Contents

1. Misc

1.1. Contest

1.1.1. Makefile

```
.PRECIOUS: ./p%

%: p%
	ulimit -s unlimited && ./$<
p%: p%.cpp
	g++ -o $@ $< -std=c++17 -Wall -Wextra -Wshadow \
		-fsanitize=address,undefined
```

1.2. How Did We Get Here?

1.2.1. Macros

Use vectorizations and math optimizations at your own peril.
For gcc≥9, there are [[likely]] and [[unlikely]] attributes.
Call gcc with `-fopt-info-optimized-missed-optall` for optimization info.

```
#define _GLIBCXX_DEBUG              1 // for debug mode
#define _GLIBCXX_SANITIZE_VECTOR 1 // for asan on vectors
#pragma GCC optimize("O3", "unroll-loops")
#pragma GCC optimize("fast-math")
#pragma GCC target("avx,avx2,abm,bmi,bmi2") // tip: `lscpu`
// before a loop
#pragma GCC unroll 16 // 0 or 1 -> no unrolling
#pragma GCC ivdep
```

1.2.2. Fast I/O

```cpp
struct scanner {
  static constexpr size_t LEN = 32 << 20;
  char *buf, *buf_ptr, *buf_end;
  scanner()
      : buf(new char[LEN]), buf_ptr(buf + LEN),
        buf_end(buf + LEN) {}
  ~scanner() { delete[] buf; }
  char getc() {
    if (buf_ptr == buf_end) [[unlikely]]
      buf_end = buf + fread_unlocked(buf, 1, LEN, stdin),
      buf_ptr = buf;
    return *(buf_ptr++);
  }
  char seek(char del) {
    char c;
    while ((c = getc()) < del) {}
    return c;
  }
  void read(int &t) {
    bool neg = false;
    char c = seek('-');
    if (c == '-') neg = true, t = 0;
    else t = c ^ '0';
    while ((c = getc()) >= '0') t = t * 10 + (c ^ '0');
    if (neg) t = -t;
  }
};
struct printer {
  static constexpr size_t CPI = 21, LEN = 32 << 20;
  char *buf, *buf_ptr, *buf_end, *tbuf;
  char *int_buf, *int_buf_end;
  printer()
      : buf(new char[LEN]), buf_ptr(buf),
        buf_end(buf + LEN), int_buf(new char[CPI + 1]()),
        int_buf_end(int_buf + CPI - 1) {}
  ~printer() {
    flush();
    delete[] buf, delete[] int_buf;
  }
  void flush() {
    fwrite_unlocked(buf, 1, buf_ptr - buf, stdout);
    buf_ptr = buf;
  }
  void write_(const char &c) {
    *buf_ptr = c;
    if (++buf_ptr == buf_end) [[unlikely]]
      flush();
  }
  void write_(const char *s) {
    for (; *s != '\0'; ++s) write_(*s);
  }
  void write(int x) {
    if (x < 0) write_('-'), x = -x;
    if (x == 0) [[unlikely]]
      return write_('0');
    for (tbuf = int_buf_end; x != 0; --tbuf, x /= 10)
      *tbuf = '0' + char(x % 10);
    write_(++tbuf);
  }
};
```

Kotlin

```kotlin
import java.io.*
import java.util.*

@JvmField val cin = System.`in`.bufferedReader()
@JvmField val cout = PrintWriter(System.out, false)
@JvmField var tokenizer: StringTokenizer = StringTokenizer("")
fun nextLine() = cin.readLine()!!
fun read(): String {
  while (!tokenizer.hasMoreTokens())
    tokenizer = StringTokenizer(nextLine())
  return tokenizer.nextToken()
}

// example
fun main() {
  val n = read().toInt()
  val a = DoubleArray(n) { read().toDouble() }
  cout.println("omg hi")
  cout.flush()
}
```

1.2.3. constexpr

Some default limits in gcc (7.x - trunk):
- constexpr recursion depth: 512
- constexpr loop iteration per function: $262\,144$
- constexpr operation count per function: $33\,554\,432$
- template recursion depth: 900 (gcc *might* segfault first)

```cpp
constexpr array<int, 10> fibonacci{[] {
  array<int, 10> a{};
  a[0] = a[1] = 1;
  for (int i = 2; i < 10; i++) a[i] = a[i - 1] + a[i - 2];
  return a;
}()};
static_assert(fibonacci[9] == 55, "CE");

template <typename F, typename INT, INT... S>
constexpr void for_constexpr(integer_sequence<INT, S...>,
                             F &&func) {
  int _[] = {(func(integral_constant<INT, S>{}), 0)...};
}
// example
template <typename... T> void print_tuple(tuple<T...> t) {
  for_constexpr(make_index_sequence<sizeof...(T)>{},
               [&](auto i) { cout << get<i>(t) << '\n'; });
}
```

1.2.4. Bump Allocator

```cpp


// global bump allocator
char mem[256 << 20]; // 256 MB
size_t rsp = sizeof mem;
void *operator new(size_t s) {
  assert(s < rsp); // MLE
  return (void *)&mem[rsp -= s];
}
void operator delete(void *) {}

// bump allocator for STL / pbds containers
char mem[256 << 20];
size_t rsp = sizeof mem;
template <typename T> struct bump {
  typedef T value_type;
  bump() {}
  template <typename U> bump(U, ...) {}
  T *allocate(size_t n) {
    rsp -= n * sizeof(T);
    rsp &= 0 - alignof(T);
    return (T *)(mem + rsp);
  }
  void deallocate(T *, size_t n) {}
};
```

1.3. Tools

1.3.1. Floating Point Binary Search

```cpp
union di {
  double d;
  ull i;
};
bool check(double);
// binary search in [L, R] with relative error 2^-eps
double binary_search(double L, double R, int eps) {
  di l = {L}, r = {R}, m;
  while (r.i - l.i > 1LL << (52 - eps)) {
    m.i = (l.i + r.i) >> 1;
```

```
11    if (check(m.d)) r = m;
      else l = m;
13  }
    return l.d;
15 }
```

1.3.2. SplitMix64

```
1 using ull = unsigned long long;
  inline ull splitmix64(ull x) {
3   // change to `static ull x = SEED;` for DRBG
    ull z = (x += 0x9E3779B97F4A7C15);
5   z = (z ^ (z >> 30)) * 0xBF58476D1CE4E5B9;
    z = (z ^ (z >> 27)) * 0x94D049BB133111EB;
7   return z ^ (z >> 31);
  }
```

1.3.3. <random>

```
1 #ifdef __unix__
  random_device rd;
3 mt19937_64 RNG(rd());
  #else
5 const auto SEED = chrono::high_resolution_clock::now()
                    .time_since_epoch()
7                   .count();
  mt19937_64 RNG(SEED);
9 #endif
  // random uint_fast64_t: RNG();
11 // uniform random of type T (int, double, ...) in [l, r]:
  // uniform_int_distribution<T> dist(l, r); dist(RNG);
```

1.3.4. x86 Stack Hack

```
1 constexpr size_t size = 200 << 20; // 200MiB
  int main() {
3   register long rsp asm("rsp");
    char *buf = new char[size];
5   asm("movq %0, %%rsp\n" ::"r"(buf + size));
    // do stuff
7   asm("movq %0, %%rsp\n" ::"r"(rsp));
    delete[] buf;
9 }
```

1.4. Algorithms

1.4.1. Bit Hacks

```
1 // next permutation of x as a bit sequence
  ull next_bits_permutation(ull x) {
3   ull c = __builtin_ctzll(x), r = x + (1ULL << c);
    return (r ^ x) >> (c + 2) | r;
5 }
  // iterate over all (proper) subsets of bitset s
7 void subsets(ull s) {
    for (ull x = s; x;) { --x &= s; /* do stuff */ }
9 }
```

1.4.2. Aliens Trick

```
1 // min dp[i] value and its i (smallest one)
  pll get_dp(int cost);
3 ll aliens(int k, int l, int r) {
    while (l != r) {
5     int m = (l + r) / 2;
      auto [f, s] = get_dp(m);
7     if (s == k) return f - m * k;
      if (s < k) r = m;
9     else l = m + 1;
    }
11  return get_dp(l).first - l * k;
  }
```

1.4.3. Hilbert Curve

```
1 ll hilbert(ll n, int x, int y) {
    ll res = 0;
3   for (ll s = n; s /= 2;) {
      int rx = !!(x & s), ry = !!(y & s);
5     res += s * s * ((3 * rx) ^ ry);
      if (ry == 0) {
7       if (rx == 1) x = s - 1 - x, y = s - 1 - y;
        swap(x, y);
9     }
    }
11  return res;
  }
```

1.4.4. Infinite Grid Knight Distance

```
1 ll get_dist(ll dx, ll dy) {
    if (++(dx = abs(dx)) > ++(dy = abs(dy))) swap(dx, dy);
3   if (dx == 1 && dy == 2) return 3;
    if (dx == 3 && dy == 3) return 4;
5   ll lb = max(dy / 2, (dx + dy) / 3);
    return ((dx ^ dy ^ lb) & 1) ? ++lb : lb;
7 }
```

1.4.5. Poker Hand

```
1


3


5


7 using namespace std;

9 struct hand {
    static constexpr auto rk = [] {
11    array<int, 256> x{};
      auto s = "23456789TJQKACDHS";
13    for (int i = 0; i < 17; i++) x[s[i]] = i % 13;
      return x;
15  }();
    vector<pair<int, int>> v;
17  vector<int> cnt, vf, vs;
    int type;
19  hand() : cnt(4), type(0) {}
    void add_card(char suit, char rank) {
21    ++cnt[rk[suit]];
      for (auto &[f, s] : v)
23      if (s == rk[rank]) return ++f, void();
      v.emplace_back(1, rk[rank]);
25  }
    void process() {
27    sort(v.rbegin(), v.rend());
      for (auto [f, s] : v) vf.push_back(f), vs.push_back(s);
29    bool str = 0, flu = find(all(cnt), 5) != cnt.end();
      if ((str = v.size() == 5))
31      for (int i = 1; i < 5; i++)
          if (vs[i] != vs[i - 1] + 1) str = 0;
33    if (vs == vector<int>{12, 3, 2, 1, 0})
        str = 1, vs = {3, 2, 1, 0, -1};
35    if (str && flu) type = 9;
      else if (vf[0] == 4) type = 8;
37    else if (vf[0] == 3 && vf[1] == 2) type = 7;
      else if (str || flu) type = 5 + flu;
39    else if (vf[0] == 3) type = 4;
      else if (vf[0] == 2) type = 2 + (vf[1] == 2);
41    else type = 1;
    }
43  bool operator<(const hand &b) const {
      return make_tuple(type, vf, vs) <
45           make_tuple(b.type, b.vf, b.vs);
    }
47 };
```

1.4.6. Longest Increasing Subsequence

```
1

3 template <class I> vi lis(const vector<I> &S) {
    if (S.empty()) return {};
5   vi prev(sz(S));
    typedef pair<I, int> p;
7   vector<p> res;
    rep(i, 0, sz(S)) {
9     // change 0 -> i for longest non-decreasing subsequence
      auto it = lower_bound(all(res), p{S[i], 0});
11    if (it == res.end())
        res.emplace_back(), it = res.end() - 1;
13    *it = {S[i], i};
      prev[i] = it == res.begin() ? 0 : (it - 1)->second;
15  }
    int L = sz(res), cur = res.back().second;
17  vi ans(L);
    while (L--) ans[L] = cur, cur = prev[cur];
19  return ans;
  }
```

1.4.7. Mo's Algorithm on Tree

```
1 void MoAlgoOnTree() {
    Dfs(0, -1);
3   vector<int> euler(tk);
    for (int i = 0; i < n; ++i) {
5     euler[tin[i]] = i;
      euler[tout[i]] = i;
7   }
    vector<int> l(q), r(q), qr(q), sp(q, -1);
```

```
 9   for (int i = 0; i < q; ++i) {
       if (tin[u[i]] > tin[v[i]]) swap(u[i], v[i]);
11     int z = GetLCA(u[i], v[i]);
       sp[i] = z[i];
13     if (z == u) l[i] = tin[u[i]], r[i] = tin[v[i]];
       else l[i] = tout[u[i]], r[i] = tin[v[i]];
15     qr[i] = i;
     }
17   sort(qr.begin(), qr.end(), [&](int i, int j) {
       if (l[i] / kB == l[j] / kB) return r[i] < r[j];
19     return l[i] / kB < l[j] / kB;
     });
21   vector<bool> used(n);
     // Add(v): add/remove v to/from the path based on used[v]
23   for (int i = 0, tl = 0, tr = -1; i < q; ++i) {
       while (tl < l[qr[i]]) Add(euler[tl++]);
25     while (tl > l[qr[i]]) Add(euler[--tl]);
       while (tr > r[qr[i]]) Add(euler[tr--]);
27     while (tr < r[qr[i]]) Add(euler[++tr]);
       // add/remove LCA(u, v) if necessary
29   }
   }
```

2. Data Structures

2.1. GNU PBDS

```cpp
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/priority_queue.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

// most std::map + order_of_key, find_by_order, split, join
template <typename T, typename U = null_type>
using ordered_map = tree<T, U, std::less<>, rb_tree_tag,
                         tree_order_statistics_node_update>;
// useful tags: rb_tree_tag, splay_tree_tag

template <typename T> struct myhash {
  size_t operator()(T x) const; // splitmix, bswap(x*R), ...
};
// most of std::unordered_map, but faster (needs good hash)
template <typename T, typename U = null_type>
using hash_table = gp_hash_table<T, U, myhash<T>>;

// most std::priority_queue + modify, erase, split, join
using heap = priority_queue<int, std::less<>>;
// useful tags: pairing_heap_tag, binary_heap_tag,
//              (rc_)?binomial_heap_tag, thin_heap_tag
```

2.2. 2D Partial Sums

```cpp
using vvi = vector<vector<int>>;
using vvll = vector<vector<ll>>;
using vll = vector<ll>;

struct PrefixSum2D {
  vvll pref;                    // 0-based 2-D prefix sum
  void build(const vvll &v) { // creates a copy
    int n = v.size(), m = v[0].size();
    pref.assign(n, vll(m, 0));
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
        pref[i][j] = v[i][j] + (i ? pref[i - 1][j] : 0) +
                     (j ? pref[i][j - 1] : 0) -
                     (i && j ? pref[i - 1][j - 1] : 0);
      }
    }
  }
  ll query(int ulx, int uly, int brx, int bry) const {
    ll ans = pref[brx][bry];
    if (ulx) ans -= pref[ulx - 1][bry];
    if (uly) ans -= pref[brx][uly - 1];
    if (ulx && uly) ans += pref[ulx - 1][uly - 1];
    return ans;
  }
  ll query(int ulx, int uly, int size) const {
    return query(ulx, uly, ulx + size - 1, uly + size - 1);
  }
};
struct PartialSum2D : PrefixSum2D {
  vvll diff; // 0 based
  int n, m;
  PartialSum2D(int _n, int _m) : n(_n), m(_m) {
    diff.assign(n + 1, vll(m + 1, 0));
  }
  // add c from [ulx,uly] to [brx,bry]
  void update(int ulx, int uly, int brx, int bry, ll c) {
    diff[ulx][uly] += c;
    diff[ulx][bry + 1] -= c;
    diff[brx + 1][uly] -= c;
    diff[brx + 1][bry + 1] += c;
  }
  void update(int ulx, int uly, int size, ll c) {
    int brx = ulx + size - 1;
    int bry = uly + size - 1;
    update(ulx, uly, brx, bry, c);
  }
  // process the grid using prefix sum
  void process() { this->build(diff); }
};
// usage
PrefixSum2D pref;
pref.build(v); // takes 2d 0-based vector as input
pref.query(x1, y1, x2, y2); // sum of region

PartialSum2D part(n, m);        // dimension of grid 0 based
part.update(x1, y1, x2, y2, 1); // add 1 in region
// must run after all updates
part.process(); // prefix sum on diff array
// only exists after processing
vvll &grid = part.pref;      // processed diff array
part.query(x1, y1, x2, y2); // gives sum of region
```

2.3. Segment Tree (ZKW)

```cpp
struct segtree {
```

```cpp
using T = int;
T f(T a, T b) { return a + b; } // any monoid operation
static constexpr T ID = 0;      // identity element
int n;
vector<T> v;
segtree(int n_) : n(n_), v(2 * n, ID) {}
segtree(vector<T> &a) : n(a.size()), v(2 * n, ID) {
  copy_n(a.begin(), n, v.begin() + n);
  for (int i = n - 1; i > 0; i--)
    v[i] = f(v[i * 2], v[i * 2 + 1]);
}
void update(int i, T x) {
  for (v[i += n] = x; i /= 2;)
    v[i] = f(v[i * 2], v[i * 2 + 1]);
}
T query(int l, int r) {
  T tl = ID, tr = ID;
  for (l += n, r += n; l < r; l /= 2, r /= 2) {
    if (l & 1) tl = f(tl, v[l++]);
    if (r & 1) tr = f(v[--r], tr);
  }
  return f(tl, tr);
}
};
```

2.4. Line Container

```cpp
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; }
};
// add: line y=kx+m, query: maximum y of given 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;
  }
};
```

