 1 || x == n - 1) return
      false;
  for (int r = 1; r < s; r++) {
    x = (u128)x * x % n;
    if (x == n - 1) return false;
  }
  return true;
};
bool MillerRabin(u64 n) { // returns
    true if n is prime, else returns
    false.
  if (n < 2) return false;

  int r = 0;
  u64 d = n - 1;
  while ((d & 1) == 0) {
    d >>= 1;
    r++;
  }

  for (int a : {2, 3, 5, 7, 11, 13,
      17, 19, 23, 29, 31, 37}) {
    if (n == a) return true;
    if (check_composite(n, a, d, r))
        return false;
  }
  return true;
}
```

7.7 Primitive Root

```
int generator(int p) {
  vector<int> fact;
  int phi = p - 1, n = phi;
  for (int i = 2; i * i <= n; ++i)
    if (n % i == 0) {
      fact.push_back(i);
      while (n % i == 0) n /= i;
    }
  if (n > 1) fact.push_back(n);

  for (int res = 2; res <= p; ++res) {
    bool ok = true;
    for (size_t i = 0; i < fact.size()
        && ok; ++i)
      ok &= powmod(res, phi / fact[i],
          p) != 1;
    if (ok) return res;
  }
  return -1;
}
```

8 Matrix

8.1 gauss any mod

```
int gauss(vector<vector<int> > &a,
    vector<int> &ans) {
  int n = (int)a.size();
  int m = (int)a[0].size() - 1;
```

```
vector<int> where(m, -1);
for (int col = 0, row = 0; col < m
    && row < n; ++col) {
  int sel = row;
  for (int i = row; i < n; ++i)
    if (a[i][col] > a[sel][col]) sel
        = i;
  if (a[sel][col] == 0) continue;
  for (int i = col; i <= m; ++i) swap
      (a[sel][i], a[row][i]);
  where[col] = row;
  for (int i = 0; i < n; ++i)
    if (i != row) {
      int c = a[i][col] * mod_inv(a[
          row][col], mod) % mod;
      for (int j = col; j <= m; ++j) {
        a[i][j] = (a[i][j] - a[row][j]
            * c % mod + mod) % mod;
      }
    }
  ++row;
}
ans.assign(m, 0);
vi out(1);
for (int i = 0; i < m; ++i)
  if (where[i] != -1)
    ans[i] = a[where[i]][m] * mod_inv
        (a[where[i]][i], mod) % mod;
for (int i = 0; i < n; ++i) {
  int sum = 0;
  for (int j = 0; j < m; ++j) sum = (
      sum + ans[j] * a[i][j]) % mod;
  if (sum != a[i][m]) return -1;
}
for (int i = 0; i < m; ++i)
  if (where[i] == -1) return 2;
return 1;
}
```

8.2 gauss mod 2

```
const int N = 500;
int gauss(vector<bitset<N> > a, int n,
    int m, bitset<N>& ans) {
  vector<int> where(m, -1);
  for (int col = 0, row = 0; col < m
      && row < n; ++col) {
    for (int i = row; i < n; ++i)
      if (a[i][col]) {
        swap(a[i], a[row]);
        break;
      }
    if (!a[row][col]) continue;
    where[col] = row;

    for (int i = 0; i < n; ++i)
      if (i != row && a[i][col]) a[i]
          ^= a[row];
    ++row;
  }
  ans.reset();
  for (int i = 0; i < m; ++i)
    if (where[i] != -1) ans[i] = a[
        where[i]][m] / a[where[i]][i];
  for (int i = 0; i < n; ++i) {
    int sum = (ans & a[i]).count();
    if (sum % 2 != a[i][m]) return 0;
  }
  for (int i = 0; i < m; ++i)
    if (where[i] == -1) return 2;
  return 1;
}
```

9 range query

9.1 Fenwick

