
Contents

1 Misc	2	4.3.1 Mod Struct	12
1.1 Contest	2	4.3.2 Miller-Rabin	13
1.1.1 Macros and debug	2	4.3.3 Linear Sieve	13
2 Data Structures	3	4.3.4 Get Factors	14
2.1 GNU PBDS	3	4.3.5 Extended GCD	14
2.2 2D Partial Sums	3	4.3.6 Chinese Remainder Theorem	14
2.3 Sparse Table	4	4.3.7 Pollard Rho	15
2.4 Fenwick Tree	5	4.3.8 De Bruijn Sequence	15
2.5 Longest Increasing Subsequence	5	4.3.9 Combinatorics	15
2.6 Xobriest	6	5 Geometry	17
3 Graph	7	5.1 convex hull	17
3.1 Kuhn-Munkres algorithm	7	6 Strings	19
3.2 Strongly Connected Components	8	6.1 Knuth-Morris-Pratt Algorithm	19
3.3 Biconnected Components	8	6.2 Z Value	19
3.3.1 Articulation Points	8	6.3 Manachers Algorithm	19
4 Math	10	6.4 Trie	20
4.1 SOS DP	10	6.5 Aho-Corasick Automaton	21
4.2 Matrix	10		
4.3 Number Theory	12		

1. Misc

1.1. Contest

1.1.1. Macros and debug

```
1 #pragma GCC optimize("O3", "unroll-loops")
```

- 1 - Pre-submit:
 - 3 - Did you make a typo when copying a template?
 - 3 - Test more cases if unsure.
 - 5 - Write a naive solution and check small cases.
 - 5 - Submit the correct file.
- 7 - General Debugging:
 - 9 - Read the whole problem again.
 - 9 - Have a teammate read the problem.
 - 9 - Have a teammate read your code.
 - 11 - Explain your solution to them (or a rubber duck).
 - 11 - Print the code and its output / debug output.
 - 13 - Go to the toilet.
- 15 - Wrong Answer:
 - 17 - Any possible overflows?
 - 17 - > `__int128` ?
 - 17 - Try `-ftrapv` or `#pragma GCC optimize("trapv")`
 - 19 - Floating point errors?
 - 21 - > `long double` ?
 - 21 - turn off math optimizations

- 23 - check for `==`, `>=`, `acos(1.000000001)`, etc.
- 23 - Did you forget to sort or unique?
- 23 - Generate large and worst "corner" cases.
- 25 - Check your `m` / `n`, `i` / `j` and `x` / `y`.
- 25 - Are everything initialized or reset properly?
- 27 - Are you sure about the STL thing you are using?
 - 27 - Read cppreference (should be available).
- 29 - Print everything and run it on pen and paper.
- 31 - Time Limit Exceeded:
 - 31 - Calculate your time complexity again.
 - 33 - Does the program actually end?
 - 33 - Check for `while(q.size())` etc.
 - 35 - Test the largest cases locally.
 - 35 - Did you do unnecessary stuff?
 - 37 - e.g. pass vectors by value
 - 37 - e.g. `memset` for every test case
 - 39 - Is your constant factor reasonable?
- 39 - Runtime Error:
 - 41 - Check memory usage.
 - 41 - Forget to clear or destroy stuff?
 - 43 - > `vector::shrink_to_fit()`
 - 43 - Stack overflow?
 - 45 - Bad pointer / array access?
 - 45 - Division by zero? NaN's?

2. Data Structures

2.1. GNU PBDS

```

1
3 using namespace __gnu_pbds;
  template <typename T>
5 using ordered_set =
  tree<T, null_type, std::less<T>, rb_tree_tag,
7     tree_order_statistics_node_update>;
  ordered_set<int> os;
9 os.insert(5);
  os.erase(5);
11 // iterator to k-th element (0-indexed)
  os.find_by_order(k);
13 // number of elements < x
  os.order_of_key(x);

```

2.2. 2D Partial Sums

```

1 struct PrefixSum2D {
  vll pref; // 0-based 2-D prefix sum
3 void build(const vll &v) { // creates a copy
  int n = v.size(), m = v[0].size();
5 pref.assign(n, vll(m, 0));
  for (int i = 0; i < n; i++) {
7     for (int j = 0; j < m; j++) {
        pref[i][j] = v[i][j] + (i ? pref[i - 1][j] : 0) +