2.5. Li-Chao Tree

```cpp
constexpr ll MAXN = 2e5, INF = 2e18;
struct Line {
  ll m, b;
  Line() : m(0), b(-INF) {}
  Line(ll _m, ll _b) : m(_m), b(_b) {}
  ll operator()(ll x) const { return m * x + b; }
};
struct Li_Chao {
  Line a[MAXN * 4];
  void insert(Line seg, int l, int r, int v = 1) {
    if (l == r) {
      if (seg(l) > a[v](l)) a[v] = seg;
      return;
    }
    int mid = (l + r) >> 1;
    if (a[v].m > seg.m) swap(a[v], seg);
    if (a[v](mid) < seg(mid)) {
      swap(a[v], seg);
      insert(seg, l, mid, v << 1);
    } else insert(seg, mid + 1, r, v << 1 | 1);
  }
  ll query(int x, int l, int r, int v = 1) {
    if (l == r) return a[v](x);
    int mid = (l + r) >> 1;
    if (x <= mid)
      return max(a[v](x), query(x, l, mid, v << 1));
    else
      return max(a[v](x), query(x, mid + 1, r, v << 1 | 1));
  }
};
```

2.6. Heavy-Light Decomposition

```
template <bool VALS_EDGES> struct HLD {
  int N, tim = 0;
  vector<vi> adj;
  vi par, siz, depth, rt, pos;
  Node *tree;
  HLD(vector<vi> adj_)
      : N(sz(adj_)), adj(adj_), par(N, -1), siz(N, 1),
        depth(N), rt(N), pos(N), tree(new Node(0, N)) {
    dfsSz(0);
    dfsHld(0);
  }
  void dfsSz(int v) {
    if (par[v] != -1)
      adj[v].erase(find(all(adj[v]), par[v]));
    for (int &u : adj[v]) {
      par[u] = v, depth[u] = depth[v] + 1;
      dfsSz(u);
      siz[v] += siz[u];
      if (siz[u] > siz[adj[v][0]]) swap(u, adj[v][0]);
    }
  }
  void dfsHld(int v) {
    pos[v] = tim++;
    for (int u : adj[v]) {
      rt[u] = (u == adj[v][0] ? rt[v] : u);
      dfsHld(u);
    }
  }
  template <class B> void process(int u, int v, B op) {
    for (; rt[u] != rt[v]; v = par[rt[v]]) {
      if (depth[rt[u]] > depth[rt[v]]) swap(u, v);
      op(pos[rt[v]], pos[v] + 1);
    }
    if (depth[u] > depth[v]) swap(u, v);
    op(pos[u] + VALS_EDGES, pos[v] + 1);
  }
  void modifyPath(int u, int v, int val) {
    process(u, v,
            [&](int l, int r) { tree->add(l, r, val); });
  }
  int queryPath(int u,
                int v) { // Modify depending on problem
    int res = -1e9;
    process(u, v, [&](int l, int r) {
      res = max(res, tree->query(l, r));
    });
    return res;
  }
  int querySubtree(int v) { // modifySubtree is similar
    return tree->query(pos[v] + VALS_EDGES,
                       pos[v] + siz[v]);
  }
};
```

2.7. Wavelet Matrix

```
#pragma GCC target("popcnt,bmi2")
#include <immintrin.h>

// T is unsigned. You might want to compress values first
template <typename T> struct wavelet_matrix {
  static_assert(is_unsigned_v<T>, "only unsigned T");
  struct bit_vector {
    static constexpr uint W = 64;
    uint n, cnt0;
    vector<ull> bits;
    vector<uint> sum;
    bit_vector(uint n_)
        : n(n_), bits(n / W + 1), sum(n / W + 1) {}
    void build() {
      for (uint j = 0; j != n / W; ++j)
        sum[j + 1] = sum[j] + _mm_popcnt_u64(bits[j]);
      cnt0 = rank0(n);
    }
    void set_bit(uint i) { bits[i / W] |= 1ULL << i % W; }
    bool operator[](uint i) const {
      return !!(bits[i / W] & 1ULL << i % W);
    }
    uint rank1(uint i) const {
      return sum[i / W] +
             _mm_popcnt_u64(_bzhi_u64(bits[i / W], i % W));
    }
    uint rank0(uint i) const { return i - rank1(i); }
  };
  uint n, lg;
  vector<bit_vector> b;
  wavelet_matrix(const vector<T> &a) : n(a.size()) {
```

```
    lg =
        __lg(max(*max_element(a.begin(), a.end()), T(1))) + 1;
    b.assign(lg, n);
    vector<T> cur = a, nxt(n);
    for (int h = lg; h--;) {
      for (uint i = 0; i < n; ++i)
        if (cur[i] & (T(1) << h)) b[h].set_bit(i);
      b[h].build();
      int il = 0, ir = b[h].cnt0;
      for (uint i = 0; i < n; ++i)
        nxt[(b[h][i] ? ir : il)++] = cur[i];
      swap(cur, nxt);
    }
  }
  T operator[](uint i) const {
    T res = 0;
    for (int h = lg; h--;)
      if (b[h][i])
        i += b[h].cnt0 - b[h].rank0(i), res |= T(1) << h;
      else i = b[h].rank0(i);
    return res;
  }
  // query k-th smallest (0-based) in a[l, r)
  T kth(uint l, uint r, uint k) const {
    T res = 0;
    for (int h = lg; h--;) {
      uint tl = b[h].rank0(l), tr = b[h].rank0(r);
      if (k >= tr - tl) {
        k -= tr - tl;
        l += b[h].cnt0 - tl;
        r += b[h].cnt0 - tr;
        res |= T(1) << h;
      } else l = tl, r = tr;
    }
    return res;
  }
  // count of i in [l, r) with a[i] < u
  uint count(uint l, uint r, T u) const {
    if (u >= T(1) << lg) return r - l;
    uint res = 0;
    for (int h = lg; h--;) {
      uint tl = b[h].rank0(l), tr = b[h].rank0(r);
      if (u & (T(1) << h)) {
        l += b[h].cnt0 - tl;
        r += b[h].cnt0 - tr;
        res += tr - tl;
      } else l = tl, r = tr;
    }
    return res;
  }
};
```

2.8. Link-Cut Tree

```
const int MXN = 100005;
const int MEM = 100005;

struct Splay {
  static Splay nil, mem[MEM], *pmem;
  Splay *ch[2], *f;
  int val, rev, size;
  Splay() : val(-1), rev(0), size(0) {
    f = ch[0] = ch[1] = &nil;
  }
  Splay(int _val) : val(_val), rev(0), size(1) {
    f = ch[0] = ch[1] = &nil;
  }
  bool isr() {
    return f->ch[0] != this && f->ch[1] != this;
  }
  int dir() { return f->ch[0] == this ? 0 : 1; }
  void setCh(Splay *c, int d) {
    ch[d] = c;
    if (c != &nil) c->f = this;
    pull();
  }
  void push() {
    if (rev) {
      swap(ch[0], ch[1]);
      if (ch[0] != &nil) ch[0]->rev ^= 1;
      if (ch[1] != &nil) ch[1]->rev ^= 1;
      rev = 0;
    }
  }
  void pull() {
    size = ch[0]->size + ch[1]->size + 1;
    if (ch[0] != &nil) ch[0]->f = this;
    if (ch[1] != &nil) ch[1]->f = this;
  }
} Splay::nil, Splay::mem[MEM], *Splay::pmem = Splay::mem;
Splay *nil = &Splay::nil;
```

```cpp
41  void rotate(Splay *x) {
      Splay *p = x->f;
43    int d = x->dir();
      if (!p->isr()) p->f->setCh(x, p->dir());
45    else x->f = p->f;
      p->setCh(x->ch[!d], d);
47    x->setCh(p, !d);
      p->pull();
49    x->pull();
    }

51
    vector<Splay *> splayVec;
53  void splay(Splay *x) {
      splayVec.clear();
55    for (Splay *q = x;; q = q->f) {
        splayVec.push_back(q);
57      if (q->isr()) break;
      }
59    reverse(begin(splayVec), end(splayVec));
      for (auto it : splayVec) it->push();
61    while (!x->isr()) {
        if (x->f->isr()) rotate(x);
63      else if (x->dir() == x->f->dir())
          rotate(x->f), rotate(x);
65      else rotate(x), rotate(x);
      }
67  }

69  Splay *access(Splay *x) {
      Splay *q = nil;
71    for (; x != nil; x = x->f) {
        splay(x);
73      x->setCh(q, 1);
        q = x;
75    }
      return q;
77  }
    void evert(Splay *x) {
79    access(x);
      splay(x);
81    x->rev ^= 1;
      x->push();
83    x->pull();
    }
85  void link(Splay *x, Splay *y) {
      //  evert(x);
87    access(x);
      splay(x);
89    evert(y);
      x->setCh(y, 1);
91  }
    void cut(Splay *x, Splay *y) {
93    //  evert(x);
      access(y);
95    splay(y);
      y->push();
97    y->ch[0] = y->ch[0]->f = nil;
    }

99
    int N, Q;
101 Splay *vt[MXN];

103 int ask(Splay *x, Splay *y) {
      access(x);
105   access(y);
      splay(x);
107   int res = x->f->val;
      if (res == -1) res = x->val;
109   return res;
    }

111
    int main(int argc, char **argv) {
113   scanf("%d%d", &N, &Q);
      for (int i = 1; i <= N; i++)
115     vt[i] = new (Splay::pmem++) Splay(i);
      while (Q--) {
117     char cmd[105];
        int u, v;
119     scanf("%s", cmd);
        if (cmd[1] == 'i') {
121       scanf("%d%d", &u, &v);
          link(vt[v], vt[u]);
123     } else if (cmd[0] == 'c') {
          scanf("%d", &v);
125       cut(vt[1], vt[v]);
        } else {
127       scanf("%d%d", &u, &v);
          int res = ask(vt[u], vt[v]);
129       printf("%d\n", res);
        }
131   }
    }
```

3. Graph

3.1. Modeling

- Maximum/Minimum flow with lower bound / Circulation problem
  1. Construct super source S and sink T.
  2. For each edge (x, y, l, u), connect $x \to y$ with capacity $u - l$.
  3. For each vertex v, denote by $in(v)$ the difference between the sum of incoming lower bounds and the sum of outgoing lower bounds.
  4. If $in(v) > 0$, connect $S \to v$ with capacity $in(v)$, otherwise, connect $v \to T$ with capacity $-in(v)$.
     - To maximize, connect $t \to s$ with capacity ∞ (skip this in circulation problem), and let f be the maximum flow from S to T. If $f \neq \sum_{v \in V, in(v) > 0} in(v)$, there's no solution. Otherwise, the maximum flow from s to t is the answer.
     - To minimize, let f be the maximum flow from S to T. Connect $t \to s$ with capacity ∞ and let the flow from S to T be f'. If $f + f' \neq \sum_{v \in V, in(v) > 0} in(v)$, there's no solution. Otherwise, f' is the answer.
  5. The solution of each edge e is $l_e + f_e$, where f_e corresponds to the flow of edge e on the graph.

- Construct minimum vertex cover from maximum matching M on bipartite graph (X, Y)
  1. Redirect every edge: $y \to x$ if $(x, y) \in M$, $x \to y$ otherwise.
  2. DFS from unmatched vertices in X.
  3. $x \in X$ is chosen iff x is unvisited.
  4. $y \in Y$ is chosen iff y is visited.

- Minimum cost cyclic flow
  1. Consruct super source S and sink T
  2. For each edge (x, y, c), connect $x \to y$ with $(cost, cap) = (c, 1)$ if $c > 0$, otherwise connect $y \to x$ with $(cost, cap) = (-c, 1)$
  3. For each edge with $c < 0$, sum these cost as K, then increase $d(y)$ by 1, decrease $d(x)$ by 1
  4. For each vertex v with $d(v) > 0$, connect $S \to v$ with $(cost, cap) = (0, d(v))$
  5. For each vertex v with $d(v) < 0$, connect $v \to T$ with $(cost, cap) = (0, -d(v))$
  6. Flow from S to T, the answer is the cost of the flow $C + K$

- Maximum density induced subgraph
  1. Binary search on answer, suppose we're checking answer T
  2. Construct a max flow model, let K be the sum of all weights
  3. Connect source $s \to v$, $v \in G$ with capacity K
  4. For each edge (u, v, w) in G, connect $u \to v$ and $v \to u$ with capacity w
  5. For $v \in G$, connect it with sink $v \to t$ with capacity $K + 2T - (\sum_{e \in E(v)} w(e)) - 2w(v)$
  6. T is a valid answer if the maximum flow $f < K|V|$

- Minimum weight edge cover
  1. For each $v \in V$ create a copy v', and connect $u' \to v'$ with weight $w(u, v)$.
  2. Connect $v \to v'$ with weight $2\mu(v)$, where $\mu(v)$ is the cost of the cheapest edge incident to v.
  3. Find the minimum weight perfect matching on G'.

- Project selection problem
  1. If $p_v > 0$, create edge (s, v) with capacity p_v; otherwise, create edge (v, t) with capacity $-p_v$.
  2. Create edge (u, v) with capacity w with w being the cost of choosing u without choosing v.
  3. The mincut is equivalent to the maximum profit of a subset of projects.

- 0/1 quadratic programming

$$\sum_x c_x x + \sum_y c_y \bar{y} + \sum_{xy} c_{xy} x\bar{y} + \sum_{xyx'y'} c_{xyx'y'}(x\bar{y} + x'\bar{y'})$$

can be minimized by the mincut of the following graph:
  1. Create edge (x, t) with capacity c_x and create edge (s, y) with capacity c_y.
  2. Create edge (x, y) with capacity c_{xy}.
  3. Create edge (x, y) and edge (x', y') with capacity $c_{xyx'y'}$.

3.2. Matching/Flows

3.2.1. Dinic's Algorithm

```cpp
struct Dinic {
  struct edge {
    int to, cap, flow, rev;
  };
  static constexpr int MAXN = 1000, MAXF = 1e9;
  vector<edge> v[MAXN];
  int top[MAXN], deep[MAXN], side[MAXN], s, t;
  void make_edge(int s, int t, int cap) {
    v[s].push_back({t, cap, 0, (int)v[t].size()});
    v[t].push_back({s, 0, 0, (int)v[s].size() - 1});
  }
  int dfs(int a, int flow) {
    if (a == t || !flow) return flow;
    for (int &i = top[a]; i < v[a].size(); i++) {
      edge &e = v[a][i];
      if (deep[a] + 1 == deep[e.to] && e.cap - e.flow) {
        int x = dfs(e.to, min(e.cap - e.flow, flow));
        if (x) {
          e.flow += x, v[e.to][e.rev].flow -= x;
          return x;
        }
      }
    }
    deep[a] = -1;
    return 0;
  }
  bool bfs() {
    queue<int> q;
    fill_n(deep, MAXN, 0);
    q.push(s), deep[s] = 1;
    int tmp;
    while (!q.empty()) {
      tmp = q.front(), q.pop();
      for (edge e : v[tmp])
        if (!deep[e.to] && e.cap != e.flow)
          deep[e.to] = deep[tmp] + 1, q.push(e.to);
    }
    return deep[t];
  }
  int max_flow(int _s, int _t) {
    s = _s, t = _t;
    int flow = 0, tflow;
    while (bfs()) {
      fill_n(top, MAXN, 0);
      while ((tflow = dfs(s, MAXF))) flow += tflow;
    }
    return flow;
  }
  void reset() {
    fill_n(side, MAXN, 0);
    for (auto &i : v) i.clear();
  }
};
```

3.2.2. Minimum Cost Flow

```cpp
struct MCF {
  struct edge {
    ll to, from, cap, flow, cost, rev;
  } *fromE[MAXN];
  vector<edge> v[MAXN];
  ll n, s, t, flows[MAXN], dis[MAXN], pi[MAXN], flowlim;
  void make_edge(int s, int t, ll cap, ll cost) {
    if (!cap) return;
    v[s].pb(edge{t, s, cap, 0LL, cost, v[t].size()});
    v[t].pb(edge{s, t, 0LL, 0LL, -cost, v[s].size() - 1});
  }
  bitset<MAXN> vis;
  void dijkstra() {
    vis.reset();
    __gnu_pbds::priority_queue<pair<ll, int>> q;
    vector<decltype(q)::point_iterator> its(n);
    q.push({0LL, s});
    while (!q.empty()) {
      int now = q.top().second;
      q.pop();
      if (vis[now]) continue;
      vis[now] = 1;
      ll ndis = dis[now] + pi[now];
      for (edge &e : v[now]) {
        if (e.flow == e.cap || vis[e.to]) continue;
        if (dis[e.to] > ndis + e.cost - pi[e.to]) {
          dis[e.to] = ndis + e.cost - pi[e.to];
          flows[e.to] = min(flows[now], e.cap - e.flow);
          fromE[e.to] = &e;
          if (its[e.to] == q.end())
            its[e.to] = q.push({-dis[e.to], e.to});
          else q.modify(its[e.to], {-dis[e.to], e.to});
        }
      }
    }
```

```
35        }
     }
37   bool AP(ll &flow) {
       fill_n(dis, n, INF);
39     fromE[s] = 0;
       dis[s] = 0;
41     flows[s] = flowlim - flow;
       dijkstra();
43     if (dis[t] == INF) return false;
       flow += flows[t];
45     for (edge *e = fromE[t]; e; e = fromE[e->from]) {
         e->flow += flows[t];
47       v[e->to][e->rev].flow -= flows[t];
       }
49     for (int i = 0; i < n; i++)
         pi[i] = min(pi[i] + dis[i], INF);
51     return true;
     }
53   pll solve(int _s, int _t, ll _flowlim = INF) {
       s = _s, t = _t, flowlim = _flowlim;
55     pll re;
       while (re.F != flowlim && AP(re.F));
57     for (int i = 0; i < n; i++)
         for (edge &e : v[i])
59         if (e.flow != 0) re.S += e.flow * e.cost;
       re.S /= 2;
61     return re;
     }
63   void init(int _n) {
       n = _n;
65     fill_n(pi, n, 0);
       for (int i = 0; i < n; i++) v[i].clear();
67   }
     void setpi(int s) {
69     fill_n(pi, n, INF);
       pi[s] = 0;
71     for (ll it = 0, flag = 1, tdis; flag && it < n; it++) {
         flag = 0;
73       for (int i = 0; i < n; i++)
           if (pi[i] != INF)
75           for (edge &e : v[i])
               if (e.cap && (tdis = pi[i] + e.cost) < pi[e.to])
77               pi[e.to] = tdis, flag = 1;
       }
79   }
   };
```