```
struct FenwickTree2D {
  vector<vector<int>> bit;
  int n, m;
  int sum(int x, int y) {
    int ret = 0;
    for (int i = x; i >= 0; i = (i & (i
        + 1)) - 1)
      for (int j = y; j >= 0; j = (j &
          (j + 1)) - 1) ret += bit[i][j
          ];
    return ret;
  }
  void add(int x, int y, int delta) {
    for (int i = x; i < n; i = i | (i +
        1))
      for (int j = y; j < m; j = j | (j
          + 1)) bit[i][j] += delta;
  }
};
```

9.2 MosAlgo

```
int block_size;
struct Query {
   int l, r, idx;
   bool operator<(Query other) const
   {
      if(l/block_size != other.l/
          block_size)return make_pair(
          l , r) <
            make_pair(other.l , other.
                r);
      return (l/block_size & 1) ? (r <
          other.r) : (r > other.r);
   }
```

```
};
```

10 string

10.1 AhoCorasick

```
template<int ALPHABET = 26, int LOW =
    'a'>
struct AhoCorasick {
 struct Node {
  int next[ALPHABET], link, parent;
  char ch; bool ends;
  Node(int par = -1, char c = LOW - 1)
      : parent(par), ch(c), link(-1),
      ends(false) {
   for(int i=0; i<ALPHABET; i++)
    next[i] = -1;
  }
 };
 vector<Node> nodes;
 int root;
 AhoCorasick(): root(0), nodes(1) {}
 void add_string(string &s, int idx) {
  int cur = root;
  for(auto c: s) {
   if(nodes[cur].next[c - LOW] == -1)
    nodes.push_back(Node(cur, c)),
        nodes[cur].next[c - LOW] = (int
        )nodes.size()-1;
   cur = nodes[cur].next[c - LOW];
  }
  nodes[cur].leaves.push_back(idx),
      nodes[cur].ends = true;
 }
 void build_links() {
  queue<int> q; q.push(0);
  while(!q.empty()) {
   int fr = q.front(); q.pop();
   if(nodes[fr].parent <= 0) {
    nodes[fr].link = 0;
    for(int i=0; i<ALPHABET; i++)
     if(nodes[fr].next[i] == -1)
      if(nodes[fr].parent == -1)
       nodes[fr].next[i] = 0;
      else
       nodes[fr].next[i] = nodes[nodes[
           fr].link].next[i];
    else
     q.push(nodes[fr].next[i]);
   }
   else {
    nodes[fr].link = nodes[nodes[nodes[
        fr].parent].link].next[nodes[fr
        ].ch - LOW];
    for(int i=0; i<ALPHABET; i++)
     if(nodes[fr].next[i] == -1)
      nodes[fr].next[i] = nodes[nodes[
          fr].link].next[i];
    else
     q.push(nodes[fr].next[i]);
   }
  }
 }
};
```

10.2 suffixArray

```
const int MAXLEN = 4e5 + 5;
template <int ALPHABET = 26, int LOW =
    'a'>
struct SuffixArray {
 vector<int> sa, order, lcp, locate;
 vector<vector<int>> sparse;
 string _s;
 SuffixArray() {}
 void build(string s) {
  s += (char)(LOW - 1);
  int n = s.size();
  _s = s;
  sa.resize(n);
  order.resize(n);
  vector<vector<int>> pos(ALPHABET +
      1);
  for (int i = 0; i < n; i++) pos[s[i
      ] - LOW + 1].push_back(i);

  int idx = -1, o_idx = -1;
  for (int i = 0; i < ALPHABET + 1; i
      ++) {
   o_idx += (pos[i].size() > 0);
   for (auto& x : pos[i]) order[x] =
       o_idx, sa[++idx] = x;
  }
  int cur = 1;
  while (cur < n) {
   cur *= 2;
   vector<pair<pair<int, int>, int>>
       w(n);
   vector<int> cnt(n), st(n), where(
       n);
   for (int i = 0; i < n; i++) {
    int from = sa[i] - cur / 2 + n;
    if (from >= n) from -= n;
    w[i] = {{order[from], order[sa[i
        ]]}, from};
    cnt[order[from]]++;
    where[from] = i;
   }
   for (int i = 1; i < n; i++) st[i]
       = st[i - 1] + cnt[i - 1];
   for (int i = 0; i < n; i++) sa[st
       [w[i].first.first]++] = w[i].
```