```

```

9         (j ? pref[i][j - 1] : 0) -
        (i && j ? pref[i - 1][j - 1] : 0);
11     }
    }
13 }
  ll query(int ulx, int uly, int brx, int bry) const {
15     ll ans = pref[brx][bry];
    if (ulx) ans -= pref[ulx - 1][bry];
17     if (uly) ans -= pref[brx][uly - 1];
    if (ulx && uly) ans += pref[ulx - 1][uly - 1];
19     return ans;
    }
21 ll query(int ulx, int uly, int size) const {
    return query(ulx, uly, ulx + size - 1, uly + size - 1);
23 }
};
25 struct PartialSum2D : PrefixSum2D {
  vll diff; // 0 based
27 int n, m;
  PartialSum2D(int _n, int _m) : n(_n), m(_m) {
29     diff.assign(n + 1, vll(m + 1, 0));
    }
31 // add c from [ulx,uly] to [brx,bry]
  void update(int ulx, int uly, int brx, int bry, ll c) {
33     diff[ulx][uly] += c;
    diff[ulx][bry + 1] -= c;
35     diff[brx + 1][uly] -= c;

```

```

diff[brx + 1][bry + 1] += c;
37 }
void update(int ulx, int uly, int size, ll c) {
39     int brx = ulx + size - 1, bry = uly + size - 1;
    update(ulx, uly, brx, bry, c);
41 }
// process the grid using prefix sum
43 void process() { this->build(diff); }
};
45 // usage
PrefixSum2D pref;
47 pref.build(v); // takes 2d 0-based vector as input
pref.query(x1, y1, x2, y2); // sum of region
49
PartialSum2D part(n, m); // dimension of grid 0 based
51 part.update(x1, y1, x2, y2, 1); // add 1 in region
// must run after all updates
53 part.process(); // prefix sum on diff array
// only exists after processing
55 vll &grid = part.pref; // processed diff array
part.query(x1, y1, x2, y2); // gives sum of region

```

2.3. Sparse Table

```

1 // Purpose: Static Range Queries
  // Idempotent (Min/Max/GCD): O(1)
3 // Non-Idempotent (Sum/Prod/XOR): O(log N)
  struct SparseTable {

```

```

5 vector<vector<long long>>> sparse;
  vector<int> Log;
7 int n, max_log;
  long long IDENTITY_VAL; // e.g., 0 for Sum, 1 for Product,
9 // INF for Min
  long long func(long long a, long long b) {
11     return (a + b); // Example: Sum
  }
13 void build(const vector<long long> &a,
    long long identity) {
15     n = a.size();
    IDENTITY_VAL = identity;
17     max_log = 32 - __builtin_clz(n);
    sparse.assign(n, vector<long long>(max_log));
19     Log.assign(n + 1, 0);
    for (int i = 2; i <= n; ++i) Log[i] = Log[i / 2] + 1;
21     for (int i = 0; i < n; ++i) sparse[i][0] = a[i];
    for (int j = 1; (1 << j) <= n; ++j) {
23         for (int i = 0; i + (1 << j) <= n; ++i) {
            sparse[i][j] =
25             func(sparse[i][j - 1],
                sparse[i + (1 << (j - 1))][j - 1]);
27         }
    }
29 }
// 3. IDEMPOTENT QUERY O(1)
31 // Use for: Min, Max, GCD, OR, AND

```

```

33 long long query_idempotent(int l, int r) {
    int k = Log[r - l + 1];
    return func(sparse[l][k], sparse[r - (1 << k) + 1][k]);
35 }
// 4. NON-IDEMPOTENT QUERY O(log N)
37 // Use for: Sum, Product, XOR, Matrix Multiplication
// Logic: Decomposes range [l, r] into disjoint power-of-2
39 long long query_non_idempotent(int l, int r) {
    long long res = IDENTITY_VAL;
41 for (int j = max_log - 1; j >= 0; --j) {
    if ((1 << j) <= r - l + 1) {
43     // Combine current result with the next block
    res = func(res, sparse[l][j]);
45     l += (1 << j); // Move L forward by 2^j
    }
47 }
    return res;
49 }
};

```

2.4. Fenwick Tree

```

1 // Interface: 0-based indexing (Internal logic handles
// 1-based conversion). for point updates and range query
3 struct FenwickTree {
    vector<long long> bit;
5 int n;
    FenwickTree(int n) {

```

```

7     this->n = n + 1;
    bit.assign(n + 1, 0);
9 }
    FenwickTree(const vector<int> &a)
11 : FenwickTree(a.size()) {
    for (size_t i = 0; i < a.size(); i++) add(i, a[i]);
13 }
    long long sum(int idx) {
15     long long ret = 0;
    for (++idx; idx > 0; idx -= idx & -idx) ret += bit[idx];
17     return ret;
    }
19 long long sum(int l, int r) {
    if (l > r) return 0; // Guard clause
21     return sum(r) - sum(l - 1);
    }
23 void add(int idx, int delta) {
    for (++idx; idx < n; idx += idx & -idx)
25     bit[idx] += delta;
    }
27 };

