Contents

1 Misc

1.1 Contest

1.1.1 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?
```

1.1.2 Makefile

```
.PRECIOUS: ./p%

%: p%
  ulimit -s unlimited && ./$<

p%: p%.cpp
  g++ -o $@ $< -std=gnu++17 -Wall -Wextra -Wshadow \
  -fsanitize=address -fsanitize=undefined

init:
  for i in a b c d e f g h; do \
    cp default.cpp "p$$i.cpp"; \
  done
```

1.1.3 Default Code

```cpp
#include <bits/stdc++.h>

#define pb push_back
#define eb emplace_back
#define F first
#define S second
#define SZ(v) ((int)(v).size())
#define ALL(v) (v).begin(),(v).end()
#define MEM(a,b) memset(a,b,sizeof a)

using namespace std;
using ll = long long;
using ld = long double;
using LL = __int128;
using pii = pair<int, int>;
using pll = pair<ll, ll>;

int main() {
  ios::sync_with_stdio(0), cin.tie(0);
}
```

1.1.4 acceleration yes

```cpp
#pragma GCC optimize ("O3", "unroll-loops")
// #pragma GCC optimize ("fast-math")
#pragma GCC target("avx,avx2,fma,mmx,abm,bmi,bmi2")
// tip: `lscpu` on the contest machine

// for math:
//  - writing your own complex, etc. can increase speed
//  - use Ofast on floating points with caution!
// stuff to put before a loop:
//  - unroll count (0 or 1 -> no unroll)
#pragma GCC unroll 10
//  - for a loop that you're sure can be vectorized
#pragma GCC ivdep

// to see what are optimized, call gcc with
//  -fopt-info -fopt-info-loop -fopt-info-loop-missed
//  -fopt-info-vec -fopt-info-vec-missed
```

1.1.5 C++17 constexpr

```cpp
// constexpr array example
constexpr array<int, 100> fibonacci {
  [] {
    array<int, 100> a{};
    a[0] = a[1] = 1;
    for (int i = 2; i < 100; i++)
      a[i] = (a[i - 1] + a[i - 2]) % 420;
    return a;
  }()
};
static_assert(fibonacci[9] == 55, "CE");

// some default limits in g++ (7.x - trunk):
// constexpr recursion depth: 512
// constexpr loop iteration (per function): 262144
// constexpr operation count (per function): 33554432
// template recursion depth: 900 (g++ might segfault)
// use recursion / unrolling / TMP for more calculations.

// WARNING:
//   g++ 10 has a bug that incorrectly rejects some usages
//   g++ 11 has it fixed, and g++ 7.x - 9.x work fine
// "for constexpr"
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 usage
template<typename ...T>
void print_tuple(tuple<T...> t) {
  for_constexpr(
    make_index_sequence<sizeof...(T)>{},
    [&](auto index) { cout << get<index>(t) << ' '; }
  );
  cout << '\n';
}
```

1.1.6 Changing Stack Size

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

1.2 Utils

1.2.1 SplitMix64

```
using ull = unsigned long long;
inline ull splitmix64(ull x) {
  // static ull x = seed;
  ull z = (x += 0x9E3779B97F4A7C15);
  z = (z ^ (z >> 30)) * 0xBF58476D1CE4E5B9;
  z = (z ^ (z >> 27)) * 0x94D049BB133111EB;
  return z ^ (z >> 31);
}
```

1.2.2 Random

```
#include <random>
#include <chrono>

using namespace std;

// most judges and contest environments run linux
// but we should not trust anyone
#ifdef __unix__
  random_device rd;
  mt19937_64 RNG(rd());
#else
  const auto SEED = chrono::high_resolution_clock::now()
      .time_since_epoch().count();
  mt19937_64 RNG(SEED);
#endif

// usage:
// random long long: RNG();
// uniformly random ll in [l, r]:
//   uniform_int_distribution<int> dt(l, r); dt(RNG);
// uniformly random double in [l, r]:
//   uniform_real_distribution<double> dt(l, r); dt(RNG);
```

1.2.3 Bit Twiddling

```
ull next_permutation(ull x) {
  ull c = __builtin_ctzll(x), r = x + (1 << c);
  return (r ^ x) >> (c + 2) | r;
}
// iterate over all (proper) subsets of bitset s
void subsets(ull s) {
  for (ull x = s; x; ) { --x &= s; /* do stuff */ }
}
```

1.2.4 Floating Point Comparison

```
long long rc(double x) { return *(long long*)&x; }
// relative error: 2^-28 ~ 3.7e-9
// absolute error: 1e-8
bool is_equal(double a, double b) {
  return abs(rc(a) - rc(b)) < (1LL << (52 - 24)) ||
    abs(a - b) < 1e-8;
}
```

1.2.5 Floating Point Binary Search

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

1.3 Misc Algorithms

1.3.1 Mo's Algorithm on Tree

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

1.3.2 Longest Increasing Subsequence

```
// source: KACTL

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

1.3.3 Aliens Trick

```
// min dp[i] value and its i (smallest one)
pll get_dp(int n);
ll aliens(int n) {
  int l = 0, r = 1000000;
  while (l != r) {
    int m = (l + r) / 2;
    pll res = get_dp(m);
    if (res.S == n) return res.F - m * n;
    if (res.S < n) r = m;
    else l = m + 1;
  }
  l--;
  return get_dp(l).F - l * n;
}
```

2 Data Structures

2.1 GNU PBDS

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

// most of std::map + order_of_key, find_by_order
template<typename T, typename U = null_type>
```

```cpp
using ordered_map = tree<T, U, std::less<T>,
rb_tree_tag, tree_order_statistics_node_update>;
// rb_tree_tag can be changed to splay_tree_tag

template<typename T> struct myhash {
  size_t operator()(T x) const;  // splitmix64, etc.
};
// mostly the same as std::unordered_map
template<typename T, typename U = null_type>
using hash_table = gp_hash_table<T, U, myhash<T>>;

// most of std::priority_queue + merge
using heap = priority_queue<int, std::less<int>>;
// the third template parameter is tag, useful ones are
// pairing_heap_tag, binary_heap_tag, binomial_heap_tag

// similar to treap, has insert/delete range, merge, etc.
using __gnu_cxx::rope;
```

2.2 Segment Tree

```cpp
template<typename T> struct tree {
  static const T ID; // identity element
  T f(T, T);  // associative operator
  int n;
  vector<T> v;
  tree(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 << 1], v[i << 1 | 1]);
  }
  void update(int pos, T val) {
    for(v[pos += n] = val; pos > 1; pos >>= 1)
      v[pos >> 1] = f(v[pos & -2], v[(pos & -2) ^ 1]);
  }
  T query(int l, int r) {
    T tl = ID, tr = ID;
    for(l += n, r += n; l < r; l >>= 1, r >>= 1) {
      if(l & 1) tl = f(tl, v[l++]);
      if(r & 1) tr = f(v[--r], tr);
    }
    return f(tl, tr);
  }
};
```

2.3 Line Container

```cpp
// source: kactl

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;
  }
};
```

2.4 Heavy-Light Decomposition

```cpp
// source: KACTL

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]);
  }
};
```

2.5 Link-cut Tree

```cpp
// source: bcw codebook

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;

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

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

Splay* access(Splay *x) {
  Splay *q = nil;
  for (;x!=nil;x=x->f) {
    splay(x);
    x->setCh(q, 1);
    q = x;
  }
  return q;
}
void evert(Splay *x) {
  access(x);
  splay(x);
  x->rev ^= 1;
  x->push(); x->pull();
}
void link(Splay *x, Splay *y) {
//  evert(x);
  access(x);
  splay(x);
  evert(y);
  x->setCh(y, 1);
}
void cut(Splay *x, Splay *y) {
//  evert(x);
  access(y);
  splay(y);
  y->push();
  y->ch[0] = y->ch[0]->f = nil;
}

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

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

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

3 Math

3.1 Number Theory

3.1.1 Modular

```
template<typename T> struct M {
  static T MOD;
  T v;
  M() : v(0) {}
  M(T x) {
    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)); }
  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<ll>;
template<>ll Mod::MOD = 1000000007;
ll &MOD = Mod::MOD;

/* Safe primes
 * 21673, 26497, 22621, 21817, 28393, 26821, 30181, 22093
 * 977680993, 971939533, 970479637, 910870273, 1041012121
 * 741266610070171837, 1110995545625882557
 * NTT prime           | p - 1       | primitive root
 * 65537               | (2^16)      | 3
 * 998244353           | (2^23)*119  | 3
 * 2748779069441       | (2^39)*5    | 3
 * 1945555039024054273 | (2^56)*27   | 5              */
```

3.1.2 Extended GCD

```
tuple<ll, ll, ll> extgcd(ll a, ll b) {
  if (b == 0) return { 1, 0, a };
  else {
    auto [p, q, g] = extgcd(b, a % b);
    return { q, p - q * (a / b), g };
  }
}
```

3.1.3 Chinese Remainder

```
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;
}
```

3.1.4 Sieve

```
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;
      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);
    }
  }
}
```

3.1.5 Miller-Rabin

```
// checks if Mod::MOD is prime
bool is_prime() {
  if (MOD <= 1 || MOD % 2 == 0) return MOD == 2;
  // Mod A[] = {2, 7, 61};
  Mod A[] = { 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 && ll(x + 1) > 2; i++, x *= x);
    if (i && x != -1) return 0;
  }
  return 1;
}
```

3.1.6 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;
  }
}
```

3.1.7 Tonelli-Shanks

```
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;
}
```

3.1.8 Baby-Step Giant-Step

```
// 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
}
```

3.1.9 Multiplicative Function Sum

```
const ll N = 1000000;
ll pre_g(ll n);
ll pre_h(ll n);
// preprocessed prefix sum of f
ll pre_f[N];
// djs: prefix sum of multiplicative function f
ll djs_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;
}
```

3.2 Combinatorics

3.2.1 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);
  }
}
int DeBruijn(int k, int n) {
  // return cyclic string of length k^n such that every
  ↪  string of length n using k character appears as a
  ↪  substring.
  if (k == 1) return res[0] = 0, 1;
  fill(aux, aux + k * n, 0);
  return sz = 0, Rec(1, 1, n, k), sz;
}
```

3.2.2 Multinomial

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

3.2.3 Not 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

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|.$$

3.2.4 Matroid Intersection

Let $M_1 = (E, \mathcal{I}_1), M_2 = (E, \mathcal{I}_2)$ be matroids. We want to find a set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ with maximum cardinality.

