Contents

1 Basic Template

1.1 C++

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

#define MAX 200100
#define MOD 1000000007
#define INF 0x7f7f7f7f7f7f7f7f
#define endl '\n'

using namespace std;
typedef pair<int, int> pr;
typedef array<int, 3> tp;

signed main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);

    return 0;
}
```

1.2 Python

```python
from sys import stdin

input = stdin.readline
```

2 Data Structure

2.1 Segment Tree

```cpp
class SegTree {
  public:
    int N;
    vector<int> arr, tree;

    SegTree(int n) : N(n), arr(n + 1, 0), tree(n * 2 + 1, 0) {}

    void init() {
        for (int i = 1; i <= N; i++)
            tree[i + N - 1] = arr[i];

        for (int i = N - 1; i > 0; --i)
            tree[i] = tree[i << 1] + tree[i << 1 | 1];
    }

    int query(int l, int r) {
        int res = 0;
        for (l += N - 1, r += N; l < r; l >>= 1, r >>= 1) {
            if (l & 1)
                res += tree[l++];
            if (r & 1)
                res += tree[--r];
        }
        return res;
    }

    int query(int pos) { return query(pos, pos); }
    void update(int pos, int val) {
        for (tree[pos += N - 1] += val; pos > 1; pos >>= 1)
            tree[pos >> 1] = tree[pos] + tree[pos ^ 1];
    }
};
```

```
class SegTree {
  private:
    void init(int n, int s, int e) {
        if (s == e)
            tree[n] = arr[s];
        else {
            init(n << 1, s, (s + e) >> 1);
            init(n << 1 | 1, ((s + e) >> 1) + 1, e);
            tree[n] = tree[n << 1] + tree[n << 1 | 1];
        }
    }

    int query(int n, int s, int e, int l, int r) {
        if (l <= s && e <= r)
            return tree[n];
        else if (r < s || e < l)
            return 0;
        else {
            int lv = query(n << 1, s, ((s + e) >> 1), l, r);
            int rv = query(n << 1 | 1, ((s + e) >> 1) + 1, e, l, r);
            return lv + rv;
        }
    }

    void update(int n, int s, int e, int pos, int val) {
        if (pos < s || e < pos)
            return;
        else if (s == e) {
            tree[n] = val;
            arr[s] = val;
        } else {
            update(n << 1, s, (s + e) >> 1, pos, val);
            update(n << 1 | 1, ((s + e) >> 1) + 1, e, pos, val);
            tree[n] = tree[n << 1] + tree[n << 1 | 1];
        }
    }

  public:
    int N;
    vector<int> arr;
    vector<int> tree;

    SegTree(int n) : N(n), arr(n + 1), tree(n * 4 + 1) {}

    void init() { init(1, 1, N); }
    int query(int pos) { return query(1, 1, N, pos, pos); }
    int query(int l, int r) { return query(1, 1, N, l, r); }
    void update(int pos, int val) { update(1, 1, N, pos, val); }
};
```

2.2 Lazy Propagation

```
class LazyPropagition {
  private:
    void init(int n, int s, int e) {
        if (s == e)
            tree[n] = arr[s];
        else {
            init(n << 1, s, (s + e) >> 1);
            init(n << 1 | 1, ((s + e) >> 1) + 1, e);
            tree[n] = tree[n << 1] + tree[n << 1 | 1];
        }
    }

    void lazy_update(int n, int s, int e) {
        if (lazy[n] != 0) {
            tree[n] += (e - s + 1) * lazy[n];
            if (s != e) {
                lazy[n << 1] += lazy[n];
                lazy[n << 1 | 1] += lazy[n];
            }
            lazy[n] = 0;
        }
    }

    int query(int n, int s, int e, int l, int r) {
        lazy_update(n, s, e);
        if (l <= s && e <= r)
            return tree[n];
        else if (r < s || e < l)
            return 0;
        else {
            int lv = query(n << 1, s, ((s + e) >> 1), l, r);
            int rv = query(n << 1 | 1, ((s + e) >> 1) + 1, e, l, r);
            return lv + rv;
        }
    }

    void update(int n, int s, int e, int l, int r, int val) {
        lazy_update(n, s, e);
        if (r < s || e < l)
            return;
        else if (l <= s && e <= r) {
            lazy[n] += val;
            lazy_update(n, s, e);
            arr[s] += val;
        } else {
            update(n << 1, s, (s + e) >> 1, l, r, val);
            update(n << 1 | 1, ((s + e) >> 1) + 1, e, l, r, val);
            tree[n] = tree[n << 1] + tree[n << 1 | 1];
        }
    }

  public:
    int N, arr[MAX];
    vector<int> arr, tree, lazy;

