

Contents			
1 Misc	2		
1.1 Contest	2	4.2 Matrix	11
1.1.1 Makefile	2	4.3 Number Theory	13
1.2 How Did We Get Here?	2	4.3.1 Mod Struct	13
1.2.1 Macros	2	4.3.2 Miller-Rabin	14
2 Data Structures	3	4.3.3 Linear Sieve	14
2.1 GNU PBDS	3	4.3.4 Get Factors	15
2.2 2D Partial Sums	3	4.3.5 Extended GCD	15
2.3 Sparse Table	4	4.3.6 Chinese Remainder Theorem	15
2.4 Fenwick Tree	5	4.3.7 Pollard Rho	16
2.5 Longest Increasing Subsequence	6	4.3.8 De Bruijn Sequence	16
2.6 Xobriest	7	4.3.9 Combinatorics	16
3 Graph	8	5 Geometry	18
3.1 Kuhn-Munkres algorithm	8	5.1 convex hull	18
3.2 Strongly Connected Components	9	6 Strings	20
3.3 Biconnected Components	9	6.1 Knuth-Morris-Pratt Algorithm	20
3.3.1 Articulation Points	9	6.2 Z Value	20
4 Math	11	6.3 Manachers Algorithm	20
4.1 SOS DP	11	6.4 Trie	21
		6.5 Aho-Corasick Automaton	22
		7 Debug List	24

1. Misc

1.1. Contest

1.1.1. Makefile

```
1 .PRECIOUS: ./p%
```



```
3 %: p%
    ulimit -s unlimited && ./$<
```



```
5 p%: p%.cpp
    g++ -o $@ $< -std=c++17 -Wall -Wextra -Wshadow \
    7 -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.

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

2. Data Structures

2.1. GNU PBDS

```

1 #include <ext/pb_ds/assoc_container.hpp>
2 #include <ext/pb_ds/priority_queue.hpp>
3 #include <ext/pb_ds/tree_policy.hpp>
4 using namespace __gnu_pbds;
5
6 // most std::map + order_of_key, find_by_order, split, join
7 template <typename T, typename U = null_type>
8 using ordered_map = tree<T, U, std::less<>, rb_tree_tag,
9                 tree_order_statistics_node_update>;
10 // useful tags: rb_tree_tag, splay_tree_tag
11
12 template <typename T> struct myhash {
13     size_t operator()(T x) const; // splitmix, bswap(x*R), ...
14 };
15 // most of std::unordered_map, but faster (needs good hash)
16 template <typename T, typename U = null_type>
17 using hash_table = gp_hash_table<T, U, myhash<T>>;
18
19 // most std::priority_queue + modify, erase, split, join
20 using heap = priority_queue<int, std::less<>>;
21 // useful tags: pairing_heap_tag, binary_heap_tag,
22 //               (rc_)?binomial_heap_tag, thin_heap_tag

```

2.2. 2D Partial Sums

```
1 using vvi = vector<vector<int>>;
```

```

2 using vvll = vector<vector<ll>>;
3 using vll = vector<ll>;
4
5 struct PrefixSum2D {
6     vvll pref;           // 0-based 2-D prefix sum
7     void build(const vvll &v) { // creates a copy
8         int n = v.size(), m = v[0].size();
9         pref.assign(n, vll(m, 0));
10        pref.assign(n, vll(m, 0));
11        for (int i = 0; i < n; i++) {
12            for (int j = 0; j < m; j++) {
13                pref[i][j] = v[i][j] + (i ? pref[i - 1][j] : 0) +
14                    (j ? pref[i][j - 1] : 0) -
15                    (i && j ? pref[i - 1][j - 1] : 0);
16            }
17        }
18    ll query(int ulx, int uly, int brx, int bry) const {
19        ll ans = pref[brx][bry];
20        if (ulx) ans -= pref[ulx - 1][bry];
21        if (uly) ans -= pref[brx][uly - 1];
22        if (ulx && uly) ans += pref[ulx - 1][uly - 1];
23        return ans;
24    }
25    ll query(int ulx, int uly, int size) const {
26        return query(ulx, uly, ulx + size - 1, uly + size - 1);
27    }
28};
```

```

29 struct PartialSum2D : PrefixSum2D {
30     vvll diff; // 0 based
31     int n, m;
32     PartialSum2D(int _n, int _m) : n(_n), m(_m) {
33         diff.assign(n + 1, vll(m + 1, 0));
34     }
35     // add c from [ulx,uly] to [brx,bry]
36     void update(int ulx, int uly, int brx, int bry, ll c) {
37         diff[ulx][uly] += c;
38         diff[ulx][bry + 1] -= c;
39         diff[brx + 1][uly] -= c;
40         diff[brx + 1][bry + 1] += c;
41     }
42     void update(int ulx, int uly, int size, ll c) {
43         int brx = ulx + size - 1;
44         int bry = uly + size - 1;
45         update(ulx, uly, brx, bry, c);
46     }
47     // process the grid using prefix sum
48     void process() { this->build(diff); }
49 };
50 // usage
51 PrefixSum2D pref;
52 pref.build(v); // takes 2d 0-based vector as input
53 pref.query(x1, y1, x2, y2); // sum of region
54
55 PartialSum2D part(n, m); // dimension of grid 0 based

```

```

56     part.update(x1, y1, x2, y2, 1); // add 1 in region
57     // must run after all updates
58     part.process(); // prefix sum on diff array
59     // only exists after processing
60     vvll &grid = part.pref; // processed diff array
61     part.query(x1, y1, x2, y2); // gives sum of region

```

2.3. Sparse Table

```

1 /**
2 * Sparse Table (Generic)
3 * * Purpose: Static Range Queries
4 * * Query Time:
5 * - Idempotent (Min/Max/GCD): O(1)
6 * - Non-Idempotent (Sum/Prod/XOR): O(log N)
7 */
8 struct SparseTable {
9     vector<vector<long long>> sparse;
10    vector<int> Log;
11    int n, max_log;
12    long long IDENTITY_VAL; // e.g., 0 for Sum, 1 for Product,
13                                // INF for Min
14
15    // 1. OPERATION FUNCTION (Change this)
16    long long func(long long a, long long b) {
17        return (a + b); // Example: Sum
18    }
19

```