Start with $S = \varnothing$. Repeat the following while feasible:

1. Let $Y_1 = \{x \notin S \mid S + x \in I_1\}$ and $Y_2 = \{x \notin S \mid S + x \in I_2\}$.
2. If $Y_1 \cap Y_2 \neq \varnothing$, we can choose any $x \in Y_1 \cap Y_2$ and add x to S, then go back to step 1.
3. Build a directed bipartite graph. For each $x \in S, y \in E \backslash S$:
   - $x \to y$ iff $S - x + y \in Y_1$.
   - $x \leftarrow y$ iff $S - x + y \in Y_2$.
4. Find a augmenting path from Y_1 to Y_2 without shortcuts (using BFS), then set S to $S \Delta P$ for where P is the set of vertices on the path.

To find the maximum weighted S: For each $x \in S$, assign weight $w(x)$ to vertex x, and for each $y \in E \backslash S$, assign weight $-w(y)$ to vertex y. Then on step 4, find the minimum weighted path from Y_1 to Y_2 (using Bellman-Ford) instead, and if there are multiple, choose the one with the least edges.

3.3 Theorems

Source: waynedisonitau123

3.3.1 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 $|\det(\tilde{L}_{rr})|$.

3.3.2 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.

3.3.3 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 $T_{n,k} = kn^{n-k-1}$.

3.3.4 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 Numeric

4.1 long long Multiplication

```
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);
}
```

4.2 Barrett Reduction

```
using ull = unsigned long long;
using uL = __uint128_t;
// very fast calculation of a % m
struct reduction {
  const ull m, d;
  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;
  }
};
```

4.3 Polynomial Interpolation

```
// source: KACTL

typedef vector<double> vd;
vd interpolate(vd x, vd y, int n) {
  vd res(n), temp(n);
  rep(k,0,n-1) rep(i,k+1,n)
    y[i] = (y[i] - y[k]) / (x[i] - x[k]);
  double last = 0; temp[0] = 1;
  rep(k,0,n) rep(i,0,n) {
    res[i] += y[k] * temp[i];
    swap(last, temp[i]);
    temp[i] -= last * x[k];
  }
  return res;
}
```

4.4 Fast Fourier Transform

```
template<typename T>
void work(int n, vector<T>& a, vector<T>& rt, bool inv) {
  for (int i = 1, r = 0; i < n; i++) {
    for (int bit = n; !(r & bit); bit >>= 1, r ^= bit);
    if (r > i) swap(a[i], a[r]);
  }
  for (int len = 2; len <= n; len <<= 1)
    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 (inv) {
    T minv = T(1) / T(n);
    for (T& x : a) x *= minv;
  }
}
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) };
  work(n, a, rt, inv);
}
void NTT(vector<Mod>& a, bool inv, Mod p_root) {
  int n = a.size();
  Mod root = p_root ^ (MOD - 1) / n;
  vector<Mod> rt(n + 1, 1);
  for (int i = 0; i < n; i++) rt[i + 1] = rt[i] * root;
  work(n, a, rt, inv);
}
```

4.5 Fast Walsh-Hadamard Transform

```
// source: waynedisonitau123

void xorfwt(int v[], int l, int r) {
  if (r - l == 1) return;
  int m = l + r >> 1;
  xorfwt(v, l, m), xorfwt(v, m, r);
  for (int i = l, j = m; i < m; ++i, ++j) {
    int x = v[i] + v[j];
    v[j] = v[i] - v[j], v[i] = x;
  }
}

void xorifwt(int v[], int l, int r) {
  if (r - l == 1) return;
  int m = l + r >> 1;
  for (int i = l, j = m; i < m; ++i, ++j) {
    int x = (v[i] + v[j]) / 2;
    v[j] = (v[i] - v[j]) / 2, v[i] = x;
  }
  xorifwt(v, l, m), xorifwt(v, m, r);
}

void andfwt(int v[], int l, int r) {
  if (r - l == 1) return;
  int m = l + r >> 1;
  andfwt(v, l, m), andfwt(v, m, r);
  for (int i = l, j = m; i < m; ++i, ++j) v[i] += v[j];
}

void andifwt(int v[], int l, int r) {
  if (r - l == 1) return;
  int m = l + r >> 1;
  andifwt(v, l, m), andifwt(v, m, r);
  for (int i = l, j = m; i < m; ++i, ++j) v[i] -= v[j];
}

void orfwt(int v[], int l, int r) {
  if (r - l == 1) return;
  int m = l + r >> 1;
  orfwt(v, l, m), orfwt(v, m, r);
  for (int i = l, j = m; i < m; ++i, ++j) v[j] += v[i];
}

void orifwt(int v[], int l, int r) {
  if (r - l == 1) return;
  int m = l + r >> 1;
  orifwt(v, l, m), orifwt(v, m, r);
  for (int i = l, j = m; i < m; ++i, ++j) v[j] -= v[i];
}
```

4.6 FFT Convolution

```
// source: waynedisonitau123

vector<long long> convolution(const vector<int> &a, const
↪  vector<int> &b) {
  // Should be able to handle N <= 10^5, C <= 10^4
  int sz = 1;
  while (sz < a.size() + b.size() - 1) sz <<= 1;
  vector<cplx> v(sz);
  for (int i = 0; i < sz; ++i) {
    double re = i < a.size() ? a[i] : 0;
    double im = i < b.size() ? b[i] : 0;
    v[i] = cplx(re, im);
  }
  fft(v, sz);
  for (int i = 0; i <= sz / 2; ++i) {
    int j = (sz - i) & (sz - 1);
    cplx x = (v[i] + v[j].conj()) * (v[i] - v[j].conj()) *
    ↪  cplx(0, -0.25);
    if (j != i) v[j] = (v[j] + v[i].conj()) * (v[j] -
    ↪  v[i].conj()) * cplx(0, -0.25);
    v[i] = x;
  }
  ifft(v, sz);
  vector<long long> c(sz);
  for (int i = 0; i < sz; ++i) c[i] = round(v[i].re);
  return c;
}
vector<int> convolution_mod(const vector<int> &a, const
↪  vector<int> &b, int p) {
  int sz = 1;
  while (sz < (int)a.size() + (int)b.size() - 1) sz <<= 1;
  vector<cplx> fa(sz), fb(sz);
  for (int i = 0; i < (int)a.size(); ++i)
    fa[i] = cplx(a[i] & ((1 << 15) - 1), a[i] >> 15);
  for (int i = 0; i < (int)b.size(); ++i)
    fb[i] = cplx(b[i] & ((1 << 15) - 1), b[i] >> 15);
  fft(fa, sz), fft(fb, sz);
  double r = 0.25 / sz;
  cplx r2(0, -1), r3(r, 0), r4(0, -r), r5(0, 1);
  for (int i = 0; i <= (sz >> 1); ++i) {
    int j = (sz - i) & (sz - 1);
    cplx a1 = (fa[i] + fa[j].conj());
    cplx a2 = (fa[i] - fa[j].conj()) * r2;
    cplx b1 = (fb[i] + fb[j].conj()) * r3;
    cplx b2 = (fb[i] - fb[j].conj()) * r4;
    if (i != j) {
      cplx c1 = (fa[j] + fa[i].conj());
      cplx c2 = (fa[j] - fa[i].conj()) * r2;
      cplx d1 = (fb[j] + fb[i].conj()) * r3;
      cplx d2 = (fb[j] - fb[i].conj()) * r4;
      fa[i] = c1 * d1 + c2 * d2 * r5;
      fb[i] = c1 * d2 + c2 * d1;
    }
    fa[j] = a1 * b1 + a2 * b2 * r5;
    fb[j] = a1 * b2 + a2 * b1;
  }
  fft(fa, sz), fft(fb, sz);
  vector<int> res(sz);
  for (int i = 0; i < sz; ++i) {
    long long a = round(fa[i].re);
    long long b = round(fb[i].re);
    long long c = round(fa[i].im);
    res[i] = (a + ((b % p) << 15) + ((c % p) << 30)) % p;
  }
  return res;
}}
```

