Competitive Programming Notebook

Compiled on March 13, 2024

Contents

1 Math

1.1 GCD, LCM

Time Complexity: $O(logN)$

```cpp
typedef long long int ll;

// Gcd(x, y) : x, y의 최대공약수를 반환
ll Gcd(ll x, ll y) {
    if (y == 0) return x;
    else return Gcd(y, x % y);
}

// Lcm(x, y) : x, y의 최대공약수를 반환
ll Lcm(ll x, ll y) {
    return x * y / Gcd(x, y);
}
```

1.2 Fast Power

Time Complexity: $O(logN)$

```cpp
typedef long long int ll;
ll mod = 1000000007;

// Mul(x, y, mod) : x*y % mod를 반환
ll Mul(ll x, ll y, ll mod) { return (__int128_t)x * y % mod; }

// Pow(x, y, mod) : x^y % mod를 반환
ll Pow(ll x, ll y, ll mod) {
    ll res = 1; x %= mod;
    while (y) {
        if (y & 1) res = Mul(res, x, mod);
        x = Mul(x, x, mod);
        y >>= 1;
    }
    return res;
}
```

1.3 Fast Combination

Time Complexity: $O(N + K)$

```cpp
typedef long long int ll;
#define MOD 1000000007

ll Pow(ll x, ll y, ll mod) {
    ll res = 1;
    x %= mod;

    while (y) {
        if (y & 1) res = (res * x) % mod;
        y >>= 1;
        x = (x * x) % mod;
    }

    return res;
}

// Comb(n, k) : nCk를 계산해서 반환
// mod값이 소수일 때만 가능
ll Comb(ll n, ll k) {
    ll A = 1, B = 1;

    for (int i = 1; i <= n; i++) A = (A * i) % MOD;
    for (int i = 1; i <= k; i++) B = (B * i) % MOD;
    for (int i = 1; i <= n - k; i++) B = (B * i) % MOD;

    return (A * Pow(B, MOD - 2, MOD)) % MOD;
}
```

1.4 Sieve of Eratosthenes

Time Complexity: $O(Nlog^2N)$

```
// primeCheck[n] == 0 : n이 소수
// primeCheck[n] != 0 : n의 최소 소인수
int primeCheck[5005000];

void Sieve() {
    for (int i = 2; i <= sqrt(5000000); i++) {
        if (primeCheck[i]) continue;
        for (int j = i * i; j <= 5000000; j += i) {
            if (!primeCheck[j]) primeCheck[j] = i;
        }
    }
}
```

1.5 Linear Sieve

Time Complexity: $O(N)$

```
// primeCheck[n] == 0 : n이 소수
// primeCheck[n] != 0 : n의 최소 소인수
int primeCheck[5005000];
vector<int> primeNum;

void LinearSieve() {
    for (int i = 2; i <= 5000000; i++) {
        if (!primeCheck[i]) primeNum.emplace_back(i);

        for (auto j: primeNum) {
            if (i * j > 5000000) break;
            primeCheck[i*j] = j;
            if (i % j == 0) break;
        }
    }
}
```

1.6 Fraction

```
typedef long long int ll;

// Fraction(a, b) = a / b    (b != 0)
struct Fraction {
    ll x, y;
    Fraction(ll x = 0, ll y = 1) { this->x = x; this->y = y; }
    Fraction operator+(const Fraction &f) const {
        ll temp = lcm(y, f.y);
        Fraction res = {temp / y * x + temp / f.y * f.x,
        temp};
        temp = gcd(res.x, res.y);
        res = {res.x / temp, res.y / temp};
        return res;
    }
    Fraction operator-(const Fraction &f) const {
        ll temp = lcm(y, f.y);
        Fraction res = {temp / y * x - temp / f.y * f.x,
        temp};
        temp = gcd(res.x, res.y);
        res = {res.x / temp, res.y / temp};
        return res;
    }
    Fraction operator*(const Fraction &f) const {
        Fraction res = {x * f.x, y * f.y};
        ll temp = gcd(res.x, res.y);
        res = {res.x / temp, res.y / temp};
        return res;
    }
    Fraction operator/(const Fraction &f) const {
        Fraction res = {x * f.y, y * f.x};
        ll temp = gcd(res.x, res.y);
        res = {res.x / temp, res.y / temp};
        return res;
    }
    bool operator<(const Fraction &f) const {
        __int128_t a = x * f.y, b = y * f.x;
        return a < b;
    }
    bool operator>(const Fraction &f) const {
        __int128_t a = x * f.y, b = y * f.x;
        return a > b;
    }
    bool operator<=(const Fraction &f) const {
        __int128_t a = x * f.y, b = y * f.x;
        return a <= b;
    }
    bool operator>=(const Fraction &f) const {
        __int128_t a = x * f.y, b = y * f.x;
        return a >= b;
    }
    bool operator==(const Fraction &f) const {
        __int128_t a = x * f.y, b = y * f.x;
        return a == b;
    }
};
```