```

// 2. BUILD
21 void build(const vector<long long> &a,
22     long long identity) {
23     n = a.size();
24     IDENTITY_VAL = identity;
25     max_log = 32 - __builtin_clz(n);
26     sparse.assign(n, vector<long long>(max_log));
27     Log.assign(n + 1, 0);
28
29     for (int i = 2; i <= n; ++i) Log[i] = Log[i / 2] + 1;
30
31     for (int i = 0; i < n; ++i) sparse[i][0] = a[i];
32
33     for (int j = 1; (1 << j) <= n; ++j) {
34         for (int i = 0; i + (1 << j) <= n; ++i) {
35             sparse[i][j] =
36                 func(sparse[i][j - 1],
37                     sparse[i + (1 << (j - 1))][j - 1]);
38         }
39     }
40
41 // 3. IDEMPOTENT QUERY O(1)
42 // Use for: Min, Max, GCD, OR, AND
43 long long query_idempotent(int l, int r) {
44     int k = Log[r - l + 1];
45     return func(sparse[l][k], sparse[r - (1 << k) + 1][k]);
46
47 }
48
49 // 4. NON-IDEMPOTENT QUERY O(log N)
50 // Use for: Sum, Product, XOR, Matrix Multiplication
51 // Logic: Decomposes range [l, r] into disjoint power-of-2
52 // blocks
53 long long query_non_idempotent(int l, int r) {
54     long long res = IDENTITY_VAL;
55     for (int j = max_log - 1; j >= 0; --j) {
56         if ((1 << j) <= r - l + 1) {
57             // Combine current result with the next block
58             res = func(res, sparse[l][j]);
59             // Move L forward by 2^j
60             l += (1 << j);
61         }
62     }
63     return res;
64 }
65

```

2.4. Fenwick Tree

```

1 /*
2 * Interface: 0-based indexing (Internal logic handles
3 * 1-based conversion). for point updates and range query
4 */
5 struct FenwickTree {
6     // Use long long to prevent overflow during sum operations
7 }
```

```

7  vector<long long> bit;
  int n;
9
// Initialize with size n. All values 0.
11 FenwickTree(int n) {
  this->n = n + 1;
13  bit.assign(n + 1, 0);
}
15
// Initialize with an existing vector
17 // Passing by reference (const &) saves memory copy time
FenwickTree(const vector<int> &a)
  : FenwickTree(a.size()) {
  for (size_t i = 0; i < a.size(); i++) add(i, a[i]);
}
21
23 // Computes sum of [0...idx]
long long sum(int idx) {
  long long ret = 0;
  for (++idx; idx > 0; idx -= idx & -idx) ret += bit[idx];
  return ret;
}
29
31 // Computes sum of range [l...r] (inclusive)
long long sum(int l, int r) {
  if (l > r) return 0; // Guard clause
  return sum(r) - sum(l - 1);
}

35   }
36
// Adds delta to the element at idx
37 void add(int idx, int delta) {
  for (++idx; idx < n; idx += idx & -idx)
    bit[idx] += delta;
}
41

```

2.5. Longest Increasing Subsequence

```

1 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;
}
17

```

{}

2.6. Xobriest

```
1 long long randInRange(long long l, long long r) {
2     static random_device rd;      // seed
3     static mt19937_64 gen(rd()); // 64-bit Mersenne Twister
4     uniform_int_distribution<long long> dist(l, r);
5     return dist(gen);
6 }
7 void solve() {
8     map<ll, pll> mp;
9     vector<pll> pref;
10    for (int i = 1; i <= n; i++) {
11        mp[a[i]] = {randInRange(0LL, (1LL << 64) - 1),
12                     randInRange(0LL, (1LL << 64) - 1)};
13        pref[i].ff = pref[i - 1].ff ^ mp[a[i]].ff;
14        pref[i].ss = pref[i - 1].ss ^ mp[a[i]].ss;
15    }
16 }
```

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
7 typedef long long ll;
8 struct KM {
9     static const int MAXN = 1050;
10    static const ll INF = 1LL << 60;
11    int n, match[MAXN], vx[MAXN], vy[MAXN];
12    ll edge[MAXN][MAXN], lx[MAXN], ly[MAXN], slack[MAXN];
13    void init(int _n) {
14        n = _n;
15        for (int i = 0; i < n; i++)
16            for (int j = 0; j < n; j++) edge[i][j] = 0;
17    }
18    void add_edge(int x, int y, ll w) { edge[x][y] = w; }
19    bool DFS(int x) {
20        vx[x] = 1;
21        for (int y = 0; y < n; y++) {
22            if (vy[y]) continue;
23            if (lx[x] + ly[y] > edge[x][y]) {
24                slack[y] =
25                    min(slack[y], lx[x] + ly[y] - edge[x][y]);
26            }
27        }
28        return match[y] == x;
29    }
30    int solve() {
31        fill(match, match + n, -1);
32        fill(lx, lx + n, -INF);
33        fill(ly, ly + n, 0);
34        for (int i = 0; i < n; i++)
35            for (int j = 0; j < n; j++)
36                lx[i] = max(lx[i], edge[i][j]);
37        for (int i = 0; i < n; i++) {
38            fill(slack, slack + n, INF);
39            while (true) {
40                fill(vx, vx + n, 0);
41                fill(vy, vy + n, 0);
42                if (DFS(i)) break;
43                ll d = INF;
44                for (int j = 0; j < n; j++)
45                    if (!vy[j]) d = min(d, slack[j]);
46                for (int j = 0; j < n; j++) {
47                    if (d == slack[j]) {
48                        lx[i] += d;
49                        ly[j] -= d;
50                        match[j] = i;
51                    }
52                }
53            }
54        }
55    }
56}
```

```

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 }
64 } 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_) : n(n_), step(0), time(n), low(n), instk(n), e(n) {}
6   void add_edge(int u, int v) { e[u].push_back(v); }
7   void dfs(int x) {
8     time[x] = low[x] = ++step;
9     stk.push_back(x);
10    instk[x] = 1;
11    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;
int dfs(int u, int p, vvi &adj, vi &vis, vi &low,
       vvi &ans) {
    if (vis[u] != -1) return low[u];
    vis[u] = cnt, low[u] = cnt;
    cnt++;
    for (auto v : adj[u]) {
        if (v == p) continue;
        int temp = dfs(v, u, adj, vis, low, ans);
        low[u] = min(low[u], low[v]);
        if (temp > vis[u]) ans.push_back({u, v});
        else low[u] = min(low[u], vis[v]);
    }
    return low[u];
}
vvi tarjanAlgorithm(int n, vvi &edges) {
    vector<vector<int>> adj(n);
    for (int i = 0; i < edges.size(); i++) {
        int u = edges[i][0], v = edges[i][1];
        adj[u].pb(v), adj[v].pb(u);
    }
    vi vis(n, -1), low(n, -1);
    vector<vector<int>> ans;
    cnt = 1;
    dfs(0, -1, adj, vis, low, ans);
    return ans;
}
```

29 };