4.7 Linear Recurrence

4.7.1 Calculation

```
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;
  }
};
```

4.7.2 Berlekamp-Massey

```
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;
}
```

4.7.3 Composite Modulus Recurrence

```
// source: min-25

using i64 = long long;
using Matrix = vector< vector<int> >;

vector<int> linear_recurrence_mod(const vector<int>& terms,
↪  const int mod) {
  const int N = terms.size() / 2;
  Matrix A(N, vector<int>(N + 1));
  for (int y = 0; y < N; ++y)
    for (int x = 0; x < N + 1; ++x)
      if ((A[y][x] = terms[x + y] % mod) < 0) A[y][x] += mod;
  int r = 0;
  for (int x = 0; x < N; ++x, ++r) {
    for (int y = x + 1; y < N; ++y) {
      while (A[y][x] > 0) {
        if (A[y][x] < A[x][x] || A[x][x] == 0) {
          for (int x2 = x; x2 < N + 1; ++x2) swap(A[x][x2],
          ↪  A[y][x2]);
        }
        int mq = mod - A[y][x] / A[x][x];
        for (int x2 = x; x2 < N + 1; ++x2) A[y][x2] =
        ↪  (A[y][x2] + i64(mq) * A[x][x2]) % mod;
      }
    }
    if (A[x][x] == 0) break;
  }
  vector<int> f(r + 1); f[0] = 1;
  for (int x = r - 1; x >= 0; --x) if (A[x][x]) {
    int g = __gcd(mod, A[x][x]); assert(A[x][r] % g == 0);
    int mc = (mod - i64(A[x][r] / g) * mod_inv(A[x][x] / g,
    ↪  mod / g) % mod) % mod;
    f[r - x] = mc;
    for (int y = x - 1; y >= 0; --y) A[y][r] = (A[y][r] +
    ↪  i64(mc) * A[y][x]) % mod;
  }
  return f;
}
```

4.8 Matrix Determinant

```
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;
}

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;
}
```

4.9 Matrix Inverse

```
// source: KACTL

// Returns rank.
// Result is stored in A unless singular (rank < n).
// For prime powers, repeatedly set
↪  A^{-1} = A^{-1}(2I - AA^{-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;
  }

  /// forget A at this point, just eliminate tmp backward
  for (int i = n-1; i > 0; --i) rep(j,0,i) {
    double v = A[j][i];
    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;
}

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;
}
```

4.10 Linear Equations

```
// source: KACTL

typedef vector<double> vd;
const double eps = 1e-12;

// solves for x: A * x = b
int solveLinear(vector<vd>& A, vd& b, vd& x) {
  int n = sz(A), m = sz(x), rank = 0, br, bc;
  if (n) assert(sz(A[0]) == m);
  vi col(m); iota(all(col), 0);

  rep(i,0,n) {
    double v, bv = 0;
    rep(r,i,n) rep(c,i,m)
      if ((v = fabs(A[r][c])) > bv)
        br = r, bc = c, bv = v;
    if (bv <= eps) {
      rep(j,i,n) if (fabs(b[j]) > eps) return -1;
      break;
    }
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    rep(j,0,n) swap(A[j][i], A[j][bc]);
    bv = 1/A[i][i];
    rep(j,i+1,n) {
      double fac = A[j][i] * bv;
      b[j] -= fac * b[i];
      rep(k,i+1,m) A[j][k] -= fac*A[i][k];
    }
    rank++;
  }

  x.assign(m, 0);
  for (int i = rank; i--;) {
    b[i] /= A[i][i];
    x[col[i]] = b[i];
    rep(j,0,i) b[j] -= A[j][i] * b[i];
  }
  return rank; // (multiple solutions if rank < m)
}
```

4.11 Simplex

```
// Two-phase simplex algorithm for solving linear programs
↪  of the form
//
//     maximize     c^T x
//     subject to   Ax <= b
//                  x >= 0
//
// INPUT: A -- an m x n matrix
//        b -- an m-dimensional vector
//        c -- an n-dimensional vector
//        x -- a vector where the optimal solution will be
↪  stored
//
// OUTPUT: value of the optimal solution (infinity if
↪  unbounded
//         above, nan if infeasible)
//
// To use this code, create an LPSolver object with A, b,
↪  and c as
// arguments.  Then, call Solve(x).

typedef long double ld;
typedef vector<ld> vd;
typedef vector<vd> vvd;
typedef vector<int> vi;

const ld EPS = 1e-9;

struct LPSolver {
  int m, n;
  vi B, N;
  vvd D;

  LPSolver(const vvd &A, const vd &b, const vd &c) :
    m(b.size()), n(c.size()), N(n + 1), B(m), D(m + 2, vd(n
    ↪  + 2)) {
    for (int i = 0; i < m; i++) for (int j = 0; j < n; j++)
    ↪  D[i][j] = A[i][j];
    for (int i = 0; i < m; i++) { B[i] = n + i; D[i][n] =
    ↪  -1; D[i][n + 1] = b[i]; }
    for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j];
    ↪  }
    N[n] = -1; D[m + 1][n] = 1;
  }

  void Pivot(int r, int s) {
    double inv = 1.0 / D[r][s];
    for (int i = 0; i < m + 2; i++) if (i != r)
      for (int j = 0; j < n + 2; j++) if (j != s)
        D[i][j] -= D[r][j] * D[i][s] * inv;
    for (int j = 0; j < n + 2; j++) if (j != s) D[r][j] *=
    ↪  inv;
    for (int i = 0; i < m + 2; i++) if (i != r) D[i][s] *=
    ↪  -inv;
    D[r][s] = inv;
    swap(B[r], N[s]);
  }

  bool Simplex(int phase) {
    int x = phase == 1 ? m + 1 : m;
    while (true) {
      int s = -1;
      for (int j = 0; j <= n; j++) {
        if (phase == 2 && N[j] == -1) continue;
        if (s == -1 || D[x][j] < D[x][s] || D[x][j] ==
        ↪  D[x][s] && N[j] < N[s]) s = j;
      }
      if (D[x][s] > -EPS) return true;
      int r = -1;
      for (int i = 0; i < m; i++) {
        if (D[i][s] < EPS) continue;
        if (r == -1 || D[i][n + 1] / D[i][s] < D[r][n + 1] /
        ↪  D[r][s] ||
          (D[i][n + 1] / D[i][s]) == (D[r][n + 1] / D[r][s])
          ↪  && B[i] < B[r]) r = i;
      }
      if (r == -1) return false;
      Pivot(r, s);
    }
  }

  ld Solve(vd &x) {
    int r = 0;
    for (int i = 1; i < m; i++) if (D[i][n + 1] < D[r][n +
    ↪  1]) r = i;
    if (D[r][n + 1] < -EPS) {
      Pivot(r, n);
      if (!Simplex(1) || D[m + 1][n + 1] < -EPS) return
      ↪  -numeric_limits<ld>::infinity();
      for (int i = 0; i < m; i++) if (B[i] == -1) {
        int s = -1;
        for (int j = 0; j <= n; j++)
          if (s == -1 || D[i][j] < D[i][s] || D[i][j] ==
          ↪  D[i][s] && N[j] < N[s]) s = j;
        Pivot(i, s);
      }
    }
    if (!Simplex(2)) return numeric_limits<ld>::infinity();
    x = vd(n);
    for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] =
    ↪  D[i][n + 1];
    return D[m][n + 1];
  }
};

int main() {

  const int m = 4;
  const int n = 3;
  ld _A[m][n] = {
```

```
    { 6, -1, 0 },
    { -1, -5, 0 },
    { 1, 5, 1 },
    { -1, -5, -1 }
  };
  ld _b[m] = { 10, -4, 5, -5 };
  ld _c[n] = { 1, -1, 0 };

  vvd A(m);
  vd b(_b, _b + m);
  vd c(_c, _c + n);
  for (int i = 0; i < m; i++) A[i] = vd(_A[i], _A[i] + n);

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

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

5 Graph

5.1 Modeling

Source: waynedisonitau123

- 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 $c_{xyx'y'}$.

5.2 Flow

5.2.1 Dinic

```
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();
  }
};
```

5.2.2 Gomory-Hu Tree

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

5.2.3 Global Minimum Cut

```cpp
// source: waynedisonitau123

int w[kN][kN], g[kN], del[kN], v[kN];
void AddEdge(int x, int y, int c) {
  w[x][y] += c;
  w[y][x] += c;
}
pair<int, int> Phase(int n) {
  fill(v, v + n, 0), fill(g, g + n, 0);
  int s = -1, t = -1;
  while (true) {
    int c = -1;
    for (int i = 0; i < n; ++i) {
      if (del[i] || v[i]) continue;
      if (c == -1 || g[i] > g[c]) c = i;
    }
    if (c == -1) break;
    v[c] = 1, s = t, t = c;
    for (int i = 0; i < n; ++i) {
      if (del[i] || v[i]) continue;
      g[i] += w[c][i];
    }
  }
  return make_pair(s, t);
}
int GlobalMinCut(int n) {
  int cut = kInf;
  fill(del, 0, sizeof(del));
  for (int i = 0; i < n - 1; ++i) {
    int s, t; tie(s, t) = Phase(n);
    del[t] = 1, cut = min(cut, g[t]);
    for (int j = 0; j < n; ++j) {
      w[s][j] += w[t][j];
      w[j][s] += w[j][t];
    }
  }
  return cut;
}
```

5.2.4 Min Cost Max Flow