```

2.5. Longest Increasing Subsequence

```

1 int lis(vector<int> const &a) {
    int n = a.size();
3     const int INF = 1e9;
    vector<int> d(n + 1, INF); // min possible ending value

```

```

5 // of inc subseq of length l, that we have seen
  d[0] = -INF;
7 // user lower bound for non-decreasing
  for (int i = 0; i < n; i++) {
9     int l =
        upper_bound(d.begin(), d.end(), a[i]) - d.begin();
11    if (d[l - 1] < a[i] && a[i] < d[l]) d[l] = a[i];
    }
13    int ans = 0;
    for (int l = 0; l <= n; l++) {
15        if (d[l] < INF) ans = l;
    }
17    return ans;
    }

```

```

        return l + splitmix64() % (r - l + 1);
13    }
    void solve() {
15        map<ll, pll> mp;
        vector<pll> pref;
17        for (int i = 1; i <= n; i++) {
            mp[a[i]] = {randInRange(0LL, (1LL << 64) - 1),
19                randInRange(0LL, (1LL << 64) - 1)};
            pref[i].ff = pref[i - 1].ff ^ mp[a[i]].ff;
21            pref[i].ss = pref[i - 1].ss ^ mp[a[i]].ss;
        }
23    }

```

2.6. Xobriest

```

1 static uint64_t seed =
  chrono::steady_clock::now().time_since_epoch().count();
3 uint64_t splitmix64() {
    seed += 0x9e3779b97f4a7c15;
5    uint64_t x = seed;
    x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
7    x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
    return x ^ (x >> 31);
9 }
long long randInRange(long long l, long long r) {
11    assert(l <= r);

```

3. Graph

3.1. Kuhn-Munkres algorithm

```

1 // Maximum Weight Perfect Bipartite Matching
2 // Detect non-perfect-matching:
3 // 1. set all edge[i][j] as INF
4 // 2. if solve() >= INF, it is not perfect matching.
5
6 typedef long long ll;
7 struct KM {
8     static const int MAXN = 1050;
9     static const ll INF = 1LL << 60;
10    int n, match[MAXN], vx[MAXN], vy[MAXN];
11    ll edge[MAXN][MAXN], lx[MAXN], ly[MAXN], slack[MAXN];
12    void init(int _n) {
13        n = _n;
14        for (int i = 0; i < n; i++)
15            for (int j = 0; j < n; j++) edge[i][j] = 0;
16    }
17    void add_edge(int x, int y, ll w) { edge[x][y] = w; }
18    bool DFS(int x) {
19        vx[x] = 1;
20        for (int y = 0; y < n; y++) {
21            if (vy[y]) continue;
22            if (lx[x] + ly[y] > edge[x][y]) {
23                slack[y] =
24                    min(slack[y], lx[x] + ly[y] - edge[x][y]);
25            } else {
26                vy[y] = 1;
27                if (match[y] == -1 || DFS(match[y])) {
28                    match[y] = x;
29                    return true;
30                }
31            }
32        }
33        return false;
34    }
35    ll solve() {
36        fill(match, match + n, -1);
37        fill(lx, lx + n, -INF);
38        fill(ly, ly + n, 0);
39        for (int i = 0; i < n; i++)
40            for (int j = 0; j < n; j++)
41                lx[i] = max(lx[i], edge[i][j]);
42        for (int i = 0; i < n; i++) {
43            fill(slack, slack + n, INF);
44            while (true) {
45                fill(vx, vx + n, 0);
46                fill(vy, vy + n, 0);
47                if (DFS(i)) break;
48                ll d = INF;
49                for (int j = 0; j < n; j++)
50                    if (!vy[j]) d = min(d, slack[j]);
51                for (int j = 0; j < n; j++) {

```

```

53     if (vx[j]) lx[j] -= d;
54     if (vy[j]) ly[j] += d;
55     else slack[j] -= d;
56 }
57 }
58 ll res = 0;
59 for (int i = 0; i < n; i++) {
60     res += edge[match[i]][i];
61 }
62 return res;
63 }
} graph;

```

3.2. Strongly Connected Components

```

1 struct TarjanScc {
2     int n, step;
3     vector<int> time, low, instk, stk;
4     vector<vector<int>> e, scc;
5     TarjanScc(int n_)
6         : n(n_), step(0), time(n), low(n), instk(n), e(n) {}
7     void add_edge(int u, int v) { e[u].push_back(v); }
8     void dfs(int x) {
9         time[x] = low[x] = ++step;
10        stk.push_back(x);
11        instk[x] = 1;
12        for (int y : e[x])