1.7 Miller-Rabin

Time Complexity: $O(logN)$

```
// MillerRabin(n) : n의 소수 여부를 반환
bool MillerRabin(ll n) {
    ll intCheck[] = {2, 7, 61};
    ll llCheck[] = {2, 325, 9375, 28178, 450775, 9780504,
    1795265022};

    bool res = true;
    for (ll a: intCheck) {
        if (a % n == 0) break;
        ll k = n - 1;
        while (1) {
            ll tmp = Pow(a, k, n);
            if (tmp == n - 1) break;
            if (k & 1) {
                res &= (tmp == 1 || tmp == n - 1); break;
            }
            k >>= 1;
        }
        if (!res) break;
    }

    return res;
}
```

1.8 Pollard-Rho

Time Complexity: $O(N^{1/4})$

```
// pollardRho(n, v) : v 배열에 n의 소인수를 반환
void PollardRho(ll n, vector<ll> &v) {
    if (n == 1) return;
    if (MillerRabin(n)) { v.push_back(n); return; }
    if (~n & 1) { v.push_back(2); PollardRho(n >> 1, v);
    return; }

    ll a, b, c, g = n;
    auto f = [&](ll x) { return (c + Mul(x, x, n)) % n; };

    a = b = Rand.randInt(0, n - 2) + 2;
    c = Rand.randInt(0, n - 1) + 1;
    do {
        a = f(a); b = f(f(b));
        g = gcd(abs(a - b), n);
    } while (g == 1);

    PollardRho(g, v); PollardRho(n / g, v);
}
```

1.9 Catalan Number

Time Complexity: $O(N^2)$

```
// 길이 2n 괄호 문자열의 경우의 수
// n+2각형 → n개 삼각형으로 분할하는 경우의 수
// 일반항 : C_n = 1/(n+1) * (2n)C(n)

#include <bits/stdc++.h>
#define MOD 987654321
using namespace std;
typedef long long ll;
```

```
ll dp[12345] = {0};

int main(){
  int N;
  cin >> N;

  dp[0] = 1;
  for(int i=1; i<=N; i++){
    for(int j=0; j<i; j++){
      dp[i] += (dp[j]*dp[i-j-1]);
      dp[i] %= MOD;
    }
  }

  cout << dp[N/2];
  return 0;
}
```

1.10 FFT

Time Complexity: $O(NlogN)$

```
typedef complex<double> cp;
const double PI = acos(-1);

void FFT(vector<cp> &v, bool inv) {
    int siz = v.size();

    for (int i = 1, j = 0; i < siz; i++) {
        int bit = siz >> 1;
        while(!((j ^= bit) & bit)) bit >>= 1;
        if (i < j) swap(v[i], v[j]);
    }

    for (int k = 1; k < siz; k <<= 1) {
        double a = (inv ? PI / k : -PI / k);
        cp w(cos(a), sin(a));

        for (int i = 0; i < siz; i += (k << 1)) {
            cp z(1, 0);

            for (int j = 0; j < k; j++) {
                cp even = v[i+j], odd = v[i+j+k];

                v[i+j] = even + z * odd;
                v[i+j+k] = even - z * odd;

                z *= w;
            }
        }
    }

    if (inv) {
        for (int i = 0; i < siz; i++) v[i] /= siz;
    }
}

// multiply(a, b) : 다항식 a, b의 convolution을 반환
vector<int> multiply(vector<int> &a, vector<int> &b) {
    vector<cp> A(a.begin(), a.end());
    vector<cp> B(b.begin(), b.end());

    int siz = 2;
    while (siz < A.size() + B.size()) siz <<= 1;

    A.resize(siz); FFT(A, false);
    B.resize(siz); FFT(B, false);

    for (int i = 0; i < siz; i++) A[i] *= B[i];
    FFT(A, true);

    vector<int> res(siz);
    for (int i = 0; i < siz; i++) res[i] = round(A[i].real());

    return res;
}
```