3.2.3. Gomory-Hu Tree

Requires: Dinic's Algorithm

```
1
 3  int e[MAXN][MAXN];
    int p[MAXN];
 5  Dinic D; // original graph
    void gomory_hu() {
 7    fill(p, p + n, 0);
      fill(e[0], e[n], INF);
 9    for (int s = 1; s < n; s++) {
        int t = p[s];
11      Dinic F = D;
        int tmp = F.max_flow(s, t);
13      for (int i = 1; i < s; i++)
          e[s][i] = e[i][s] = min(tmp, e[t][i]);
15      for (int i = s + 1; i <= n; i++)
          if (p[i] == t && F.side[i]) p[i] = s;
17    }
    }
```

3.2.4. Global Minimum Cut

```
1
 3  // weights is an adjacency matrix, undirected
    pair<int, vi> getMinCut(vector<vi> &weights) {
 5    int N = sz(weights);
      vi used(N), cut, best_cut;
 7    int best_weight = -1;

 9    for (int phase = N - 1; phase >= 0; phase--) {
        vi w = weights[0], added = used;
11      int prev, k = 0;
        rep(i, 0, phase) {
13        prev = k;
          k = -1;
15        rep(j, 1, N) if (!added[j] &&
                          (k == -1 || w[j] > w[k])) k = j;
17        if (i == phase - 1) {
            rep(j, 0, N) weights[prev][j] += weights[k][j];
19          rep(j, 0, N) weights[j][prev] = weights[prev][j];
            used[k] = true;
21          cut.push_back(k);
            if (best_weight == -1 || w[k] < best_weight) {
```

```
23            best_cut = cut;
              best_weight = w[k];
25          }
          } else {
27          rep(j, 0, N) w[j] += weights[k][j];
            added[k] = true;
29        }
        }
31    }
      return {best_weight, best_cut};
33  }
```

3.2.5. Bipartite Minimum Cover

Requires: Dinic's Algorithm

```
1
 3  // maximum independent set = all vertices not covered
    // x : [0, n), y : [0, m]
 5  struct Bipartite_vertex_cover {
      Dinic D;
 7    int n, m, s, t, x[maxn], y[maxn];
      void make_edge(int x, int y) { D.make_edge(x, y + n, 1); }
 9    int matching() {
        int re = D.max_flow(s, t);
11      for (int i = 0; i < n; i++)
          for (Dinic::edge &e : D.v[i])
13          if (e.to != s && e.flow == 1) {
              x[i] = e.to - n, y[e.to - n] = i;
15            break;
            }
17      return re;
      }
19    // init() and matching() before use
      void solve(vector<int> &vx, vector<int> &vy) {
21      bitset<maxn * 2 + 10> vis;
        queue<int> q;
23      for (int i = 0; i < n; i++)
          if (x[i] == -1) q.push(i), vis[i] = 1;
25      while (!q.empty()) {
          int now = q.front();
27        q.pop();
          if (now < n) {
29          for (Dinic::edge &e : D.v[now])
              if (e.to != s && e.to - n != x[now] && !vis[e.to])
31              vis[e.to] = 1, q.push(e.to);
          } else {
33          if (!vis[y[now - n]])
              vis[y[now - n]] = 1, q.push(y[now - n]);
35        }
        }
37      for (int i = 0; i < n; i++)
          if (!vis[i]) vx.pb(i);
39      for (int i = 0; i < m; i++)
          if (vis[i + n]) vy.pb(i);
41    }
      void init(int _n, int _m) {
43      n = _n, m = _m, s = n + m, t = s + 1;
        for (int i = 0; i < n; i++)
45        x[i] = -1, D.make_edge(s, i, 1);
        for (int i = 0; i < m; i++)
47        y[i] = -1, D.make_edge(i + n, t, 1);
      }
49  };
```

3.2.6. Edmonds' Algorithm

```
1
 3  struct Edmonds {
      int n, T;
 5    vector<vector<int>> g;
      vector<int> pa, p, used, base;
 7    Edmonds(int n)
          : n(n), T(0), g(n), pa(n, -1), p(n), used(n),
 9          base(n) {}
      void add(int a, int b) {
11      g[a].push_back(b);
        g[b].push_back(a);
13    }
      int getBase(int i) {
15      while (i != base[i])
          base[i] = base[base[i]], i = base[i];
17      return i;
      }
19    vector<int> toJoin;
      void mark_path(int v, int x, int b, vector<int> &path) {
21      for (; getBase(v) != b; v = p[x]) {
          p[v] = x, x = pa[v];
23        toJoin.push_back(v);
          toJoin.push_back(x);
25        if (!used[x]) used[x] = ++T, path.push_back(x);
        }
```

```cpp
27      }
     bool go(int v) {
29      for (int x : g[v]) {
          int b, bv = getBase(v), bx = getBase(x);
31        if (bv == bx) {
            continue;
33        } else if (used[x]) {
            vector<int> path;
35          toJoin.clear();
            if (used[bx] < used[bv])
37            mark_path(v, x, b = bx, path);
            else mark_path(x, v, b = bv, path);
39          for (int z : toJoin) base[getBase(z)] = b;
            for (int z : path)
41            if (go(z)) return 1;
          } else if (p[x] == -1) {
43          p[x] = v;
            if (pa[x] == -1) {
45            for (int y; x != -1; x = v)
                y = p[x], v = pa[y], pa[x] = y, pa[y] = x;
47            return 1;
            }
49          if (!used[pa[x]]) {
              used[pa[x]] = ++T;
51            if (go(pa[x])) return 1;
            }
53        }
        }
55      return 0;
      }
57    void init_dfs() {
        for (int i = 0; i < n; i++)
59        used[i] = 0, p[i] = -1, base[i] = i;
      }
61    bool dfs(int root) {
        used[root] = ++T;
63      return go(root);
      }
65    void match() {
        int ans = 0;
67      for (int v = 0; v < n; v++)
          for (int x : g[v])
69          if (pa[v] == -1 && pa[x] == -1) {
              pa[v] = x, pa[x] = v, ans++;
71            break;
            }
73      init_dfs();
        for (int i = 0; i < n; i++)
75        if (pa[i] == -1 && dfs(i)) ans++, init_dfs();
        cout << ans * 2 << "\n";
77      for (int i = 0; i < n; i++)
          if (pa[i] > i)
79          cout << i + 1 << " " << pa[i] + 1 << "\n";
      }
81  };
```

3.2.7. Minimum Weight Matching

```cpp
1  struct Graph {
     static const int MAXN = 105;
3    int n, e[MAXN][MAXN];
     int match[MAXN], d[MAXN], onstk[MAXN];
5    vector<int> stk;
     void init(int _n) {
7      n = _n;
       for (int i = 0; i < n; i++)
9        for (int j = 0; j < n; j++)
           // change to appropriate infinity
11         // if not complete graph
           e[i][j] = 0;
13   }
     void add_edge(int u, int v, int w) {
15     e[u][v] = e[v][u] = w;
     }
17   bool SPFA(int u) {
       if (onstk[u]) return true;
19     stk.push_back(u);
       onstk[u] = 1;
21     for (int v = 0; v < n; v++) {
         if (u != v && match[u] != v && !onstk[v]) {
23         int m = match[v];
           if (d[m] > d[u] - e[v][m] + e[u][v]) {
25           d[m] = d[u] - e[v][m] + e[u][v];
             onstk[v] = 1;
27           stk.push_back(v);
             if (SPFA(m)) return true;
29           stk.pop_back();
             onstk[v] = 0;
31         }
         }
33     }
       onstk[u] = 0;
35     stk.pop_back();
       return false;
```

```cpp
37    }
    int solve() {
39    for (int i = 0; i < n; i += 2) {
        match[i] = i + 1;
41      match[i + 1] = i;
      }
43    while (true) {
        int found = 0;
45      for (int i = 0; i < n; i++) onstk[i] = d[i] = 0;
        for (int i = 0; i < n; i++) {
47        stk.clear();
          if (!onstk[i] && SPFA(i)) {
49          found = 1;
            while (stk.size() >= 2) {
51            int u = stk.back();
              stk.pop_back();
53            int v = stk.back();
              stk.pop_back();
55            match[u] = v;
              match[v] = u;
57          }
          }
59      }
        if (!found) break;
61    }
      int ret = 0;
63    for (int i = 0; i < n; i++) ret += e[i][match[i]];
      ret /= 2;
65    return ret;
    }
67 } graph;
```

3.2.8. Stable Marriage

```cpp
1  // normal stable marriage problem
   /* input:
3  3
   Albert Laura Nancy Marcy
5  Brad Marcy Nancy Laura
   Chuck Laura Marcy Nancy
7  Laura Chuck Albert Brad
   Marcy Albert Chuck Brad
9  Nancy Brad Albert Chuck
   */
11
13 using namespace std;
   const int MAXN = 505;
15
   int n;
17 int favor[MAXN][MAXN]; // favor[boy_id][rank] = girl_id;
   int order[MAXN][MAXN]; // order[girl_id][boy_id] = rank;
19 int current[MAXN];     // current[boy_id] = rank;
   // boy_id will pursue current[boy_id] girl.
21 int girl_current[MAXN]; // girl[girl_id] = boy_id;
23 void initialize() {
     for (int i = 0; i < n; i++) {
25     current[i] = 0;
       girl_current[i] = n;
27     order[i][n] = n;
     }
29 }
31 map<string, int> male, female;
   string bname[MAXN], gname[MAXN];
33 int fit = 0;
35 void stable_marriage() {
37   queue<int> que;
     for (int i = 0; i < n; i++) que.push(i);
39   while (!que.empty()) {
       int boy_id = que.front();
41     que.pop();
43     int girl_id = favor[boy_id][current[boy_id]];
       current[boy_id]++;
45
       if (order[girl_id][boy_id] <
47         order[girl_id][girl_current[girl_id]]) {
         if (girl_current[girl_id] < n)
49         que.push(girl_current[girl_id]);
         girl_current[girl_id] = boy_id;
51     } else {
         que.push(boy_id);
53     }
     }
55 }
57 int main() {
     cin >> n;
59
     for (int i = 0; i < n; i++) {
```

```
61    string p, t;
      cin >> p;
63    male[p] = i;
      bname[i] = p;
65    for (int j = 0; j < n; j++) {
        cin >> t;
67      if (!female.count(t)) {
          gname[fit] = t;
69        female[t] = fit++;
        }
71      favor[i][j] = female[t];
      }
73  }

75  for (int i = 0; i < n; i++) {
      string p, t;
77    cin >> p;
      for (int j = 0; j < n; j++) {
79      cin >> t;
        order[female[p]][male[t]] = j;
81    }
    }
83
    initialize();
85    stable_marriage();

87    for (int i = 0; i < n; i++) {
      cout << bname[i] << " "
89         << gname[favor[i][current[i] - 1]] << endl;
    }
91  }
```

3.2.9. Kuhn-Munkres algorithm

```
1  // Maximum Weight Perfect Bipartite Matching
   // Detect non-perfect-matching:
3  // 1. set all edge[i][j] as INF
   // 2. if solve() >= INF, it is not perfect matching.
5
   typedef long long ll;
7  struct KM {
     static const int MAXN = 1050;
9    static const ll INF = 1LL << 60;
     int n, match[MAXN], vx[MAXN], vy[MAXN];
11   ll edge[MAXN][MAXN], lx[MAXN], ly[MAXN], slack[MAXN];
     void init(int _n) {
13     n = _n;
       for (int i = 0; i < n; i++)
15       for (int j = 0; j < n; j++) edge[i][j] = 0;
     }
17   void add_edge(int x, int y, ll w) { edge[x][y] = w; }
     bool DFS(int x) {
19     vx[x] = 1;
       for (int y = 0; y < n; y++) {
21       if (vy[y]) continue;
         if (lx[x] + ly[y] > edge[x][y]) {
23         slack[y] =
           min(slack[y], lx[x] + ly[y] - edge[x][y]);
25       } else {
           vy[y] = 1;
27         if (match[y] == -1 || DFS(match[y])) {
             match[y] = x;
29           return true;
           }
31       }
       }
33     return false;
     }
35   ll solve() {
       fill(match, match + n, -1);
37     fill(lx, lx + n, -INF);
       fill(ly, ly + n, 0);
39     for (int i = 0; i < n; i++)
         for (int j = 0; j < n; j++)
41         lx[i] = max(lx[i], edge[i][j]);
       for (int i = 0; i < n; i++) {
43       fill(slack, slack + n, INF);
         while (true) {
45         fill(vx, vx + n, 0);
           fill(vy, vy + n, 0);
47         if (DFS(i)) break;
           ll d = INF;
49         for (int j = 0; j < n; j++)
             if (!vy[j]) d = min(d, slack[j]);
51         for (int j = 0; j < n; j++) {
             if (vx[j]) lx[j] -= d;
53           if (vy[j]) ly[j] += d;
             else slack[j] -= d;
55         }
         }
57     }
       ll res = 0;
59     for (int i = 0; i < n; i++) {
         res += edge[match[i]][i];
```

```
61      }
        return res;
63    }
    } graph;
```

3.3. Shortest Path Faster Algorithm

```
1  struct SPFA {
     static const int maxn = 1010, INF = 1e9;
3    int dis[maxn];
     bitset<maxn> inq, inneg;
5    queue<int> q, tq;
     vector<pii> v[maxn];
7    void make_edge(int s, int t, int w) {
       v[s].emplace_back(t, w);
9    }
     void dfs(int a) {
11     inneg[a] = 1;
       for (pii i : v[a])
13       if (!inneg[i.F]) dfs(i.F);
     }
15   bool solve(int n, int s) { // true if have neg-cycle
       for (int i = 0; i <= n; i++) dis[i] = INF;
17     dis[s] = 0, q.push(s);
       for (int i = 0; i < n; i++) {
19       inq.reset();
         int now;
21       while (!q.empty()) {
           now = q.front(), q.pop();
23         for (pii &i : v[now]) {
             if (dis[i.F] > dis[now] + i.S) {
25             dis[i.F] = dis[now] + i.S;
               if (!inq[i.F]) tq.push(i.F), inq[i.F] = 1;
27           }
           }
29       }
         q.swap(tq);
31     }
       bool re = !q.empty();
33     inneg.reset();
       while (!q.empty()) {
35       if (!inneg[q.front()]) dfs(q.front());
         q.pop();
37     }
       return re;
39   }
     void reset(int n) {
41     for (int i = 0; i <= n; i++) v[i].clear();
     }
43 };
```

3.4. Strongly Connected Components

```
1  struct TarjanScc {
     int n, step;
3    vector<int> time, low, instk, stk;
     vector<vector<int>> e, scc;
5    TarjanScc(int n_)
         : n(n_), step(0), time(n), low(n), instk(n), e(n) {}
7    void add_edge(int u, int v) { e[u].push_back(v); }
     void dfs(int x) {
9      time[x] = low[x] = ++step;
       stk.push_back(x);
11     instk[x] = 1;
       for (int y : e[x])
13       if (!time[y]) {
           dfs(y);
15         low[x] = min(low[x], low[y]);
         } else if (instk[y]) {
17         low[x] = min(low[x], time[y]);
         }
19     if (time[x] == low[x]) {
         scc.emplace_back();
21       for (int y = -1; y != x;) {
           y = stk.back();
23         stk.pop_back();
           instk[y] = 0;
25         scc.back().push_back(y);
         }
27     }
     }
29   void solve() {
       for (int i = 0; i < n; i++)
31       if (!time[i]) dfs(i);
       reverse(scc.begin(), scc.end());
33     // scc in topological order
     }
35 };
```