```

```

13        if (!time[y]) {
14            dfs(y);
15            low[x] = min(low[x], low[y]);
16        } else if (instk[y]) {
17            low[x] = min(low[x], time[y]);
18        }
19        if (time[x] == low[x]) {
20            scc.emplace_back();
21            for (int y = -1; y != x; ) {
22                y = stk.back();
23                stk.pop_back();
24                instk[y] = 0;
25                scc.back().push_back(y);
26            }
27        }
28    }
29    void solve() {
30        for (int i = 0; i < n; i++)
31            if (!time[i]) dfs(i);
32        reverse(scc.begin(), scc.end());
33        // scc in topological order
34    }
35 };

```

3.3. Biconnected Components

3.3.1. Articulation Points

```

1 class Solution {

```



```
int cnt;
3 int dfs(int u, int p, vvi &adj, vi &vis, vi &low,
    vvi &ans) {
5     if (vis[u] != -1) return low[u];
    vis[u] = cnt, low[u] = cnt;
7     cnt++;
    for (auto v : adj[u]) {
9         if (v == p) continue;
        int temp = dfs(v, u, adj, vis, low, ans);
11        low[u] = min(low[u], low[v]);
        if (temp > vis[u]) ans.push_back({u, v});
13        else low[u] = min(low[u], vis[v]);
    }
15    return low[u];
}
17 vvi tarjanAlgorithm(int n, vvi &edges) {
    vector<vector<int>> adj(n);
19    for (int i = 0; i < edges.size(); i++) {
        int u = edges[i][0], v = edges[i][1];
21        adj[u].pb(v), adj[v].pb(u);
    }
23    vi vis(n, -1), low(n, -1);
    vector<vector<int>> ans;
25    cnt = 1;
    dfs(0, -1, adj, vis, low, ans);
27    return ans;
}
```

```
29 };
```

4. Math

4.1. SOS DP

```

1  const long long LOG = 20;
   const long long sz = (1 << LOG);
3  void forward1(
   vector<long long> &dp) { // subSet contribution to superset
5     for (int b = 0; b <= LOG; b++)
         for (int i = 0; i <= sz; i++)
7         if (i & (1 << b)) dp[i] += dp[i ^ (1 << b)];
   }
9  void backward1(vector<long long> &dp) { // undo of forward 1
   for (int b = LOG; b >= 0; b--)
11     for (int i = sz; i >= 0; i--)
         if (i & (1 << b)) dp[i] -= dp[i ^ (1 << b)];
13 }
   void forward2(
15 vector<long long> &dp) { // superset contributes to subset
   for (int b = 0; b <= LOG; b++)
17     for (int i = 0; i <= sz; i++)
         if (i & (1 << b)) dp[i ^ (1 << b)] += dp[i];
19 }
   void backward2(vector<long long> &dp) { // undo of forward 2
21     for (int b = LOG; b >= 0; b--)
         for (int i = sz; i >= 0; i--)
23         if (i & (1 << b)) dp[i ^ (1 << b)] -= dp[i];
   }

```

4.2. Matrix

```

1  /**
   * Generic Matrix Template
3  * * Purpose: Matrix Exponentiation and Operations
   * * Usage:
5  * Matrix A(n, n, 1e9+7); // Modular Arithmetic
   * Matrix B(n, n); // Standard Arithmetic (mod = 0)
7  * * Complexity: Multiplication  $O(N^3)$ , Power  $O(N^3 \log Exp)$ 
   */
9  struct Matrix {
   using ll = long long;
11     vector<vector<ll>> mat;
   int rows, cols;
13     ll mod; // mod = 0 implies Standard Arithmetic (No Modulo)

15     // Constructor: Default mod is 0 (No Mod)
   Matrix(int r, int c, ll m = 0)
17         : rows(r), cols(c), mod(m) {
         mat.assign(rows, vector<ll>(cols, 0));
19     }

21     // Input Matrix
   void input() {
23         for (int i = 0; i < rows; ++i)
             for (int j = 0; j < cols; ++j) cin >> mat[i][j];
25     }

```

```

27 // Print Matrix
void print() const {
29     for (const auto &row : mat) {
        for (const auto &val : row) cout << val << " ";
31     cout << endl;
    }
33 }

```

```

35 // Addition
Matrix operator+(const Matrix &other) const {
37     Matrix result(rows, cols, mod);
    for (int i = 0; i < rows; ++i) {
39         for (int j = 0; j < cols; ++j) {
            result.mat[i][j] = mat[i][j] + other.mat[i][j];
41             if (mod) result.mat[i][j] %= mod;
        }
43     }
    return result;
45 }