2 Graph

2.1 Dijkstra

Time Complexity: $O(VlogE)$

```
// 1. 인접 리스트 e에 간선 {음이 아닌 가중치, 도착점} 형태로 저장
// 2. Dijkstra(s)으로 실행
// 3. dist[i]에 s -> i의 최단경로 가중치 저장

typedef long long int ll;
typedef pair<ll, ll> pr;
#define D first
#define P second

vector<vector<pr>> e(200001);
vector<ll> dist(200001);
bitset<200001> visited;

void Dijkstra(int s) {
    visited.reset(); fill(dist.begin(), dist.end(), 1e18);
    priority_queue<pr, vector<pr>, greater<pr>> pq;

    pq.push({0, s}); dist[s] = 0;

    while (!pq.empty()) {
        pr t = pq.top();
        pq.pop();

        if (visited[t.P]) continue;
        visited[t.P] = true;

        for (auto i: e[t.P]) {
            if (dist[i.P] > dist[t.P] + i.D) {
                dist[i.P] = dist[t.P] + i.D;
                pq.push({dist[i.P], i.P});
            }
        }
    }
}
```

2.2 Bellman-Ford

Time Complexity: $O(VE)$

```
// 1. 인접 리스트 e에 간선 {가중치, 도착점} 형태로 저장
// 2. BellmanFord(n, s)으로 실행
// 3. dist[i]에 s -> i의 최단경로 가중치 저장
// 4. 음수 사이클이 생기면 cycle = true임

typedef long long int ll;
typedef pair<ll, ll> pr;
#define D first
#define P second

vector<vector<pr>> e(200001);
vector<ll> dist(200001);
bitset<200001> visited;
bool cycle = false;

void BellmanFord(int n, int s) {
    fill(dist.begin(), dist.end(), 1e18);
    dist[s] = 0;

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (dist[j] == 2e18) continue;

            for (auto k: e[j]) {
                if (dist[k.P] > dist[j] + k.D) {
                    dist[k.P] = dist[j] + k.D;
                    if (i == n) cycle = true;
                }
            }
        }
    }
}
```

2.3 Floyd-Warshall

Time Complexity: $O(V^3)$

```
// dist[i][j] = INF로 초기화
// 1. 인접 행렬 dist[s][e]에 가중치를 저장
// 2. FloydWarshall(n)으로 실행
// 3. dist[i][j]에 i -> j의 최단경로 가중치 저장

typedef long long int ll;
ll dist[1010][1010];

void FloydWarshall(int n) {
    for (int k = 1; k <= n; k++) {
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                dist[i][j] = min(dist[i][j], dist[i][k] +
                dist[k][j]);
            }
        }
    }
}
```