4. Math

4.1. SOS DP

```

1 const long long LOG = 20;
2 const long long sz = (1 << LOG);
3 void forward1(
4     vector<long long> &dp) { // subSet contribution to superset
5     for (int b = 0; b <= LOG; b++)
6         for (int i = 0; i <= sz; i++)
7             if (i & (1 << b)) dp[i] += dp[i ^ (1 << b)];
8 }
9 void backward1(vector<long long> &dp) { // undo of forward 1
10    for (int b = LOG; b >= 0; b--)
11        for (int i = sz; i >= 0; i--)
12            if (i & (1 << b)) dp[i] -= dp[i ^ (1 << b)];
13 }
14 void forward2(
15     vector<long long> &dp) { // superset contributes to subset
16     for (int b = 0; b <= LOG; b++)
17         for (int i = 0; i <= sz; i++)
18             if (i & (1 << b)) dp[i ^ (1 << b)] += dp[i];
19 }
20 void backward2(vector<long long> &dp) { // undo of forward 2
21    for (int b = LOG; b >= 0; b--)
22        for (int i = sz; i >= 0; i--)
23            if (i & (1 << b)) dp[i ^ (1 << b)] += dp[i];
24 }
```

4.2. Matrix

```

1 /**
2 * Generic Matrix Template
3 * * Purpose: Matrix Exponentiation and Operations
4 * * Usage:
5 * Matrix A(n, n, 1e9+7); // Modular Arithmetic
6 * Matrix B(n, n);      // Standard Arithmetic (mod = 0)
7 * * Complexity: Multiplication O(N^3), Power O(N^3 log Exp)
8 */
9 struct Matrix {
10     using ll = long long;
11     vector<vector<ll>> mat;
12     int rows, cols;
13     ll mod; // mod = 0 implies Standard Arithmetic (No Modulo)
14
15     // Constructor: Default mod is 0 (No Mod)
16     Matrix(int r, int c, ll m = 0)
17         : rows(r), cols(c), mod(m) {
18             mat.assign(rows, vector<ll>(cols, 0));
19         }
20
21     // Input Matrix
22     void input() {
23         for (int i = 0; i < rows; ++i)
24             for (int j = 0; j < cols; ++j) cin >> mat[i][j];
25     }
26 }
```

```

27 // Print Matrix
28 void print() const {
29     for (const auto &row : mat) {
30         for (const auto &val : row) cout << val << " ";
31         cout << endl;
32     }
33 }
34
35 // Addition
36 Matrix operator+(const Matrix &other) const {
37     Matrix result(rows, cols, mod);
38     for (int i = 0; i < rows; ++i) {
39         for (int j = 0; j < cols; ++j) {
40             result.mat[i][j] = mat[i][j] + other.mat[i][j];
41             if (mod) result.mat[i][j] %= mod;
42         }
43     }
44     return result;
45 }
46
47 // Subtraction
48 Matrix operator-(const Matrix &other) const {
49     Matrix result(rows, cols, mod);
50     for (int i = 0; i < rows; ++i) {
51         for (int j = 0; j < cols; ++j) {
52             result.mat[i][j] = mat[i][j] - other.mat[i][j];
53             if (mod)
54                 result.mat[i][j] =
55                     (result.mat[i][j] % mod + mod) % mod;
56         }
57     }
58     return result;
59 }
60
61 // Multiplication O(N^3)
62 Matrix operator*(const Matrix &other) const {
63     // Assert matching dimensions if needed: assert(cols ==
64     // other.rows);
65     Matrix result(rows, other.cols, mod);
66     for (int i = 0; i < rows; ++i) {
67         for (int k = 0; k < cols;
68              ++k) { // Optimized loop order (i-k-j is cache
69                  // friendly)
70                 if (mat[i][k] == 0)
71                     continue; // Optimization for sparse matrices
72                 for (int j = 0; j < other.cols; ++j) {
73                     if (mod) {
74                         result.mat[i][j] =
75                             (result.mat[i][j] +
76                             mat[i][k] * other.mat[k][j]) %
77                             mod;
78                     } else {
79                         result.mat[i][j] += mat[i][k] * other.mat[k][j];
80                     }
81                 }
82             }
83         }
84     }
85     return result;
86 }

```

```

81      }
82  }
83  return result;
84 }
85

// Matrix Exponentiation O(N^3 log Exp)
86 Matrix power(ll exp) const {
87     // Identity Matrix
88     Matrix result(rows, cols, mod);
89     for (int i = 0; i < rows; ++i) result.mat[i][i] = 1;
90
91     Matrix base = *this;
92     while (exp > 0) {
93         if (exp & 1) result = result * base;
94         base = base * base;
95         exp >>= 1;
96     }
97     return result;
98 }
99
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)); }

```

```

19 // change above implementation to this if MOD is not prime
M inv() {
    auto [p, _, g] = extgcd(v, MOD);
    return assert(g == 1), p;
}
21 friend M operator^(M a, ll b) {
    M ans(1);
    for (; b; b >= 1, a *= a)
        if (b & 1) ans *= a;
    return ans;
}
23 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;

```

```

7 Mod A[] = {2, 7, 61}; // for int values (< 2^31)
// ll: 2, 325, 9375, 28178, 450775, 9780504, 1795265022
9 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;
15

```

4.3.3. Linear Sieve

```

1 constexpr ll MAXN = 1000000;
bitset<MAXN> is_prime;
3 vector<ll> primes;
5 ll mpf[MAXN], phi[MAXN], mu[MAXN];
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;
    primes.push_back(i);
    phi[i] = i - 1;
    mu[i] = -1;
13
15

```

```

17    }
18  for (ll p : primes) {
19      if (p > mpf[i] || i * p >= MAXN) break;
20      is_prime[i * p] = 0;
21      mpf[i * p] = p;
22      mu[i * p] = -mu[i];
23      if (i % p == 0)
24          phi[i * p] = phi[i] * p, mu[i * p] = 0;
25      else phi[i * p] = phi[i] * (p - 1);
26  }
27 }

```

4.3.4. Get Factors

Requires: Linear Sieve

```

1
2
3 vector<ll> all_factors(ll n) {
4     vector<ll> fac = {1};
5     while (n > 1) {
6         const ll p = mpf[n];
7         vector<ll> cur = {1};
8         while (n % p == 0) {
9             n /= p;
10            cur.push_back(cur.back() * p);
11        }
12        vector<ll> tmp;

```

```

13     for (auto x : fac)
14         for (auto y : cur) tmp.push_back(x * y);
15     tmp.swap(fac);
16 }
17 return fac;
18 }

```

4.3.5. Extended GCD

```

1 // returns (p, q, g): p * a + q * b == g == gcd(a, b)
2 // g is not guaranteed to be positive when a < 0 or b < 0
3 tuple<ll, ll, ll> extgcd(ll a, ll b) {
4     ll s = 1, t = 0, u = 0, v = 1;
5     while (b) {
6         ll q = a / b;
7         swap(a -= q * b, b);
8         swap(s -= q * t, t);
9         swap(u -= q * v, v);
10    }
11    return {s, u, a};
12 }