```cpp
// source: KACTL

const ll INF = numeric_limits<ll>::max() / 4;
typedef vector<ll> VL;

struct MCMF {
  int N;
  vector<vi> ed, red;
  vector<VL> cap, flow, cost;
  vi seen;
  VL dist, pi;
  vector<pii> par;

  MCMF(int N) :
    N(N), ed(N), red(N), cap(N, VL(N)), flow(cap), cost(cap),
    seen(N), dist(N), pi(N), par(N) {}

  void addEdge(int from, int to, ll cap, ll cost) {
    this->cap[from][to] = cap;
    this->cost[from][to] = cost;
    ed[from].push_back(to);
    red[to].push_back(from);
  }

  void path(int s) {
    fill(all(seen), 0);
    fill(all(dist), INF);
    dist[s] = 0; ll di;

    __gnu_pbds::priority_queue<pair<ll, int>> q;
    vector<decltype(q)::point_iterator> its(N);
    q.push({0, s});

    auto relax = [&](int i, ll cap, ll cost, int dir) {
      ll val = di - pi[i] + cost;
      if (cap && val < dist[i]) {
        dist[i] = val;
        par[i] = {s, dir};
        if (its[i] == q.end()) its[i] = q.push({-dist[i],
        ↪  i});
        else q.modify(its[i], {-dist[i], i});
      }
    };

    while (!q.empty()) {
      s = q.top().second; q.pop();
      seen[s] = 1; di = dist[s] + pi[s];
      for (int i : ed[s]) if (!seen[i])
        relax(i, cap[s][i] - flow[s][i], cost[s][i], 1);
      for (int i : red[s]) if (!seen[i])
        relax(i, flow[i][s], -cost[i][s], 0);
    }
    rep(i,0,N) pi[i] = min(pi[i] + dist[i], INF);
  }

  pair<ll, ll> maxflow(int s, int t) {
    ll totflow = 0, totcost = 0;
    while (path(s), seen[t]) {
      ll fl = INF;
      for (int p,r,x = t; tie(p,r) = par[x], x != s; x = p)
        fl = min(fl, r ? cap[p][x] - flow[p][x] :
        ↪  flow[x][p]);
      totflow += fl;
      for (int p,r,x = t; tie(p,r) = par[x], x != s; x = p)
        if (r) flow[p][x] += fl;
        else flow[x][p] -= fl;
    }
    rep(i,0,N) rep(j,0,N) totcost += cost[i][j] * flow[i][j];
    return {totflow, totcost};
  }

  // If some costs can be negative, call this before maxflow:
  void setpi(int s) { // (otherwise, leave this out)
    fill(all(pi), INF); pi[s] = 0;
    int it = N, ch = 1; ll v;
    while (ch-- && it--)
      rep(i,0,N) if (pi[i] != INF)
        for (int to : ed[i]) if (cap[i][to])
          if ((v = pi[i] + cost[i][to]) < pi[to])
            pi[to] = v, ch = 1;
    assert(it >= 0); // negative cost cycle
  }
};
```

5.3 Matching

5.3.1 Kuhn-Munkres

```cpp
// Maximum Weight Perfect Bipartite Matching
// Detect non-perfect-matching:
// 1. set all edge[i][j] as INF
// 2. if solve() >= INF, it is not perfect matching.

typedef long long ll;
struct KM {
  static const int MAXN = 1050;
  static const ll INF = 1LL<<60;
  int n, match[MAXN], vx[MAXN], vy[MAXN];
  ll edge[MAXN][MAXN], lx[MAXN], ly[MAXN], slack[MAXN];
  void init(int _n) {
    n = _n;
    for ( int i = 0 ; i < n ; i++ )
      for ( int j = 0; j < n ; j++ )
        edge[i][j] = 0;
  }
```

```
  void add_edge(int x, int y, ll w) {
    edge[x][y] = w;
  }
  bool DFS(int x) {
    vx[x] = 1;
    for ( int y = 0 ; y < n ; y++ ) {
      if ( vy[y] ) continue;
      if ( lx[x] + ly[y] > edge[x][y] ) {
        slack[y] = min(slack[y], lx[x] + ly[y] - edge[x][y]);
      } else {
        vy[y] = 1;
        if ( match[y] == -1 || DFS(match[y]) ) {
          match[y] = x;
          return true;
        }
      }
    }
    return false;
  }
  ll solve() {
    fill(match, match + n, -1);
    fill(lx, lx + n, -INF);
    fill(ly, ly + n, 0);
    for ( int i = 0; i < n; i++ )
      for ( int j = 0; j < n; j++ )
        lx[i] = max(lx[i], edge[i][j]);
    for ( int i = 0 ; i < n; i++ ) {
      fill(slack, slack + n, INF);
      while (true) {
        fill(vx, vx + n, 0);
        fill(vy, vy + n, 0);
        if ( DFS(i) ) break;
        ll d = INF;
        for ( int j = 0 ; j < n ; j++ )
          if ( !vy[j] ) d = min(d, slack[j]);
        for ( int j = 0 ; j < n ; j++ ) {
          if (vx[j]) lx[j] -= d;
          if (vy[j]) ly[j] += d;
          else slack[j] -= d;
        }
      }
    }
    ll res = 0;
    for ( int i = 0 ; i < n ; i++ ) {
      res += edge[ match[i] ][i];
    }
    return res;
  }
} graph;
```

5.3.2 Bipartite Minimum Vertex Cover

```
// maximum independent set = all vertices not covered
// include Dinic, x : [0, n), y : [0, m]
struct Bipartite_vertex_cover {
  Dinic D;
  int n, m, s, t, x[maxn], y[maxn];
  void make_edge(int x, int y) {D.make_edge(x, y + n, 1);}
  int matching() {
    int re = D.max_flow(s, t);
    for(int i = 0; i < n; i++)
      for(Dinic::edge &e : D.v[i])
        if(e.to != s && e.flow == 1) {
          x[i] = e.to - n, y[e.to - n] = i;
          break;
        }
    return re;
  }
  // init() and matching() before use
  void solve(vector<int> &vx, vector<int> &vy) {
    bitset<maxn * 2 + 10> vis;
    queue<int> q;
    for(int i = 0; i < n; i ++)
      if(x[i] == -1)
        q.push(i), vis[i] = 1;
    while(!q.empty()) {
      int now = q.front();
      q.pop();
      if(now < n) {
        for(Dinic::edge &e : D.v[now])
          if(e.to != s && e.to - n != x[now] && !vis[e.to])
            vis[e.to] = 1, q.push(e.to);
      } else {
        if(!vis[y[now - n]])
          vis[y[now - n]] = 1, q.push(y[now - n]);
      }
    }
    for(int i = 0; i < n; i++)
      if(!vis[i])
        vx.pb(i);
    for(int i = 0; i < m; i++)
      if(vis[i + n])
        vy.pb(i);
  }
  void init(int _n, int _m) {
    n = _n, m = _m, s = n + m, t = s + 1;
    for(int i = 0; i < n; i++)
      x[i] = -1, D.make_edge(s, i, 1);
    for(int i = 0; i < m; i++)
      y[i] = -1, D.make_edge(i + n, t, 1);
  }
};
```

5.3.3 Maximum General Matching

```
// source: Sergey Kopeliovich (burunduk30@gmail.com)

struct Edmonds {
  int n, T;
  vector<vector<int>> g;
  vector<int> pa, p, used, base;
  Edmonds(int n) :
    n(n), T(0), g(n), pa(n, -1), p(n), used(n), base(n) {}
  void add(int a, int b) {
    g[a].push_back(b);
    g[b].push_back(a);
  }
  int getBase(int i) {
    while (i != base[i])
      base[i] = base[base[i]], i = base[i];
    return i;
  }
  vector<int> toJoin;
  void mark_path(int v, int x, int b, vector<int> &path) {
    for (; getBase(v) != b; v = p[x]) {
      p[v] = x, x = pa[v];
      toJoin.push_back(v);
      toJoin.push_back(x);
      if (!used[x])
        used[x] = ++T, path.push_back(x);
    }
  }
  bool go(int v) {
    for (int x : g[v]) {
      int b, bv = getBase(v), bx = getBase(x);
      if (bv == bx) {
        continue;
      } else if (used[x]) {
        vector<int> path;
        toJoin.clear();
        if (used[bx] < used[bv])
          mark_path(v, x, b = bx, path);
        else
          mark_path(x, v, b = bv, path);
        for (int z : toJoin)
          base[getBase(z)] = b;
        for (int z : path)
          if (go(z))
            return 1;
      } else if (p[x] == -1) {
        p[x] = v;
        if (pa[x] == -1) {
          for (int y; x != -1; x = v)
            y = p[x], v = pa[y], pa[x] = y, pa[y] = x;
          return 1;
        }
        if (!used[pa[x]]) {
          used[pa[x]] = ++T;
          if (go(pa[x]))
            return 1;
        }
      }
    }
    return 0;
  }
  void init_dfs() {
    for (int i = 0; i < n; i++)
      used[i] = 0, p[i] = -1, base[i] = i;
  }
  bool dfs(int root) {
```