2.4 SPFA

Time Complexity: Average : $O(V+E)$, Worst : $O(VE)$

```
// 1. 인접 리스트 e에 간선 {가중치, 도착점} 형태로 저장
// 2. SPFA(n, s)으로 실행
// 3. dist[i]에 s -> i의 최단경로 가중치 저장
// 4. 음수 사이클이 생기면 cycle = true임

typedef long long int ll;
typedef pair<ll, ll> pr;
#define D first
#define P second

vector<vector<pr>> e(200001);
vector<ll> dist(200001), cnt(200001);
bitset<200001> inq; // 큐 안에 들어가 있는지 저장
bool cycle = false;

void SPFA(int n, int s) {
    fill(dist.begin(), dist.end(), 1e18);
    fill(cnt.begin(), cnt.end(), 0); inq.reset();

    queue<int> q;
    q.push(s); inq[s] = true; dist[s] = 0;

    while (!q.empty()) {
        int t = q.front();
        q.pop(); inq[t] = false;

        if (++cnt[t] >= n || dist[t] < -1e18) {
            cycle = true; return;
        }

        for (auto i: e[t]) {
            if (dist[i.P] > i.D + dist[t]) {
                dist[i.P] = i.D + dist[t];
                if (!inq[i.P]) {
                    q.push(i.P);
                    inq[i.P] = true;
                }
            }
        }
    }
}
```

2.5 Lowest Common Ancestor

Time Complexity: Init : $O(NlogN)$, Query : $O(logN)$

```
// 1. 트리의 간선을 인접리스트 e에 저장
// 2. makeSparseTable(root)로 희소 배열을 만든 후, LCA(x, y)

int n, m;
vector<vector<int>> e;
int sp_table[200200][20], dep[200200];
bitset<200200> visited;

void dfs(int p, int d) {
    visited[p] = true;
    dep[p] = d;

    for (int nxt: e[p]) {
        if (!visited[nxt]) {
            sp_table[nxt][0] = p;
            dfs(nxt, d + 1);
        }
    }
}

void makeSparseTable(int root) {
    visited.reset();
    sp_table[root][0] = 1;
    dfs(root, 0);

    for (int i = 1; i < 20; i++) {
        for (int j = 1; j <= n; j++) sp_table[j][i] =
        sp_table[sp_table[j][i-1]][i-1];
    }
}

int LCA(int x, int y) {
    int dist = 0;
    if (dep[x] < dep[y]) swap(x, y);

    for (int i = 19; i >= 0; i--) {
        if (dep[x] - (1 << i) >= dep[y]) {
            dist += 1 << i;
            x = sp_table[x][i];
        }

        if (dep[x] == dep[y]) break;
    }

    for (int i = 19; i >= 0; i--) {
        if (sp_table[x][i] != sp_table[y][i]) {
            dist += 2 << i;
            x = sp_table[x][i];
            y = sp_table[y][i];
        }
    }

    if (x != y) {
        dist += 2;
        x = sp_table[x][0];
        y = sp_table[y][0];
    }

    return x; // return LCA of (x, y)
}
```

2.6 Euler Tour Technique

Time Complexity: $O(N)$

```
// 트리의 인접 리스트 e가 주어졌을 때, ETT(rootNode)로 실행하면
// l[x] ~ r[x]가 정점 x의 서브트리의 정점 범위를 나타냄

int cnt = 0;
int l[200200], r[200200];
vector<vector<int>> e;
bitset<200200> visited;

void ETT(int x) {
    l[x] = ++cnt;
    visited[x] = true;

    for (auto i : e[x]) {
        if (!visited[i]) ETT(i);
    }

    r[x] = cnt;
}
```

3 Flow

3.1 Edmonds-Karp

```
#define MAX 501
#define INF 1e9
struct FlowEdge { int p, c, r; }; // pos, cost, rev
vector<vector<FlowEdge>> graph(MAX);
vector<int> dist(MAX), prv(MAX), idx(MAX);

// addEdge(s, e, c) : s -> e, cost = c인 단방향 간선 추가
// addEdge(s, e, c, c) : s - e, cost = c인 무방향 간선 추가
void addEdge(int s, int e, int c1, int c2 = 0) {
    graph[s].push_back({e, c1, (int)graph[e].size()});
    graph[e].push_back({s, c2, (int)graph[s].size() - 1});
}

int augment(int s, int t) {
    fill(dist.begin(), dist.end(), INF);
    queue<int> q; q.push(s); dist[s] = 0;

    while (q.size()) {
        int pos = q.front(); q.pop();

        for (int i = 0; i < graph[pos].size(); i++) {
            auto &nxt = graph[pos][i];

            if (nxt.c > 0 && dist[nxt.p] == INF) {
                q.push(nxt.p); dist[nxt.p] = dist[pos] + 1;
                prv[nxt.p] = pos; idx[nxt.p] = i;
            }
        }
    }

    if (dist[t] == INF) return 0;

    int flow = INF;
    for (int pos = t; pos != s; pos = prv[pos]) {
        auto &nxt = graph[prv[pos]][idx[pos]];
        flow = min(flow, nxt.c);
    }

    for (int pos = t; pos != s; pos = prv[pos]) {
        auto &nxt = graph[prv[pos]][idx[pos]];
        nxt.c -= flow; graph[nxt.p][nxt.r].c += flow;
    }

    return flow;
}

// s -> t로 흘릴 수 있는 최대 유량 반환
int EdmondKarp(int s, int t) {
    int flow = 0, tmp = 0;

    while (true) {
        tmp = augment(s, t);
        if (tmp) flow += tmp;
        else break;
    }

    return flow;
}
```