```

4.3.6. Chinese Remainder Theorem

Requires: Extended GCD

```

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

```

```

5 | if (n > m) swap(a, b), swap(m, n);
6 | auto [x, y, g] = extgcd(m, n);
7 | assert((a - b) % g == 0); // no solution
8 | x = ((b - a) / g * x) % (n / g) * m + a;
9 | return x < 0 ? x + m / g * n : x;
10|

```

4.3.7. Pollard Rho

```

1 | ll f(ll x, ll mod) { return (x * x + 1) % mod; }
2 | // n should be composite
3 | ll pollard_rho(ll n) {
4 |   if (!(n & 1)) return 2;
5 |   while (1) {
6 |     ll y = 2, x = RNG() % (n - 1) + 1, res = 1;
7 |     for (int sz = 2; res == 1; sz *= 2) {
8 |       for (int i = 0; i < sz && res <= 1; i++) {
9 |         x = f(x, n);
10|         res = __gcd(abs(x - y), n);
11|       }
12|       y = x;
13|     }
14|     if (res != 0 && res != n) return res;
15|   }
16|

```

4.3.8. De Bruijn Sequence

```

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

```

```

3 | void Rec(int t, int p, int n, int k) {
4 |   if (t > n) {
5 |     if (n % p == 0)
6 |       for (int i = 1; i <= p; ++i) res[sz++] = aux[i];
7 |     else {
8 |       aux[t] = aux[t - p];
9 |       Rec(t + 1, p, n, k);
10|       for (aux[t] = aux[t - p] + 1; aux[t] < k; ++aux[t])
11|         Rec(t + 1, t, n, k);
12|     }
13|   }
14|   int DeBruijn(int k, int n) {
15|     // return cyclic string of length k^n such that every
16|     // string of length n using k character appears as a
17|     // substring.
18|     if (k == 1) return res[0] = 0, 1;
19|     fill(aux, aux + k * n, 0);
20|     return sz = 0, Rec(1, 1, n, k), sz;
21|

```

4.3.9. Combinatorics

```

1 | struct Combinatorics {
2 |   const int MOD;
3 |   vector<long long> fact, invFact;
4 |
5 |   // Constructor
6 |   Combinatorics(int maxN, int mod)

```

```

7   : MOD(mod), fact(maxN + 1), invFact(maxN + 1) {
9     precompute(maxN);
11 // Function to perform modular exponentiation: a^b % MOD
12   long long modpow(long long a, long long b) const {
13     long long res = 1;
14     while (b) {
15       if (b & 1) res = res * a % MOD;
16       a = a * a % MOD;
17       b >>= 1;
18     }
19     return res;
20   }
21 // Precomputing factorials and modular inverses
22   void precompute(int maxN) {
23     fact[0] = 1;
24     for (int i = 1; i <= maxN; i++) {
25       fact[i] = fact[i - 1] * i % MOD;
26     }
27     invFact[maxN] =
28     modpow(fact[maxN], MOD - 2); // Fermat's Little Theorem
29     for (int i = maxN - 1; i >= 0; i--) {
30       invFact[i] = invFact[i + 1] * (i + 1) % MOD;
31     }
32   }
33 }

35 // Function to calculate nCk % MOD
36   long long nCk(int n, int k) const {
37     if (k > n || k < 0) return 0;
38     return fact[n] * invFact[k] % MOD * invFact[n - k] %
39           MOD;
40   }
41 // Function to calculate nPk % MOD
42   long long nPk(int n, int k) const {
43     if (k > n || k < 0) return 0;
44     return fact[n] * invFact[n - k] % MOD;
45   }
46 // Function to calculate n! % MOD
47   long long factorial(int n) const { return fact[n]; }
48 };
49 // Combinatorics comb(maxN,mod)
50
51

```

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
17      → B → C. If cross > 0 → left turn (counter-clockwise) If
18      cross < 0 → right turn (clockwise) If cross = 0 →
19      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

```

```

26
27     const Point &C) {
28     // computes (B - A) · (C - A)
29     return (B.x - A.x) * (C.x - A.x) +
30             (B.y - A.y) * (C.y - A.y);
31 }
32 vector<Point> ConvexHullAndrowChain(vector<Point> pts) {
33     sort(pts);
34     pts.erase(unique(pts.begin(), pts.end()), pts.end());
35     int n = pts.size();
36     if (n <= 1) return pts;
37     vector<Point> lower, upper;
38     // Build lower hull
39     for (int i = 0; i < n; ++i) {
40         const Point &p = pts[i];
41         while (lower.size() >= 2 &&
42                 cross_product(lower[lower.size() - 2],
43                               lower[lower.size() - 1], p) <= 0) {
44             lower.pop_back();
45         }
46         lower.push_back(p);
47     }
48     // Build upper hull
49     for (int i = n - 1; i >= 0; --i) {
50         const Point &p = pts[i];
51         while (upper.size() >= 2 &&
52                 cross_product(upper[upper.size() - 2],
53                               upper[upper.size() - 1], p) <= 0) {
54

```

```
    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) {
2     vector<int> p(s.size());
3     for (int i = 1; i < s.size(); i++) {
4         int g = p[i - 1];
5         while (g && s[i] != s[g]) g = p[g - 1];
6         p[i] = g + (s[i] == s[g]);
7     }
8     return p;
9 }
10 vector<int> match(const string &s, const string &pat) {
11     vector<int> p = pi(pat + '\0' + s), res;
12     for (int i = p.size() - s.size(); i < p.size(); i++)
13         if (p[i] == pat.size())
14             res.push_back(i - 2 * pat.size());
15     return res;
16 }
```

6.2. Z Value

```

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

```

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

6.3. Manachers Algorithm