    LazyPropagition(int n) : N(n), arr(n + 1, 0), tree(4 * n + 1, 0), lazy(4 * n
       + 1, 0) {}
```

```
    void init() { init(1, 1, N); }
    int query(int pos) { return query(1, 1, N, pos, pos); }
    int query(int l, int r) { return query(1, 1, N, l, r); }
    void update(int l, int r, int val) { update(1, 1, N, l, r, val); }
};
```

2.3 Merge Sort Tree

```
class MergeSortTree {
  public:
    int N;
    vector<int> arr;
    vector<vector<int>> tree;

    MergeSortTree(int n) : N(n), arr(n + 1), tree(n * 2 + 1) {}

    void init() {
        for (int i = 1; i <= N; i++)
            tree[i + N - 1].push_back(arr[i]);

        for (int i = N - 1; i > 0; --i) {
            tree[i].resize(tree[i << 1].size() + tree[i << 1 | 1].size());
            merge(tree[i << 1].begin(), tree[i << 1].end(), tree[i << 1 | 1].
              begin(), tree[i << 1 | 1].end(), tree[i].begin());
        }
    }

    int query(int l, int r, int k) {
        int res = 0;
        for (l += N - 1, r += N; l < r; l >>= 1, r >>= 1) {
            if (l & 1)
                res += tree[l].end() - upper_bound(tree[l].begin(), tree[l].end
                  (), k), l++;
            if (r & 1)
                --r, res += tree[r].end() - upper_bound(tree[r].begin(), tree[r
                  ].end(), k);
        }
        return res;
    }
    int query(int pos, int k) { return query(pos, pos, k); }
};
```

2.4 SQRT Decomposition

```
class SqrtDecomposition {
  public:
    int N, S, size;
    vector<int> arr, sqrt_arr;

    SqrtDecomposition(int n) : N(n), arr(n + 1), sqrt_arr(n + 1) {}

    void init() {
        S = sqrt(N);
        size = N / S;
        if (N % S)
            size++;

        for (int i = 1; i <= N; i++)
            sqrt_arr[i / S] += arr[i];
    }

    void update(int pos, int val) {
        sqrt_arr[pos / S] += val - arr[pos];
        arr[pos] = val;
    }

    int query(int l, int r) {
        int res = 0;
        for (; l % S && l <= r; l++)
            res += arr[l];
        for (; (r + 1) % S && l <= r; r--)
            res += arr[r];
        for (; l <= r; l += S)
            res += sqrt_arr[l / S];
        return res;
    }
};
```

3 Graph

3.1 Dijkstra

```
int V;
vector<pr> arr[MAX];

vector<int> get_dis(int K) {
    vector<int> dis(V + 1, LLONG_MAX);
    priority_queue<pr, vector<pr>, greater<pr>> pq;
    pq.push({0, K});
    dis[K] = 0;

    while (pq.size()) {
        pr p = pq.top();
        pq.pop();

        if (dis[p.second] != p.first)
            continue;

        for (pr i : arr[p.second]) {
            if (i.first + p.first < dis[i.second]) {
                dis[i.second] = i.first + p.first;
                pq.push({dis[i.second], i.second});
            }
        }
    }

    return dis;
```

```
}
```

3.2 Kosaraju

```
vector<int> arr[MAX], rvt_arr[MAX], scc[MAX];
stack<int> st;
bool checked[MAX];
int scc_num = 0, scc_id[MAX];

void kosaraju_first_dfs(int K) {
    checked[K] = true;
    for (int i : arr[K]) {
        if (!checked[i])
            kosaraju_first_dfs(i);
    }
    st.push(K);
}

void kosaraju_second_dfs(int K) {
    checked[K] = true;
    for (int i : rvt_arr[K]) {
        if (!checked[i])
            kosaraju_second_dfs(i);
    }
    scc[scc_num].push_back(K);
    scc_id[K] = scc_num;
}

void kosaraju(int V) {
    int A;
    for (int i = 1; i <= V; i++) {
        if (!checked[i])
            kosaraju_first_dfs(i);
    }
    fill(checked, checked + V + 1, false);
    while (!st.empty()) {
        A = st.top();
        st.pop();
        if (!checked[A]) {
            kosaraju_second_dfs(A);
            scc_num++;
        }
    }
}
```

3.3 Floyd Warshall