4 String

4.1 KMP

Time Complexity: $O(|A| + |B|)$

```
// getFail(b) : 문자열 b의 실패함수 반환
vector<int> getFail(string &b) {
    int n = b.length();
    vector<int> fail(n);

    for (int i = 1, j = 0; i < n; i++) {
        while (j && b[i] != b[j]) j = fail[j-1];
        if (b[i] == b[j]) fail[i] = ++j;
    }
    return fail;
}

// KMP(a, b) : 문자열 a에서 문자열 b가 나오는 위치 반환
// KMP("AAABAA", "AA") = {0, 1, 4}
vector<int> KMP(string &a, string &b) {
    int n = a.length(); int m = b.length();
    vector<int> fail, ret; fail = getFail(b);

    for (int i = 0, j = 0; i < n; i++) {
        while (j && a[i] != b[j]) j = fail[j-1];
        if (a[i] == b[j]) {
            if (j + 1 == m) {
                ret.push_back(i - j + 1);
                j = fail[j];
            }
            else j++;
        }
    }

    return ret;
}
```

4.2 Trie

Time Complexity: $O(VlogE)$

```
// 선언: Trie* root = new Trie;
// 추가: root->Insert(문자열.c_str());
// 찾기: root->Find(문자열.c_str());
// 삭제: root->Delete(문자열.c_str());

struct Trie {
    int child_num;
    bool finish;
    Trie* next[26];

    Trie() {
        child_num = 0;
        finish = false;
        for (int i = 0; i < 26; i++) next[i] = NULL;
    }
    ~Trie() {
        for (int i = 0; i < 26; i++) {
            if (next[i]) delete next[i];
        }
    }
    void Insert(const char* s) {
        child_num++;

        if (!*s) {
            this->finish = true;
            return;
        }

        int now = *s - 'A';
        if (!next[now]) next[now] = new Trie;
        next[now]->Insert(s + 1);
    }
    bool Find(const char* s) {
        if (!*s) return this->finish;

        int now = *s - 'A';
        if (!next[now]) return false;
        return next[now]->Find(s + 1);
    }
    bool Delete(const char* s) {
        this->child_num--;

        if (!*s) this->finish = false;
        else {
            int now = *s - 'A';
            if (!next[now]->Delete(s + 1)) {
                delete next[now];
                next[now] = nullptr;
            }
        }
        return this->child_num;
    }
};
```

4.3 Manacher

Time Complexity: $O(N)$

```
// 각 문자를 중심으로 하는 최장 팰린드롬의 길이를 반환
// Manacher("ASDDSA") = {0, 1, 0, 1, 0, 1, 6, 1, 0, 1, 0, 1,
0}
vector<int> Manacher(string &s) {
    int n = s.length() * 2 + 1;
    vector<int> ret(n);
    string tmp = "#";

    for (auto i: s) {
        tmp += i; tmp += '#';
    }

    for (int i = 0, p = -1, r = -1; i < n; i++) {
        if (i <= r) ret[i] = min(r - i, ret[2*p-i]);
        else ret[i] = 0;

        while (i - ret[i] - 1 >= 0 && i + ret[i] + 1 < n &&
        tmp[i-ret[i]-1] == tmp[i+ret[i]+1]) ret[i]++;
        if (i + ret[i] > r) {
            r = i + ret[i]; p = i;
        }
    }

    return ret;
}
```

