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

```
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);
  }
};
```

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)

```
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

```
// 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

```
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;
}
```

1.3.2. SplitMix64

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

1.3.3. <random>

```
#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
// random uint_fast64_t: RNG();
// uniform random of type T (int, double, ...) in [l, r]:
// uniform_int_distribution<T> dist(l, r); dist(RNG);
```

1.4. Algorithms

1.4.1. Bit Hacks

```
// next permutation of x as a bit sequence
ull next_bits_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 */ }
}
```

1.4.2. Aliens Trick

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

1.4.3. Hilbert Curve

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

1.4.4. Infinite Grid Knight Distance

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

1.4.5. Poker Hand

```
using namespace std;

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

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. Segment Tree (ZKW)

```cpp
struct segtree {
  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);
  }
};
```

2.3. Wavelet Matrix

```cpp
#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;
  }
};
```

3. Graph

3.1. Strongly Connected Components

```cpp
struct TarjanScc {
  int n, step;
  vector<int> time, low, instk, stk;
  vector<vector<int>> e, scc;
  TarjanScc(int n_)
      : n(n_), step(0), time(n), low(n), instk(n), e(n) {}
  void add_edge(int u, int v) { e[u].push_back(v); }
  void dfs(int x) {
    time[x] = low[x] = ++step;
    stk.push_back(x);
    instk[x] = 1;
    for (int y : e[x])
      if (!time[y]) {
        dfs(y);
        low[x] = min(low[x], low[y]);
      } else if (instk[y]) {
        low[x] = min(low[x], time[y]);
      }
    if (time[x] == low[x]) {
      scc.emplace_back();
      for (int y = -1; y != x;) {
        y = stk.back();
        stk.pop_back();
        instk[y] = 0;
        scc.back().push_back(y);
      }
    }
  }
  void solve() {
    for (int i = 0; i < n; i++)
      if (!time[i]) dfs(i);
    reverse(scc.begin(), scc.end());
    // scc in topological order
  }
};
```

3.2. Triconnected Components

```cpp
// requires a union-find data structure
struct ThreeEdgeCC {
  int V, ind;
  vector<int> id, pre, post, low, deg, path;
  vector<vector<int>> components;
  UnionFind uf;
  template <class Graph>
  void dfs(const Graph &G, int v, int prev) {
    pre[v] = ++ind;
    for (int w : G[v])
      if (w != v) {
        if (w == prev) {
          prev = -1;
          continue;
        }
        if (pre[w] != -1) {
          if (pre[w] < pre[v]) {
            deg[v]++;
            low[v] = min(low[v], pre[w]);
          } else {
            deg[v]--;
            int &u = path[v];
            for (; u != -1 && pre[u] <= pre[w] &&
                   pre[w] <= post[u];) {
              uf.join(v, u);
              deg[v] += deg[u];
              u = path[u];
            }
          }
          continue;
        }
        dfs(G, w, v);
        if (path[w] == -1 && deg[w] <= 1) {
          deg[v] += deg[w];
          low[v] = min(low[v], low[w]);
          continue;
        }
        if (deg[w] == 0) w = path[w];
        if (low[v] > low[w]) {
          low[v] = min(low[v], low[w]);
          swap(w, path[v]);
        }
        for (; w != -1; w = path[w]) {
          uf.join(v, w);
          deg[v] += deg[w];
        }
      }
    post[v] = ind;
  }
  template <class Graph>
  ThreeEdgeCC(const Graph &G)
      : V(G.size()), ind(-1), id(V, -1), pre(V, -1),
        post(V), low(V, INT_MAX), deg(V, 0), path(V, -1),
        uf(V) {
    for (int v = 0; v < V; v++)
      if (pre[v] == -1) dfs(G, v, -1);
    components.reserve(uf.cnt);
    for (int v = 0; v < V; v++)
      if (uf.find(v) == v) {
        id[v] = components.size();
        components.emplace_back(1, v);
        components.back().reserve(uf.getSize(v));
      }
    for (int v = 0; v < V; v++)
      if (id[v] == -1)
        components[id[v] = id[uf.find(v)]].push_back(v);
  }
};
```

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. 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};
}
```