```

1 int z[n];
2 void manacher(string s) {
3     // z[i] => longest odd palindrome centered at i is
4     //      s[i - z[i] ... i + z[i]]
5     // to get all palindromes (including even length),
6     // insert a '#' between each s[i] and s[i + 1]
7     int n = s.size();
8     z[0] = 0;
9     for (int b = 0, i = 1; i < n; i++) {
10        if (z[b] + b >= i)
11            z[i] = min(z[2 * b - i], b + z[b] - i);
12        else z[i] = 0;
13        while (i + z[i] + 1 < n && i - z[i] - 1 >= 0 &&
14              s[i + z[i] + 1] == s[i - z[i] - 1])
15            z[i]++;
16        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 - 'A'; // 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;
54
55             for (char &ch : s) {
56                 int base = getBase(ch);
57                 if (nodes[curr].next[base] == -1) {
58                     return false;
59                 }
60                 curr = nodes[curr].next[base];
61             }
62             return nodes[curr].wordCnt != 0;
63         }
64
65         int getDistWords() const {
66             return distWords;
67         }
68
69         int getMaxChars() const {
70             return maxChars;
71         }
72
73         void printTrie() {
74             for (const auto &node : nodes) {
75                 cout << "Node at index " << node.pfxCnt << ": ";
76                 for (int i = 0; i < maxChars; ++i) {
77                     cout << node.next[i];
78                 }
79                 cout << endl;
80             }
81         }
82
83         void printWords() {
84             for (const auto &node : nodes) {
85                 if (node.wordCnt != 0) {
86                     cout << "Word ends at index " << node.wordCnt << endl;
87                 }
88             }
89         }
90
91         void printPfxCnt() {
92             for (const auto &node : nodes) {
93                 cout << "PfxCnt at index " << node.pfxCnt << endl;
94             }
95         }
96
97         void printWordCnt() {
98             for (const auto &node : nodes) {
99                 cout << "WordCnt at index " << node.wordCnt << endl;
100            }
101        }
102
103        void printNext() {
104            for (const auto &node : nodes) {
105                cout << "Next at index " << node.next << endl;
106            }
107        }
108
109        void printMaxChars() {
110            cout << "MaxChars: " << maxChars << endl;
111        }
112
113        void printNodes() {
114            for (const auto &node : nodes) {
115                cout << "Node: " << node << endl;
116            }
117        }
118
119        void printAll() {
120            printTrie();
121            printWords();
122            printPfxCnt();
123            printWordCnt();
124            printNext();
125            printMaxChars();
126            printNodes();
127        }
128
129        void printBase() {
130            cout << "Base: " << getBase('A') << endl;
131        }
132
133        void printGetBase() {
134            cout << "GetBase: " << getBase('A') << endl;
135        }
136
137        void printDistWords() {
138            cout << "DistWords: " << distWords << endl;
139        }
140
141        void printMaxChars() {
142            cout << "MaxChars: " << maxChars << endl;
143        }
144
145        void printWordCnt() {
146            cout << "WordCnt: " << wordCnt << endl;
147        }
148
149        void printPfxCnt() {
150            cout << "PfxCnt: " << pfxCnt << endl;
151        }
152
153        void printNext() {
154            cout << "Next: " << next << endl;
155        }
156
157        void printWordEnd() {
158            cout << "WordEnd: " << wordEnd << endl;
159        }
160
161        void printPfxEnd() {
162            cout << "PfxEnd: " << pfxEnd << endl;
163        }
164
165        void printWordStart() {
166            cout << "WordStart: " << wordStart << endl;
167        }
168
169        void printPfxStart() {
170            cout << "PfxStart: " << pfxStart << endl;
171        }
172
173        void printWordMiddle() {
174            cout << "WordMiddle: " << wordMiddle << endl;
175        }
176
177        void printPfxMiddle() {
178            cout << "PfxMiddle: " << pfxMiddle << endl;
179        }
180
181        void printWordEnd() {
182            cout << "WordEnd: " << wordEnd << endl;
183        }
184
185        void printPfxEnd() {
186            cout << "PfxEnd: " << pfxEnd << endl;
187        }
188
189        void printWordStart() {
190            cout << "WordStart: " << wordStart << endl;
191        }
192
193        void printPfxStart() {
194            cout << "PfxStart: " << pfxStart << endl;
195        }
196
197        void printWordMiddle() {
198            cout << "WordMiddle: " << wordMiddle << endl;
199        }
200
201        void printPfxMiddle() {
202            cout << "PfxMiddle: " << pfxMiddle << endl;
203        }
204
205        void printWordEnd() {
206            cout << "WordEnd: " << wordEnd << endl;
207        }
208
209        void printPfxEnd() {
210            cout << "PfxEnd: " << pfxEnd << endl;
211        }
212
213        void printWordStart() {
214            cout << "WordStart: " << wordStart << endl;
215        }
216
217        void printPfxStart() {
218            cout << "PfxStart: " << pfxStart << endl;
219        }
220
221        void printWordMiddle() {
222            cout << "WordMiddle: " << wordMiddle << endl;
223        }
224
225        void printPfxMiddle() {
226            cout << "PfxMiddle: " << pfxMiddle << endl;
227        }
228
229        void printWordEnd() {
230            cout << "WordEnd: " << wordEnd << endl;
231        }
232
233        void printPfxEnd() {
234            cout << "PfxEnd: " << pfxEnd << endl;
235        }
236
237        void printWordStart() {
238            cout << "WordStart: " << wordStart << endl;
239        }
240
241        void printPfxStart() {
242            cout << "PfxStart: " << pfxStart << endl;
243        }
244
245        void printWordMiddle() {
246            cout << "WordMiddle: " << wordMiddle << endl;
247        }
248
249        void printPfxMiddle() {
250            cout << "PfxMiddle: " << pfxMiddle << endl;
251        }
252
253        void printWordEnd() {
254            cout << "WordEnd: " << wordEnd << endl;
255        }
256
257        void printPfxEnd() {
258            cout << "PfxEnd: " << pfxEnd << endl;
259        }
260
261        void printWordStart() {
262            cout << "WordStart: " << wordStart << endl;
263        }
264
265        void printPfxStart() {
266            cout << "PfxStart: " << pfxStart << endl;
267        }
268
269        void printWordMiddle() {
270            cout << "WordMiddle: " << wordMiddle << endl;
271        }
272
273        void printPfxMiddle() {
274            cout << "PfxMiddle: " << pfxMiddle << endl;
275        }
276
277        void printWordEnd() {
278            cout << "WordEnd: " << wordEnd << endl;
279        }
280
281        void printPfxEnd() {
282            cout << "PfxEnd: " << pfxEnd << endl;
283        }
284
285        void printWordStart() {
286            cout << "WordStart: " << wordStart << endl;
287        }
288
289        void printPfxStart() {
290            cout << "PfxStart: " << pfxStart << endl;
291        }
292
293        void printWordMiddle() {
294            cout << "WordMiddle: " << wordMiddle << endl;
295        }
296
297        void printPfxMiddle() {
298            cout << "PfxMiddle: " << pfxMiddle << endl;
299        }
300
301        void printWordEnd() {
302            cout << "WordEnd: " << wordEnd << endl;
303        }
304
305        void printPfxEnd() {
306            cout << "PfxEnd: " << pfxEnd << endl;
307        }
308
309        void printWordStart() {
310            cout << "WordStart: " << wordStart << endl;
311        }
312
313        void printPfxStart() {
314            cout << "PfxStart: " << pfxStart << endl;
315        }
316
317        void printWordMiddle() {
318            cout << "WordMiddle: " << wordMiddle << endl;
319        }
320
321        void printPfxMiddle() {
322            cout << "PfxMiddle: " << pfxMiddle << endl;