5 Data Structure

5.1 Union Find

Time Complexity: $O(a(N))$

```
// 생성 : UnionFind* uf = new UnionFind(Siz);
// uf->Find(x) : x의 부모를 반환
// uf->Union(x, y, dir) : x, y 그룹을 하나로 합침
// uf->isSameSet(x, y) : x, y가 한 그룹에 있는지 반환

struct UnionFind {
    vector<int> parent;
    int group_count = 0; int size = 0;

    UnionFind(int siz) {
        parent.resize(siz);
        group_count = siz; size = siz;

        for (int i = 0; i < siz; i++) parent[i] = i;
    }

    ~UnionFind() {
        parent.clear();
        group_count = 0;
    }

    int Find(int x) {
        if (x == parent[x]) return x;
        else return parent[x] = Find(parent[x]);
    }

    // dir == false : x -> y
    // dir == true : x <- y
    void Union(int x, int y, bool dir) {
        x = Find(x); y = Find(y);
        if (x > y) swap(x, y);

        if (x != y) {
            dir ? (parent[x] = y) : (parent[y] = x);
            group_count--;
        }
    }

    bool isSameSet(int x, int y) {
        return Find(x) == Find(y);
    }
};
```

5.2 Segment Tree

Time Complexity: Update : $O(logN)$, Query : $O(logN)$

```
// 생성 : Segtree* seg = new Segtree(n);
// seg->Update(pos, val) : p 위치에 v 값으로 업데이트
// seg->Query(l, r) : l ~ r 범위의 구간 쿼리 결과를 반환

typedef long long int ll;

struct Segtree {
    vector<ll> st;
    int tmp;

    Segtree(int n) {
        for (tmp = 1; tmp < n; tmp *= 2) {}
        st.resize(tmp * 2);
    }

    // 용도 별로 Update함수 고쳐쓰기
    void Update(int pos, ll val) {
        pos = pos + tmp - 1;
        st[pos] = val;

        while (pos != 1) {
            pos /= 2;
            st[pos] = st[pos*2] + st[pos*2+1];
        }
    }

    // 용도 별로 쿼리를 처리하는 Search함수 고쳐쓰기
    ll Search(int l, int r, int s, int e, int p) {
        if (e < l || r < s) return 0;
        else if (s <= l && r <= e) return st[p];
        else return Search(l, (l + r) / 2, s, e, p * 2) +
        Search((l + r) / 2 + 1, r, s, e, p * 2 + 1);
    }

    ll Query(int l, int r) {
        return Search(1, tmp, l, r, 1);
    }
};
```

5.3 Fenwick Tree

```
// dk10211 작성
void add(int k, int x) {
  while (k <= n) {
    ft[k] = (ft[k] + x) % MOD;
    k += k & -k;
  }
}
int sum(int k) {
  int s = 0;
  while (k >= 1) {
    s = (s + ft[k]) % MOD;
    k -= k & -k;
  }
  return s;
}
```