4.1.4. 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;
}
```

4.1.5. Rational Number Binary Search

```cpp
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;
}
```

4.1.6. 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;
}
```

5. Numeric

5.1. Fast Fourier Transform

```
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

```
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);
}
```

5.2. Fast Walsh-Hadamard Transform

Requires: Mod Struct

```
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
}
```

6. Geometry

6.1. Point

```
template <typename T> struct P {
  T x, y;
  P(T x = 0, T y = 0) : x(x), y(y) {}
  bool operator<(const P &p) const {
    return tie(x, y) < tie(p.x, p.y);
  }
  bool operator==(const P &p) const {
    return tie(x, y) == tie(p.x, p.y);
  }
  P operator-() const { return {-x, -y}; }
  P operator+(P p) const { return {x + p.x, y + p.y}; }
  P operator-(P p) const { return {x - p.x, y - p.y}; }
  P operator*(T d) const { return {x * d, y * d}; }
  P operator/(T d) const { return {x / d, y / d}; }
  T dist2() const { return x * x + y * y; }
  double len() const { return sqrt(dist2()); }
  P unit() const { return *this / len(); }
  friend T dot(P a, P b) { return a.x * b.x + a.y * b.y; }
  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) {
    return cross(a - o, b - o);
  }
};
using pt = P<ll>;
```

6.1.1. Quarternion

```cpp
constexpr double PI = 3.141592653589793;
constexpr double EPS = 1e-7;
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() <= EPS;
  }
  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 = abs(a.y) > EPS ? cross(a, Q(1, 0, 0))
                               : cross(a, Q(0, 1, 0));
      return rotation_around(ortho, PI);
    }
    return (a * (a + b)).conj();
  }
};
```

7. Strings

7.1. 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';
  }
}
```

7.2. Suffix Array

```cpp
// sa[i]: starting index of suffix at rank i
//        0-indexed, sa[0] = n (empty string)
// lcp[i]: lcp of sa[i] and sa[i - 1], lcp[0] = 0
struct SuffixArray {
  vector<int> sa, lcp;
  SuffixArray(string &s,
              int lim = 256) { // or basic_string<int>
    int n = sz(s) + 1, k = 0, a, b;
    vector<int> 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);
      for (int i = 0; i < n; i++)
        if (sa[i] >= j) y[p++] = sa[i] - j;
      fill(all(ws), 0);
      for (int i = 0; i < n; i++) ws[x[i]]++;
      for (int i = 1; i < lim; i++) 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;
      for (int i = 1; i < n; i++)
        a = sa[i - 1], b = sa[i],

        x[b] = (y[a] == y[b] && y[a + j] == y[b + j])
               ? p - 1 : p++;

    }
    for (int i = 1; i < n; i++) 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++)
        ;
  }
};
```