```
    used[root] = ++T;
    return go(root);
  }
  void match() {
    int ans = 0;
    for (int v = 0; v < n; v++)
      for (int x : g[v])
        if (pa[v] == -1 && pa[x] == -1) {
          pa[v] = x, pa[x] = v, ans++;
          break;
        }
    init_dfs();
    for (int i = 0; i < n; i++)
      if (pa[i] == -1 && dfs(i))
        ans++, init_dfs();
    cout << ans * 2 << "\n";
    for (int i = 0; i < n; i++)
      if (pa[i] > i)
        cout << i + 1 << " " << pa[i] + 1 << "\n";
  }
};
```

5.3.4 Min Weight Perfect Matching

```
struct Graph {
  static const int MAXN = 105;
  int n, e[MAXN][MAXN];
  int match[MAXN], d[MAXN], onstk[MAXN];
  vector<int> stk;
  void init(int _n) {
    n = _n;
    for( int i = 0 ; i < n ; i ++ )
      for( int j = 0 ; j < n ; j ++ )
        // change to appropriate infinity
        // if not complete graph
        e[i][j] = 0;
  }
  void add_edge(int u, int v, int w) {
    e[u][v] = e[v][u] = w;
  }
  bool SPFA(int u){
    if (onstk[u]) return true;
    stk.push_back(u);
    onstk[u] = 1;
    for ( int v = 0 ; v < n ; v++ ) {
      if (u != v && match[u] != v && !onstk[v] ) {
        int m = match[v];
        if ( d[m] > d[u] - e[v][m] + e[u][v] ) {
          d[m] = d[u] - e[v][m] + e[u][v];
          onstk[v] = 1;
          stk.push_back(v);
          if (SPFA(m)) return true;
          stk.pop_back();
          onstk[v] = 0;
        }
      }
    }
    onstk[u] = 0;
    stk.pop_back();
    return false;
  }
  int solve() {
    for ( int i = 0 ; i < n ; i += 2 ) {
      match[i] = i+1;
      match[i+1] = i;
    }
    while (true){
      int found = 0;
      for ( int i = 0 ; i < n ; i++ )
        onstk[ i ] = d[ i ] = 0;
      for ( int i = 0 ; i < n ; i++ ) {
        stk.clear();
        if ( !onstk[i] && SPFA(i) ) {
          found = 1;
          while ( stk.size() >= 2 ) {
            int u = stk.back(); stk.pop_back();
            int v = stk.back(); stk.pop_back();
            match[u] = v;
            match[v] = u;
          }
        }
      }
      if (!found) break;
    }
    int ret = 0;
    for ( int i = 0 ; i < n ; i++ )
      ret += e[i][match[i]];
    ret /= 2;
    return ret;
  }
} graph;
```

5.3.5 Stable Marriage

```
// normal stable marriage problem
/* input:
3
Albert Laura Nancy Marcy
Brad Marcy Nancy Laura
Chuck Laura Marcy Nancy
Laura Chuck Albert Brad
Marcy Albert Chuck Brad
Nancy Brad Albert Chuck
*/

#include<bits/stdc++.h>
using namespace std;
const int MAXN = 505;

int n;
int favor[MAXN][MAXN]; // favor[boy_id][rank] = girl_id;
int order[MAXN][MAXN]; // order[girl_id][boy_id] = rank;
int current[MAXN]; // current[boy_id] = rank;
// boy_id will pursue current[boy_id] girl.
int girl_current[MAXN]; // girl[girl_id] = boy_id;

void initialize() {
  for ( int i = 0 ; i < n ; i++ ) {
    current[i] = 0;
    girl_current[i] = n;
    order[i][n] = n;
  }
}

map<string, int> male, female;
string bname[MAXN], gname[MAXN];
int fit = 0;

void stable_marriage() {

  queue<int> que;
  for ( int i = 0 ; i < n ; i++ ) que.push(i);
  while ( !que.empty() ) {
    int boy_id = que.front();
    que.pop();

    int girl_id = favor[boy_id][current[boy_id]];
    current[boy_id] ++;

    if (order[girl_id][boy_id] <
        order[girl_id][girl_current[girl_id]]) {
      if ( girl_current[girl_id] < n )
        que.push(girl_current[girl_id]);
      girl_current[girl_id] = boy_id;
    } else {
      que.push(boy_id);
    }
  }

}

int main() {
  cin >> n;

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

  for ( int i = 0 ; i < n ; i++ ) {
```

```
    string p, t;
    cin >> p;
    for ( int j = 0 ; j < n ; j++ ) {
      cin >> t;
      order[female[p]][male[t]] = j;
    }
  }

  initialize();
  stable_marriage();

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

}
```

5.4 Centroid Decomposition

```
void get_center(int now) {
  v[now] = true; vtx.push_back(now);
  sz[now] = 1; mx[now] = 0;
  for (int u : G[now]) if (!v[u]) {
    get_center(u);
    mx[now] = max(mx[now], sz[u]);
    sz[now] += sz[u];
  }
}
void get_dis(int now, int d, int len) {
  dis[d][now] = cnt;
  v[now] = true;
  for (auto u : G[now]) if (!v[u.first]) {
    get_dis(u, d, len + u.second);
  }
}
void dfs(int now, int fa, int d) {
  get_center(now);
  int c = -1;
  for (int i : vtx) {
    if (max(mx[i], (int)vtx.size() - sz[i]) <=
    ↪   (int)vtx.size() / 2) c = i;
    v[i] = false;
  }
  get_dis(c, d, 0);
  for (int i : vtx) v[i] = false;
  v[c] = true; vtx.clear();
  dep[c] = d; p[c] = fa;
  for (auto u : G[c]) if (u.first != fa && !v[u.first]) {
    dfs(u.first, c, d + 1);
  }
}
```

5.5 Minimum Mean Cycle

```
// source: waynedisonitau123

// d[i][j] == 0 if {i,j} !in E
long long d[1003][1003],dp[1003][1003];

pair<long long,long long> MMWC(){
  memset(dp,0x3f,sizeof(dp));
  for(int i=1;i<=n;++i)dp[0][i]=0;
  for(int i=1;i<=n;++i){
    for(int j=1;j<=n;++j){
      for(int k=1;k<=n;++k){
        dp[i][k]=min(dp[i-1][j]+d[j][k],dp[i][k]);
      }
    }
  }
  long long au=1ll<<31,ad=1;
  for(int i=1;i<=n;++i){
    if(dp[n][i]==0x3f3f3f3f3f3f3f3f)continue;
    long long u=0,d=1;
    for(int j=n-1;j>=0;--j){
      if((dp[n][i]-dp[j][i])*d>u*(n-j)){
        u=dp[n][i]-dp[j][i];
        d=n-j;
      }
    }
    if(u*ad<au*d)au=u,ad=d;
  }
  long long g=__gcd(au,ad);
  return make_pair(au/g,ad/g);
}
```

5.6 Bellman-Ford

```
// source: KACTL

const ll inf = LLONG_MAX;
struct Ed { int a, b, w, s() { return a < b ? a : -a; }};
struct Node { ll dist = inf; int prev = -1; };

void bellmanFord(vector<Node>& nodes, vector<Ed>& eds, int
↪  s) {
  nodes[s].dist = 0;
  sort(all(eds), [](Ed a, Ed b) { return a.s() < b.s(); });

  int lim = sz(nodes) / 2 + 2; // /3+100 with shuffled
  ↪  vertices
  rep(i,0,lim) for (Ed ed : eds) {
    Node cur = nodes[ed.a], &dest = nodes[ed.b];
    if (abs(cur.dist) == inf) continue;
    ll d = cur.dist + ed.w;
    if (d < dest.dist) {
      dest.prev = ed.a;
      dest.dist = (i < lim-1 ? d : -inf);
    }
  }
  rep(i,0,lim) for (Ed e : eds) {
    if (nodes[e.a].dist == -inf)
      nodes[e.b].dist = -inf;
  }
}
```

5.7 Directed Minimum Spanning Tree

```
template <typename T> struct DMST {
  T g[maxn][maxn], fw[maxn];
  int n, fr[maxn];
  bool vis[maxn], inc[maxn];
  void clear() {
    for(int i = 0; i < maxn; ++i) {
      for(int j = 0; j < maxn; ++j) g[i][j] = inf;
      vis[i] = inc[i] = false;
    }
  }
  void addedge(int u, int v, T w) {
    g[u][v] = min(g[u][v], w);
  }
  T operator()(int root, int _n) {
    n = _n;
    if (dfs(root) != n) return -1;
    T ans = 0;
    while (true) {
      for (int i = 1; i <= n; ++i) fw[i] = inf, fr[i] = i;
      for (int i = 1; i <= n; ++i) if (!inc[i]) {
        for (int j = 1; j <= n; ++j) {
          if (!inc[j] && i != j && g[j][i] < fw[i]) {
            fw[i] = g[j][i];
            fr[i] = j;
          }
        }
      }
      int x = -1;
      for (int i = 1; i <= n; ++i) if (i != root && !inc[i])
      ↪  {
        int j = i, c = 0;
        while (j != root && fr[j] != i && c <= n) ++c, j =
        ↪  fr[j];
        if (j == root || c > n) continue;
        else { x = i; break; }
      }
      if (!~x) {
        for (int i = 1; i <= n; ++i) if (i != root &&
        ↪  !inc[i]) ans += fw[i];
        return ans;
      }
      int y = x;
      for (int i = 1; i <= n; ++i) vis[i] = false;
      do { ans += fw[y]; y = fr[y]; vis[y] = inc[y] = true;
      ↪  } while (y != x);
      inc[x] = false;
      for (int k = 1; k <= n; ++k) if (vis[k]) {
        for (int j = 1; j <= n; ++j) if (!vis[j]) {
```