5.4 Segment Tree with Lazy Propagation

Time Complexity: Update : $O(logN)$, Query : $O(logN)$

```
// 생성 : LazySegtree* seg = new LazySegtree(n);
// seg->Update(l, r, val) : l ~ r 범위의 구간에 v 값으로 업데이트
// seg->Query(l, r) : l ~ r 범위의 구간 쿼리 결과를 반환

typedef long long int ll;

struct LazySegtree {
    vector<ll> st, lazy;
    int tmp;

    LazySegtree(int n) {
        for (tmp = 1; tmp < n; tmp *= 2) {}
        st.resize(tmp * 2); lazy.resize(tmp * 2);
    }
```

```
    void prop(int l, int r, int p) {
        st[p] += lazy[p] * (r - l + 1);
        if (l != r) {
            lazy[p*2] += lazy[p]; lazy[p*2+1] += lazy[p];
        }
        lazy[p] = 0;
    }

    // 용도 별로 업데이트를 처리하는 RangeUpdate함수 고쳐쓰기
    void RangeUpdate(int l, int r, int s, int e, int p, ll
    val) {
        if (lazy[p]) prop(l, r, p);

        if (e < l || r < s) return;
        else if (s <= l && r <= e) {
            lazy[p] += val;
            prop(l, r, p);
            return;
        }
        else {
            RangeUpdate(l, (l + r) / 2, s, e, p * 2, val);
            RangeUpdate((l + r) / 2 + 1, r, s, e, p * 2 + 1,
            val);
            st[p] = st[p*2] + st[p*2+1];
        }
    }


    void Update(int l, int r, ll val) {
        RangeUpdate(1, tmp, l, r, 1, val);
    }

    // 용도 별로 쿼리를 처리하는 Search함수 고쳐쓰기
    ll Search(int l, int r, int s, int e, int p) {
        if (lazy[p]) prop(l, r, p);

        if (e < l || r < s) return 0;
        else if (s <= l && r <= e) return st[p];
        else return Search(l, (l + r) / 2, s, e, p * 2) +
        Search((l + r) / 2 + 1, r, s, e, p * 2 + 1);
    }

    ll Query(int l, int r) {
        return Search(1, tmp, l, r, 1);
    }
};
```

5.5 Merge Sort Tree

Time Complexity: Query : $O(log^2 N)$

```
// 생성 : MergeSortTree* seg = new MergeSortTree(v);
// seg->Query(l, r, c) : l ~ r 범위의 구간 쿼리 결과를 반환
// 예제 쿼리 : l ~ r 범위 구간에서 c보다 큰 원소의 개수

typedef long long int ll;

struct MergeSortTree {
    vector<vector<ll>> st;
    int tmp, siz;

    MergeSortTree(vector<ll> v) {
        siz = v.size();
        for (tmp = 1; tmp < siz; tmp *= 2) {}
        st.resize(tmp * 2);

        for (int i = tmp; i < tmp + siz; i++)
        st[i].push_back(v[i-tmp]);
        for (int i = tmp - 1; i >= 1; i--) {
            st[i].resize(st[i*2].size()*2);
            merge(st[i*2].begin(), st[i*2].end(),
            st[i*2+1].begin(), st[i*2+1].end(),
            st[i].begin());
        }
    }

    // 용도 별로 쿼리를 처리하는 Search함수 고쳐쓰기
    ll Search(int l, int r, int s, int e, int p, int c) {
        if (e < l || r < s) return 0;
        if (s <= l && r <= e) return st[p].size() -
        (lower_bound(st[p].begin(), st[p].end(), c + 1) -
        st[p].begin());
        return Search(l, (l + r) / 2, s, e, p * 2, c) +
        Search((l + r) / 2 + 1, r, s, e, p * 2 + 1, c);
    }

    ll Query(int l, int r, int c) {
        return Search(1, tmp, l, r, 1, c);
    }
};
```

5.6 Li-Chao Tree

Time Complexity: Update : $O(logN)$, Query : $O(logN)$

```
// 생성 : LiChaoTree* l = new LiChaoTree(Min, Max);
// Update({a, b}) : y = ax + b을 업데이트
// Query(x) : 트리에 있는 직선 중 가장 큰 ax + b의 값을 리턴

typedef long long int ll;
const ll INF = 2e18;

struct LiChaoTree {
    struct Line {
        ll a, b;
        ll get(ll x) { return a * x + b; }
    };
    struct Node {
        LiChaoTree* l; LiChaoTree* r;
        ll s, e;
        Line line;
    };

    Node node;

    LiChaoTree(ll s, ll e) {
        node.l = NULL; node.r = NULL;
        node.s = s; node.e = e;
        node.line = {0, -INF};
    }

    void Update(Line v) {
        ll s = node.s; ll e = node.e;
        ll m = (s + e) / 2;

        Line low = node.line; Line high = v;
        if (low.get(s) > high.get(s)) swap(low, high);

        if (low.get(e) <= high.get(e)) {
            node.line = high; return;
        }

        if (low.get(m) < high.get(m)) {
            node.line = high;
            if (!node.r) node.r = new LiChaoTree(m + 1, e);
            node.r->Update(low);
        }
        else {
            node.line = low;
            if (!node.l) node.l = new LiChaoTree(s, m);
            node.l->Update(high);
        }
    }

    ll Query(ll x) {
        ll m = (node.s + node.e) / 2;
        ll ret = node.line.get(x);

        if (x <= m) {
            if (node.l) ret = max(ret, node.l->Query(x));
            else ret = max(ret, -INF);
        }
        else {
            if (node.r) ret = max(ret, node.r->Query(x));
            else ret = max(ret, -INF);
        }

        return ret;
    }
};
```