323        }
324
325        void printWordEnd() {
326            cout << "WordEnd: " << wordEnd << endl;
327        }
328
329        void printPfxEnd() {
330            cout << "PfxEnd: " << pfxEnd << endl;
331        }
332
333        void printWordStart() {
334            cout << "WordStart: " << wordStart << endl;
335        }
336
337        void printPfxStart() {
338            cout << "PfxStart: " << pfxStart << endl;
339        }
340
341        void printWordMiddle() {
342            cout << "WordMiddle: " << wordMiddle << endl;
343        }
344
345        void printPfxMiddle() {
346            cout << "PfxMiddle: " << pfxMiddle << endl;
347        }
348
349        void printWordEnd() {
350            cout << "WordEnd: " << wordEnd << endl;
351        }
352
353        void printPfxEnd() {
354            cout << "PfxEnd: " << pfxEnd << endl;
355        }
356
357        void printWordStart() {
358            cout << "WordStart: " << wordStart << endl;
359        }
360
361        void printPfxStart() {
362            cout << "PfxStart: " << pfxStart << endl;
363        }
364
365        void printWordMiddle() {
366            cout << "WordMiddle: " << wordMiddle << endl;
367        }
368
369        void printPfxMiddle() {
370            cout << "PfxMiddle: " << pfxMiddle << endl;
371        }
372
373        void printWordEnd() {
374            cout << "WordEnd: " << wordEnd << endl;
375        }
376
377        void printPfxEnd() {
378            cout << "PfxEnd: " << pfxEnd << endl;
379        }
380
381        void printWordStart() {
382            cout << "WordStart: " << wordStart << endl;
383        }
384
385        void printPfxStart() {
386            cout << "PfxStart: " << pfxStart << endl;
387        }
388
389        void printWordMiddle() {
390            cout << "WordMiddle: " << wordMiddle << endl;
391        }
392
393        void printPfxMiddle() {
394            cout << "PfxMiddle: " << pfxMiddle << endl;
395        }
396
397        void printWordEnd() {
398            cout << "WordEnd: " << wordEnd << endl;
399        }
400
401        void printPfxEnd() {
402            cout << "PfxEnd: " << pfxEnd << endl;
403        }
404
405        void printWordStart() {
406            cout << "WordStart: " << wordStart << endl;
407        }
408
409        void printPfxStart() {
410            cout << "PfxStart: " << pfxStart << endl;
411        }
412
413        void printWordMiddle() {
414            cout << "WordMiddle: " << wordMiddle << endl;
415        }
416
417        void printPfxMiddle() {
418            cout << "PfxMiddle: " << pfxMiddle << endl;
419        }
420
421        void printWordEnd() {
422            cout << "WordEnd: " << wordEnd << endl;
423        }
424
425        void printPfxEnd() {
426            cout << "PfxEnd: " << pfxEnd << endl;
427        }
428
429        void printWordStart() {
430            cout << "WordStart: " << wordStart << endl;
431        }
432
433        void printPfxStart() {
434            cout << "PfxStart: " << pfxStart << endl;
435        }
436
437        void printWordMiddle() {
438            cout << "WordMiddle: " << wordMiddle << endl;
439        }
440
441        void printPfxMiddle() {
442            cout << "PfxMiddle: " << pfxMiddle << endl;
443        }
444
445        void printWordEnd() {
446            cout << "WordEnd: " << wordEnd << endl;
447        }
448
449        void printPfxEnd() {
450            cout << "PfxEnd: " << pfxEnd << endl;
451        }
452
453        void printWordStart() {
454            cout << "WordStart: " << wordStart << endl;
455        }
456
457        void printPfxStart() {
458            cout << "PfxStart: " << pfxStart << endl;
459        }
460
461        void printWordMiddle() {
462            cout << "WordMiddle: " << wordMiddle << endl;
463        }
464
465        void printPfxMiddle() {
466            cout << "PfxMiddle: " << pfxMiddle << endl;
467        }
468
469        void printWordEnd() {
470            cout << "WordEnd: " << wordEnd << endl;
471        }
472
473        void printPfxEnd() {
474            cout << "PfxEnd: " << pfxEnd << endl;
475        }
476
477        void printWordStart() {
478            cout << "WordStart: " << wordStart << endl;
479        }
480
481        void printPfxStart() {
482            cout << "PfxStart: " << pfxStart << endl;
483        }
484
485        void printWordMiddle() {
486            cout << "WordMiddle: " << wordMiddle << endl;
487        }
488
489        void printPfxMiddle() {
490            cout << "PfxMiddle: " << pfxMiddle << endl;
491        }
492
493        void printWordEnd() {
494            cout << "WordEnd: " << wordEnd << endl;
495        }
496
497        void printPfxEnd() {
498            cout << "PfxEnd: " << pfxEnd << endl;
499        }
500
501        void printWordStart() {
502            cout << "WordStart: " << wordStart << endl;
503        }
504
505        void printPfxStart() {
506            cout << "PfxStart: " << pfxStart << endl;
507        }
508
509        void printWordMiddle() {
510            cout << "WordMiddle: " << wordMiddle << endl;
511        }
512
513        void printPfxMiddle() {
514            cout << "PfxMiddle: " << pfxMiddle << endl;
515        }
516
517        void printWordEnd() {
518            cout << "WordEnd: " << wordEnd << endl;
519        }
520
521        void printPfxEnd() {
522            cout << "PfxEnd: " << pfxEnd << endl;
523        }
524
525        void printWordStart() {
526            cout << "WordStart: " << wordStart << endl;
527        }
528
529        void printPfxStart() {
530            cout << "PfxStart: " << pfxStart << endl;
531        }
532
533        void printWordMiddle() {
534            cout << "WordMiddle: " << wordMiddle << endl;
535        }
536
537        void printPfxMiddle() {
538            cout << "PfxMiddle: " << pfxMiddle << endl;
539        }
540
541        void printWordEnd() {
542            cout << "WordEnd: " << wordEnd << endl;
543        }
544
545        void printPfxEnd() {
546            cout << "PfxEnd: " << pfxEnd << endl;
547        }
548
549        void printWordStart() {
550            cout << "WordStart: " << wordStart << endl;
551        }
552
553        void printPfxStart() {
554            cout << "PfxStart: " << pfxStart << endl;
555        }
556
557        void printWordMiddle() {
558            cout << "WordMiddle: " << wordMiddle << endl;
559        }
560
561        void printPfxMiddle() {
562            cout << "PfxMiddle: " << pfxMiddle << endl;
563        }
564
565        void printWordEnd() {
566            cout << "WordEnd: " << wordEnd << endl;
567        }
568
569        void printPfxEnd() {
570            cout << "PfxEnd: " << pfxEnd << endl;
571        }
572
573        void printWordStart() {
574            cout << "WordStart: " << wordStart << endl;
575        }
576
577        void printPfxStart() {
578            cout << "PfxStart: " << pfxStart << endl;
579        }
580
581        void printWordMiddle() {
582            cout << "WordMiddle: " << wordMiddle << endl;
583        }
584
585        void printPfxMiddle() {
586            cout << "PfxMiddle: " << pfxMiddle << endl;
587        }
588
589        void printWordEnd() {
590            cout << "WordEnd: " << wordEnd << endl;
591        }
592
593        void printPfxEnd() {
594            cout << "PfxEnd: " << pfxEnd << endl;
595        }
596
597        void printWordStart() {
598            cout << "WordStart: " << wordStart << endl;
599        }
600
601        void printPfxStart() {
602            cout << "PfxStart: " << pfxStart << endl;
603        }
604
605        void printWordMiddle() {
606            cout << "WordMiddle: " << wordMiddle << endl;
607        }
608
609        void printPfxMiddle() {
610            cout << "PfxMiddle: " << pfxMiddle << endl;
611        }
612
613        void printWordEnd() {
614            cout << "WordEnd: " << wordEnd << endl;
615        }
616
617        void printPfxEnd() {
618            cout << "PfxEnd: " << pfxEnd << endl;
619        }
620
621        void printWordStart() {
622            cout << "WordStart: " << wordStart << endl;