```

```

47 // Subtraction
Matrix operator-(const Matrix &other) const {
49     Matrix result(rows, cols, mod);
    for (int i = 0; i < rows; ++i) {
51         for (int j = 0; j < cols; ++j) {
            result.mat[i][j] = mat[i][j] - other.mat[i][j];
53             if (mod)

```

```

        result.mat[i][j] =
55         (result.mat[i][j] % mod + mod) % mod;
    }
57 }
    return result;
59 }

61 // Multiplication  $O(N^3)$ 
Matrix operator*(const Matrix &other) const {
63     // Assert matching dimensions if needed: assert(cols ==
        // other.rows);
65     Matrix result(rows, other.cols, mod);
    for (int i = 0; i < rows; ++i) {
67         for (int k = 0; k < cols;
            ++k) { // Optimized loop order (i-k-j is cache
                // friendly)
69             if (mat[i][k] == 0)
71                 continue; // Optimization for sparse matrices
            for (int j = 0; j < other.cols; ++j) {
73                 if (mod) {
                    result.mat[i][j] =
75                     (result.mat[i][j] +
                        mat[i][k] * other.mat[k][j]) %
77                     mod;
                } else {
79                     result.mat[i][j] += mat[i][k] * other.mat[k][j];
                }

```

```

81     }
82     }
83 }
84 return result;
85 }

87 // Matrix Exponentiation  $O(N^3 \log \text{Exp})$ 
Matrix power(ll exp) const {
89     // Identity Matrix
    Matrix result(rows, cols, mod);
91     for (int i = 0; i < rows; ++i) result.mat[i][i] = 1;

93     Matrix base = *this;
    while (exp > 0) {
95         if (exp & 1) result = result * base;
        base = base * base;
97         exp >>= 1;
    }
99     return result;
100 }
101 };

```

4.3. Number Theory

4.3.1. 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

```

1
2
3 template <typename T> struct M {
4     static T MOD; // change to constexpr if already known
5     T v;
6     M(T x = 0) {
7         v = (-MOD <= x && x < MOD) ? x : x % MOD;
8         if (v < 0) v += MOD;
9     }
10    explicit operator T() const { return v; }
11    bool operator==(const M &b) const { return v == b.v; }
12    bool operator!=(const M &b) const { return v != b.v; }
13    M operator-() { return M(-v); }
14    M operator+(M b) { return M(v + b.v); }
15    M operator-(M b) { return M(v - b.v); }
16    M operator*(M b) { return M((__int128)v * b.v % MOD); }
17    M operator/(M b) { return *this * (b ^ (MOD - 2)); }

```

```

// change above implementation to this if MOD is not prime
19 M inv() {
    auto [p, _, g] = extgcd(v, MOD);
21     return assert(g == 1), p;
    }
23 friend M operator^(M a, ll b) {
    M ans(1);
25     for (; b >>= 1, a *= a)
        if (b & 1) ans *= a;
27     return ans;
    }
29 friend M &operator+=(M &a, M b) { return a = a + b; }
    friend M &operator-=(M &a, M b) { return a = a - b; }
31 friend M &operator*=(M &a, M b) { return a = a * b; }
    friend M &operator/=(M &a, M b) { return a = a / b; }
33 };
using Mod = M<int>;
35 template<> int Mod::MOD = 1'000'000'007;
int &MOD = Mod::MOD;

```

4.3.2. Miller-Rabin

Requires: Mod Struct

```

1
3 // checks if Mod::MOD is prime
bool is_prime() {
5     if (MOD < 2 || MOD % 2 == 0) return MOD == 2;

```

```

Mod A[] = {2, 7, 61}; // for int values (< 2^31)
7 // ll: 2, 325, 9375, 28178, 450775, 9780504, 1795265022
int s = __builtin_ctzll(MOD - 1), i;
9 for (Mod a : A) {
    Mod x = a ^ (MOD >> s);
11     for (i = 0; i < s && (x + 1).v > 2; i++) x *= x;
        if (i && x != -1) return 0;
13     }
    return 1;
15 }

```

4.3.3. Linear Sieve

```

1 constexpr ll MAXN = 1000000;
    bitset<MAXN> is_prime;
3 vector<ll> primes;
    ll mpf[MAXN], phi[MAXN], mu[MAXN];
5
void sieve() {
7     is_prime.set();
    is_prime[1] = 0;
9     mu[1] = phi[1] = 1;
    for (ll i = 2; i < MAXN; i++) {
11         if (is_prime[i]) {
            mpf[i] = i;
13             primes.push_back(i);
            phi[i] = i - 1;
15             mu[i] = -1;