3.4.1. 2-Satisfiability

Requires: Strongly Connected Components

```
 1
    // 1 based, vertex in SCC = MAXN * 2
 3  // (not i) is i + n
    struct two_SAT {
 5    int n, ans[MAXN];
      SCC S;
 7    void imply(int a, int b) { S.make_edge(a, b); }
      bool solve(int _n) {
 9      n = _n;
        S.solve(n * 2);
11      for (int i = 1; i <= n; i++) {
          if (S.scc[i] == S.scc[i + n]) return false;
13        ans[i] = (S.scc[i] < S.scc[i + n]);
        }
15      return true;
      }
17    void init(int _n) {
        n = _n;
19      fill_n(ans, n + 1, 0);
        S.init(n * 2);
21    }
    } SAT;
23
```

3.5. Biconnected Components

3.5.1. Articulation Points

```
 1  void dfs(int x, int p) {
      tin[x] = low[x] = ++t;
 3    int ch = 0;
      for (auto u : g[x])
 5      if (u.first != p) {
          if (!ins[u.second])
 7          st.push(u.second), ins[u.second] = true;
          if (tin[u.first]) {
 9          low[x] = min(low[x], tin[u.first]);
            continue;
11        }
          ++ch;
13        dfs(u.first, x);
          low[x] = min(low[x], low[u.first]);
15        if (low[u.first] >= tin[x]) {
            cut[x] = true;
17          ++sz;
            while (true) {
19            int e = st.top();
              st.pop();
21            bcc[e] = sz;
              if (e == u.second) break;
23          }
          }
25      }
      if (ch == 1 && p == -1) cut[x] = false;
27  }
```

3.5.2. Bridges

```
 1  // if there are multi-edges, then they are not bridges
    void dfs(int x, int p) {
 3    tin[x] = low[x] = ++t;
      st.push(x);
 5    for (auto u : g[x])
        if (u.first != p) {
 7        if (tin[u.first]) {
            low[x] = min(low[x], tin[u.first]);
 9          continue;
          }
11        dfs(u.first, x);
          low[x] = min(low[x], low[u.first]);
13        if (low[u.first] == tin[u.first]) br[u.second] = true;
        }
15    if (tin[x] == low[x]) {
        ++sz;
17      while (st.size()) {
          int u = st.top();
19        st.pop();
          bcc[u] = sz;
21        if (u == x) break;
        }
23    }
    }
```

3.6. Triconnected Components

```
 1
 3
    // requires a union-find data structure
 5  struct ThreeEdgeCC {
      int V, ind;
 7    vector<int> id, pre, post, low, deg, path;
```

```
      vector<vector<int>> components;
 9    UnionFind uf;
      template <class Graph>
11    void dfs(const Graph &G, int v, int prev) {
        pre[v] = ++ind;
13      for (int w : G[v])
          if (w != v) {
15          if (w == prev) {
              prev = -1;
17            continue;
            }
19          if (pre[w] != -1) {
              if (pre[w] < pre[v]) {
21              deg[v]++;
                low[v] = min(low[v], pre[w]);
23            } else {
                deg[v]--;
25              int &u = path[v];
                for (; u != -1 && pre[u] <= pre[w] &&
27                    pre[w] <= post[u];) {
                  uf.join(v, u);
29                deg[v] += deg[u];
                  u = path[u];
31              }
              }
33            continue;
            }
35          dfs(G, w, v);
            if (path[w] == -1 && deg[w] <= 1) {
37            deg[v] += deg[w];
              low[v] = min(low[v], low[w]);
39            continue;
            }
41          if (deg[w] == 0) w = path[w];
            if (low[v] > low[w]) {
43            low[v] = min(low[v], low[w]);
              swap(w, path[v]);
45          }
            for (; w != -1; w = path[w]) {
47            uf.join(v, w);
              deg[v] += deg[w];
49          }
          }
51      post[v] = ind;
      }
53    template <class Graph>
      ThreeEdgeCC(const Graph &G)
55        : V(G.size()), ind(-1), id(V, -1), pre(V, -1),
            post(V), low(V, INT_MAX), deg(V, 0), path(V, -1),
57          uf(V) {
        for (int v = 0; v < V; v++)
59        if (pre[v] == -1) dfs(G, v, -1);
        components.reserve(uf.cnt);
61      for (int v = 0; v < V; v++)
          if (uf.find(v) == v) {
63          id[v] = components.size();
            components.emplace_back(1, v);
65          components.back().reserve(uf.getSize(v));
          }
67      for (int v = 0; v < V; v++)
          if (id[v] == -1)
69          components[id[v] = id[uf.find(v)]].push_back(v);
      }
71  };
```

3.7. Centroid Decomposition

```
 1  void get_center(int now) {
      v[now] = true;
 3    vtx.push_back(now);
      sz[now] = 1;
 5    mx[now] = 0;
      for (int u : G[now])
 7      if (!v[u]) {
          get_center(u);
 9        mx[now] = max(mx[now], sz[u]);
          sz[now] += sz[u];
11      }
    }
13  void get_dis(int now, int d, int len) {
      dis[d][now] = cnt;
15    v[now] = true;
      for (auto u : G[now])
17      if (!v[u.first]) { get_dis(u, d, len + u.second); }
    }
19  void dfs(int now, int fa, int d) {
      get_center(now);
21    int c = -1;
      for (int i : vtx) {
23      if (max(mx[i], (int)vtx.size() - sz[i]) <=
            (int)vtx.size() / 2)
25        c = i;
        v[i] = false;
27    }
```

```
29    get_dis(c, d, 0);
      for (int i : vtx) v[i] = false;
31    v[c] = true;
      vtx.clear();
33    dep[c] = d;
      p[c] = fa;
35    for (auto u : G[c])
        if (u.first != fa && !v[u.first]) {
37        dfs(u.first, c, d + 1);
        }
    }
```

3.8.   Minimum Mean Cycle

```
1
3  // d[i][j] == 0 if {i,j} !in E
   long long d[1003][1003], dp[1003][1003];
5
   pair<long long, long long> MMWC() {
7    memset(dp, 0x3f, sizeof(dp));
     for (int i = 1; i <= n; ++i) dp[0][i] = 0;
9    for (int i = 1; i <= n; ++i) {
       for (int j = 1; j <= n; ++j) {
11       for (int k = 1; k <= n; ++k) {
           dp[i][k] = min(dp[i - 1][j] + d[j][k], dp[i][k]);
13       }
       }
15   }
     long long au = 1ll << 31, ad = 1;
17   for (int i = 1; i <= n; ++i) {
       if (dp[n][i] == 0x3f3f3f3f3f3f3f3f) continue;
19     long long u = 0, d = 1;
       for (int j = n - 1; j >= 0; --j) {
21       if ((dp[n][i] - dp[j][i]) * d > u * (n - j)) {
           u = dp[n][i] - dp[j][i];
23         d = n - j;
         }
25     }
       if (u * ad < au * d) au = u, ad = d;
27   }
     long long g = __gcd(au, ad);
29   return make_pair(au / g, ad / g);
   }
```

3.9.   Directed MST

```
1  template <typename T> struct DMST {
     T g[maxn][maxn], fw[maxn];
3    int n, fr[maxn];
     bool vis[maxn], inc[maxn];
5    void clear() {
       for (int i = 0; i < maxn; ++i) {
7        for (int j = 0; j < maxn; ++j) g[i][j] = inf;
         vis[i] = inc[i] = false;
9      }
     }
11   void addedge(int u, int v, T w) {
       g[u][v] = min(g[u][v], w);
13   }
     T operator()(int root, int _n) {
15     n = _n;
       if (dfs(root) != n) return -1;
17     T ans = 0;
       while (true) {
19       for (int i = 1; i <= n; ++i) fw[i] = inf, fr[i] = i;
         for (int i = 1; i <= n; ++i)
21         if (!inc[i]) {
             for (int j = 1; j <= n; ++j) {
23             if (!inc[j] && i != j && g[j][i] < fw[i]) {
                 fw[i] = g[j][i];
25               fr[i] = j;
               }
27           }
           }
29       int x = -1;
         for (int i = 1; i <= n; ++i)
31         if (i != root && !inc[i]) {
             int j = i, c = 0;
33           while (j != root && fr[j] != i && c <= n)
               ++c, j = fr[j];
35           if (j == root || c > n) continue;
             else {
37             x = i;
               break;
39           }
           }
41       if (!~x) {
           for (int i = 1; i <= n; ++i)
43           if (i != root && !inc[i]) ans += fw[i];
           return ans;
45       }
         int y = x;
```

```
47      for (int i = 1; i <= n; ++i) vis[i] = false;
        do {
49        ans += fw[y];
          y = fr[y];
51        vis[y] = inc[y] = true;
        } while (y != x);
53      inc[x] = false;
        for (int k = 1; k <= n; ++k)
55        if (vis[k]) {
            for (int j = 1; j <= n; ++j)
57            if (!vis[j]) {
                if (g[x][j] > g[k][j]) g[x][j] = g[k][j];
59              if (g[j][k] < inf &&
                    g[j][k] - fw[k] < g[j][x])
61                g[j][x] = g[j][k] - fw[k];
              }
63          }
        }
65      return ans;
      }
67    int dfs(int now) {
        int r = 1;
69      vis[now] = true;
        for (int i = 1; i <= n; ++i)
71        if (g[now][i] < inf && !vis[i]) r += dfs(i);
        return r;
73    }
   };
```

3.10.   Maximum Clique

```
1  // source: KACTL
3  typedef vector<bitset<200>> vb;
   struct Maxclique {
5    double limit = 0.025, pk = 0;
     struct Vertex {
7      int i, d = 0;
     };
9    typedef vector<Vertex> vv;
     vb e;
11   vv V;
     vector<vi> C;
13   vi qmax, q, S, old;
     void init(vv &r) {
15     for (auto &v : r) v.d = 0;
       for (auto &v : r)
17       for (auto j : r) v.d += e[v.i][j.i];
       sort(all(r), [](auto a, auto b) { return a.d > b.d; });
19     int mxD = r[0].d;
       rep(i, 0, sz(r)) r[i].d = min(i, mxD) + 1;
21   }
     void expand(vv &R, int lev = 1) {
23     S[lev] += S[lev - 1] - old[lev];
       old[lev] = S[lev - 1];
25     while (sz(R)) {
         if (sz(q) + R.back().d <= sz(qmax)) return;
27       q.push_back(R.back().i);
         vv T;
29       for (auto v : R)
           if (e[R.back().i][v.i]) T.push_back({v.i});
31       if (sz(T)) {
           if (S[lev]++ / ++pk < limit) init(T);
33         int j = 0, mxk = 1,
               mnk = max(sz(qmax) - sz(q) + 1, 1);
35         C[1].clear(), C[2].clear();
           for (auto v : T) {
37           int k = 1;
             auto f = [&](int i) { return e[v.i][i]; };
39           while (any_of(all(C[k]), f)) k++;
             if (k > mxk) mxk = k, C[mxk + 1].clear();
41           if (k < mnk) T[j++].i = v.i;
             C[k].push_back(v.i);
43         }
           if (j > 0) T[j - 1].d = 0;
45         rep(k, mnk, mxk + 1) for (int i : C[k]) T[j].i = i,
                                                   T[j++].d =
47                                                       k;
           expand(T, lev + 1);
49       } else if (sz(q) > sz(qmax)) qmax = q;
         q.pop_back(), R.pop_back();
51     }
     }
53   vi maxClique() {
       init(V), expand(V);
55     return qmax;
     }
     Maxclique(vb conn)
57       : e(conn), C(sz(e) + 1), S(sz(C)), old(S) {
       rep(i, 0, sz(e)) V.push_back({i});
59   }
61 };
```

3.11. Dominator Tree

```
 1  // idom[n] is the unique node that strictly dominates n but
    // does not strictly dominate any other node that strictly
 3  // dominates n. idom[n] = 0 if n is entry or the entry
    // cannot reach n.
 5  struct DominatorTree {
      static const int MAXN = 200010;
 7    int n, s;
      vector<int> g[MAXN], pred[MAXN];
 9    vector<int> cov[MAXN];
      int dfn[MAXN], nfd[MAXN], ts;
11    int par[MAXN];
      int sdom[MAXN], idom[MAXN];
13    int mom[MAXN], mn[MAXN];

15    inline bool cmp(int u, int v) { return dfn[u] < dfn[v]; }

17    int eval(int u) {
        if (mom[u] == u) return u;
19      int res = eval(mom[u]);
        if (cmp(sdom[mn[mom[u]]], sdom[mn[u]]))
21        mn[u] = mn[mom[u]];
        return mom[u] = res;
23    }

25    void init(int _n, int _s) {
        n = _n;
27      s = _s;
        REP1(i, 1, n) {
29        g[i].clear();
          pred[i].clear();
31        idom[i] = 0;
        }
33    }
      void add_edge(int u, int v) {
35      g[u].push_back(v);
        pred[v].push_back(u);
37    }
      void DFS(int u) {
39      ts++;
        dfn[u] = ts;
41      nfd[ts] = u;
        for (int v : g[u])
43        if (dfn[v] == 0) {
            par[v] = u;
45          DFS(v);
          }
47    }
      void build() {
49      ts = 0;
        REP1(i, 1, n) {
51        dfn[i] = nfd[i] = 0;
          cov[i].clear();
53        mom[i] = mn[i] = sdom[i] = i;
        }
55      DFS(s);
        for (int i = ts; i >= 2; i--) {
57        int u = nfd[i];
          if (u == 0) continue;
59        for (int v : pred[u])
            if (dfn[v]) {
61            eval(v);
              if (cmp(sdom[mn[v]], sdom[u]))
63              sdom[u] = sdom[mn[v]];
            }
65        cov[sdom[u]].push_back(u);
          mom[u] = par[u];
67        for (int w : cov[par[u]]) {
            eval(w);
69          if (cmp(sdom[mn[w]], par[u])) idom[w] = mn[w];
            else idom[w] = par[u];
71        }
          cov[par[u]].clear();
73      }
        REP1(i, 2, ts) {
75        int u = nfd[i];
          if (u == 0) continue;
77        if (idom[u] != sdom[u]) idom[u] = idom[idom[u]];
        }
79    }
    } dom;
```

3.12. Manhattan Distance MST

```
 1
 3  // returns [(dist, from, to), ...]
    // then do normal mst afterwards
 5  typedef Point<int> P;
    vector<array<int, 3>> manhattanMST(vector<P> ps) {
 7    vi id(sz(ps));
      iota(all(id), 0);
 9    vector<array<int, 3>> edges;
      rep(k, 0, 4) {
11      sort(all(id), [&](int i, int j) {
          return (ps[i] - ps[j]).x < (ps[j] - ps[i]).y;
13      });
        map<int, int> sweep;
15      for (int i : id) {
          for (auto it = sweep.lower_bound(-ps[i].y);
17            it != sweep.end(); sweep.erase(it++)) {
            int j = it->second;
19          P d = ps[i] - ps[j];
            if (d.y > d.x) break;
21          edges.push_back({d.y + d.x, i, j});
          }
23        sweep[-ps[i].y] = i;
        }
25      for (P &p : ps)
          if (k & 1) p.x = -p.x;
27        else swap(p.x, p.y);
      }
29    return edges;
    }
```

4.   Math

4.1.   Number Theory

4.1.1.   Mod Struct

A list of safe primes: 26003, 27767, 28319, 28979, 29243, 29759, 30467
910927547, 919012223, 947326223, 990669467, 1007939579, 1019126699
929760389146037459, 975500632317046523, 989312547895528379

NTT prime p	$p - 1$	primitive root
65537	$1 \ll 16$	3
998244353	$119 \ll 23$	3
2748779069441	$5 \ll 39$	3
1945555039024054273	$27 \ll 56$	5

Requires: Extended GCD

```cpp
template <typename T> struct M {
  static T MOD; // change to constexpr if already known
  T v;
  M(T x = 0) {
    v = (-MOD <= x && x < MOD) ? x : x % MOD;
    if (v < 0) v += MOD;
  }
  explicit operator T() const { return v; }
  bool operator==(const M &b) const { return v == b.v; }
  bool operator!=(const M &b) const { return v != b.v; }
  M operator-() { return M(-v); }
  M operator+(M b) { return M(v + b.v); }
  M operator-(M b) { return M(v - b.v); }
  M operator*(M b) { return M((__int128)v * b.v % MOD); }
  M operator/(M b) { return *this * (b ^ (MOD - 2)); }
  // change above implementation to this if MOD is not prime
  M inv() {
    auto [p, _, g] = extgcd(v, MOD);
    return assert(g == 1), p;
  }
  friend M operator^(M a, ll b) {
    M ans(1);
    for (; b; b >>= 1, a *= a)
      if (b & 1) ans *= a;
    return ans;
  }
  friend M &operator+=(M &a, M b) { return a = a + b; }
  friend M &operator-=(M &a, M b) { return a = a - b; }
  friend M &operator*=(M &a, M b) { return a = a * b; }
  friend M &operator/=(M &a, M b) { return a = a / b; }
};
using Mod = M<int>;
template <> int Mod::MOD = 1'000'000'007;
int &MOD = Mod::MOD;
```

4.1.2.   Miller-Rabin

Requires: Mod Struct

```cpp
// checks if Mod::MOD is prime
bool is_prime() {
  if (MOD < 2 || MOD % 2 == 0) return MOD == 2;
  Mod A[] = {2, 7, 61}; // for int values (< 2^31)
  // ll: 2, 325, 9375, 28178, 450775, 9780504, 1795265022
  int s = __builtin_ctzll(MOD - 1), i;
  for (Mod a : A) {
    Mod x = a ^ (MOD >> s);
    for (i = 0; i < s && (x + 1).v > 2; i++) x *= x;
    if (i && x != -1) return 0;
  }
  return 1;
}
```