```
      second;

  order[sa[0]] = 0;
  for (int i = 1; i < n; i++)
   order[sa[i]] = order[sa[i - 1]]
       +
              (w[where[sa[i]]].
                 first != w[where[
                 sa[i - 1]]].first
                 );
 }
}
void build_lcp() {
 int n = sa.size();
 lcp.resize(n);
 locate.resize(n);
 for (int i = 0; i < n; i++) locate[
    sa[i]] = i;
 for (int i = 0; i < n - 1; i++) {
  int wh = locate[i], up = sa[wh -
     1];
  if (i > 0) lcp[wh] = max(lcp[wh],
      lcp[locate[i - 1]] - 1);
  while (_s[i + lcp[wh]] == _s[up +
      lcp[wh]]) ++lcp[wh];
 }
}
void build_sparse() {
 int n = _s.size();
 sparse.resize(20, vector<int>(n));
 for (int i = 0; i < n; i++) sparse
    [0][i] = lcp[i];
 for (int i = 1, len = 2; i < 20; i
    ++, len *= 2)
  for (int j = 0; j + len <= n; j
     ++)
      sparse[i][j] = min(sparse[i -
         1][j], sparse[i - 1][j + len
          / 2]);
 }
 int find_lcp(int a, int b) {
  if (a == b)
   return _s.size() - 1 - a; //-1
      because sentinel is added to
      string
  a = locate[a];
  b = locate[b];
  if (a > b) {
   swap(a, b);
  }
  a++;
  int which = log2(b - a + 1);
  return min(sparse[which][a], sparse
     [which][b - (1 << which) + 1]);
 }
};
```

10.3 suffixAutomaton

```
template <int MAXLEN = 1000000>
struct SuffixAutomaton {
 struct node_SA {
  int len, link, cnt;
  int next[26]; // map<char, int>
     next;
  node_SA() {
   for (int i = 0; i < 26; i++) next
      [i] = -2;
  }
 };
 vector<node_SA> v;
 int sz, last;
 SuffixAutomaton(int MAX_SIZE =
    MAXLEN) : sz(1), last(0), v(2 *
    MAX_SIZE + 5) {
  v[0].len = 0, v[0].link = -1;
 }
 int minlen(const int& idx) {
  return (v[idx].link == -1 ? 0 : v[v
     [idx].link].len + 1);
 }
 int minlen(const node_SA& n) {
  return (n.link == -1 ? 0 : v[n.link
     ].len + 1);
 }
 void add_char(char c) {
  int cur = sz++;
  v[cur].len = v[last].len + 1;
  v[cur].cnt = 1;
  int temp = last;
  while (temp != -1 && v[temp].next[c
      - 'a'] == -2) {
   v[temp].next[c - 'a'] = cur;
   temp = v[temp].link;
  }
  if (temp == -1)
   v[cur].link = 0;
  else {
   int nx = v[temp].next[c - 'a'];
   if (v[temp].len + 1 == v[nx].len)
    v[cur].link = nx;
   else {
    int clone = sz++;
    v[clone].len = v[temp].len + 1;
    v[clone].link = v[nx].link;
    for (int i = 0; i < 26; i++) v[
       clone].next[i] = v[nx].next[
       i];
    while (temp != -1 && v[temp].
       next[c - 'a'] == nx) {
     v[temp].next[c - 'a'] = clone;
     temp = v[temp].link;
```

```
      }
      v[nx].link = v[cur].link = clone
          ;
    }
   }
   last = cur;
  }
  void build(std::string& s) {
   for (char c : s) add_char(c);
  }
};
```

10.4 z kmp manacher