6 Etc

6.1 좌표 압축

Time Complexity: $O(NlogN)$

```
typedef long long int ll;

// comp : 압축한 좌표값들을 저장
vector<ll> comp;

void CoordComp(vector<ll> &v)
{
    for (auto i: v) comp.emplace_back(i);

    // 좌표를 압축
    sort(comp.begin(), comp.end());
    comp.erase(unique(comp.begin(), comp.end()), comp.end());

    // 원래 값을 재인덱싱
    for (int i = 0; i < v.size(); i++) {
        v[i] = lower_bound(comp.begin(), comp.end(), v[i]) -
        comp.begin();
    }
}
```

6.2 Binary Search

Time Complexity: $O(logN)$

```
// 0 0 0 0 1 1 1 형태에서 첫번째 1의 위치
int l = 0, r = n;

while (l + 1 < r) {
    int m = (l + r) >> 1;

    bool ok;
    // 여기에 조건 작성

    if (ok) l = m;
    else r = m;
} // 1에 결과값 저장됨


// 1 1 1 1 0 0 0 형태에서 마지막 1의 위치
int l = -1, r = n - 1;

while (l + 1 < r) {
    int m = (l + r) >> 1;

    bool ok;
    // 여기에 조건 작성

    if (ok) l = m;
    else r = m;
} // r에 결과값 저장됨
```

6.3 Tenary Search

Time Complexity: $O(logN)$

```
// 볼록, 오목 개형에서 극점을 찾을때 사용
int l = 0; int r = n;
int a, b;

while (r - l >= 3) {
    a = (l * 2 + r) / 3;
    b = (l + r * 2) / 3;

    bool ok;
    // 여기에 조건 작성

    if (ok) r = b;
    else l = a;
}
```

6.4 랜덤 템플릿

```
// Rand.randInt(l, r) : l~r 범위 안의 랜덤숫자 반환
struct Random {
  mt19937 rd;
  Random() :
  rd(chrono::steady_clock::now().time_since_epoch().count())
  {}
  ll randInt(ll l, ll r) { return
  uniform_int_distribution<ll>(l, r)(rd); }
  double randDouble(double l, double r) { return
  uniform_real_distribution<double>(l, r)(rd); }
} Rand;
```

6.5 Simulated Annealing

```
double t, d, k, lim, local_pow;
int local_ans = 1e9;

void reset_sa() {
    t = 1, d = 0.9999, k = 4, lim = 0.005;
    local_pow = 0;
}

int scoring() {

}

void simulated_annealing() {
    reset_sa(); int e1, e2;
    e1 = scoring();

    while (t > lim) {
        // 현재 상태를 백업 후, 상태를 변형

        // 상태를 변형한 후, e2로 점수 측정
        e2 = scoring();

        if (e2 < local_ans) {
            local_pow = 0; local_ans = e2;

            e1 = e2;
        }
        else local_pow++;

        double p = exp((e1 - e2) / (t * k * log(local_pow)));
        if (p < Rand.randDouble(0, 1)) {
            // 복구
        }
        else e1 = e2;
        t *= d;
    }
}
```

6.6 C++ 코드 템플릿 + Ofast

```
#pragma GCC optimize("O3")
#pragma GCC optimize("Ofast")
#pragma GCC optimize("unroll-loops")
#include <bits/stdc++.h>
using namespace std;

typedef long long int ll;

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);

    return 0;
}
```