4.1.3.   Linear Sieve

```cpp
constexpr ll MAXN = 1000000;
bitset<MAXN> is_prime;
vector<ll> primes;
ll mpf[MAXN], phi[MAXN], mu[MAXN];

void sieve() {
  is_prime.set();
  is_prime[1] = 0;
  mu[1] = phi[1] = 1;
  for (ll i = 2; i < MAXN; i++) {
    if (is_prime[i]) {
      mpf[i] = i;
      primes.push_back(i);
      phi[i] = i - 1;
      mu[i] = -1;
    }
    for (ll p : primes) {
      if (p > mpf[i] || i * p >= MAXN) break;
```

```cpp
      is_prime[i * p] = 0;
      mpf[i * p] = p;
      mu[i * p] = -mu[i];
      if (i % p == 0)
        phi[i * p] = phi[i] * p, mu[i * p] = 0;
      else phi[i * p] = phi[i] * (p - 1);
    }
  }
}
```

4.1.4.   Get Factors

Requires: Linear Sieve

```cpp
vector<ll> all_factors(ll n) {
  vector<ll> fac = {1};
  while (n > 1) {
    const ll p = mpf[n];
    vector<ll> cur = {1};
    while (n % p == 0) {
      n /= p;
      cur.push_back(cur.back() * p);
    }
    vector<ll> tmp;
    for (auto x : fac)
      for (auto y : cur) tmp.push_back(x * y);
    tmp.swap(fac);
  }
  return fac;
}
```

4.1.5.   Binary GCD

```cpp
// returns the gcd of non-negative a, b
ull bin_gcd(ull a, ull b) {
  if (!a || !b) return a + b;
  int s = __builtin_ctzll(a | b);
  a >>= __builtin_ctzll(a);
  while (b) {
    if ((b >>= __builtin_ctzll(b)) < a) swap(a, b);
    b -= a;
  }
  return a << s;
}
```

4.1.6.   Extended GCD

```cpp
// returns (p, q, g): p * a + q * b == g == gcd(a, b)
// g is not guaranteed to be positive when a < 0 or b < 0
tuple<ll, ll, ll> extgcd(ll a, ll b) {
  ll s = 1, t = 0, u = 0, v = 1;
  while (b) {
    ll q = a / b;
    swap(a -= q * b, b);
    swap(s -= q * t, t);
    swap(u -= q * v, v);
  }
  return {s, u, a};
}
```

4.1.7.   Chinese Remainder Theorem

Requires: Extended GCD

```cpp
// for 0 <= a < m, 0 <= b < n, returns the smallest x >= 0
// such that x % m == a and x % n == b
ll crt(ll a, ll m, ll b, ll n) {
  if (n > m) swap(a, b), swap(m, n);
  auto [x, y, g] = extgcd(m, n);
  assert((a - b) % g == 0); // no solution
  x = ((b - a) / g * x) % (n / g) * m + a;
  return x < 0 ? x + m / g * n : x;
}
```

4.1.8.   Baby-Step Giant-Step

Requires: Mod Struct

```cpp
// returns x such that a ^ x = b where x \in [l, r]
ll bsgs(Mod a, Mod b, ll l = 0, ll r = MOD - 1) {
  int m = sqrt(r - l) + 1, i;
  unordered_map<ll, ll> tb;
  Mod d = (a ^ l) / b;
  for (i = 0, d = (a ^ l) / b; i < m; i++, d *= a)
    if (d == 1) return l + i;
    else tb[(ll)d] = l + i;
  Mod c = Mod(1) / (a ^ m);
  for (i = 0, d = 1; i < m; i++, d *= c)
    if (auto j = tb.find((ll)d); j != tb.end())
      return j->second + i * m;
  return assert(0), -1; // no solution
}
```

4.1.9. Pollard's Rho

```
ll f(ll x, ll mod) { return (x * x + 1) % mod; }
// n should be composite
ll pollard_rho(ll n) {
  if (!(n & 1)) return 2;
  while (1) {
    ll y = 2, x = RNG() % (n - 1) + 1, res = 1;
    for (int sz = 2; res == 1; sz *= 2) {
      for (int i = 0; i < sz && res <= 1; i++) {
        x = f(x, n);
        res = __gcd(abs(x - y), n);
      }
      y = x;
    }
    if (res != 0 && res != n) return res;
  }
}
```

4.1.10. Tonelli-Shanks Algorithm

Requires: Mod Struct

```
int legendre(Mod a) {
  if (a == 0) return 0;
  return (a ^ ((MOD - 1) / 2)) == 1 ? 1 : -1;
}
Mod sqrt(Mod a) {
  assert(legendre(a) != -1); // no solution
  ll p = MOD, s = p - 1;
  if (a == 0) return 0;
  if (p == 2) return 1;
  if (p % 4 == 3) return a ^ ((p + 1) / 4);
  int r, m;
  for (r = 0; !(s & 1); r++) s >>= 1;
  Mod n = 2;
  while (legendre(n) != -1) n += 1;
  Mod x = a ^ ((s + 1) / 2), b = a ^ s, g = n ^ s;
  while (b != 1) {
    Mod t = b;
    for (m = 0; t != 1; m++) t *= t;
    Mod gs = g ^ (1LL << (r - m - 1));
    g = gs * gs, x *= gs, b *= g, r = m;
  }
  return x;
}
// to get sqrt(X) modulo p^k, where p is an odd prime:
// c = x^2 (mod p), c = X^2 (mod p^k), q = p^(k-1)
// X = x^q * c^((p^k-2q+1)/2) (mod p^k)
```

4.1.11. Chinese Sieve

```
const ll N = 1000000;
// f, g, h multiplicative, h = f (dirichlet convolution) g
ll pre_g(ll n);
ll pre_h(ll n);
// preprocessed prefix sum of f
ll pre_f[N];
// prefix sum of multiplicative function f
ll solve_f(ll n) {
  static unordered_map<ll, ll> m;
  if (n < N) return pre_f[n];
  if (m.count(n)) return m[n];
  ll ans = pre_h(n);
  for (ll l = 2, r; l <= n; l = r + 1) {
    r = n / (n / l);
    ans -= (pre_g(r) - pre_g(l - 1)) * djs_f(n / l);
  }
  return m[n] = ans;
}
```

4.1.12. Rational Number Binary Search

```
struct QQ {
  ll p, q;
  QQ go(QQ b, ll d) { return {p + b.p * d, q + b.q * d}; }
};
bool pred(QQ);
// returns smallest p/q in [lo, hi] such that
// pred(p/q) is true, and 0 <= p,q <= N
QQ frac_bs(ll N) {
  QQ lo{0, 1}, hi{1, 0};
  if (pred(lo)) return lo;
  assert(pred(hi));
  bool dir = 1, L = 1, H = 1;
  for (; L || H; dir = !dir) {
    ll len = 0, step = 1;
    for (int t = 0; t < 2 && (t ? step /= 2 : step *= 2);)
      if (QQ mid = hi.go(lo, len + step);
          mid.p > N || mid.q > N || dir ^ pred(mid))
        t++;
      else len += step;
```

```
    swap(lo, hi = hi.go(lo, len));
    (dir ? L : H) = !!len;
  }
  return dir ? hi : lo;
}
```

4.1.13. Farey Sequence

```
// returns (e/f), where (a/b, c/d, e/f) are
// three consecutive terms in the order n farey sequence
// to start, call next_farey(n, 0, 1, 1, n)
pll next_farey(ll n, ll a, ll b, ll c, ll d) {
  ll p = (n + b) / d;
  return pll(p * c - a, p * d - b);
}
```

4.2. Combinatorics

4.2.1. Matroid Intersection

This template assumes 2 weighted matroids of the same type, and that removing an element is much more expensive than checking if one can be added. **Remember to change the implementation details.**

The ground set is $0, 1, \ldots, n - 1$, where element i has weight $w[i]$. For the unweighted version, remove weights and change BF/SPFA to BFS.

```
constexpr int N = 100;
constexpr int INF = 1e9;

struct Matroid {           // represents an independent set
  Matroid(bitset<N>);      // initialize from an independent set
  bool can_add(int);       // if adding will break independence
  Matroid remove(int);     // removing from the set
};

auto matroid_intersection(int n, const vector<int> &w) {
  bitset<N> S;
  for (int sz = 1; sz <= n; sz++) {
    Matroid M1(S), M2(S);

    vector<vector<pii>> e(n + 2);
    for (int j = 0; j < n; j++)
      if (!S[j]) {
        if (M1.can_add(j)) e[n].emplace_back(j, -w[j]);
        if (M2.can_add(j)) e[j].emplace_back(n + 1, 0);
      }
    for (int i = 0; i < n; i++)
      if (S[i]) {
        Matroid T1 = M1.remove(i), T2 = M2.remove(i);
        for (int j = 0; j < n; j++)
          if (!S[j]) {
            if (T1.can_add(j)) e[i].emplace_back(j, -w[j]);
            if (T2.can_add(j)) e[j].emplace_back(i, w[i]);
          }
      }

    vector<pii> dis(n + 2, {INF, 0});
    vector<int> prev(n + 2, -1);
    dis[n] = {0, 0};
    // change to SPFA for more speed, if necessary
    bool upd = 1;
    while (upd) {
      upd = 0;
      for (int u = 0; u < n + 2; u++)
        for (auto [v, c] : e[u]) {
          pii x(dis[u].first + c, dis[u].second + 1);
          if (x < dis[v]) dis[v] = x, prev[v] = u, upd = 1;
        }
    }

    if (dis[n + 1].first < INF)
      for (int x = prev[n + 1]; x != n; x = prev[x])
        S.flip(x);
    else break;

    // S is the max-weighted independent set with size sz
  }
  return S;
}
```

4.2.2. De Brujin Sequence

```
int res[kN], aux[kN], a[kN], sz;
void Rec(int t, int p, int n, int k) {
  if (t > n) {
    if (n % p == 0)
      for (int i = 1; i <= p; ++i) res[sz++] = aux[i];
  } else {
    aux[t] = aux[t - p];
    Rec(t + 1, p, n, k);
    for (aux[t] = aux[t - p] + 1; aux[t] < k; ++aux[t])
      Rec(t + 1, t, n, k);
```

```
11    }
    }
13  int DeBruijn(int k, int n) {
      // return cyclic string of length k^n such that every
15    // string of length n using k character appears as a
      // substring.
17    if (k == 1) return res[0] = 0, 1;
      fill(aux, aux + k * n, 0);
19    return sz = 0, Rec(1, 1, n, k), sz;
    }
```

4.2.3. Multinomial

```
1
3  // ways to permute v[i]
    ll multinomial(vi &v) {
5    ll c = 1, m = v.empty() ? 1 : v[0];
      for (int i = 1; i < v.size(); i++)
7      for (int j = 0; i < v[i]; j++) c = c * ++m / (j + 1);
      return c;
9  }
```

4.3.   Algebra

4.3.1.   Formal Power Series

```
1
3
    template <typename mint>
5  struct FormalPowerSeries : vector<mint> {
      using vector<mint>::vector;
7    using FPS = FormalPowerSeries;

9    FPS &operator+=(const FPS &r) {
        if (r.size() > this->size()) this->resize(r.size());
11      for (int i = 0; i < (int)r.size(); i++)
          (*this)[i] += r[i];
13      return *this;
      }
15
      FPS &operator+=(const mint &r) {
17      if (this->empty()) this->resize(1);
        (*this)[0] += r;
19      return *this;
      }
21
      FPS &operator-=(const FPS &r) {
23      if (r.size() > this->size()) this->resize(r.size());
        for (int i = 0; i < (int)r.size(); i++)
25        (*this)[i] -= r[i];
        return *this;
27    }

29    FPS &operator-=(const mint &r) {
        if (this->empty()) this->resize(1);
31      (*this)[0] -= r;
        return *this;
33    }

35    FPS &operator*=(const mint &v) {
        for (int k = 0; k < (int)this->size(); k++)
37        (*this)[k] *= v;
        return *this;
39    }

41    FPS &operator/=(const FPS &r) {
        if (this->size() < r.size()) {
43        this->clear();
          return *this;
45      }
        int n = this->size() - r.size() + 1;
47      if ((int)r.size() <= 64) {
          FPS f(*this), g(r);
49        g.shrink();
          mint coeff = g.back().inverse();
51        for (auto &x : g) x *= coeff;
          int deg = (int)f.size() - (int)g.size() + 1;
53        int gs = g.size();
          FPS quo(deg);
55        for (int i = deg - 1; i >= 0; i--) {
            quo[i] = f[i + gs - 1];
57          for (int j = 0; j < gs; j++)
              f[i + j] -= quo[i] * g[j];
59        }
          *this = quo * coeff;
61        this->resize(n, mint(0));
          return *this;
63      }
        return *this = ((*this).rev().pre(n) * r.rev().inv(n))
65                        .pre(n)
                          .rev();
```

```
67    }

69    FPS &operator%=(const FPS &r) {
        *this -= *this / r * r;
71      shrink();
        return *this;
73    }

75    FPS operator+(const FPS &r) const {
        return FPS(*this) += r;
77    }
    FPS operator+(const mint &v) const {
79      return FPS(*this) += v;
      }
81    FPS operator-(const FPS &r) const {
        return FPS(*this) -= r;
83    }
    FPS operator-(const mint &v) const {
85      return FPS(*this) -= v;
      }
87    FPS operator*(const FPS &r) const {
        return FPS(*this) *= r;
89    }
    FPS operator*(const mint &v) const {
91      return FPS(*this) *= v;
      }
93    FPS operator/(const FPS &r) const {
        return FPS(*this) /= r;
95    }
    FPS operator%(const FPS &r) const {
97      return FPS(*this) %= r;
      }
99    FPS operator-() const {
        FPS ret(this->size());
101     for (int i = 0; i < (int)this->size(); i++)
          ret[i] = -(*this)[i];
103     return ret;
      }
105
      void shrink() {
107     while (this->size() && this->back() == mint(0))
          this->pop_back();
109   }

111   FPS rev() const {
        FPS ret(*this);
113     reverse(begin(ret), end(ret));
        return ret;
115   }

117   FPS dot(FPS r) const {
        FPS ret(min(this->size(), r.size()));
119     for (int i = 0; i < (int)ret.size(); i++)
          ret[i] = (*this)[i] * r[i];
121     return ret;
      }
123
      FPS pre(int sz) const {
125     return FPS(begin(*this),
                   begin(*this) + min((int)this->size(), sz));
127   }

129   FPS operator>>(int sz) const {
        if ((int)this->size() <= sz) return {};
131     FPS ret(*this);
        ret.erase(ret.begin(), ret.begin() + sz);
133     return ret;
      }
135
      FPS operator<<(int sz) const {
137     FPS ret(*this);
        ret.insert(ret.begin(), sz, mint(0));
139     return ret;
      }
141
      FPS diff() const {
143     const int n = (int)this->size();
        FPS ret(max(0, n - 1));
145     mint one(1), coeff(1);
        for (int i = 1; i < n; i++) {
147       ret[i - 1] = (*this)[i] * coeff;
          coeff += one;
149     }
        return ret;
151   }

153   FPS integral() const {
        const int n = (int)this->size();
155     FPS ret(n + 1);
        ret[0] = mint(0);
157     if (n > 0) ret[1] = mint(1);
        auto mod = mint::get_mod();
159     for (int i = 2; i <= n; i++)
          ret[i] = (-ret[mod % i]) * (mod / i);
```

```
161    for (int i = 0; i < n; i++) ret[i + 1] *= (*this)[i];
       return ret;
163  }

165  mint eval(mint x) const {
       mint r = 0, w = 1;
167    for (auto &v : *this) r += w * v, w *= x;
       return r;
169  }

171  FPS log(int deg = -1) const {
       assert((*this)[0] == mint(1));
173    if (deg == -1) deg = (int)this->size();
       return (this->diff() * this->inv(deg))
175    .pre(deg - 1)
       .integral();
177  }

179  FPS pow(int64_t k, int deg = -1) const {
       const int n = (int)this->size();
181    if (deg == -1) deg = n;
       for (int i = 0; i < n; i++) {
183      if ((*this)[i] != mint(0)) {
           if (i * k > deg) return FPS(deg, mint(0));
185        mint rev = mint(1) / (*this)[i];
           FPS ret =
187        (((*this * rev) >> i).log(deg) * k).exp(deg) *
           ((*this)[i].pow(k));
189        ret = (ret << (i * k)).pre(deg);
           if ((int)ret.size() < deg) ret.resize(deg, mint(0));
191        return ret;
         }
193    }
       return FPS(deg, mint(0));
195  }

197  static void *ntt_ptr;
     static void set_fft();
199  FPS &operator*=(const FPS &r);
     void ntt();
201  void intt();
     void ntt_doubling();
203  static int ntt_pr();
     FPS inv(int deg = -1) const;
205  FPS exp(int deg = -1) const;
   };
207  template <typename mint>
   void *FormalPowerSeries<mint>::ntt_ptr = nullptr;
```

4.4. Theorems

4.4.1. Kirchhoff's Theorem

Denote L be a $n \times n$ matrix as the Laplacian matrix of graph G, where $L_{ii} = d(i)$, $L_{ij} = -c$ where c is the number of edge (i, j) in G.