```

```

    }
17  for (ll p : primes) {
    if (p > mpf[i] || i * p >= MAXN) break;
19  is_prime[i * p] = 0;
    mpf[i * p] = p;
21  mu[i * p] = -mu[i];
    if (i % p == 0)
23     phi[i * p] = phi[i] * p, mu[i * p] = 0;
    else phi[i * p] = phi[i] * (p - 1);
25  }
    }
27 }

```

4.3.4. Get Factors

Requires: Linear Sieve

```

1  vector<ll> all_factors(ll n) {
3  vector<ll> fac = {1};
5  while (n > 1) {
    const ll p = mpf[n];
7  vector<ll> cur = {1};
    while (n % p == 0) {
9     n /= p;
    cur.push_back(cur.back() * p);
11  }
    vector<ll> tmp;

```

```

13  for (auto x : fac)
    for (auto y : cur) tmp.push_back(x * y);
15  tmp.swap(fac);
    }
17  return fac;
    }

```

4.3.5. Extended GCD

```

1  // 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
3  tuple<ll, ll, ll> extgcd(ll a, ll b) {
    ll s = 1, t = 0, u = 0, v = 1;
5  while (b) {
    ll q = a / b;
7  swap(a -= q * b, b);
    swap(s -= q * t, t);
9  swap(u -= q * v, v);
    }
11  return {s, u, a};
    }

```

4.3.6. Chinese Remainder Theorem

Requires: Extended GCD

```

1  // for 0 <= a < m, 0 <= b < n, returns the smallest x >= 0
   // such that x % m == a and x % n == b
3  ll crt(ll a, ll m, ll b, ll n) {

```

```

5 if (n > m) swap(a, b), swap(m, n);
  auto [x, y, g] = extgcd(m, n);
7 assert((a - b) % g == 0); // no solution
  x = ((b - a) / g * x) % (n / g) * m + a;
9 return x < 0 ? x + m / g * n : x;
}

```

4.3.7. Pollard Rho

```

1 ll f(ll x, ll mod) { return (x * x + 1) % mod; }
  // n should be composite
3 ll pollard_rho(ll n) {
  if (!(n & 1)) return 2;
5 while (1) {
  ll y = 2, x = RNG() % (n - 1) + 1, res = 1;
7 for (int sz = 2; res == 1; sz *= 2) {
  for (int i = 0; i < sz && res <= 1; i++) {
9 x = f(x, n);
  res = __gcd(abs(x - y), n);
11 }
  y = x;
13 }
  if (res != 0 && res != n) return res;
15 }
}

```

4.3.8. De Bruijn Sequence

```

1 int res[kN], aux[kN], a[kN], sz;

```

```

void Rec(int t, int p, int n, int k) {
3 if (t > n) {
  if (n % p == 0)
5 for (int i = 1; i <= p; ++i) res[sz++] = aux[i];
  } else {
7 aux[t] = aux[t - p];
  Rec(t + 1, p, n, k);
9 for (aux[t] = aux[t - p] + 1; aux[t] < k; ++aux[t])
  Rec(t + 1, t, n, k);
11 }
}
13 int DeBruijn(int k, int n) {
  // return cyclic string of length k^n such that every
15 // string of length n using k character appears as a
  // substring.
17 if (k == 1) return res[0] = 0, 1;
  fill(aux, aux + k * n, 0);
19 return sz = 0, Rec(1, 1, n, k), sz;
}

```

4.3.9. Combinatorics

```

1 struct Combinatorics {
  const int MOD;
3 vector<long long> fact, invFact;
5 // Constructor
  Combinatorics(int maxN, int mod)

```

```

7      : MOD(mod), fact(maxN + 1), invFact(maxN + 1) {
      precompute(maxN);
9  }

11 // Function to perform modular exponentiation:  $a^b \% MOD$ 
    long long modpow(long long a, long long b) const {
13     long long res = 1;
        while (b) {
15         if (b & 1) res = res * a % MOD;
            a = a * a % MOD;
17         b >>= 1;
        }
19     return res;
    }

21 // Precomputing factorials and modular inverses
    void precompute(int maxN) {
23         fact[0] = 1;
            for (int i = 1; i <= maxN; i++) {
25                 fact[i] = fact[i - 1] * i % MOD;
            }
27         invFact[maxN] =
            modpow(fact[maxN], MOD - 2); // Fermat's Little Theorem
            for (int i = maxN - 1; i >= 0; i--) {
31                 invFact[i] = invFact[i + 1] * (i + 1) % MOD;
            }
33     }