623        }
624
625        void printPfxStart() {
626            cout << "PfxStart: " << pfxStart << endl;
627        }
628
629        void printWordMiddle() {
630            cout << "WordMiddle: " << wordMiddle << endl;
631        }
632
633        void printPfxMiddle() {
634            cout << "PfxMiddle: " << pfxMiddle << endl;
635        }
636
637        void printWordEnd() {
638            cout << "WordEnd: " << wordEnd << endl;
639        }
640
641        void printPfxEnd() {
642            cout << "PfxEnd: " << pfxEnd << endl;
643        }
644
645        void printWordStart() {
646            cout << "WordStart: " << wordStart << endl;
647        }
648
649        void printPfxStart() {
650            cout << "PfxStart: " << pfxStart << endl;
651        }
652
653        void printWordMiddle() {
654            cout << "WordMiddle: " << wordMiddle << endl;
655        }
656
657        void printPfxMiddle() {
658            cout << "PfxMiddle: " << pfxMiddle << endl;
659        }
660
661        void printWordEnd() {
662            cout << "WordEnd: " << wordEnd << endl;
663        }
664
665        void printPfxEnd() {
666            cout << "PfxEnd: " << pfxEnd << endl;
667        }
668
669        void printWordStart() {
670            cout << "WordStart: " << wordStart << endl;
671        }
672
673        void printPfxStart() {
674            cout << "PfxStart: " << pfxStart << endl;
675        }
676
677        void printWordMiddle() {
678            cout << "WordMiddle: " << wordMiddle << endl;
679        }
680
681        void printPfxMiddle() {
682            cout << "PfxMiddle: " << pfxMiddle << endl;
683        }
684
685        void printWordEnd() {
686            cout << "WordEnd: " << wordEnd << endl;
687        }
688
689        void printPfxEnd() {
690            cout << "PfxEnd: " << pfxEnd << endl;
691        }
692
693        void printWordStart() {
694            cout << "WordStart: " << wordStart << endl;
695        }
696
697        void printPfxStart() {
698            cout << "PfxStart: " << pfxStart << endl;
699        }
700
701        void printWordMiddle() {
702            cout << "WordMiddle: " << wordMiddle << endl;
703        }
704
705        void printPfxMiddle() {
706            cout << "PfxMiddle: " << pfxMiddle << endl;
707        }
708
709        void printWordEnd() {
710            cout << "WordEnd: " << wordEnd << endl;
711        }
712
713        void printPfxEnd() {
714            cout << "PfxEnd: " << pfxEnd << endl;
715        }
716
717        void printWordStart() {
718            cout << "WordStart: " << wordStart << endl;
719        }
720
721        void printPfxStart() {
722            cout << "PfxStart: " << pfxStart << endl;
723        }
724
725        void printWordMiddle() {
726            cout << "WordMiddle: " << wordMiddle << endl;
727        }
728
729        void printPfxMiddle() {
730            cout << "PfxMiddle: " << pfxMiddle << endl;
731        }
732
733        void printWordEnd() {
734            cout << "WordEnd: " << wordEnd << endl;
735        }
736
737        void printPfxEnd() {
738            cout << "PfxEnd: " << pfxEnd << endl;
739        }
740
741        void printWordStart() {
742            cout << "WordStart: " << wordStart << endl;
743        }
744
745        void printPfxStart() {
746            cout << "PfxStart: " << pfxStart << endl;
747        }
748
749        void printWordMiddle() {
750            cout << "WordMiddle: " << wordMiddle << endl;
751        }
752
753        void printPfxMiddle() {
754            cout << "PfxMiddle: " << pfxMiddle << endl;
755        }
756
757        void printWordEnd() {
758            cout << "WordEnd: " << wordEnd << endl;
759        }
760
761        void printPfxEnd() {
762            cout << "PfxEnd: " << pfxEnd << endl;
763        }
764
765        void printWordStart() {
766            cout << "WordStart: " << wordStart << endl;
767        }
768
769        void printPfxStart() {
770            cout << "PfxStart: " << pfxStart << endl;
771        }
772
773        void printWordMiddle() {
774            cout << "WordMiddle: " << wordMiddle << endl;
775        }
776
777        void printPfxMiddle() {
778            cout << "PfxMiddle: " << pfxMiddle << endl;
779        }
780
781        void printWordEnd() {
782            cout << "WordEnd: " << wordEnd << endl;
783        }
784
785        void printPfxEnd() {
786            cout << "PfxEnd: " << pfxEnd << endl;
787        }
788
789        void printWordStart() {
790            cout << "WordStart: " << wordStart << endl;
791        }
792
793        void printPfxStart() {
794            cout << "PfxStart: " << pfxStart << endl;
795        }
796
797        void printWordMiddle() {
798            cout << "WordMiddle: " << wordMiddle << endl;
799        }
800
801        void printPfxMiddle() {
802            cout << "PfxMiddle: " << pfxMiddle << endl;
803        }
804
805        void printWordEnd() {
806            cout << "WordEnd: " << wordEnd << endl;
807        }
808
809        void printPfxEnd() {
810            cout << "PfxEnd: " << pfxEnd << endl;
811        }
812
813        void printWordStart() {
814            cout << "WordStart: " << wordStart << endl;
815        }
816
817        void printPfxStart() {
818            cout << "PfxStart: " << pfxStart << endl;
819        }
820
821        void printWordMiddle() {
822            cout << "WordMiddle: " << wordMiddle << endl;
823        }
824
825        void printPfxMiddle() {
826            cout << "PfxMiddle: " << pfxMiddle << endl;
827        }
828
829        void printWordEnd() {
830            cout << "WordEnd: " << wordEnd << endl;
831        }
832
833        void printPfxEnd() {
834            cout << "PfxEnd: " << pfxEnd << endl;
835        }
836
837        void printWordStart() {
838            cout << "WordStart: " << wordStart << endl;
839        }
840
841        void printPfxStart() {
842            cout << "PfxStart: " << pfxStart << endl;
843        }
844
845        void printWordMiddle() {
846            cout << "WordMiddle: " << wordMiddle << endl;
847        }
848
849        void printPfxMiddle() {
850            cout << "PfxMiddle: " << pfxMiddle << endl;
851        }
852
853        void printWordEnd() {
854            cout << "WordEnd: " << wordEnd << endl;
855        }
856
857        void printPfxEnd() {
858            cout << "PfxEnd: " << pfxEnd << endl;
859        }
860
861        void printWordStart() {
862            cout << "WordStart: " << wordStart << endl;
863        }
864
865        void printPfxStart() {
866            cout << "PfxStart: " << pfxStart << endl;
867        }
868
869        void printWordMiddle() {
870            cout << "WordMiddle: " << wordMiddle << endl;
871        }
872
873        void printPfxMiddle() {
874            cout << "PfxMiddle: " << pfxMiddle << endl;
875        }
876
877        void printWordEnd() {
878            cout << "WordEnd: " << wordEnd << endl;
879        }
880
881        void printPfxEnd() {
882            cout << "PfxEnd: " << pfxEnd << endl;
883        }
884
885        void printWordStart() {
886            cout << "WordStart: " << wordStart << endl;
887        }
888
889        void printPfxStart() {
890            cout << "PfxStart: " << pfxStart << endl;
891        }
892
893        void printWordMiddle() {
894            cout << "WordMiddle: " << wordMiddle << endl;
895        }
896
897        void printPfxMiddle() {
898            cout << "PfxMiddle: " << pfxMiddle << endl;
899        }
900
901        void printWordEnd() {
902            cout << "WordEnd: " << wordEnd << endl;
903        }
904
905        void printPfxEnd() {
906            cout << "PfxEnd: " << pfxEnd << endl;
907        }
908
909        void printWordStart() {
910            cout << "WordStart: " << wordStart << endl;
911        }
912
913        void printPfxStart() {
914            cout << "PfxStart: " << pfxStart << endl;
915        }
916
917        void printWordMiddle() {
918            cout << "WordMiddle: " << wordMiddle << endl;
919        }
920
921        void printPfxMiddle() {
922            cout << "PfxMiddle: " <&lt
```