7.3. Suffix Tree

```cpp
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;
  }
};
```

7.4. Cocke-Younger-Kasami Algorithm

```cpp
struct rule {
  // s -> xy
  // if y == -1, then s -> x (unit rule)
  int s, x, y, cost;
};
int state;
// state (id) for each letter (variable)
// lowercase letters are terminal symbols
map<char, int> rules;
vector<rule> cnf;
void init() {
  state = 0;
  rules.clear();
  cnf.clear();
}
// convert a cfg rule to cnf (but with unit rules) and add
// it
void add_to_cnf(char s, const string &p, int cost) {
  if (!rules.count(s)) rules[s] = state++;
  for (char c : p)
    if (!rules.count(c)) rules[c] = state++;
  if (p.size() == 1) {
    cnf.push_back({rules[s], rules[p[0]], -1, cost});
  } else {
    // length >= 3 -> split
    int left = rules[s];
    int sz = p.size();
    for (int i = 0; i < sz - 2; i++) {
      cnf.push_back({left, rules[p[i]], state, 0});
      left = state++;
    }
    cnf.push_back(
    {left, rules[p[sz - 2]], rules[p[sz - 1]], cost});
  }
}

constexpr int MAXN = 55;
vector<long long> dp[MAXN][MAXN];
// unit rules with negative costs can cause negative cycles
vector<bool> neg_INF[MAXN][MAXN];

void relax(int l, int r, rule c, long long cost,
           bool neg_c = 0) {
  if (!neg_INF[l][r][c.s] &&
      (neg_INF[l][r][c.x] || cost < dp[l][r][c.s])) {
    if (neg_c || neg_INF[l][r][c.x]) {
      dp[l][r][c.s] = 0;
      neg_INF[l][r][c.s] = true;
    } else {
      dp[l][r][c.s] = cost;
    }
  }
}
void bellman(int l, int r, int n) {
  for (int k = 1; k <= state; k++)
    for (rule c : cnf)
      if (c.y == -1)
        relax(l, r, c, dp[l][r][c.x] + c.cost, k == n);
}
void cyk(const string &s) {
  vector<int> tok;
  for (char c : s) tok.push_back(rules[c]);
  for (int i = 0; i < tok.size(); i++) {
    for (int j = 0; j < tok.size(); j++) {
      dp[i][j] = vector<long long>(state + 1, INT_MAX);
      neg_INF[i][j] = vector<bool>(state + 1, false);
    }
    dp[i][i][tok[i]] = 0;
    bellman(i, i, tok.size());
  }
  for (int r = 1; r < tok.size(); r++) {
    for (int l = r - 1; l >= 0; l--) {
      for (int k = l; k < r; k++)
        for (rule c : cnf)
          if (c.y != -1)
            relax(l, r, c,
                  dp[l][k][c.x] + dp[k + 1][r][c.y] +
                  c.cost);
      bellman(l, r, tok.size());
    }
  }
}

// usage example
int main() {
  init();
  add_to_cnf('S', "aSc", 1);
  add_to_cnf('S', "BBB", 1);
  add_to_cnf('S', "SB", 1);
  add_to_cnf('B', "b", 1);
  cyk("abbbbc");
  // dp[0][s.size() - 1][rules[start]] = min cost to
  // generate s
  cout << dp[0][5][rules['S']] << '\n'; // 7
  cyk("acbc");
  cout << dp[0][3][rules['S']] << '\n'; // INT_MAX
  add_to_cnf('S', "S", -1);
  cyk("abbbbc");
  cout << neg_INF[0][5][rules['S']] << '\n'; // 1
}
```

7.5. 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;
  }
}
```

7.6. Manacher's Algorithm

```
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;
  }
}
```

7.7. 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;
}
```

7.8. Palindromic Tree

```
struct palindromic_tree {
  struct node {
    int next[26], fail, len;
    int cnt,
    num; // cnt: appear times, num: number of pal. suf.
    node(int l = 0) : fail(0), len(l), cnt(0), num(0) {
      for (int i = 0; i < 26; ++i) next[i] = 0;
    }
  };
  vector<node> St;
  vector<char> s;
  int last, n;
  palindromic_tree() : St(2), last(1), n(0) {
    St[0].fail = 1, St[1].len = -1, s.pb(-1);
  }
  inline void clear() {
    St.clear(), s.clear(), last = 1, n = 0;
    St.pb(0), St.pb(-1);
    St[0].fail = 1, s.pb(-1);
  }
  inline int get_fail(int x) {
    while (s[n - St[x].len - 1] != s[n]) x = St[x].fail;
    return x;
  }
  inline void add(int c) {
    s.push_back(c -= 'a'), ++n;
    int cur = get_fail(last);
    if (!St[cur].next[c]) {
      int now = SZ(St);
      St.pb(St[cur].len + 2);
      St[now].fail = St[get_fail(St[cur].fail)].next[c];
      St[cur].next[c] = now;
      St[now].num = St[St[now].fail].num + 1;
    }
    last = St[cur].next[c], ++St[last].cnt;
  }
  inline void count() { // counting cnt
    auto i = St.rbegin();
    for (; i != St.rend(); ++i) {
      St[i->fail].cnt += i->cnt;
    }
  }
  inline int size() { // The number of diff. pal.
    return SZ(St) - 2;
  }
};
```