```
int V;
int dis[MAX][MAX];

void get_dis() {
    for (int i = 1; i <= V; i++)
        dis[i][i] = 0;
    for (int i = 1; i <= V; i++) {
        for (int j = 1; j <= V; j++) {
            for (int k = 1; k <= V; k++)
                dis[j][k] = min(dis[j][k], dis[j][i] + dis[i][k]);
        }
    }
}
```

3.4 Centroid Decomposition

```
vector<int> arr[MAX];
int sz[MAX], C[MAX];
bool checked[MAX];

int get_size(int node, int parent) {
    sz[node] = 1;
    for (int i : arr[node]) {
        if (i == parent || checked[i])
            continue;
        sz[node] += get_size(i, node);
    }
    return sz[node];
}

int get_centroid(int node, int parent, int cap) {
    for (int i : arr[node]) {
        if (i == parent || checked[i])
            continue;
        if (sz[i] * 2 > cap)
            return get_centroid(i, node, cap);
    }
    return node;
}

int get_res(int node, int parent) {
    // 분할정복
}

int divide_and_conquer(int node) {
    get_size(node, -1);
    int res = LLONG_MAX, cent = get_centroid(node, -1, sz[node]);
    checked[cent] = true;

    for (int i : arr[cent]) {
        if (checked[i])
            continue;
    }

    for (int i : arr[cent]) {
        if (checked[i])
            continue;
        res = min(res, divide_and_conquer(i));
    }
    return res;
}
```

3.5 Heavy Light Decomposition

```
class HLD {
  private:
    int pv;
    vector<bool> checked;

    void dfs(int cur) {
        in[cur] = ++pv;
        seg.arr[pv] = val[cur];
        for (int i : child[cur]) {
            top[i] = i == child[cur][0] ? top[cur] : i;
            dfs(i);
        }
        out[cur] = pv;
    }

  public:
    SegTree seg;

    vector<vector<int>> arr, child;
    vector<int> parent, depth, sz, top, in, out, val;
    int N, root;

    HLD(int n, int rt = 1) : N(n), pv(0), seg(n), root(rt), arr(n + 1), child(n
      + 1), parent(n + 1), depth(n + 1), sz(n + 1), top(n + 1), in(n + 1), out(n
       + 1), val(n + 1), checked(n + 1) {}

    void add_edge(int u, int v) {
        arr[u].push_back(v);
        arr[v].push_back(u);
    }

    void set_node(int n, int v) {
        val[n] = v;
    }

    void build_tree() {
        int cur;

        queue<int> q;
        stack<int> st;

        q.push(root);
        checked[root] = true;

        while (!q.empty()) {
            cur = q.front(), q.pop();
            st.push(cur);

            for (int i : arr[cur]) {
                if (checked[i])
                    continue;
                checked[i] = true;
                parent[i] = cur;
                depth[i] = depth[cur] + 1;
                child[cur].push_back(i);
                q.push(i);
            }
        }

        while (!st.empty()) {
            cur = st.top(), st.pop();
            sz[cur] = 1;
            for (int i = 0; i < child[cur].size(); i++) {
                sz[cur] += sz[child[cur][i]];
                if (sz[child[cur][i]] > sz[child[cur][0]])
                    swap(child[cur][i], child[cur][0]);
            }
        }

        dfs(root);
    }

    void build_seg() {
        for (int i = 1; i <= N; i++)
            seg.arr[in[i]] = val[i];
        seg.init();
    }

    void update(int pos, int val) {
        seg.update(in[pos], val);
    }

    int query(int u, int v) {
        int res = 0;
        while (top[u] ^ top[v]) {
            if (depth[top[u]] < depth[top[v]])
                swap(u, v);
            res = res + seg.query(in[top[u]], in[u]);
            u = parent[top[u]];
        }
        if (depth[u] > depth[v])
            swap(u, v);
        res = res + seg.query(in[u] + 1, in[v]);
        return res;
    }

