Team Note of DART

2e2guk, rlatkdgus, flareon078

Compiled on October 26, 2024

Contents

ALL BELOW HERE ARE USELESS IF YOU READ THE STATEMENT WRONG

1 수학

1.1 에라토스테네스의 체

```
//1~n까지의 소수를 빠르게 찾는 알고리즘, O(N * log log N)
using ll = long long;
void erathos(vector<int> &result, int n){
    vector<bool> notPrime(n + 1, false);
    notPrime[1] = true;
    for(int i = 2; i <= sqrt(n); i++){
        if(notPrime[i]) continue;
        for(int j = (i << 1); j <= n; j += i){
            notPrime[j] = true;
        }
    }
    for(int i = 1; i <= n; i++){
        if(!notPrime[i])    result.emplace_back(i);
    }
}
int main() {
    ios::sync_with_stdio(false); cin.tie(nullptr); ll n; cin >> n;
    vector<int> primeList;
    erathos(primeList, n);
    for(int x : primeList) cout << x << " ";
    return 0;
}
```

1.2 소수판정 알고리즘(확장유클리드)

```
// gcd(A, B) ->  O(max(logA, logB)) -> 확장 유클리드 호제법도 이거 따라감
typedef long long ll;

ll gcd(ll x, ll y) { return __gcd(x, y); }
ll lcm(ll x, ll y) { return x / gcd(x, y) * y; }
ll mod(ll a, ll b) { return ((a%b) + b) % b; }
ll ext_gcd(ll a, ll b, ll &x, ll &y) { // ax + by = gcd(a, b)
  ll g = a; x = 1, y = 0;
  if (b) g = ext_gcd(b, a % b, y, x), y -= a / b * x;
  return g;
}
ll inv(ll a, ll m){ //return x when ax mod m = 1, fail -> -1
    ll x, y;
    ll g = ext_gcd(a, m, x, y);
    if(g > 1) return -1;
    return mod(x, m);
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    ll mod, a; cin >> mod >> a;
    cout << mod - a << " " << inv(a, mod);
    return 0;
}
```

1.3 밀러라빈 + 폴라드로

```
// 밀러-라빈 소수 판별법, pollard-rho 고속 소인수분해
// MillerRabin = O(log^2 N), pollard-rho = O(N^(1/4))
typedef long long ll;
typedef unsigned long long ull;
constexpr int SZ = 10000000;
bool PrimeCheck[SZ+1]; vector<int> Primes;
void Sieve(){
    memset(PrimeCheck, true, sizeof PrimeCheck);
    PrimeCheck[0] = PrimeCheck[1] = false;
    for(int i=2; i<=SZ; i++){
        if(PrimeCheck[i]) Primes.push_back(i);
        for(auto j : Primes){
            if(i*j > SZ) break;
            PrimeCheck[i*j] = false;
            if(i % j == 0) break;
        }
    }
}
ull MulMod(ull a, ull b, ull c){ return (__uint128_t)a * b % c; }
ull PowMod(ull a, ull b, ull c){
    ull res = 1; a %= c;
    for(; b; b>>=1, a=MulMod(a,a,c)) if(b & 1) res = MulMod(res,a,c);
    return res;
}
bool MillerRabin(ull n, ull a){
    if(a % n == 0) return true;
    int cnt = __builtin_ctzll(n - 1);
    ull p = PowMod(a, n >> cnt, n);
    if(p == 1 || p == n - 1) return true;
    while(cnt--) if((p=MulMod(p,p,n)) == n - 1) return true;
    return false;
}
bool IsPrime(ll n){
    if(n <= SZ) return PrimeCheck[n];
    if(n <= 2) return n == 2;
    if(n % 2 == 0 || n % 3 == 0 || n % 5 == 0 || n % 7 == 0 || n % 11 == 0) return false;
    // 32bit integer: {2, 7, 61}
    for(int p : {2, 325, 9375, 28178, 450775, 9780504, 1795265022}) if(!MillerRabin(n, p))
    return false;
    return true;
}
ll Rho(ll n){
    while(true){
        ll x = rand() % (n - 2) + 2, y = x, c = rand() % (n - 1) + 1;
        while(true){
            x = (MulMod(x, x, n) + c) % n;
            y = (MulMod(y, y, n) + c) % n;
            y = (MulMod(y, y, n) + c) % n;
            ll d = __gcd(abs(x - y), n);
            if(d == 1) continue;
            if(IsPrime(d)) return d;
            else{ n = d; break; }
        }
    }
}
vector<pair<ll,ll>> Factorize(ll n){
    vector<pair<ll,ll>> v;
    int two = __builtin_ctzll(n);
    if(two > 0) v.emplace_back(2, two), n >>= two;
```

```
    if(n == 1) return v;
    while(!IsPrime(n)){
        ll d = Rho(n), cnt = 0;
        while(n % d == 0) cnt++, n /= d;
        v.emplace_back(d, cnt);
        if(n == 1) break;
    }
    if(n != 1) v.emplace_back(n, 1);
    return v;
}
int main() {
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    ll input; cin >> input;
    Sieve();
    // 입력된 수를 소인수분해
    vector<pair<ll,ll>> factors = Factorize(input);
    sort(factors.begin(), factors.end());
    // 결과 출력
    for (const auto& factor : factors) {
        for (ll i = 0; i < factor.second; ++i) cout << factor.first << '\n';
    }
    return 0;
}
```