- The number of undirected spanning in G is $|\det(\tilde{L}_{11})|$.
- The number of directed spanning tree rooted at r in G is $|\det(\tilde{L}_{rr})|$.

4.4.2. Tutte's Matrix

Let D be a $n \times n$ matrix, where $d_{ij} = x_{ij}$ (x_{ij} is chosen uniformly at random) if $i < j$ and $(i, j) \in E$, otherwise $d_{ij} = -d_{ji}$. $\frac{rank(D)}{2}$ is the maximum matching on G.

4.4.3. Cayley's Formula

- Given a degree sequence $d_1, d_2, \ldots, d_n$ for each *labeled* vertices, there are
$$\frac{(n-2)!}{(d_1 - 1)!(d_2 - 1)! \cdots (d_n - 1)!}$$
  spanning trees.
- Let $T_{n,k}$ be the number of *labeled* forests on n vertices with k components, such that vertex $1, 2, \ldots, k$ belong to different components. Then $T_{n,k} = kn^{n-k-1}$.

4.4.4. Erdős–Gallai Theorem

A sequence of non-negative integers $d_1 \geq d_2 \geq \ldots \geq d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1 + d_2 + \ldots + d_n$ is even and
$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k)$$
holds for all $1 \leq k \leq n$.

4.4.5. Burnside's Lemma

Let X be a set and G be a group that acts on X. For $g \in G$, denote by X^g the elements fixed by g:
$$X^g = \{x \in X \mid gx \in X\}$$
Then
$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|.$$

5. Numeric

5.1. Barrett Reduction

```cpp
using ull = unsigned long long;
using uL = __uint128_t;
// very fast calculation of a % m
struct reduction {
  const ull m, d;
  explicit reduction(ull m) : m(m), d(((uL)1 << 64) / m) {}
  inline ull operator()(ull a) const {
    ull q = (ull)(((uL)d * a) >> 64);
    return (a -= q * m) >= m ? a - m : a;
  }
};
```

5.2. Long Long Multiplication

```cpp
using ull = unsigned long long;
using ll = long long;
using ld = long double;
// returns a * b % M where a, b < M < 2**63
ull mult(ull a, ull b, ull M) {
  ll ret = a * b - M * ull(ld(a) * ld(b) / ld(M));
  return ret + M * (ret < 0) - M * (ret >= (ll)M);
}
```

5.3. Fast Fourier Transform

```cpp
template <typename T>
void fft_(int n, vector<T> &a, vector<T> &rt, bool inv) {
  vector<int> br(n);
  for (int i = 1; i < n; i++) {
    br[i] = (i & 1) ? br[i - 1] + n / 2 : br[i / 2] / 2;
    if (br[i] > i) swap(a[i], a[br[i]]);
  }
  for (int len = 2; len <= n; len *= 2)
    for (int i = 0; i < n; i += len)
      for (int j = 0; j < len / 2; j++) {
        int pos = n / len * (inv ? len - j : j);
        T u = a[i + j], v = a[i + j + len / 2] * rt[pos];
        a[i + j] = u + v, a[i + j + len / 2] = u - v;
      }
  if (T minv = T(1) / T(n); inv)
    for (T &x : a) x *= minv;
}
```

Requires: Mod Struct

```cpp
void ntt(vector<Mod> &a, bool inv, Mod primitive_root) {
  int n = a.size();
  Mod root = primitive_root ^ (MOD - 1) / n;
  vector<Mod> rt(n + 1, 1);
  for (int i = 0; i < n; i++) rt[i + 1] = rt[i] * root;
  fft_(n, a, rt, inv);
}
void fft(vector<complex<double>> &a, bool inv) {
  int n = a.size();
  vector<complex<double>> rt(n + 1);
  double arg = acos(-1) * 2 / n;
  for (int i = 0; i <= n; i++)
    rt[i] = {cos(arg * i), sin(arg * i)};
  fft_(n, a, rt, inv);
}
```

5.4. Fast Walsh-Hadamard Transform

Requires: Mod Struct

```cpp
void fwht(vector<Mod> &a, bool inv) {
  int n = a.size();
  for (int d = 1; d < n; d <<= 1)
    for (int m = 0; m < n; m++)
      if (!(m & d)) {
        inv ? a[m] -= a[m | d] : a[m] += a[m | d]; // AND
        inv ? a[m | d] -= a[m] : a[m | d] += a[m]; // OR
        Mod x = a[m], y = a[m | d];                // XOR
        a[m] = x + y, a[m | d] = x - y;            // XOR
      }
  if (Mod iv = Mod(1) / n; inv) // XOR
    for (Mod &i : a) i *= iv;    // XOR
}
```

5.5. Subset Convolution

Requires: Mod Struct

```cpp
#pragma GCC target("popcnt")
#include <immintrin.h>

void fwht(int n, vector<vector<Mod>> &a, bool inv) {
  for (int h = 0; h < n; h++)
    for (int i = 0; i < (1 << n); i++)
      if (!(i & (1 << h)))
        for (int k = 0; k <= n; k++)
          inv ? a[i | (1 << h)][k] -= a[i][k]
              : a[i | (1 << h)][k] += a[i][k];
}
// c[k] = sum(popcnt(i & j) == sz && i | j == k) a[i] * b[j]
vector<Mod> subset_convolution(int n, int sz,
                               const vector<Mod> &a_,
                               const vector<Mod> &b_) {
  int len = n + sz + 1, N = 1 << n;
  vector<vector<Mod>> a(1 << n, vector<Mod>(len, 0)), b = a;
  for (int i = 0; i < N; i++)
    a[i][_mm_popcnt_u64(i)] = a_[i],
    b[i][_mm_popcnt_u64(i)] = b_[i];
  fwht(n, a, 0), fwht(n, b, 0);
  for (int i = 0; i < N; i++) {
    vector<Mod> tmp(len);
    for (int j = 0; j < len; j++)
      for (int k = 0; k <= j; k++)
        tmp[j] += a[i][k] * b[i][j - k];
    a[i] = tmp;
  }
  fwht(n, a, 1);
  vector<Mod> c(N);
  for (int i = 0; i < N; i++)
    c[i] = a[i][_mm_popcnt_u64(i) + sz];
  return c;
}
```

5.6. Linear Recurrences

5.6.1. Berlekamp-Massey Algorithm

```cpp
template <typename T>
vector<T> berlekamp_massey(const vector<T> &s) {
  int n = s.size(), l = 0, m = 1;
  vector<T> r(n), p(n);
  r[0] = p[0] = 1;
  T b = 1, d = 0;
  for (int i = 0; i < n; i++, m++, d = 0) {
    for (int j = 0; j <= l; j++) d += r[j] * s[i - j];
    if ((d /= b) == 0) continue; // change if T is float
    auto t = r;
    for (int j = m; j < n; j++) r[j] -= d * p[j - m];
    if (l * 2 <= i) l = i + 1 - l, b *= d, m = 0, p = t;
  }
  return r.resize(l + 1), reverse(r.begin(), r.end()), r;
}
```

5.6.2. Linear Recurrence Calculation

```cpp
template <typename T> struct lin_rec {
  using poly = vector<T>;
  poly mul(poly a, poly b, poly m) {
    int n = m.size();
    poly r(n);
    for (int i = n - 1; i >= 0; i--) {
      r.insert(r.begin(), 0), r.pop_back();
      T c = r[n - 1] + a[n - 1] * b[i];
      // c /= m[n - 1];  if m is not monic
      for (int j = 0; j < n; j++)
        r[j] += a[j] * b[i] - c * m[j];
    }
    return r;
  }
  poly pow(poly p, ll k, poly m) {
    poly r(m.size());
    r[0] = 1;
    for (; k; k >>= 1, p = mul(p, p, m))
      if (k & 1) r = mul(r, p, m);
    return r;
  }
  T calc(poly t, poly r, ll k) {
    int n = r.size();
    poly p(n);
    p[1] = 1;
    poly q = pow(p, k, r);
    T ans = 0;
    for (int i = 0; i < n; i++) ans += t[i] * q[i];
    return ans;
  }
};
```

5.7. Matrices

5.7.1. Determinant

Requires: Mod Struct

```cpp
Mod det(vector<vector<Mod>> a) {
  int n = a.size();
  Mod ans = 1;
  for (int i = 0; i < n; i++) {
    int b = i;
    for (int j = i + 1; j < n; j++)
      if (a[j][i] != 0) {
        b = j;
        break;
      }
    if (i != b) swap(a[i], a[b]), ans = -ans;
    ans *= a[i][i];
    if (ans == 0) return 0;
    for (int j = i + 1; j < n; j++) {
      Mod v = a[j][i] / a[i][i];
      if (v != 0)
        for (int k = i + 1; k < n; k++)
          a[j][k] -= v * a[i][k];
    }
  }
  return ans;
}
```

```cpp
double det(vector<vector<double>> a) {
  int n = a.size();
  double ans = 1;
  for (int i = 0; i < n; i++) {
    int b = i;
    for (int j = i + 1; j < n; j++)
      if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
    if (i != b) swap(a[i], a[b]), ans = -ans;
    ans *= a[i][i];
    if (ans == 0) return 0;
    for (int j = i + 1; j < n; j++) {
      double v = a[j][i] / a[i][i];
      if (v != 0)
        for (int k = i + 1; k < n; k++)
          a[j][k] -= v * a[i][k];
    }
  }
  return ans;
}
```

5.7.2. Inverse

```cpp
// Returns rank.
// Result is stored in A unless singular (rank < n).
// For prime powers, repeatedly set
// A^{-1} = A^{-1} (2I - A*A^{-1}) (mod p^k)
// where A^{-1} starts as the inverse of A mod p,
// and k is doubled in each step.

int matInv(vector<vector<double>> &A) {
  int n = sz(A);
  vi col(n);
  vector<vector<double>> tmp(n, vector<double>(n));
  rep(i, 0, n) tmp[i][i] = 1, col[i] = i;

  rep(i, 0, n) {
    int r = i, c = i;
    rep(j, i, n)
    rep(k, i, n) if (fabs(A[j][k]) > fabs(A[r][c])) r = j,
                                                     c = k;
    if (fabs(A[r][c]) < 1e-12) return i;
    A[i].swap(A[r]);
    tmp[i].swap(tmp[r]);
    rep(j, 0, n) swap(A[j][i], A[j][c]),
    swap(tmp[j][i], tmp[j][c]);
    swap(col[i], col[c]);
    double v = A[i][i];
    rep(j, i + 1, n) {
      double f = A[j][i] / v;
      A[j][i] = 0;
      rep(k, i + 1, n) A[j][k] -= f * A[i][k];
      rep(k, 0, n) tmp[j][k] -= f * tmp[i][k];
    }
    rep(j, i + 1, n) A[i][j] /= v;
    rep(j, 0, n) tmp[i][j] /= v;
    A[i][i] = 1;
  }

  for (int i = n - 1; i > 0; --i) rep(j, 0, i) {
    double v = A[j][i];
```

```cpp
    rep(k, 0, n) tmp[j][k] -= v * tmp[i][k];
  }
  rep(i, 0, n) rep(j, 0, n) A[col[i]][col[j]] = tmp[i][j];
  return n;
}
```

```cpp
int matInv_mod(vector<vector<ll>> &A) {
  int n = sz(A);
  vi col(n);
  vector<vector<ll>> tmp(n, vector<ll>(n));
  rep(i, 0, n) tmp[i][i] = 1, col[i] = i;

  rep(i, 0, n) {
    int r = i, c = i;
    rep(j, i, n) rep(k, i, n) if (A[j][k]) {
      r = j;
      c = k;
      goto found;
    }
    return i;
  found:
    A[i].swap(A[r]);
    tmp[i].swap(tmp[r]);
    rep(j, 0, n) swap(A[j][i], A[j][c]),
    swap(tmp[j][i], tmp[j][c]);
    swap(col[i], col[c]);
    ll v = modpow(A[i][i], mod - 2);
    rep(j, i + 1, n) {
      ll f = A[j][i] * v % mod;
      A[j][i] = 0;
      rep(k, i + 1, n) A[j][k] =
      (A[j][k] - f * A[i][k]) % mod;
      rep(k, 0, n) tmp[j][k] =
      (tmp[j][k] - f * tmp[i][k]) % mod;
    }
    rep(j, i + 1, n) A[i][j] = A[i][j] * v % mod;
    rep(j, 0, n) tmp[i][j] = tmp[i][j] * v % mod;
    A[i][i] = 1;
  }

  for (int i = n - 1; i > 0; --i) rep(j, 0, i) {
    ll v = A[j][i];
    rep(k, 0, n) tmp[j][k] =
    (tmp[j][k] - v * tmp[i][k]) % mod;
  }

  rep(i, 0, n) rep(j, 0, n) A[col[i]][col[j]] =
  tmp[i][j] % mod + (tmp[i][j] < 0 ? mod : 0);
  return n;
}
```

5.7.3. Characteristic Polynomial

```cpp
// calculate det(a - xI)
template <typename T>
vector<T> CharacteristicPolynomial(vector<vector<T>> a) {
  int N = a.size();

  for (int j = 0; j < N - 2; j++) {
    for (int i = j + 1; i < N; i++) {
      if (a[i][j] != 0) {
        swap(a[j + 1], a[i]);
        for (int k = 0; k < N; k++)
          swap(a[k][j + 1], a[k][i]);
        break;
      }
    }
    if (a[j + 1][j] != 0) {
      T inv = T(1) / a[j + 1][j];
      for (int i = j + 2; i < N; i++) {
        if (a[i][j] == 0) continue;
        T coe = inv * a[i][j];
        for (int l = j; l < N; l++)
          a[i][l] -= coe * a[j + 1][l];
        for (int k = 0; k < N; k++)
          a[k][j + 1] += coe * a[k][i];
      }
    }
  }

  vector<vector<T>> p(N + 1);
  p[0] = {T(1)};
  for (int i = 1; i <= N; i++) {
    p[i].resize(i + 1);
    for (int j = 0; j < i; j++) {
      p[i][j + 1] -= p[i - 1][j];
      p[i][j] += p[i - 1][j] * a[i - 1][i - 1];
    }
    T x = 1;
    for (int m = 1; m < i; m++) {
```

```
41        x *= -a[i - m][i - m - 1];
          T coe = x * a[i - m - 1][i - 1];
43        for (int j = 0; j < i - m; j++)
            p[i][j] += coe * p[i - m - 1][j];
45      }
      }
47    return p[N];
    }
```

5.7.4.   Solve Linear Equation

```
1
3   typedef vector<double> vd;
    const double eps = 1e-12;
5
    // solves for x: A * x = b
7   int solveLinear(vector<vd> &A, vd &b, vd &x) {
      int n = sz(A), m = sz(x), rank = 0, br, bc;
9     if (n) assert(sz(A[0]) == m);
      vi col(m);
11    iota(all(col), 0);
13    rep(i, 0, n) {
        double v, bv = 0;
15      rep(r, i, n) rep(c, i, m) if ((v = fabs(A[r][c])) > bv)
        br = r,
17      bc = c, bv = v;
        if (bv <= eps) {
19        rep(j, i, n) if (fabs(b[j]) > eps) return -1;
          break;
21      }
        swap(A[i], A[br]);
23      swap(b[i], b[br]);
        swap(col[i], col[bc]);
25      rep(j, 0, n) swap(A[j][i], A[j][bc]);
        bv = 1 / A[i][i];
27      rep(j, i + 1, n) {
          double fac = A[j][i] * bv;
29        b[j] -= fac * b[i];
          rep(k, i + 1, m) A[j][k] -= fac * A[i][k];
31      }
        rank++;
33    }
35    x.assign(m, 0);
      for (int i = rank; i--;) {
37      b[i] /= A[i][i];
        x[col[i]] = b[i];
39      rep(j, 0, i) b[j] -= A[j][i] * b[i];
      }
41    return rank; // (multiple solutions if rank < m)
    }
```

5.8.   Polynomial Interpolation

```
1
3   // returns a, such that a[0]x^0 + a[1]x^1 + a[2]x^2 + ...
    // passes through the given points
5   typedef vector<double> vd;
    vd interpolate(vd x, vd y, int n) {
7     vd res(n), temp(n);
      rep(k, 0, n - 1) rep(i, k + 1, n) y[i] =
9     (y[i] - y[k]) / (x[i] - x[k]);
      double last = 0;
11    temp[0] = 1;
      rep(k, 0, n) rep(i, 0, n) {
13      res[i] += y[k] * temp[i];
        swap(last, temp[i]);
15      temp[i] -= last * x[k];
      }
17    return res;
    }
```