    int lca(int u, int v) {
        while (top[u] ^ top[v]) {
            if (depth[top[u]] < depth[top[v]])
                swap(u, v);
            u = parent[top[u]];
        }
        return depth[u] < depth[v] ? u : v;
    }
};

class LazyHLD {
  private:
```

```
    int pv;
    bool checked[MAX];

    void dfs(int cur) {
        in[cur] = ++pv;
        seg.arr[pv] = val[cur];
        for (int i : child[cur]) {
            top[i] = i == child[cur][0] ? top[cur] : i;
            dfs(i);
        }
        out[cur] = pv;
    }

public:
    LazyPropagition seg;

    vector<int> arr[MAX], child[MAX];
    int N, root, parent[MAX], depth[MAX], sz[MAX], top[MAX], in[MAX], out[MAX],
      val[MAX];

    LazyHLD(int n, int rt = 1) : N(n), pv(0), seg(n), root(rt) {}

    void add_edge(int u, int v) {
        arr[u].push_back(v);
        arr[v].push_back(u);
    }

    void set_node(int n, int v) {
        val[n] = v;
    }

    void build_tree() {
        int cur;

        queue<int> q;
        stack<int> st;

        q.push(root);
        checked[root] = true;

        while (!q.empty()) {
            cur = q.front(), q.pop();
            st.push(cur);

            for (int i : arr[cur]) {
                if (checked[i])
                    continue;
                checked[i] = true;
                parent[i] = cur;
                depth[i] = depth[cur] + 1;
                child[cur].push_back(i);
                q.push(i);
            }
        }

        while (!st.empty()) {
            cur = st.top(), st.pop();
            sz[cur] = 1;
            for (int i = 0; i < child[cur].size(); i++) {
                sz[cur] += sz[child[cur][i]];
                if (sz[child[cur][i]] > sz[child[cur][0]])
                    swap(child[cur][i], child[cur][0]);
            }
        }

        dfs(root);
    }

    void build_seg() {
        for (int i = 1; i <= N; i++)
            seg.arr[in[i]] = val[i];
        seg.init();
    }

    void update(int u, int v, int val) {
        while (top[u] ^ top[v]) {
            if (depth[top[u]] < depth[top[v]])
                swap(u, v);
            seg.update(in[top[u]], in[u], val);
            u = parent[top[u]];
        }
        if (depth[u] > depth[v])
            swap(u, v);
        seg.update(in[u] + 1, in[v], val);
    }

    void update_sub(int pos, int val) {
        seg.update(in[pos], out[pos], val);
    }

    int query(int u, int v) {
        int res = 0;
        while (top[u] ^ top[v]) {
            if (depth[top[u]] < depth[top[v]])
                swap(u, v);
            res = res + seg.query(in[top[u]], in[u]);
            u = parent[top[u]];
        }
        if (depth[u] > depth[v])
            swap(u, v);
        res = res + seg.query(in[u] + 1, in[v]);
        return res;
    }

    int lca(int u, int v) {
        while (top[u] ^ top[v]) {
            if (depth[top[u]] < depth[top[v]])
                swap(u, v);
            u = parent[top[u]];
        }
```

```
        return depth[u] < depth[v] ? u : v;
    }
};
```

3.6 LCA

```
int N, parent[MAX][MAX_LOG], depth[MAX];
vector<int> arr[MAX];
bool checked[MAX];

void dfs(int K) {
    int A;
    stack<int> st;

    st.push(K);
    checked[K] = true;

    while (!st.empty()) {
        A = st.top();
        st.pop();

        for (int i : arr[A]) {
            if (checked[i])
                continue;
            parent[i][0] = A;
            checked[i] = true;
            depth[i] = depth[A] + 1;

            for (int j = 1; j < MAX_LOG; j++) {
                if (!parent[i][j - 1])
                    continue;
                parent[i][j] = parent[parent[i][j - 1]][j - 1];
            }

            st.push(i);
        }
    }
}

int LCA(int A, int B) {
    if (depth[A] < depth[B])
        swap(A, B);

    int diff = depth[A] - depth[B];
    for (int i = 0; diff; i++) {
        if (diff & 1)
            A = parent[A][i];
        diff >>= 1;
    }

    for (int i = MAX_LOG - 1; i >= 0; i--) {
        if (parent[A][i] != parent[B][i])
            A = parent[A][i], B = parent[B][i];
    }
    if (A != B)
        A = parent[A][0];
    return A;
}

int get_dis(int A, int B) {
    int X = LCA(A, B);
    return depth[A] + depth[B] - 2 * depth[X];
}
```

4 Math

4.1 Combination