1.4 중국인의 나머지 정리

```
// 문자열 패턴 매칭 알고리즘, Rolling- Hash 기법
const int BASE = 401, MOD = 1e9 + 7, MAX = 1e6 + 6;
int HASH, len, HASH2, POWER[MAX];

void solve() {
     POWER[0] = 1;
     for (int i = 1; i < MAX; i++) POWER[i] = md(MOD, POWER[i - 1] * BASE);
     string s, t;
     getline(cin, s);
     getline(cin, t);
     len = sz(t);
     if (sz(s) < len) {
      cout << 0;
        return;
     }
     for (int i = len - 1; i >= 0; i--) {
        HASH = md(MOD, HASH + t[i] * POWER[len - 1 - i]);
        HASH2 = md(MOD, HASH2 + s[i] * POWER[len - 1 - i]);
     }
     vi ans;
     if (HASH == HASH2) ans.pb(0);
     for (int i = 1; i + len - 1 < sz(s); i++) {
        HASH2 = md(MOD, md(MOD, HASH2 - s[i - 1] * POWER[len - 1]) * BASE + s[i + len - 1]);
        if (HASH == HASH2) ans.pb(i);
     }
     cout << sz(ans) << endl;
     for (int i: ans) cout << i + 1 << ' ';
}
int main() {
  return 0;
}
```

1.5 오일러 피 함수

```
// LCA -> main 함수는 boj 11437(https://www.acmicpc.net/problem/11437)
template<typename ValueType, typename IndexType>
struct LCA {
    int N;
    int MAXLN;
    vector<vector<int>> tree;
    vector<vector<int>> par;
    vector<int> depth;
    vector<ValueType> node_values;

    unordered_map<IndexType, int> idx_map;
    vector<IndexType> idx_reverse_map;

    int idx_counter;

    // 생성자
    LCA() : idx_counter(0) {
    }

    int get_idx(IndexType idx) {
        auto it = idx_map.find(idx);
        if(it != idx_map.end()) {
            return it->second;
        } else {
            int new_idx = idx_counter++;
            idx_map[idx] = new_idx;
            idx_reverse_map.push_back(idx);
            tree.emplace_back();
            depth.push_back(0);
            node_values.emplace_back();
            return new_idx;
        }
    }

    // LCA 구성 간선 추가.
    void add_edge(IndexType u_idx, IndexType v_idx) {
        int u = get_idx(u_idx);
        int v = get_idx(v_idx);
        tree[u].push_back(v);
        tree[v].push_back(u);
    }

    // 노드에 값 부여
    void set_node_value(IndexType idx, ValueType value) {
        int u = get_idx(idx);
        node_values[u] = value;
    }

    void dfs(int node, int parent) {
        for(int next : tree[node]) {
            if(next == parent) continue;
            depth[next] = depth[node] + 1;
            par[0][next] = node;
            dfs(next, node);
        }
```

```
    }

    // LCA 전처리 -> LCA 쿼리 날리기 전에 호출 필수
    void prepare_LCA(IndexType root_idx) {
        N = idx_counter;
        MAXLN = ceil(log2(N)) + 1;
        par.assign(MAXLN, vector<int>(N));
        int root = get_idx(root_idx);
        depth[root] = 0;
        par[0][root] = root;
        dfs(root, -1);
        for(int i = 1; i < MAXLN; i++) {
            for(int j = 0; j < N; j++) {
                par[i][j] = par[i - 1][par[i - 1][j]];
            }
        }
    }

    // LCA 구하기
    IndexType LCA_query(IndexType u_idx, IndexType v_idx) {
        int u = get_idx(u_idx);
        int v = get_idx(v_idx);

        if(depth[u] < depth[v]) swap(u, v);
        for(int i = MAXLN - 1; i >= 0; i--) {
            if(depth[u] - (1 << i) >= depth[v]) {
                u = par[i][u];
            }
        }
        if(u == v) return idx_reverse_map[u];
        for(int i = MAXLN - 1; i >= 0; i--) {
            if(par[i][u] != par[i][v]) {
                u = par[i][u];
                v = par[i][v];
            }
        }
        return idx_reverse_map[par[0][u]];
    }

    // 노드에 들어있는 값 참조
    ValueType get_node_value(IndexType idx) {
        int u = get_idx(idx);
        return node_values[u];
    }
};
int main() {
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    LCA<int, int> lca;
    int n; cin >> n;
    for(int i = 0; i < n - 1; i++) {
        int u, v; cin >> u >> v;
        lca.add_edge(u, v);
    }
    lca.prepare_LCA(1);
    int m; cin >> m;
    for(int i = 0; i < m; i++) {
        int u, v; cin >> u >> v;
        cout << lca.LCA_query(u, v) << "\n";
    }

    return 0;
}
```