5.9.   Simplex Algorithm

```
1   // Two-phase simplex algorithm for solving linear programs
    // of the form
3   //
    //       maximize      c^T x
5   //       subject to    Ax <= b
    //                     x >= 0
7   //
    // INPUT: A -- an m x n matrix
9   //        b -- an m-dimensional vector
    //        c -- an n-dimensional vector
11  //        x -- a vector where the optimal solution will be
    //             stored
13  //
    // OUTPUT: value of the optimal solution (infinity if
15  // unbounded
    //          above, nan if infeasible)
```

```
17  //
    // To use this code, create an LPSolver object with A, b,
19  // and c as arguments.  Then, call Solve(x).
21  typedef long double ld;
    typedef vector<ld> vd;
23  typedef vector<vd> vvd;
    typedef vector<int> vi;
25
    const ld EPS = 1e-9;
27
    struct LPSolver {
29    int m, n;
      vi B, N;
31    vvd D;
33    LPSolver(const vvd &A, const vd &b, const vd &c)
        : m(b.size()), n(c.size()), N(n + 1), B(m),
35        D(m + 2, vd(n + 2)) {
        for (int i = 0; i < m; i++)
37        for (int j = 0; j < n; j++) D[i][j] = A[i][j];
        for (int i = 0; i < m; i++) {
39        B[i] = n + i;
          D[i][n] = -1;
41        D[i][n + 1] = b[i];
        }
43      for (int j = 0; j < n; j++) {
          N[j] = j;
45        D[m][j] = -c[j];
        }
47      N[n] = -1;
        D[m + 1][n] = 1;
49    }
51    void Pivot(int r, int s) {
        double inv = 1.0 / D[r][s];
53      for (int i = 0; i < m + 2; i++)
          if (i != r)
55          for (int j = 0; j < n + 2; j++)
              if (j != s) D[i][j] -= D[r][j] * D[i][s] * inv;
57      for (int j = 0; j < n + 2; j++)
          if (j != s) D[r][j] *= inv;
59      for (int i = 0; i < m + 2; i++)
          if (i != r) D[i][s] *= -inv;
61      D[r][s] = inv;
        swap(B[r], N[s]);
63    }
65    bool Simplex(int phase) {
        int x = phase == 1 ? m + 1 : m;
67      while (true) {
          int s = -1;
69        for (int j = 0; j <= n; j++) {
            if (phase == 2 && N[j] == -1) continue;
71          if (s == -1 || D[x][j] < D[x][s] ||
                D[x][j] == D[x][s] && N[j] < N[s])
73            s = j;
          }
75        if (D[x][s] > -EPS) return true;
          int r = -1;
77        for (int i = 0; i < m; i++) {
            if (D[i][s] < EPS) continue;
79          if (r == -1 ||
                D[i][n + 1] / D[i][s] < D[r][n + 1] / D[r][s] ||
81            (D[i][n + 1] / D[i][s]) ==
                (D[r][n + 1] / D[r][s]) &&
83            B[i] < B[r])
              r = i;
85        }
          if (r == -1) return false;
87        Pivot(r, s);
        }
89    }
91    ld Solve(vd &x) {
        int r = 0;
93      for (int i = 1; i < m; i++)
          if (D[i][n + 1] < D[r][n + 1]) r = i;
95      if (D[r][n + 1] < -EPS) {
          Pivot(r, n);
97        if (!Simplex(1) || D[m + 1][n + 1] < -EPS)
            return -numeric_limits<ld>::infinity();
99        for (int i = 0; i < m; i++)
            if (B[i] == -1) {
101          int s = -1;
              for (int j = 0; j <= n; j++)
103            if (s == -1 || D[i][j] < D[i][s] ||
                    D[i][j] == D[i][s] && N[j] < N[s])
105              s = j;
              Pivot(i, s);
107        }
        }
109      if (!Simplex(2)) return numeric_limits<ld>::infinity();
        x = vd(n);
```

```
111        for (int i = 0; i < m; i++)
             if (B[i] < n) x[B[i]] = D[i][n + 1];
113      return D[m][n + 1];
       }
115 };

117 int main() {

119    const int m = 4;
       const int n = 3;
121    ld _A[m][n] = {
       {6, -1, 0}, {-1, -5, 0}, {1, 5, 1}, {-1, -5, -1}};
123    ld _b[m] = {10, -4, 5, -5};
       ld _c[n] = {1, -1, 0};
125
       vvd A(m);
127    vd b(_b, _b + m);
       vd c(_c, _c + n);
129    for (int i = 0; i < m; i++) A[i] = vd(_A[i], _A[i] + n);

131    LPSolver solver(A, b, c);
       vd x;
133    ld value = solver.Solve(x);

135    cerr << "VALUE: " << value << endl; // VALUE: 1.29032
       cerr << "SOLUTION:"; // SOLUTION: 1.74194 0.451613 1
137    for (size_t i = 0; i < x.size(); i++) cerr << " " << x[i];
       cerr << endl;
139    return 0;
     }
```

6.   Geometry

6.1.   Point

```
1  template <typename T> struct P {
     T x, y;
3    P(T x = 0, T y = 0) : x(x), y(y) {}
     bool operator<(const P &p) const {
5      return tie(x, y) < tie(p.x, p.y);
     }
7    bool operator==(const P &p) const {
       return tie(x, y) == tie(p.x, p.y);
9    }
     P operator-() const { return {-x, -y}; }
11   P operator+(P p) const { return {x + p.x, y + p.y}; }
     P operator-(P p) const { return {x - p.x, y - p.y}; }
13   P operator*(T d) const { return {x * d, y * d}; }
     P operator/(T d) const { return {x / d, y / d}; }
15   T dist2() const { return x * x + y * y; }
     double len() const { return sqrt(dist2()); }
17   P unit() const { return *this / len(); }
     friend T dot(P a, P b) { return a.x * b.x + a.y * b.y; }
19   friend T cross(P a, P b) { return a.x * b.y - a.y * b.x; }
     friend T cross(P a, P b, P o) {
21     return cross(a - o, b - o);
     }
23 };
   using pt = P<ll>;
```

6.1.1.   Quarternion

```
1  constexpr double PI = 3.141592653589793;
   constexpr double EPS = 1e-7;
3  struct Q {
     using T = double;
5    T x, y, z, r;
     Q(T r = 0) : x(0), y(0), z(0), r(r) {}
7    Q(T x, T y, T z, T r = 0) : x(x), y(y), z(z), r(r) {}
     friend bool operator==(const Q &a, const Q &b) {
9      return (a - b).abs2() <= EPS;
     }
11   friend bool operator!=(const Q &a, const Q &b) {
       return !(a == b);
13   }
     Q operator-() { return Q(-x, -y, -z, -r); }
15   Q operator+(const Q &b) const {
       return Q(x + b.x, y + b.y, z + b.z, r + b.r);
17   }
     Q operator-(const Q &b) const {
19     return Q(x - b.x, y - b.y, z - b.z, r - b.r);
     }
21   Q operator*(const T &t) const {
       return Q(x * t, y * t, z * t, r * t);
23   }
     Q operator*(const Q &b) const {
25     return Q(r * b.x + x * b.r + y * b.z - z * b.y,
              r * b.y - x * b.z + y * b.r + z * b.x,
27            r * b.z + x * b.y - y * b.x + z * b.r,
              r * b.r - x * b.x - y * b.y - z * b.z);
29   }
     Q operator/(const Q &b) const { return *this * b.inv(); }
31   T abs2() const { return r * r + x * x + y * y + z * z; }
     T len() const { return sqrt(abs2()); }
33   Q conj() const { return Q(-x, -y, -z, r); }
     Q unit() const { return *this * (1.0 / len()); }
35   Q inv() const { return conj() * (1.0 / abs2()); }
     friend T dot(Q a, Q b) {
37     return a.x * b.x + a.y * b.y + a.z * b.z;
     }
39   friend Q cross(Q a, Q b) {
       return Q(a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z,
41            a.x * b.y - a.y * b.x);
     }
43   friend Q rotation_around(Q axis, T angle) {
       return axis.unit() * sin(angle / 2) + cos(angle / 2);
45   }
     Q rotated_around(Q axis, T angle) {
47     Q u = rotation_around(axis, angle);
       return u * *this / u;
49   }
     friend Q rotation_between(Q a, Q b) {
51     a = a.unit(), b = b.unit();
       if (a == -b) {
53       // degenerate case
         Q ortho = abs(a.y) > EPS ? cross(a, Q(1, 0, 0))
55                                 : cross(a, Q(0, 1, 0));
         return rotation_around(ortho, PI);
57     }
       return (a * (a + b)).conj();
59   }
   };
```

6.1.2.   Spherical Coordinates

```
1  struct car_p {
     double x, y, z;
3  };
   struct sph_p {
5    double r, theta, phi;
   };
7
   sph_p conv(car_p p) {
9    double r = sqrt(p.x * p.x + p.y * p.y + p.z * p.z);
     double theta = asin(p.y / r);
11   double phi = atan2(p.y, p.x);
     return {r, theta, phi};
13 }
   car_p conv(sph_p p) {
15   double x = p.r * cos(p.theta) * sin(p.phi);
     double y = p.r * cos(p.theta) * cos(p.phi);
17   double z = p.r * sin(p.theta);
     return {x, y, z};
19 }
```

6.2.   Segments

```
1  // for non-collinear ABCD, if segments AB and CD intersect
   bool intersects(pt a, pt b, pt c, pt d) {
3    if (cross(b, c, a) * cross(b, d, a) > 0) return false;
     if (cross(d, a, c) * cross(d, b, c) > 0) return false;
5    return true;
   }
7  // the intersection point of lines AB and CD
   pt intersect(pt a, pt b, pt c, pt d) {
9    auto x = cross(b, c, a), y = cross(b, d, a);
     if (x == y) {
11     // if(abs(x, y) < 1e-8) {
       // is parallel
13   } else {
       return d * (x / (x - y)) - c * (y / (x - y));
15   }
   }
```

6.3.   Convex Hull

```
1  // returns a convex hull in counterclockwise order
   // for a non-strict one, change cross >= to >
3  vector<pt> convex_hull(vector<pt> p) {
     sort(ALL(p));
5    if (p[0] == p.back()) return {p[0]};
     int n = p.size(), t = 0;
7    vector<pt> h(n + 1);
     for (int _ = 2, s = 0; _--; s = --t, reverse(ALL(p)))
9      for (pt i : p) {
         while (t > s + 1 && cross(i, h[t - 1], h[t - 2]) >= 0)
11         t--;
         h[t++] = i;
13     }
     return h.resize(t), h;
15 }
```

6.3.1.   3D Hull

```
1
3  typedef Point3D<double> P3;

5  struct PR {
     void ins(int x) { (a == -1 ? a : b) = x; }
7    void rem(int x) { (a == x ? a : b) = -1; }
     int cnt() { return (a != -1) + (b != -1); }
9    int a, b;
   };
11
   struct F {
13   P3 q;
     int a, b, c;
15 };
17 vector<F> hull3d(const vector<P3> &A) {
     assert(sz(A) >= 4);
19   vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));
   #define E(x, y) E[f.x][f.y]
21   vector<F> FS;
     auto mf = [&](int i, int j, int k, int l) {
23     P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
       if (q.dot(A[l]) > q.dot(A[i])) q = q * -1;
25     F f{q, i, j, k};
       E(a, b).ins(k);
27     E(a, c).ins(j);
       E(b, c).ins(i);
29     FS.push_back(f);
     };
31   rep(i, 0, 4) rep(j, i + 1, 4) rep(k, j + 1, 4)
     mf(i, j, k, 6 - i - j - k);
```

```
33    rep(i, 4, sz(A)) {
35      rep(j, 0, sz(FS)) {
        F f = FS[j];
37        if (f.q.dot(A[i]) > f.q.dot(A[f.a])) {
          E(a, b).rem(f.c);
39          E(a, c).rem(f.b);
          E(b, c).rem(f.a);
41          swap(FS[j--], FS.back());
          FS.pop_back();
43        }
      }
45      int nw = sz(FS);
      rep(j, 0, nw) {
47        F f = FS[j];
#define C(a, b, c)                                       \
49        if (E(a, b).cnt() != 2) mf(f.a, f.b, i, f.c);
        C(a, b, c);
51        C(a, c, b);
        C(b, c, a);
53      }
    }
55    for (F &it : FS)
      if ((A[it.b] - A[it.a])
57          .cross(A[it.c] - A[it.a])
          .dot(it.q) <= 0)
59        swap(it.c, it.b);
    return FS;
61 };
```

6.4. Angular Sort

```
1 auto angle_cmp = [](const pt &a, const pt &b) {
  auto btm = [](const pt &a) {
3    return a.y < 0 || (a.y == 0 && a.x < 0);
  };
5  return make_tuple(btm(a), a.y * b.x, abs2(a)) <
         make_tuple(btm(b), a.x * b.y, abs2(b));
7 };
  void angular_sort(vector<pt> &p) {
9  sort(p.begin(), p.end(), angle_cmp);
}
```

6.5. Convex Polygon Minkowski Sum

```
1 // O(n) convex polygon minkowski sum
  // must be sorted and counterclockwise
3 vector<pt> minkowski_sum(vector<pt> p, vector<pt> q) {
  auto diff = [](vector<pt> &c) {
5    auto rcmp = [](pt a, pt b) {
      return pt{a.y, a.x} < pt{b.y, b.x};
7    };
    rotate(c.begin(), min_element(ALL(c), rcmp), c.end());
9    c.push_back(c[0]);
    vector<pt> ret;
11    for (int i = 1; i < c.size(); i++)
      ret.push_back(c[i] - c[i - 1]);
13    return ret;
  };
15  auto dp = diff(p), dq = diff(q);
  pt cur = p[0] + q[0];
17  vector<pt> d(dp.size() + dq.size()), ret = {cur};
  // include angle_cmp from angular-sort.cpp
19  merge(ALL(dp), ALL(dq), d.begin(), angle_cmp);
  // optional: make ret strictly convex (UB if degenerate)
21  int now = 0;
  for (int i = 1; i < d.size(); i++) {
23    if (cross(d[i], d[now]) == 0) d[now] = d[now] + d[i];
    else d[++now] = d[i];
25  }
  d.resize(now + 1);
27  // end optional part
  for (pt v : d) ret.push_back(cur = cur + v);
29  return ret.pop_back(), ret;
}
```

6.6. Point In Polygon

```
1 bool on_segment(pt a, pt b, pt p) {
  return cross(a, b, p) == 0 && dot((p - a), (p - b)) <= 0;
3 }
  // p can be any polygon, but this is O(n)
5 bool inside(const vector<pt> &p, pt a) {
  int cnt = 0, n = p.size();
7  for (int i = 0; i < n; i++) {
    pt l = p[i], r = p[(i + 1) % n];
9    // change to return 0; for strict version
    if (on_segment(l, r, a)) return 1;
11    cnt ^= ((a.y < l.y) - (a.y < r.y)) * cross(l, r, a) > 0;
  }
13  return cnt;
}
```

6.6.1. Convex Version

```
1 // no preprocessing version
  // p must be a strict convex hull, counterclockwise
3 // if point is inside or on border
  bool is_inside(const vector<pt> &c, pt p) {
5  int n = c.size(), l = 1, r = n - 1;
  if (cross(c[0], c[1], p) < 0) return false;
7  if (cross(c[n - 1], c[0], p) < 0) return false;
  while (l < r - 1) {
9    int m = (l + r) / 2;
    T a = cross(c[0], c[m], p);
11    if (a > 0) l = m;
    else if (a < 0) r = m;
13    else return dot(c[0] - p, c[m] - p) <= 0;
  }
15  if (l == r) return dot(c[0] - p, c[l] - p) <= 0;
  else return cross(c[l], c[r], p) >= 0;
17 }

19 // with preprocessing version
  vector<pt> vecs;
21 pt center;
  // p must be a strict convex hull, counterclockwise
23 // BEWARE OF OVERFLOWS!!
  void preprocess(vector<pt> p) {
25  for (auto &v : p) v = v * 3;
  center = p[0] + p[1] + p[2];
27  center.x /= 3, center.y /= 3;
  for (auto &v : p) v = v - center;
29  vecs = (angular_sort(p), p);
}
31 bool intersect_strict(pt a, pt b, pt c, pt d) {
  if (cross(b, c, a) * cross(b, d, a) > 0) return false;
33  if (cross(d, a, c) * cross(d, b, c) >= 0) return false;
  return true;
35 }
  // if point is inside or on border
37 bool query(pt p) {
  p = p * 3 - center;
39  auto pr = upper_bound(ALL(vecs), p, angle_cmp);
  if (pr == vecs.end()) pr = vecs.begin();
41  auto pl = (pr == vecs.begin()) ? vecs.back() : *(pr - 1);
  return !intersect_strict({0, 0}, p, pl, *pr);
43 }
```