```
        if (g[x][j] > g[k][j]) g[x][j] = g[k][j];
        if (g[j][k] < inf && g[j][k] - fw[k] < g[j][x])
        ↪  g[j][x] = g[j][k] - fw[k];
      }
    }
  }
  return ans;
}
int dfs(int now) {
  int r = 1;
  vis[now] = true;
  for (int i = 1; i <= n; ++i) if (g[now][i] < inf &&
  ↪  !vis[i]) r += dfs(i);
  return r;
}
};
```

5.8 Maximum Clique

```
// source: KACTL

typedef vector<bitset<200>> vb;
struct Maxclique {
  double limit=0.025, pk=0;
  struct Vertex { int i, d=0; };
  typedef vector<Vertex> vv;
  vb e;
  vv V;
  vector<vi> C;
  vi qmax, q, S, old;
  void init(vv& r) {
    for (auto& v : r) v.d = 0;
    for (auto& v : r) for (auto j : r) v.d += e[v.i][j.i];
    sort(all(r), [](auto a, auto b) { return a.d > b.d; });
    int mxD = r[0].d;
    rep(i,0,sz(r)) r[i].d = min(i, mxD) + 1;
  }
  void expand(vv& R, int lev = 1) {
    S[lev] += S[lev - 1] - old[lev];
    old[lev] = S[lev - 1];
    while (sz(R)) {
      if (sz(q) + R.back().d <= sz(qmax)) return;
      q.push_back(R.back().i);
      vv T;
      for(auto v:R) if (e[R.back().i][v.i])
      ↪  T.push_back({v.i});
      if (sz(T)) {
        if (S[lev]++ / ++pk < limit) init(T);
        int j = 0, mxk = 1, mnk = max(sz(qmax) - sz(q) + 1,
        ↪  1);
        C[1].clear(), C[2].clear();
        for (auto v : T) {
          int k = 1;
          auto f = [&](int i) { return e[v.i][i]; };
          while (any_of(all(C[k]), f)) k++;
          if (k > mxk) mxk = k, C[mxk + 1].clear();
          if (k < mnk) T[j++].i = v.i;
          C[k].push_back(v.i);
        }
        if (j > 0) T[j - 1].d = 0;
        rep(k,mnk,mxk + 1) for (int i : C[k])
          T[j].i = i, T[j++].d = k;
        expand(T, lev + 1);
      } else if (sz(q) > sz(qmax)) qmax = q;
      q.pop_back(), R.pop_back();
    }
  }
  vi maxClique() { init(V), expand(V); return qmax; }
  Maxclique(vb conn) : e(conn), C(sz(e)+1), S(sz(C)), old(S)
  ↪  {
    rep(i,0,sz(e)) V.push_back({i});
  }
};
```

5.9 Tarjan

5.9.1 Strongly Connected Components

```
struct Tarjan {
  static const int MAXN = 1000006;
  // 0-based
  int n, dfn[MAXN], low[MAXN], scc[MAXN], scn, count;
  vector<int> G[MAXN];
  stack<int> stk;
  bool ins[MAXN];

  void tarjan(int u) {
    dfn[u] = low[u] = ++count;
    stk.push(u);
    ins[u] = true;

    for(auto v : G[u]) {
      if(!dfn[v]) {
        tarjan(v);
        low[u] = min(low[u], low[v]);
      } else if(ins[v]) {
        low[u] = min(low[u], dfn[v]);
      }
    }

    if(dfn[u] == low[u]){
      int v;
      do {
        v = stk.top();
        stk.pop();
        scc[v] = scn;
        ins[v] = false;
      } while(v != u);
      scn++;
    }
  }

  void getSCC() {
    memset(dfn,0,sizeof(dfn));
    memset(low,0,sizeof(low));
    memset(ins,0,sizeof(ins));
    memset(scc,0,sizeof(scc));
    count = scn = 0;
    for (int i = 0; i < n; i++) {
      if(!dfn[i]) tarjan(i);
    }
  }
} SCC;
```

5.9.2 Articulation Point

```
void dfs(int x, int p) {
  tin[x] = low[x] = ++t;
  int ch = 0;
  for (auto u : g[x]) if (u.first != p) {
    if (!ins[u.second]) st.push(u.second), ins[u.second] =
    ↪  true;
    if (tin[u.first]) {
      low[x] = min(low[x], tin[u.first]);
      continue;
    }
    ++ch;
    dfs(u.first, x);
    low[x] = min(low[x], low[u.first]);
    if (low[u.first] >= tin[x]) {
      cut[x] = true;
      ++sz;
      while (true) {
        int e = st.top(); st.pop();
        bcc[e] = sz;
        if (e == u.second) break;
      }
    }
  }
  if (ch == 1 && p == -1) cut[x] = false;
}
```

5.9.3 Bridge

```
// if there are multi-edges, then they are not bridges
void dfs(int x, int p) {
  tin[x] = low[x] = ++t;
  st.push(x);
  for (auto u : g[x]) if (u.first != p) {
    if (tin[u.first]) {
      low[x] = min(low[x], tin[u.first]);
      continue;
    }
    dfs(u.first, x);
    low[x] = min(low[x], low[u.first]);
    if (low[u.first] == tin[u.first]) br[u.second] = true;
  }
```

```
  if (tin[x] == low[x]) {
    ++sz;
    while (st.size()) {
      int u = st.top(); st.pop();
      bcc[u] = sz;
      if (u == x) break;
    }
  }
}
```

5.10 2-SAT

```
const int MAXN = 2020;

struct TwoSAT{
  static const int MAXv = 2*MAXN;
  vector<int> GO[MAXv],BK[MAXv],stk;
  bool vis[MAXv];
  int SC[MAXv];

  void imply(int u,int v){ // u imply v
    GO[u].push_back(v);
    BK[v].push_back(u);
  }
  int dfs(int u,vector<int>*G,int sc){
    vis[u]=1, SC[u]=sc;
    for (int v:G[u])if (!vis[v])
      dfs(v,G,sc);
    if (G==GO)stk.push_back(u);
  }
  int scc(int n=MAXv){
    memset(vis,0,sizeof(vis));
    for (int i=0; i<n; i++)if (!vis[i])
      dfs(i,GO,-1);
    memset(vis,0,sizeof(vis));
    int sc=0;
    while (!stk.empty()){
      if (!vis[stk.back()])
        dfs(stk.back(),BK,sc++);
      stk.pop_back();
    }
  }
}SAT;

int main(){
  SAT.scc(2*n);
  bool ok=1;
  for (int i=0; i<n; i++){
    if (SAT.SC[2*i]==SAT.SC[2*i+1])ok=0;
  }
  if (ok){
    for (int i=0; i<n; i++){
      if (SAT.SC[2*i]>SAT.SC[2*i+1]){
        cout << i << endl;
      }
    }
  }
  else puts("NO");
}
```

5.11 Dominator Tree

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

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

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

  void init(int _n, int _s) {
    n = _n;
    s = _s;
    REP1(i,1,n) {
      g[i].clear();
      pred[i].clear();
      idom[i] = 0;
    }
  }
  void add_edge(int u, int v) {
    g[u].push_back(v);
    pred[v].push_back(u);
  }
  void DFS(int u) {
    ts++;
    dfn[u] = ts;
    nfd[ts] = u;
    for(int v:g[u]) if(dfn[v] == 0) {
      par[v] = u;
      DFS(v);
    }
  }
  void build() {
    ts = 0;
    REP1(i,1,n) {
      dfn[i] = nfd[i] = 0;
      cov[i].clear();
      mom[i] = mn[i] = sdom[i] = i;
    }
    DFS(s);
    for (int i=ts; i>=2; i--) {
      int u = nfd[i];
      if(u == 0) continue ;
      for(int v:pred[u]) if(dfn[v]) {
        eval(v);
        if(cmp(sdom[mn[v]],sdom[u])) sdom[u] = sdom[mn[v]];
      }
      cov[sdom[u]].push_back(u);
      mom[u] = par[u];
      for(int w:cov[par[u]]) {
        eval(w);
        if(cmp(sdom[mn[w]],par[u])) idom[w] = mn[w];
        else idom[w] = par[u];
      }
      cov[par[u]].clear();
    }
    REP1(i,2,ts) {
      int u = nfd[i];
      if(u == 0) continue ;
      if(idom[u] != sdom[u]) idom[u] = idom[idom[u]];
    }
  }
}dom;
```

5.12 Biconnected Components

```
// source: KACTL

vi num, st;
vector<vector<pii>> ed;
int Time;
template<class F>
int dfs(int at, int par, F& f) {
  int me = num[at] = ++Time, e, y, top = me;
  for (auto pa : ed[at]) if (pa.second != par) {
    tie(y, e) = pa;
    if (num[y]) {
      top = min(top, num[y]);
      if (num[y] < me)
        st.push_back(e);
    } else {
      int si = sz(st);
      int up = dfs(y, e, f);
      top = min(top, up);
      if (up == me) {
        st.push_back(e);
        f(vi(st.begin() + si, st.end()));
        st.resize(si);
      }
      else if (up < me) st.push_back(e);
```

```
      else { /* e is a bridge */ }
    }
  }
  return top;
}

template<class F>
void bicomps(F f) {
  num.assign(sz(ed), 0);
  rep(i,0,sz(ed)) if (!num[i]) dfs(i, -1, f);
}
```