1.6 행렬곱셈 (Strassen's algorithm)

```
/*
O(r^(log7=2.81)) nxm * mxk 행렬곱에서 max(n,m,k)에 가장가깝고 더 큰 2^x값이 r이 된다
정사각 행렬을 큰 차원의 행렬에서 기존의 O(n^3)보다 조금 더 빠름, main함수 예제는 BOJ 2704
*/
typedef long long ll;
typedef vector<vector<ll>> Matrix;

// 행렬 덧셈
Matrix add(const Matrix &A, const Matrix &B) {
    ll n = A.size();
    ll m = A[0].size();
    Matrix C(n, vector<ll>(m, 0));
    for (ll i = 0; i < n; ++i) {
        for (ll j = 0; j < m; ++j) {
            C[i][j] = A[i][j] + B[i][j]; //여기 마이너스로 바꾸면 뺄셈됨 (subtract)
        }
    }
    return C;
}

// 행렬 뺄셈
Matrix subtract(const Matrix &A, const Matrix &B) {
    ll n = A.size();
    ll m = A[0].size();
    Matrix C(n, vector<ll>(m, 0));
    for (ll i = 0; i < n; ++i) {
        for (ll j = 0; j < m; ++j) {
            C[i][j] = A[i][j] - B[i][j];
        }
    }
    return C;
}

// 행렬 크기를 2^n 크기로 맞추는 함수
Matrix resizeMatrix(const Matrix &A, ll newSize) {
    ll oldRows = A.size();
    ll oldCols = A[0].size();
    Matrix resized(newSize, vector<ll>(newSize, 0));

    for (ll i = 0; i < oldRows; ++i) {
        for (ll j = 0; j < oldCols; ++j) {
            resized[i][j] = A[i][j];
        }
    }
    return resized;
}

// Strassen 알고리즘을 사용한 행렬 곱셈
```

```cpp
Matrix strassen(const Matrix &A, const Matrix &B) {
    ll n = A.size();

    if (n == 1) {
        return {{A[0][0] * B[0][0]}};
    }

    // 행렬을 4개의 하위 블록으로 나눔
    ll half = n / 2;
    Matrix A11(half, vector<ll>(half)), A12(half, vector<ll>(half)),
           A21(half, vector<ll>(half)), A22(half, vector<ll>(half)),
           B11(half, vector<ll>(half)), B12(half, vector<ll>(half)),
           B21(half, vector<ll>(half)), B22(half, vector<ll>(half));

    // 행렬 분할
    for (ll i = 0; i < half; ++i) {
        for (ll j = 0; j < half; ++j) {
            A11[i][j] = A[i][j];
            A12[i][j] = A[i][j + half];
            A21[i][j] = A[i + half][j];
            A22[i][j] = A[i + half][j + half];

            B11[i][j] = B[i][j];
            B12[i][j] = B[i][j + half];
            B21[i][j] = B[i + half][j];
            B22[i][j] = B[i + half][j + half];
        }
    }

    // Strassen의 7개의 중간 행렬 계산
    Matrix M1 = strassen(add(A11, A22), add(B11, B22));
    Matrix M2 = strassen(add(A21, A22), B11);
    Matrix M3 = strassen(A11, subtract(B12, B22));
    Matrix M4 = strassen(A22, subtract(B21, B11));
    Matrix M5 = strassen(add(A11, A12), B22);
    Matrix M6 = strassen(subtract(A21, A11), add(B11, B12));
    Matrix M7 = strassen(subtract(A12, A22), add(B21, B22));

    // 결과 행렬 조합
    Matrix C(n, vector<ll>(n, 0));
    for (ll i = 0; i < half; ++i) {
        for (ll j = 0; j < half; ++j) {
            C[i][j] = M1[i][j] + M4[i][j] - M5[i][j] + M7[i][j];           // C11
            C[i][j + half] = M3[i][j] + M5[i][j];                           // C12
            C[i + half][j] = M2[i][j] + M4[i][j];                           // C21
            C[i + half][j + half] = M1[i][j] - M2[i][j] + M3[i][j] + M6[i][j]; // C22
        }
    }

    return C;
}

// 새로운 함수: 행렬 크기 맞추기 및 Strassen 알고리즘 실행
Matrix prepareAndExecuteStrassen(const Matrix &A, const Matrix &B, ll N, ll M, ll K) {
    // 행렬 크기를 2의 제곱수로 맞추기 위해 새로운 크기 계산
    ll maxSize = max({N, M, K});
    ll newSize = 1;
    while (newSize < maxSize) newSize *= 2;

    // 행렬 크기를 2^n 크기로 맞추기 위한 함수 호출
    Matrix A_resized = resizeMatrix(A, newSize);
    Matrix B_resized = resizeMatrix(B, newSize);

    // Strassen 알고리즘 실행
    Matrix C_resized = strassen(A_resized, B_resized);

    // 결과 행렬을 원래의 크기로 자르기
    Matrix C_final(N, vector<ll>(K, 0));
    for (ll i = 0; i < N; ++i) {
        for (ll j = 0; j < K; ++j) {
            C_final[i][j] = C_resized[i][j];
        }
    }

    return C_final;
}

// 행렬 곱셈 문제 해결을 위한 main 함수
int main() {
    ll N, M, K;

    // 행렬 A 입력 받기
    cin >> N >> M;
    Matrix A(N, vector<ll>(M));
    for (ll i = 0; i < N; ++i) {
        for (ll j = 0; j < M; ++j) {
            cin >> A[i][j];
        }
    }

    // 행렬 B 입력 받기
    cin >> M >> K;
    Matrix B(M, vector<ll>(K));
    for (ll i = 0; i < M; ++i) {
        for (ll j = 0; j < K; ++j) {
            cin >> B[i][j];
        }
    }

    // 행렬 크기를 맞추고 Strassen 알고리즘 실행
    Matrix C = prepareAndExecuteStrassen(A, B, N, M, K);

    // 결과 출력
    for (ll i = 0; i < N; ++i) {
        for (ll j = 0; j < K; ++j) {
            cout << C[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

1.7 FFT