```

```

35 // Function to calculate  $nCk \% MOD$ 
    long long nCk(int n, int k) const {
37         if (k > n || k < 0) return 0;
            return fact[n] * invFact[k] % MOD * invFact[n - k] %
39                 MOD;
        }

41 // Function to calculate  $nPk \% MOD$ 
    long long nPk(int n, int k) const {
43         if (k > n || k < 0) return 0;
            return fact[n] * invFact[n - k] % MOD;
45         }

47 // Function to calculate  $n! \% MOD$ 
    long long factorial(int n) const { return fact[n]; }
49 };

51 // Combinatorics comb(maxN, mod)

```


5. Geometry

5.1. convex hull

```

1 struct Point {
2     ll x, y;
3     Point() : x(0), y(0) {}
4     Point(ll _x, ll _y) : x(_x), y(_y) {}
5     bool operator==(const Point &other) const {
6         return x == other.x && y == other.y;
7     }
8     bool operator<(const Point &other) const {
9         if (x != other.x) return x < other.x;
10        return y < other.y;
11    }
12};
13 ll cross_product(const Point &A, const Point &B,
14                 const Point &C) {
15    /*
16     *  $cross(A, B, C)$  tells you how the angle turns when you go  $A \rightarrow B \rightarrow C$ .
17     * If  $cross > 0 \rightarrow$  left turn (counter-clockwise)
18     * If  $cross < 0 \rightarrow$  right turn (clockwise)
19     * If  $cross = 0 \rightarrow$  collinear
20     */
21    return (B.x - A.x) * (C.y - A.y) -
22           (B.y - A.y) * (C.x - A.x);
23 }
24 long long dot_product(const Point &A, const Point &B,
25                      const Point &C) {
26    // computes  $(B - A) \cdot (C - A)$ 
27    return (B.x - A.x) * (C.x - A.x) +
28           (B.y - A.y) * (C.y - A.y);
29 }
30 vector<Point> ConvexHullAndrowChain(vector<Point> pts) {
31    sort(pts);
32    pts.erase(unique(pts.begin(), pts.end()), pts.end());
33    int n = pts.size();
34    if (n <= 1) return pts;
35    vector<Point> lower, upper;
36    // Build lower hull
37    for (int i = 0; i < n; ++i) {
38        const Point &p = pts[i];
39        while (lower.size() >= 2 &&
40             cross_product(lower[lower.size() - 2],
41                          lower[lower.size() - 1], p) <= 0) {
42            lower.pop_back();
43        }
44        lower.push_back(p);
45    }
46    // Build upper hull
47    for (int i = n - 1; i >= 0; --i) {
48        const Point &p = pts[i];
49        while (upper.size() >= 2 &&
50             cross_product(upper[upper.size() - 2],
51                          upper[upper.size() - 1], p) <= 0) {

```

```
    upper.pop_back();
53 }
    upper.push_back(p);
55 }
    vector<Point> hull = lower;
57 for (int i = 1; i + 1 < (int)upper.size(); ++i) {
    hull.push_back(upper[i]);
59 }

61 return hull; // CCW order
}
```

6. Strings

6.1. Knuth-Morris-Pratt Algorithm

```

1 vector<int> pi(const string &s) {
  vector<int> p(s.size());
3  for (int i = 1; i < s.size(); i++) {
    int g = p[i - 1];
5    while (g && s[i] != s[g]) g = p[g - 1];
    p[i] = g + (s[i] == s[g]);
7  }
  return p;
9 }

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

```

6.2. Z Value

```

1 int z[n];
void zval(string s) {
3  // z[i] => longest common prefix of s and s[i:], i > 0
  int n = s.size();
  z[0] = 0;
5  for (int b = 0, i = 1; i < n; i++) {

```

```

7   if (z[b] + b <= i) z[i] = 0;
   else z[i] = min(z[i - b], z[b] + b - i);
9   while (s[i + z[i]] == s[z[i]]) z[i]++;
   if (i + z[i] > b + z[b]) b = i;
11  }
}

```

6.3. Manachers Algorithm

```

1 int z[n];
void manacher(string s) {
3  // z[i] => longest odd palindrome centered at i is
  //      s[i - z[i] ... i + z[i]]
5  // to get all palindromes (including even length),
  // insert a '#' between each s[i] and s[i + 1]
7  int n = s.size();
  z[0] = 0;
9  for (int b = 0, i = 1; i < n; i++) {
    if (z[b] + b >= i)
11     z[i] = min(z[2 * b - i], b + z[b] - i);
    else z[i] = 0;
13    while (i + z[i] + 1 < n && i - z[i] - 1 >= 0 &&
      s[i + z[i] + 1] == s[i - z[i] - 1])
15     z[i]++;
    if (z[i] + i > z[b] + b) b = i;
17  }
}