```
vector<int> kmp(const string &s) {
  int n = (int)s.size();
  vector<int> ans(n, 0);
  for (int i = 1; i < n; i++) {
   int k = ans[i - 1];
   while (k && s[k] != s[i]) k = ans[k
       - 1];
   ans[i] = k + (s[k] == s[i]);
  }
  return ans;
}
vector<int> zfunc(const string &s) {
  int n = (int)s.size();
  vector<int> z(n, 0);
  z[0] = n;
  for (int i = 1, l = 0, r = 0; i < n;
      i++) {
   z[i] = max(0, min(r - i + 1, z[i -
       l]));
   while (s[i + z[i]] == s[z[i]]) ++z[
       i];
   if (i + z[i] - 1 > r) l = i, r = i
       + z[i] - 1;
  }
  return z;
}
pair<vector<int>, vector<int>>
    manacher(const string &s) {
  string t = "$";
  for (auto c : s) t += c, t += '^';
    // Only odd manacher will do the
    trick now
  int N = (int)t.size();
  vector<int> ans(N, 1);
  int l = 1, r = 1;
  for (int i = 1; i < N; i++) {
   ans[i] = max(0, min(r - i, ans[l +
       (r - i)]));
   while (t[i - ans[i]] == t[i + ans[i
       ]]) ++ans[i];
   if (i + ans[i] > r) l = i - ans[i],
       r = i + ans[i];
  }
  vector<int> odd, even;
  for (int i = 1; i < N - 1; i++) {
   if (i & 1)
    odd.push_back(1 + 2 * ((ans[i] -
       1) / 2));
   else
    even.push_back(2 * (ans[i] / 2));
  }
  return {odd, even}; // odd[i] :
    length of palindrome centred at
    ith character
} // even[i]: length of palindrome
   centred after ith character (0-
   indexed)
```

11 Template

11.1 build system

```
"g++ -std=c++17 -Wshadow -Wall -
   fsanitize=address,undefined"
"-static-libasan -g3 -fno-omit-frame-
   pointer -fmax-errors=2"
```

11.2 template yatin

```
#include <bits/stdc++.h>
using namespace std;
#include <ext/pb_ds/assoc_container.
   hpp>
#include <ext/pb_ds/tree_policy.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
   >;

//use only if code is very short
#pragma GCC optimize("Ofast")
#pragma GCC target("avx,avx2,fma")
#pragma GCC optimization ("unroll-
   loops")

#define start_clock() auto start_time
   = std::chrono::
   high_resolution_clock::now();
#define measure() auto end_time = std
   ::chrono::high_resolution_clock::
   now(); cerr << (end_time -
   start_time)/std::chrono::
   milliseconds(1) << "ms" << endl;

typedef long long ll;

#ifndef ONLINE_JUDGE
```

```
template<typename T>
void __p(T a) {cout<<a;}

template<typename T, typename F>
void __p(pair<T, F> a) {
 cout<<"{";
 __p(a.first);
 cout<<",";
 __p(a.second);
 cout<<"}";
}
template<typename T>
void __p(vector<T> a) { //reuse for
    set and map
 cout<<"{";
 for(T i:a) __p(i),cout<<",";
   cout<<"}";
}
template<typename Arg1>
void __f(const char *name, Arg1 &&arg1
    ) {
 cout<<name<<" : ";
 __p(arg1);
 cout<<endl;
}
template<typename Arg1, typename ...
    Args>
void __f(const char *names, Arg1 &&
    arg1, Args &&... args) {
 const char *comma=strchr(names,',');
 cout.write(names,comma-names)<<" : ";
 __p(arg1);cout<<" | ";__f(comma+1,
     args...);
}
#define trace(...) cout<<"Line:"<<
    __LINE__<<" ", __f(#__VA_ARGS__,
    __VA_ARGS__)
#else
#define trace(...)
#endif

mt19937_64 rng(chrono::steady_clock::
    now().time_since_epoch().count());

struct custom_hash {
  static uint64_t splitmix64(uint64_t
      x) {
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) * 0
        xbf58476d1ce4e5b9;
    x = (x ^ (x >> 27)) * 0
        x94d049bb133111eb;
    return x ^ (x >> 31);
  }
  size_t operator()(uint64_t x) const
      {
    static const uint64_t FIXED_RANDOM
        =
        chrono::steady_clock::now().
            time_since_epoch().count();
    return splitmix64(x + FIXED_RANDOM)
        ;
  }
};

int main() {
 ios_base::sync_with_stdio(false);
   cin.tie(NULL);
 return 0;
}
```