6.6.2. Offline Multiple Points Version

Requires: Point, GNU PBDS

```
1

3

5
  using Double = __float128;
7  using Point = pt<Double, Double>;

9  int n, m;
  vector<Point> poly;
11 vector<Point> query;
  vector<int> ans;

13
  struct Segment {
15  Point a, b;
  int id;
17 };
  vector<Segment> segs;

19
  Double Xnow;
21 inline Double get_y(const Segment &u, Double xnow = Xnow) {
  const Point &a = u.a;
23  const Point &b = u.b;
  return (a.y * (b.x - xnow) + b.y * (xnow - a.x)) /
25         (b.x - a.x);
}
27 bool operator<(Segment u, Segment v) {
  Double yu = get_y(u);
29  Double yv = get_y(v);
  if (yu != yv) return yu < yv;
31  return u.id < v.id;
}
33 ordered_map<Segment> st;

35 struct Event {
  int type; // +1 insert seg, -1 remove seg, 0 query
37  Double x, y;
  int id;
39 };
  bool operator<(Event a, Event b) {
41  if (a.x != b.x) return a.x < b.x;
  if (a.type != b.type) return a.type < b.type;
43  return a.y < b.y;
}
```

```
45  vector<Event> events;

47  void solve() {
      set<Double> xs;
49    set<Point> ps;
      for (int i = 0; i < n; i++) {
51      xs.insert(poly[i].x);
        ps.insert(poly[i]);
53    }
      for (int i = 0; i < n; i++) {
55      Segment s{poly[i], poly[(i + 1) % n], i};
        if (s.a.x > s.b.x ||
57          (s.a.x == s.b.x && s.a.y > s.b.y)) {
          swap(s.a, s.b);
59      }
        segs.push_back(s);
61
        if (s.a.x != s.b.x) {
63        events.push_back({+1, s.a.x + 0.2, s.a.y, i});
          events.push_back({-1, s.b.x - 0.2, s.b.y, i});
65      }
      }
67    for (int i = 0; i < m; i++) {
        events.push_back({0, query[i].x, query[i].y, i});
69    }
      sort(events.begin(), events.end());
71    int cnt = 0;
      for (Event e : events) {
73      int i = e.id;
        Xnow = e.x;
75      if (e.type == 0) {
          Double x = e.x;
77        Double y = e.y;
          Segment tmp = {{x - 1, y}, {x + 1, y}, -1};
79        auto it = st.lower_bound(tmp);

81        if (ps.count(query[i]) > 0) {
            ans[i] = 0;
83        } else if (xs.count(x) > 0) {
            ans[i] = -2;
85        } else if (it != st.end() &&
                     get_y(*it) == get_y(tmp)) {
87          ans[i] = 0;
          } else if (it != st.begin() &&
89                   get_y(*prev(it)) == get_y(tmp)) {
            ans[i] = 0;
91        } else {
            int rk = st.order_of_key(tmp);
93          if (rk % 2 == 1) {
              ans[i] = 1;
95          } else {
              ans[i] = -1;
97          }
          }
99      } else if (e.type == 1) {
          st.insert(segs[i]);
101       assert((int)st.size() == ++cnt);
        } else if (e.type == -1) {
103       st.erase(segs[i]);
          assert((int)st.size() == --cnt);
105     }
      }
107 }
```

6.7. Closest Pair

```
1  vector<pll> p; // sort by x first!
   bool cmpy(const pll &a, const pll &b) const {
3    return a.y < b.y;
   }
5  ll sq(ll x) { return x * x; }
   // returns (minimum dist)^2 in [l, r)
7  ll solve(int l, int r) {
     if (r - l <= 1) return 1e18;
9    int m = (l + r) / 2;
     ll mid = p[m].x, d = min(solve(l, m), solve(m, r));
11   auto pb = p.begin();
     inplace_merge(pb + l, pb + m, pb + r, cmpy);
13   vector<pll> s;
     for (int i = l; i < r; i++)
15     if (sq(p[i].x - mid) < d) s.push_back(p[i]);
     for (int i = 0; i < s.size(); i++)
17     for (int j = i + 1;
            j < s.size() && sq(s[j].y - s[i].y) < d; j++)
19       d = min(d, dis(s[i], s[j]));
     return d;
21 }
```

6.8. Minimum Enclosing Circle

```
1
3  typedef Point<double> P;
```

```
   double ccRadius(const P &A, const P &B, const P &C) {
5    return (B - A).dist() * (C - B).dist() * (A - C).dist() /
            abs((B - A).cross(C - A)) / 2;
7  }
   P ccCenter(const P &A, const P &B, const P &C) {
9    P b = C - A, c = B - A;
     return A + (b * c.dist2() - c * b.dist2()).perp() /
11             b.cross(c) / 2;
   }
13 pair<P, double> mec(vector<P> ps) {
     shuffle(all(ps), mt19937(time(0)));
15   P o = ps[0];
     double r = 0, EPS = 1 + 1e-8;
17   rep(i, 0, sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
       o = ps[i], r = 0;
19     rep(j, 0, i) if ((o - ps[j]).dist() > r * EPS) {
         o = (ps[i] + ps[j]) / 2;
21       r = (o - ps[i]).dist();
         rep(k, 0, j) if ((o - ps[k]).dist() > r * EPS) {
23         o = ccCenter(ps[i], ps[j], ps[k]);
           r = (o - ps[i]).dist();
25       }
       }
27   }
     return {o, r};
29 }
```

6.9. Delaunay Triangulation

```
1
3  typedef Point<ll> P;
   typedef struct Quad *Q;
5  typedef __int128_t lll; // (can be ll if coords are < 2e4)
   P arb(LLONG_MAX, LLONG_MAX); // not equal to any other point
7
   struct Quad {
9    bool mark;
     Q o, rot;
11   P p;
     P F() { return r()->p; }
13   Q r() { return rot->rot; }
     Q prev() { return rot->o->rot; }
15   Q next() { return r()->prev(); }
   };
17
   bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
19   lll p2 = p.dist2(), A = a.dist2() - p2,
         B = b.dist2() - p2, C = c.dist2() - p2;
21   return p.cross(a, b) * C + p.cross(b, c) * A +
          p.cross(c, a) * B >
23        0;
   }
25 Q makeEdge(P orig, P dest) {
     Q q[] = {new Quad{0, 0, 0, orig}, new Quad{0, 0, 0, arb},
27           new Quad{0, 0, 0, dest}, new Quad{0, 0, 0, arb}};
     rep(i, 0, 4) q[i]->o = q[-i & 3],
29                 q[i]->rot = q[(i + 1) & 3];
     return *q;
31 }
   void splice(Q a, Q b) {
33   swap(a->o->rot->o, b->o->rot->o);
     swap(a->o, b->o);
35 }
   Q connect(Q a, Q b) {
37   Q q = makeEdge(a->F(), b->p);
     splice(q, a->next());
39   splice(q->r(), b);
     return q;
41 }

43 pair<Q, Q> rec(const vector<P> &s) {
     if (sz(s) <= 3) {
45     Q a = makeEdge(s[0], s[1]),
         b = makeEdge(s[1], s.back());
47     if (sz(s) == 2) return {a, a->r()};
       splice(a->r(), b);
49     auto side = s[0].cross(s[1], s[2]);
       Q c = side ? connect(b, a) : 0;
51     return {side < 0 ? c->r() : a, side < 0 ? c : b->r()};
     }
53
   #define H(e)     e->F(), e->p
55 #define valid(e) (e->F().cross(H(base)) > 0)
     Q A, B, ra, rb;
57   int half = sz(s) / 2;
     tie(ra, A) = rec({all(s) - half});
59   tie(B, rb) = rec({sz(s) - half + all(s)});
     while ((B->p.cross(H(A)) < 0 && (A = A->next())) ||
61          (A->p.cross(H(B)) > 0 && (B = B->r()->o)));
     Q base = connect(B->r(), A);
63   if (A->p == ra->p) ra = base->r();
     if (B->p == rb->p) rb = base;
65
```

```
67  #define DEL(e, init, dir)                                    \
      Q e = init->dir;                                           \
      if (valid(e))                                              \
69        while (circ(e->dir->F(), H(base), e->F())) {           \
            Q t = e->dir;                                        \
71          splice(e, e->prev());                                \
            splice(e->r(), e->r()->prev());                      \
73          e = t;                                               \
          }                                                      \
75    for (;;) {
        DEL(LC, base->r(), o);
77      DEL(RC, base, prev());
        if (!valid(LC) && !valid(RC)) break;
79      if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
          base = connect(RC, base->r());
81      else base = connect(base->r(), LC->r());
      }
83    return {ra, rb};
    }
85
    // returns [A_0, B_0, C_0, A_1, B_1, ...]
87  // where A_i, B_i, C_i are counter-clockwise triangles
    vector<P> triangulate(vector<P> pts) {
89    sort(all(pts));
      assert(unique(all(pts)) == pts.end());
91    if (sz(pts) < 2) return {};
      Q e = rec(pts).first;
93    vector<Q> q = {e};
      int qi = 0;
95    while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
    #define ADD                                                  \
97    {                                                          \
        Q c = e;                                                 \
99      do {                                                     \
          c->mark = 1;                                           \
101       pts.push_back(c->p);                                    \
          q.push_back(c->r());                                   \
103       c = c->next();                                          \
        } while (c != e);                                        \
105   }
      ADD;
107   pts.clear();
      while (qi < sz(q))
109     if (!(e = q[qi++])->mark) ADD;
      return pts;
111 }
```

```
25      last--;
      while (first < last && !OnLeft(L[i], p[first])) first++;
27    q[++last] = L[i];
      if (fabs(Cross(q[last].v, q[last - 1].v)) < EPS) {
29      last--;
        if (OnLeft(q[last], L[i].P)) q[last] = L[i];
31    }
      if (first < last)
33      p[last - 1] = GetIntersection(q[last - 1], q[last]);
    }
35  while (first < last && !OnLeft(q[first], p[last - 1]))
      last--;
37  if (last - first <= 1) return 0;
    p[last] = GetIntersection(q[last], q[first]);
39
    int m = 0;
41  for (int i = first; i <= last; i++) poly[m++] = p[i];
    return m;
43 }
```

6.9.1. Slower Version

```
3  template <class P, class F>
   void delaunay(vector<P> &ps, F trifun) {
5    if (sz(ps) == 3) {
       int d = (ps[0].cross(ps[1], ps[2]) < 0);
7      trifun(0, 1 + d, 2 - d);
     }
9    vector<P3> p3;
     for (P p : ps) p3.emplace_back(p.x, p.y, p.dist2());
11   if (sz(ps) > 3)
       for (auto t : hull3d(p3))
13       if ((p3[t.b] - p3[t.a])
             .cross(p3[t.c] - p3[t.a])
15           .dot(P3(0, 0, 1)) < 0)
           trifun(t.a, t.c, t.b);
17 }
```

6.10. Half Plane Intersection

```
1  struct Line {
     Point P;
3    Vector v;
     bool operator<(const Line &b) const {
5      return atan2(v.y, v.x) < atan2(b.v.y, b.v.x);
     }
7  };
   bool OnLeft(const Line &L, const Point &p) {
9    return Cross(L.v, p - L.P) > 0;
   }
11 Point GetIntersection(Line a, Line b) {
     Vector u = a.P - b.P;
13   Double t = Cross(b.v, u) / Cross(a.v, b.v);
     return a.P + a.v * t;
15 }
   int HalfplaneIntersection(Line *L, int n, Point *poly) {
17   sort(L, L + n);
19   int first, last;
     Point *p = new Point[n];
21   Line *q = new Line[n];
     q[first = last = 0] = L[0];
23   for (int i = 1; i < n; i++) {
       while (first < last && !OnLeft(L[i], p[last - 1]))
```

7. Strings

7.1. Knuth-Morris-Pratt Algorithm

```cpp
vector<int> pi(const string &s) {
  vector<int> p(s.size());
  for (int i = 1; i < s.size(); i++) {
    int g = p[i - 1];
    while (g && s[i] != s[g]) g = p[g - 1];
    p[i] = g + (s[i] == s[g]);
  }
  return p;
}
vector<int> match(const string &s, const string &pat) {
  vector<int> p = pi(pat + '\0' + s), res;
  for (int i = p.size() - s.size(); i < p.size(); i++)
    if (p[i] == pat.size())
      res.push_back(i - 2 * pat.size());
  return res;
}
```

7.2. Z Value

```cpp
int z[n];
void zval(string s) {
  // z[i] => longest common prefix of s and s[i:], i > 0
  int n = s.size();
  z[0] = 0;
  for (int b = 0, i = 1; i < n; i++) {
    if (z[b] + b <= i) z[i] = 0;
    else z[i] = min(z[i - b], z[b] + b - i);
    while (s[i + z[i]] == s[z[i]]) z[i]++;
    if (i + z[i] > b + z[b]) b = i;
  }
}
```

7.3. Manacher's Algorithm

```cpp
int z[n];
void manacher(string s) {
  // z[i] => longest odd palindrome centered at i is
  //         s[i - z[i] ... i + z[i]]
  // to get all palindromes (including even length),
  // insert a '#' between each s[i] and s[i + 1]
  int n = s.size();
  z[0] = 0;
  for (int b = 0, i = 1; i < n; i++) {
    if (z[b] + b >= i)
      z[i] = min(z[2 * b - i], b + z[b] - i);
    else z[i] = 0;
    while (i + z[i] + 1 < n && i - z[i] - 1 >= 0 &&
           s[i + z[i] + 1] == s[i - z[i] - 1])
      z[i]++;
    if (z[i] + i > z[b] + b) b = i;
  }
}
```

7.4. Minimum Rotation

```cpp
int min_rotation(string s) {
  int a = 0, n = s.size();
  s += s;
  for (int b = 0; b < n; b++) {
    for (int k = 0; k < n; k++) {
      if (a + k == b || s[a + k] < s[b + k]) {
        b += max(0, k - 1);
        break;
      }
      if (s[a + k] > s[b + k]) {
        a = b;
        break;
      }
    }
  }
  return a;
}
```

7.5. Aho-Corasick Automaton

```cpp
struct Aho_Corasick {
  static const int maxc = 26, maxn = 4e5;
  struct NODES {
    int Next[maxc], fail, ans;
  };
  NODES T[maxn];
  int top, qtop, q[maxn];
  int get_node(const int &fail) {
    fill_n(T[top].Next, maxc, 0);
    T[top].fail = fail;
    T[top].ans = 0;
    return top++;
  }
  int insert(const string &s) {
    int ptr = 1;
    for (char c : s) { // change char id
      c -= 'a';
      if (!T[ptr].Next[c]) T[ptr].Next[c] = get_node(ptr);
      ptr = T[ptr].Next[c];
    }
    return ptr;
  } // return ans_last_place
  void build_fail(int ptr) {
    int tmp;
    for (int i = 0; i < maxc; i++)
      if (T[ptr].Next[i]) {
        tmp = T[ptr].fail;
        while (tmp != 1 && !T[tmp].Next[i])
          tmp = T[tmp].fail;
        if (T[tmp].Next[i] != T[ptr].Next[i])
          if (T[tmp].Next[i]) tmp = T[tmp].Next[i];
        T[T[ptr].Next[i]].fail = tmp;
        q[qtop++] = T[ptr].Next[i];
      }
  }
  void AC_auto(const string &s) {
    int ptr = 1;
    for (char c : s) {
      while (ptr != 1 && !T[ptr].Next[c]) ptr = T[ptr].fail;
      if (T[ptr].Next[c]) {
        ptr = T[ptr].Next[c];
        T[ptr].ans++;
      }
    }
  }
  void Solve(string &s) {
    for (char &c : s) // change char id
      c -= 'a';
    for (int i = 0; i < qtop; i++) build_fail(q[i]);
    AC_auto(s);
    for (int i = qtop - 1; i > -1; i--)
      T[T[q[i]].fail].ans += T[q[i]].ans;
  }
  void reset() {
    qtop = top = q[0] = 1;
    get_node(1);
  }
} AC;
// usage example
string s, S;
int n, t, ans_place[50000];
int main() {
  Tie cin >> t;
  while (t--) {
    AC.reset();
    cin >> S >> n;
    for (int i = 0; i < n; i++) {
      cin >> s;
      ans_place[i] = AC.insert(s);
    }
    AC.Solve(S);
    for (int i = 0; i < n; i++)
      cout << AC.T[ans_place[i]].ans << '\n';
  }
}
```

8.   Debug List

```
- Pre-submit:
  - Did you make a typo when copying a template?
  - Test more cases if unsure.
    - Write a naive solution and check small cases.
  - Submit the correct file.

- General Debugging:
  - Read the whole problem again.
  - Have a teammate read the problem.
  - Have a teammate read your code.
    - Explain you solution to them (or a rubber duck).
  - Print the code and its output / debug output.
  - Go to the toilet.

- Wrong Answer:
  - Any possible overflows?
    - > `__int128` ?
    - Try `-ftrapv` or `#pragma GCC optimize("trapv")`
  - Floating point errors?
    - > `long double` ?
    - turn off math optimizations
    - check for `==`, `>=`, `acos(1.000000001)`, etc.
  - Did you forget to sort or unique?
  - Generate large and worst "corner" cases.
  - Check your `m` / `n`, `i` / `j` and `x` / `y`.
  - Are everything initialized or reset properly?
  - Are you sure about the STL thing you are using?
    - Read cppreference (should be available).
  - Print everything and run it on pen and paper.

- Time Limit Exceeded:
  - Calculate your time complexity again.
  - Does the program actually end?
    - Check for `while(q.size())` etc.
  - Test the largest cases locally.
  - Did you do unnecessary stuff?
    - e.g. pass vectors by value
    - e.g. `memset` for every test case
  - Is your constant factor reasonable?

- Runtime Error:
  - Check memory usage.
    - Forget to clear or destroy stuff?
    - > `vector::shrink_to_fit()`
  - Stack overflow?
  - Bad pointer / array access?
    - Try `-fsanitize=address`
  - Division by zero? NaN's?
```