```

6.4. Trie

```

1 class Trie {
2 public:
3     struct Node {
4         vector<int> next; // Indices of children nodes
5         int pfxCnt = 0; // How many words pass through this node
6         int wordCnt =
7             0; // How many words end exactly at this node
8
9         Node(int maxChars) {
10             next.assign(maxChars, -1);
11             pfxCnt = 0;
12             wordCnt = 0;
13         }
14     };
15     vector<Node> nodes;
16     int
17     distWords; // Count of distinct words currently in Trie
18     int maxChars; // Alphabet size (usually 26)
19     int getBase(char c) {
20         return c - 'Change'; // based on problem
21     }
22
23     Trie(int maxChars = 26) {
24         this->maxChars = maxChars;
25         nodes.clear();
26         distWords = 0;
27         // Create Root Node (Index 0)
28         nodes.emplace_back(maxChars);
29     }
30
31     // Insert string s into Trie
32     void insert(string s) {
33         int curr = 0;
34         nodes[curr].pfxCnt++;
35         for (char &ch : s) {
36             int base = getBase(ch);
37             // If path doesn't exist, create new node
38             if (nodes[curr].next[base] == -1) {
39                 nodes[curr].next[base] = nodes.size();
40                 nodes.emplace_back(maxChars);
41             }
42             curr = nodes[curr].next[base];
43             nodes[curr].pfxCnt++;
44         }
45         if (nodes[curr].wordCnt == 0) {
46             distWords++; // New distinct word found
47         }
48         nodes[curr].wordCnt++;
49     }
50
51     // Check if string s exists
52     bool search(string s) {
53         int curr = 0;

```

```

55  for (char &ch : s) {
    int base = getBase(ch);
    if (nodes[curr].next[base] == -1) return false;
57  curr = nodes[curr].next[base];
    }
59  return nodes[curr].wordCnt > 0;
    }
61  // Delete one occurrence of s
    void erase(string s) {
63  if (!search(s)) return; // Check existence first
    int curr = 0;
65  nodes[curr].pfxCnt--;
    for (char &ch : s) {
67  int base = getBase(ch);
        curr = nodes[curr].next[base];
69  nodes[curr].pfxCnt--;
    }
71  nodes[curr].wordCnt--;
    if (nodes[curr].wordCnt == 0)
73  distWords--; // Word completely removed
    }
75  // Count words that have s as a prefix
77  int prefixCount(string s) {
    int curr = 0;
79  for (char &ch : s) {
        int base = getBase(ch);

```

```

81  if (nodes[curr].next[base] == -1)
    return 0; // Prefix not found
83  curr = nodes[curr].next[base];
    }
85  return nodes[curr].pfxCnt;
    }
87  };

```

6.5. Aho-Corasick Automaton

```

1  const int ALPHA = 26, MAXNODES = 500000 + 5;
    int nxt[MAXNODES][ALPHA];
3  int linkS[MAXNODES];
    ll cntNode[MAXNODES];
5  vector<int> adjSL[MAXNODES];
    vector<int> patEnd;
7  int nodes = 1;

9  void build_trie(const vector<string> &P) {
    // clear
11  for (int i = 0; i < nodes; i++) {
        memset(nxt[i], 0, sizeof nxt[i]);
13  cntNode[i] = 0;
        adjSL[i].clear();
15  }
        nodes = 1;
17  patEnd.clear();
        patEnd.reserve(P.size());

```

```

19 // insert
   for (auto &pat : P) {
21     int u = 0;
     for (char ch : pat) {
23       int c = ch - 'a';
       if (!nxt[u][c]) nxt[u][c] = nodes++;
25       u = nxt[u][c];
     }
27     patEnd.pb(u);
   }
29 }
vector<int> bfsOrder;
31 void build_links() {
   queue<int> q;
33   linkS[0] = 0;
   // first layer
35   for (int c = 0; c < ALPHA; c++) {
     int v = nxt[0][c];
37     if (v) {
       linkS[v] = 0;
39       q.push(v);
     }
41   }
   // BFS
43   while (!q.empty()) {
     int u = q.front();
45     q.pop();

```

```

   bfsOrder.pb(u);
47   for (int c = 0; c < ALPHA; c++) {
     int v = nxt[u][c];
49     if (!v) continue;
     int j = linkS[u];
51     while (j && !nxt[j][c]) j = linkS[j];
     if (nxt[j][c]) j = nxt[j][c];
53     linkS[v] = j;
     q.push(v);
55   }
   }
57   for (int u : bfsOrder) { adjSL[linkS[u]].pb(u); }
   }
59   void solve() {
61     string S;
     ll k;
63     cin >> S >> k;
     vector<string> P(k);
65     for (int i = 0; i < k; i++) cin >> P[i];
     build_trie(P);
67     bfsOrder.clear();
     build_links();
69   }

```