```

55   for (char &ch : s) {
56     int base = getBase(ch);
57     if (nodes[curr].next[base] == -1) return false;
58     curr = nodes[curr].next[base];
59   }
60   return nodes[curr].wordCnt > 0;
61 }
62 // Delete one occurrence of s
63 void erase(string s) {
64   if (!search(s)) return; // Check existence first
65   int curr = 0;
66   nodes[curr].pfxCnt--;
67   for (char &ch : s) {
68     int base = getBase(ch);
69     curr = nodes[curr].next[base];
70     nodes[curr].pfxCnt--;
71   }
72   nodes[curr].wordCnt--;
73   if (nodes[curr].wordCnt == 0)
74     distWords--; // Word completely removed
75 }
76
77 // Count words that have s as a prefix
78 int prefixCount(string s) {
79   int curr = 0;
80   for (char &ch : s) {
81     int base = getBase(ch);

```

```

81     if (nodes[curr].next[base] == -1)
82       return 0; // Prefix not found
83     curr = nodes[curr].next[base];
84   }
85   return nodes[curr].pfxCnt;
86 }
87 }

```

6.5. Aho-Corasick Automaton

```

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

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

```

```

19 // insert
20 for (auto &pat : P) {
21     int u = 0;
22     for (char ch : pat) {
23         int c = ch - 'a';
24         if (!nxt[u][c]) nxt[u][c] = nodes++;
25         u = nxt[u][c];
26     }
27     patEnd.pb(u);
28 }
29 }
30 vector<int> bfsOrder;
31 void build_links() {
32     queue<int> q;
33     linkS[0] = 0;
34     //first layer
35     for (int c = 0; c < ALPHA; c++) {
36         int v = nxt[0][c];
37         if (v) {
38             linkS[v] = 0;
39             q.push(v);
40         }
41     }
42     //BFS
43     while (!q.empty()) {
44         int u = q.front();
45         q.pop();
46     }
47     bfsOrder.pb(u);
48     for (int c = 0; c < ALPHA; c++) {
49         int v = nxt[u][c];
50         if (!v) continue;
51         int j = linkS[u];
52         while (j && !nxt[j][c]) j = linkS[j];
53         if (nxt[j][c]) j = nxt[j][c];
54         linkS[v] = j;
55         q.push(v);
56     }
57     for (int u : bfsOrder) { adjSL[linkS[u]].pb(u); }
58 }
59
60 void solve() {
61     string S;
62     ll k;
63     cin >> S >> k;
64     vector<string> P(k);
65     for (int i = 0; i < k; i++) cin >> P[i];
66     build_trie(P);
67     bfsOrder.clear();
68     build_links();
69 }

```

7. Debug List

- 1 - 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.

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

- 15 - 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?

- | | |
|----|---|
| 27 | <ul style="list-style-type: none"> - Are you sure about the STL thing you are using? - Read cppreference (should be available). - Print everything and run it on pen and paper. |
| 29 | <ul style="list-style-type: none"> - Time Limit Exceeded: <ul style="list-style-type: none"> - Calculate your time complexity again. - Does the program actually end? <ul style="list-style-type: none"> - Check for `while(q.size())` etc. - Test the largest cases locally. - Did you do unnecessary stuff? <ul style="list-style-type: none"> - e.g. pass vectors by value - e.g. `memset` for every test case - Is your constant factor reasonable? |
| 31 | <ul style="list-style-type: none"> - Runtime Error: <ul style="list-style-type: none"> - Check memory usage. <ul style="list-style-type: none"> - Forget to clear or destroy stuff? <ul style="list-style-type: none"> -> `vector::shrink_to_fit()` - Stack overflow? - Bad pointer / array access? <ul style="list-style-type: none"> - Try `"-fsanitize=address` - Division by zero? NaN's? |
| 33 | <ul style="list-style-type: none"> - |
| 35 | <ul style="list-style-type: none"> - |
| 37 | <ul style="list-style-type: none"> - |
| 39 | <ul style="list-style-type: none"> - |
| 41 | <ul style="list-style-type: none"> - |
| 43 | <ul style="list-style-type: none"> - |
| 45 | <ul style="list-style-type: none"> - |
| 47 | <ul style="list-style-type: none"> - |