5.13 Edge BCC

```
struct BccEdge {
  static const int MXN = 100005;
  struct Edge { int v,eid; };
  int n,m,step,par[MXN],dfn[MXN],low[MXN];
  vector<Edge> E[MXN];
  DisjointSet djs;
  void init(int _n) {
    n = _n; m = 0;
    for (int i=0; i<n; i++) E[i].clear();
    djs.init(n);
  }
  void add_edge(int u, int v) {
    E[u].PB({v, m});
    E[v].PB({u, m});
    m++;
  }
  void DFS(int u, int f, int f_eid) {
    par[u] = f;
    dfn[u] = low[u] = step++;
    for (auto it:E[u]) {
      if (it.eid == f_eid) continue;
      int v = it.v;
      if (dfn[v] == -1) {
        DFS(v, u, it.eid);
        low[u] = min(low[u], low[v]);
      } else {
        low[u] = min(low[u], dfn[v]);
      }
    }
  }
  void solve() {
    step = 0;
    memset(dfn, -1, sizeof(int)*n);
    for (int i=0; i<n; i++) {
      if (dfn[i] == -1) DFS(i, i, -1);
    }
    djs.init(n);
    for (int i=0; i<n; i++) {
      if (low[i] < dfn[i]) djs.uni(i, par[i]);
    }
  }
}graph;
```

5.14 Manhattan MST

```
// source: KACTL

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

5.15 Notes

```
Maximum Independent Set
General: [NPC] maximum clique of complement of G
Tree: [P] Greedy
Bipartite Graph: [P] Maximum Cardinality Bipartite Matching
-------------------------------------------------------
Minimum Dominating Set
General: [NPC]
Tree: [P] DP
Bipartite Graph: [NPC]
-------------------------------------------------------
Minimum Vertex Cover
General: [NPC] (?)maximum clique of complement of G
Tree: [P] Greedy, from leaf to root
Bipartite Graph: [P] Maximum Cardinality Bipartite Matching
-------------------------------------------------------
Minimum Edge Cover
General: [P] V - Maximum Matching
Bipartite Graph: [P] Greedy, strategy: cover small degree
↪  node first.
(Min/Max)Weighted: [P]: Minimum/Minimum Weight Matching
```

6 Geometry

6.1 Basic 2D

```
using T = long long;
using pt = pair<T, T>;
#define x first
#define y second

pt operator-(pt a) { return { -a.x, -a.y }; }
pt operator+(pt a, pt b) {return { a.x + b.x, a.y + b.y};}
pt operator-(pt a, pt b) {return { a.x - b.x, a.y - b.y};}
pt operator*(pt a, T t) { return { a.x * t, a.y * t }; }
T abs2(pt a) { return a.x * a.x + a.y * a.y; }
T len(pt a) { return sqrt(abs2(a)); }
T dot(pt a, pt b) { return a.x * b.x + a.y * b.y; }
T cross(pt a, pt b) { return a.x * b.y - b.x * a.y; }
T cross(pt a, pt b, pt o) { return cross(a - o, b - o); }

// if segment AB and CD intersects
// for a nondegenrate version, change > to >=
bool intersects(pt a, pt b, pt c, pt d) {
  if(cross(b, c, a) * cross(b, d, a) > 0) return false;
  if(cross(d, a, c) * cross(d, b, c) > 0) return false;
  return true;
}
// the intersect pt of lines AB and CD
pt intersect(pt a, pt b, pt c, pt d) {
  auto x = cross(b, c, a), y = cross(b, d, a);
  if(x == y) {
  // if(abs(x, y) < 1e-8) {
    // is parallel
  } else {
    return d * (x/(x-y)) - c * (y/(x-y));
  }
}
```

6.2 Angular Sort

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

6.3 Convex Hull

```
// returns a convex hull in counterclockwise order
// for a non-strict one, change cross >= to >
```

```
vector<pt> convex_hull(vector<pt> p) {
  sort(ALL(p));
  if (p[0] == p.back()) return { p[0] };
  int n = p.size(), t = 0;
  vector<pt> h(n + 1);
  for (int _ = 2, s = 0; _--; s = --t, reverse(ALL(p)))
    for (pt i : p) {
      while (t > s+1 && cross(i, h[t-1], h[t-2]) >= 0) t--;
      h[t++] = i;
    }
  return h.resize(t), h;
}
```

6.4 Convex Polygon Point Inclusion

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

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

6.5 Convex Polygon Minkowski Sum

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

6.6 Closest Pair

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

6.7 Minimum Enclosing Circle

```
// source: waynedisonitau123

pt center(const pt &a, const pt &b, const pt &c) {
  pt p0 = b - a, p1 = c - a;
  double c1 = abs2(p0) * 0.5, c2 = abs2(p1) * 0.5;
  double d = cross(p0, p1);
  double x = a.x + (c1 * p1.y - c2 * p0.y) / d;
  double y = a.y + (c2 * p0.x - c1 * p1.x) / d;
  return pt(x, y);
}

pair<double, double> solve(vector<pt> &p) {
  shuffle(p.begin(), p.end(), RNG);
  double r = 0.0;
  pt cent;
  for (int i = 0; i < p.size(); ++i) {
    if (abs2(cent - p[i]) <= r) continue;
    cent = p[i];
    r = 0.0;
    for (int j = 0; j < i; ++j) {
      if (abs2(cent - p[j]) <= r) continue;
      cent = (p[i] + p[j]) / 2;
      r = abs2(p[j] - cent);
      for (int k = 0; k < j; ++k) {
        if (abs2(cent - p[k]) <= r) continue;
        cent = center(p[i], p[j], p[k]);
        r = abs2(p[k] - cent);
      }
    }
  }
  return {cent, sqrt(r)};
}
```

6.8 Half Plane Intersection

```
// source: waynedisonitau123

bool jizz(L l1,L l2,L l3){
  P p=Intersect(l2,l3);
  return ((l1.pb-l1.pa)^(p-l1.pa))<-eps;
}

bool cmp(const L &a,const L &b){
  return
  ↪   same(a.o,b.o)?(((b.pb-b.pa)^(a.pb-b.pa))>eps):a.o<b.o;
}
```

```
// availble area for L l is (l.pb-l.pa)^(p-l.pa)>0
vector<P> HPI(vector<L> &ls){
  sort(ls.begin(),ls.end(),cmp);
  vector<L> pls(1,ls[0]);
  for(int i=0;i<(int)ls.size();++i)if(!same(ls[i].o,pls.back
  ↪  ().o))pls.push_back(ls[i]);
  deque<int> dq; dq.push_back(0); dq.push_back(1);
#define meow(a,b,c) while(dq.size()>1u &&
↪  jizz(pls[a],pls[b],pls[c]))
  for(int i=2;i<(int)pls.size();++i){
    meow(i,dq.back(),dq[dq.size()-2])dq.pop_back();
    meow(i,dq[0],dq[1])dq.pop_front();
    dq.push_back(i);
  }
  meow(dq.front(),dq.back(),dq[dq.size()-2])dq.pop_back();
  meow(dq.back(),dq[0],dq[1])dq.pop_front();
  if(dq.size()<3u)return vector<P>(); // no solution or
  ↪  solution is not a convex
  vector<P> rt;
  for(int i=0;i<(int)dq.size();++i)rt.push_back(Intersect(pl
  ↪  s[dq[i]],pls[dq[(i+1)%dq.size()]]));
  return rt;
}
```

6.9 Delaunay Triangulation

```
// source: KACTL

typedef Point<ll> P;
typedef struct Quad* Q;
typedef __int128_t lll; // (can be ll if coords are < 2e4)
P arb(LLONG_MAX,LLONG_MAX); // not equal to any other point

struct Quad {
  bool mark; Q o, rot; P p;
  P F() { return r()->p; }
  Q r() { return rot->rot; }
  Q prev() { return rot->o->rot; }
  Q next() { return r()->prev(); }
};

bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
  lll p2 = p.dist2(), A = a.dist2()-p2,
      B = b.dist2()-p2, C = c.dist2()-p2;
  return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B >
  ↪  0;
}
Q makeEdge(P orig, P dest) {
  Q q[] = {new Quad{0,0,0,orig}, new Quad{0,0,0,arb},
           new Quad{0,0,0,dest}, new Quad{0,0,0,arb}};
  rep(i,0,4)
    q[i]->o = q[-i & 3], q[i]->rot = q[(i+1) & 3];
  return *q;
}
void splice(Q a, Q b) {
  swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
}
Q connect(Q a, Q b) {
  Q q = makeEdge(a->F(), b->p);
  splice(q, a->next());
  splice(q->r(), b);
  return q;
}

pair<Q,Q> rec(const vector<P>& s) {
  if (sz(s) <= 3) {
    Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.back());
    if (sz(s) == 2) return { a, a->r() };
    splice(a->r(), b);
    auto side = s[0].cross(s[1], s[2]);
    Q c = side ? connect(b, a) : 0;
    return {side < 0 ? c->r() : a, side < 0 ? c : b->r() };
  }