```
int fac[MAX], inv_fac[MAX];

int fpow(int N, int K) {
    int res = 1;
    while (K) {
        if (K & 1)
            res = res * N % MOD;
        K >>= 1;
        N = N * N % MOD;
    }
    return res;
}

int prime_inverse(int K, int X) { return fpow(K, X - 2); }

tp extended_gcd(int A, int B) {
    if (B == 0)
        return {A, 1, 0};
    tp res = extended_gcd(B, A % B);
    return {res[0], res[2], res[1] - A / B * res[2]};
}

int modular_inverse(int K, int X) {
    tp res = extended_gcd(K, X);
    if (res[0] != 1)
        return -1;
    return (res[1] % X + X) % X;
}

void init(int L) {
    fac[0] = 1;
    for (int i = 1; i <= L; i++)
        fac[i] = fac[i - 1] * i % MOD;
    inv_fac[L] = fpow(fac[L], MOD - 2);
    for (int i = L - 1; i >= 0; i--)
        inv_fac[i] = inv_fac[i + 1] * (i + 1) % MOD;
}

int comb(int A, int B) {
    int X = fac[A], Y, Z;
    Y = fac[B], Z = fac[A - B];
```

```
    return X * prime_inverse(Y, MOD) % MOD * prime_inverse(Z, MOD) % MOD;
}
```

4.2 Linear Sieve

```
int min_prime_factor[MAX + 1];
vector<int> primes;

void linear_sieve(int N) {
    for (int i = 2; i <= N; i++) {
        if (min_prime_factor[i] == 0) {
            min_prime_factor[i] = i;
            primes.push_back(i);
        }

        for (int p : primes) {
            if (i * p > N)
                break;
            min_prime_factor[i * p] = p;
            if (i % p == 0)
                break;
        }
    }
}
```

5 String

5.1 KMP

```
vector<int> get_pi(string P) {
    int size = P.size(), j = 0;
    vector<int> pi(size, 0);

    for (int i = 1; i < size; i++) {
        while (j > 0 && P[i] != P[j])
            j = pi[j - 1];
        if (P[i] == P[j])
            pi[i] = ++j;
    }
    return pi;
}

vector<int> kmp(string S, string P) {
    vector<int> pi = get_pi(P), ans;
    int S_size = S.size(), P_size = P.size(), j = 0;

    for (int i = 0; i < S_size; i++) {
        while (j > 0 && S[i] != P[j])
            j = pi[j - 1];
        if (S[i] == P[j]) {
            if (j == P_size - 1) {
                ans.push_back(i - P_size + 2);
                j = pi[j];
            } else
                j++;
        }
    }

    return ans;
}
```

5.2 Manacher

```
class Manacher {
  public:
    string S, K;
    vector<int> rad;

    Manacher(string S) : S(S) {
        K = "#";
        for (char i : S) {
            K.push_back(i);
            K.push_back('#');
        }
        rad.resize(K.size());
    }

    void build() {
        int r = -1, c = -1;
        for (int i = 0; i < K.size(); i++) {
            if (i <= r)
                rad[i] = min(r - i, rad[2 * c - i]);
            while (i - rad[i] - 1 >= 0 && i + rad[i] + 1 < K.size() && K[i - rad
              [i] - 1] == K[i + rad[i] + 1])
                rad[i]++;
            if (r < i + rad[i]) {
                r = i + rad[i];
                c = i;
            }
        }
    }
};
```

6 Etc

6.1 Convex Hull Trick

```
// X 좌표단조증가
class ConvexHullTrick {
  public:
    vector<tp> F;
    int ftop = 0;

    void insert(pr X) {
        tp K = {X.first, X.second, 0};
        while (!F.empty()) {
            K[2] = (F.back()[1] - K[1]) / (K[0] - F.back()[0]);
            if (F.back()[2] < K[2])
```

```
                break;
            F.pop_back();
            if (F.size() == ftop)
                --ftop;
        }
        F.push_back(K);
    }

    int query(int x) {
        while (ftop + 1 < F.size() && F[ftop + 1][2] < x)
            ++ftop;
        return F[ftop][0] * x + F[ftop][1];
    }
};