#define H(e) e->F(), e->p
#define valid(e) (e->F().cross(H(base)) > 0)
  Q A, B, ra, rb;
  int half = sz(s) / 2;
  tie(ra, A) = rec({all(s) - half});
  tie(B, rb) = rec({sz(s) - half + all(s)});
  while ((B->p.cross(H(A)) < 0 && (A = A->next())) ||
         (A->p.cross(H(B)) > 0 && (B = B->r()->o)));
  Q base = connect(B->r(), A);
  if (A->p == ra->p) ra = base->r();
  if (B->p == rb->p) rb = base;

#define DEL(e, init, dir) Q e = init->dir; if (valid(e)) \
    while (circ(e->dir->F(), H(base), e->F())) { \
      Q t = e->dir; \
      splice(e, e->prev()); \
      splice(e->r(), e->r()->prev()); \
      e = t; \
    }
  for (;;) {
    DEL(LC, base->r(), o);  DEL(RC, base, prev());
    if (!valid(LC) && !valid(RC)) break;
    if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
      base = connect(RC, base->r());
    else
      base = connect(base->r(), LC->r());
  }
  return { ra, rb };
}

// returns [A_0, B_0, C_0, A_1, B_1, ...]
// where A_i, B_i, C_i are counter-clockwise triangles
vector<P> triangulate(vector<P> pts) {
  sort(all(pts));  assert(unique(all(pts)) == pts.end());
  if (sz(pts) < 2) return {};
  Q e = rec(pts).first;
  vector<Q> q = {e};
  int qi = 0;
  while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
#define ADD { Q c = e; do { c->mark = 1;
↪  pts.push_back(c->p); \
  q.push_back(c->r()); c = c->next(); } while (c != e); }
  ADD; pts.clear();
  while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD;
  return pts;
}
```

6.10 Spherical Coordinates

```
struct car_p { double x, y, z; };
struct sph_p { double r, theta, phi; };

sph_p conv(car_p p) {
  double r = sqrt(p.x*p.x + p.y*p.y + p.z*p.z);
  double theta = asin(p.y / r);
  double phi = atan2(p.y, p.x);
  return { r, theta, phi };
}
car_p conv(sph_p p) {
  double x = p.r * cos(p.theta) * sin(p.phi);
  double y = p.r * cos(p.theta) * cos(p.phi);
  double z = p.r * sin(p.theta);
  return { x, y, z };
}
```

6.11 Quaternion

```
struct Q {
  using T = double;
  T x, y, z, r;
  Q(T r = 0) : x(0), y(0), z(0), r(r) {}
  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) {
    return (a - b).abs2() <= 1e-8; }
  friend bool operator!=(const Q& a, const Q& b) {
    return !(a == b); }
  Q operator-() { return Q(-x, -y, -z, -r); }
  Q operator+(const Q& b) const {
    return Q(x + b.x, y + b.y, z + b.z, r + b.r); }
  Q operator-(const Q& b) const {
    return Q(x - b.x, y - b.y, z - b.z, r - b.r); }
  Q operator*(const T& t) const {
    return Q(x * t, y * t, z * t, r * t); }
  Q operator*(const Q& b) const {
    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,
      r * b.z + x * b.y - y * b.x + z * b.r,
      r * b.r - x * b.x - y * b.y - z * b.z
    );
  }
  Q operator/(const Q& b) const {
```

```
    return *this * b.inv(); }
  T abs2() const {
    return r * r + x * x + y * y + z * z; }
  T len() const { return sqrt(abs2()); }
  Q conj() const { return Q(-x, -y, -z, r); }
  Q unit() const { return *this * (1.0 / len()); }
  Q inv() const { return conj() * (1.0 / abs2()); }
  friend T dot(Q a, Q b) {
    return a.x * b.x + a.y * b.y + a.z * b.z; }
  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,
      a.x * b.y - a.y * b.x
    );
  }
  friend Q rotation_around(Q axis, T angle) {
    return axis.unit() * sin(angle / 2) + cos(angle / 2);
  }
  Q rotated_around(Q axis, T angle) {
    Q u = rotation_around(axis, angle);
    return u * *this / u;
  }
  friend Q rotation_between(Q a, Q b) {
    a = a.unit(), b = b.unit();
    if(a == -b) {
      // degenerate case
      Q ortho;
      if(abs(a.y) > 1e-8) ortho = cross(a, Q(1, 0, 0));
      else ortho = cross(a, Q(0, 1, 0));
      return rotation_around(ortho, pi);
    } else {
      return (a * (a + b)).conj();
    }
  }
};
```

6.12 3D Convex Hull

```
// source: KACTL

typedef Point3D<double> P3;

struct PR {
  void ins(int x) { (a == -1 ? a : b) = x; }
  void rem(int x) { (a == x ? a : b) = -1; }
  int cnt() { return (a != -1) + (b != -1); }
  int a, b;
};

struct F { P3 q; int a, b, c; };

vector<F> hull3d(const vector<P3>& A) {
  assert(sz(A) >= 4);
  vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));
#define E(x,y) E[f.x][f.y]
  vector<F> FS;
  auto mf = [&](int i, int j, int k, int l) {
    P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
    if (q.dot(A[l]) > q.dot(A[i]))
      q = q * -1;
    F f{q, i, j, k};
    E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
    FS.push_back(f);
  };
  rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
    mf(i, j, k, 6 - i - j - k);

  rep(i,4,sz(A)) {
    rep(j,0,sz(FS)) {
      F f = FS[j];
      if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
        E(a,b).rem(f.c);
        E(a,c).rem(f.b);
        E(b,c).rem(f.a);
        swap(FS[j--], FS.back());
        FS.pop_back();
      }
    }
    int nw = sz(FS);
    rep(j,0,nw) {
      F f = FS[j];
#define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i,
↪  f.c);
      C(a, b, c); C(a, c, b); C(b, c, a);
    }
  }
  for (F& it : FS) if ((A[it.b] - A[it.a]).cross(
    A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);
  return FS;
};
```

7 Strings

7.1 Z-value

```
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;
  }
}
```

7.2 Manacher

```
int z[n];
void zval_pal(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;
  }
}
```

7.3 Minimum Rotation

```
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;
}
```

7.4 Aho-Corasick

```
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';
  }
}
```

7.5 Suffix Array

```
// source: KACTL

struct SuffixArray {
  vi sa, lcp;
  SuffixArray(string& s, int lim=256) { // or
  ↪ basic_string<int>
    int n = sz(s) + 1, k = 0, a, b;
    vi x(all(s)+1), y(n), ws(max(n, lim)), rank(n);
    sa = lcp = y, iota(all(sa), 0);
    for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim =
    ↪ p) {
      p = j, iota(all(y), n - j);
      rep(i,0,n) if (sa[i] >= j) y[p++] = sa[i] - j;
      fill(all(ws), 0);
      rep(i,0,n) ws[x[i]]++;
      rep(i,1,lim) ws[i] += ws[i - 1];
      for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
      swap(x, y), p = 1, x[sa[0]] = 0;
      rep(i,1,n) a = sa[i - 1], b = sa[i], x[b] =
        (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
    }
    rep(i,1,n) rank[sa[i]] = i;
    for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
      for (k && k--, j = sa[rank[i] - 1];
          s[i + k] == s[j + k]; k++);
  }
};
```

7.6 Suffix Tree

```
struct SAM {
  static const int maxc = 26; // char range
  static const int maxn = 10010; // string len
  struct Node {
    Node *green,*edge[maxc];
    int max_len, in, times;
  }*root, *last, reg[maxn * 2];
  int top;
  Node* get_node(int _max) {
    Node *re = &reg[top++];
    re -> in  = 0, re -> times = 1;
    re -> max_len = _max, re -> green = 0;
    for(int i = 0; i < maxc; i++)
      re -> edge[i] = 0;
    return re;
  }
  void insert(const char c) { // c in range [0, maxc)
    Node *p = last;
    last = get_node(p -> max_len + 1);
    while(p && !p -> edge[c])
      p -> edge[c] = last, p = p -> green;
    if(!p) last -> green = root;
    else {
      Node *pot_green = p -> edge[c];
      if((pot_green -> max_len) == (p -> max_len + 1))
        last -> green = pot_green;
      else {
        Node *wish = get_node(p -> max_len + 1);
        wish -> times = 0;
        while(p && p -> edge[c] == pot_green)
          p -> edge[c] = wish, p = p -> green;
        for(int i = 0; i < maxc; i++)
          wish -> edge[i] = pot_green -> edge[i];
        wish -> green = pot_green -> green;
        pot_green -> green = wish;
        last -> green = wish;
      }
    }
  }
  Node *q[maxn * 2];
  int ql, qr;
  void get_times(Node *p) {
    ql = 0, qr = -1, reg[0].in = 1;
    for(int i = 1; i < top; i++)
      reg[i].green -> in++;
    for(int i = 0; i < top; i++)
      if(!reg[i].in)
        q[++qr] = &reg[i];
    while(ql <= qr) {
      q[ql] -> green -> times += q[ql] -> times;
      if(!(--q[ql] -> green -> in))
        q[++qr] = q[ql] -> green;
      ql++;
    }
  }
  void build(const string &s) {
    top = 0;
    root = last = get_node(0);
    for(char c : s)
      insert(c - 'a'); // change char id
    get_times(root);
  }
  // call build before solve
  int solve(const string &s) {
    Node *p = root;
    for(char c : s)
      if(!(p = p -> edge[c - 'a'])) // change char id
        return 0;
    return p -> times;
  }
};
```