// 그렇지않은경우
class ConvexHullTrick {
  public:
    vector<tp> F;

    void insert(tp K) {
        while (!F.empty()) {
            K[2] = (F.back()[1] - K[1]) / (K[0] - F.back()[0]);
            if (F.back()[2] < K[2])
                break;
            F.pop_back();
        }
        F.push_back(K);
    }

    int query(int x) {
        int res = F.size() - 1, st = 0, en = F.size() - 1, mid;
        if (x < F.back()[2]) {
            while (st + 1 < en) {
                mid = (st + en) / 2;
                if (x < F[mid][2])
                    en = mid;
                else
                    st = mid;
            }
            res = st;
        }
        return F[res][0] * x + F[res][1];
    }
};
```

6.2 Union Find

```
int subst(int A, int B, int N) { return A * (N + 1) + B; }
pr to_pair(int K, int N) { return {K / (N + 1), K % (N + 1)}; }

// 경로압축
class UnionFind {
  private:
    vector<int> uf_parent;

  public:
    UnionFind(int N) : uf_parent(N + 1) { clear(N); }

    int find(int K) {
        if (uf_parent[K] != K)
            uf_parent[K] = find(uf_parent[K]);
        return uf_parent[K];
    }

    void uni(int A, int B) {
        A = find(A), B = find(B);
        if (A > B)
            swap(A, B);
        uf_parent[B] = A;
    }

    void clear(int N) {
        for (int i = 1; i <= N; i++)
            uf_parent[i] = i;
    }
};

// union-by-rank
class UnionFind {
  private:
    vector<int> uf_parent, rank;
    stack<tp> st;

  public:
    UnionFind(int N) : uf_parent(N + 1), rank(N + 1, 0) { clear(N); }

    int find(int K) { return uf_parent[K] == K ? K : find(uf_parent[K]); }

    bool uni(int A, int B) {
        A = find(A), B = find(B);
        if (A == B)
            return false;

        if (rank[A] < rank[B])
            swap(A, B);
        st.push({A, B, rank[A] == rank[B]});
        uf_parent[B] = A;
        rank[A] += rank[A] == rank[B];
        return true;
    }

    void rollback(int cnt = 1) {
        while (cnt--) {
            tp cur = st.top();
            st.pop();
            uf_parent[cur[1]] = cur[1];
            rank[cur[0]] -= cur[2];
        }
    }
```

```
    void clear(int N) {
        for (int i = 1; i <= N; i++) {
            uf_parent[i] = i;
            rank[i] = 0;
        }
    }
};
```

6.3 mo's algorithm

```
int ans[MAX];

signed main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);

    int N, Q, S, A, B, nl, nr, l, r;
    vector<tp> query;

    cin >> N;
    S = sqrt(N);

    cin >> Q;
    for (int i = 1; i <= Q; i++) {
        cin >> A >> B;
        query.push_back({A, B, i});
    }

    sort(query.begin(), query.end(), [S](tp a, tp b) {
        int af = a[0] / S, bf = b[0] / S;
        if (af == bf)
            return a[1] / S < b[1] / S;
        return af < bf;
    });

    for (int i = query[0][0]; i <= query[0][1]; i++)
        continue;
    ans[query[0][2]] = ans[0];
    nl = query[0][0], nr = query[0][1];

    for (int i = 1; i < M; i++) {
        l = query[i][0], r = query[i][1];
        while (nl < l)
            continue;
        while (nr > r)
            continue;
        while (nl > l)
            continue;
        while (nr < r)
            continue;

        ans[query[i][2]] = ans[0];
    }

    for (int i = 1; i <= Q; i++)
        cout << ans[i] << '\n';

    return 0;
}
```

7 체계적인 접근을 위한 질문들

"알고리즘 문제 해결 전략" 에서 발췌함

- 비슷한 문제를 풀어본 적이 있던가?
- 단순한 방법에서 시작할 수 있을까? (brute force)
- 내가 문제를 푸는 과정을 수식화할 수 있을까? (예제를 직접 해결해보면서)
- 문제를 단순화할 수 없을까?
- 그림으로 그려볼 수 있을까?
- 수식으로 표현할 수 있을까?
- 문제를 분해할 수 있을까?
- 뒤에서부터 생각해서 문제를 풀 수 있을까?
- 순서를 강제할 수 있을까?
- 특정 형태의 답만을 고려할 수 있을까? (정규화)