```cpp
// Fast Fourier Transform(FFT, 고속 푸리에 변환), O(N log N)
const double PI = acos(-1);
typedef complex<double> cpx;
typedef long long ll;
/*
input : f => Coefficient, w => principal n-th root of unity
output : f => f(x_0), f(x_1), f(x_2), ... , f(x_n-1)
T(N) = 2T(N/2) + O(N)
*/
void FFT(vector<cpx> &v, bool inv) {
    ll S = v.size(); // ll 타입으로 선언

    for(ll i=1, j=0; i<S; i++) {
        ll bit = S >> 1;
        while(j >= bit) {
            j -= bit;
            bit >>= 1;
        }
        j += bit;
        if(i < j) swap(v[i], v[j]);
    }

    for(ll k=1; k<S; k<<=1) {
        double angle = inv ? PI/k : -PI/k;
        cpx dir(cos(angle), sin(angle));
        for(ll i=0; i<S; i+=(k<<1)) {
            cpx unit(1, 0);
            for(ll j=0; j<k; j++) {
                cpx a = v[i+j], b = v[i+j+k] * unit;
                v[i+j] = a + b;
                v[i+j+k] = a - b;
                unit *= dir;
            }
        }
    }

    if(inv) {
        for(ll i=0; i<S; i++) v[i] /= S;
    }
}
/*
input : a => A's Coefficient, b => B's Coefficient
output : A * B
*/
vector<cpx> mul(vector<cpx> &v, vector<cpx> &u) {
    ll S = 1;
    while(S < max(v.size(), u.size())) S <<= 1;
    S <<= 1; // 벡터의 길이를 조정하여 곱셈을 위한 충분한 길이를 확보

    v.resize(S); FFT(v, false);
    u.resize(S); FFT(u, false);

    vector<cpx> w(S);
    for(ll i=0; i<S; i++) w[i] = v[i] * u[i];
    FFT(w, true); // 역 FFT 수행
    return w;
}

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

    int N; cin >> N;
    vector<cpx> v(N*2), u(N);

    for(int i=0; i<N; i++) {
        int x; cin >> x;
        v[i] = v[i+N] = cpx(x, 0);
    }
    for(int i=0; i<N; i++) {
        int x; cin >> x;
        u[N-1-i] = cpx(x, 0);
    }

    vector<cpx> w = mul(v, u);

    int ans = 0;
    for(int i=0; i<w.size(); i++) ans = max(ans, (int)round(w[i].real()));

    cout << ans << "\n";
}
```

2 문자열

2.1 KMP

```cpp
// 일대일 문자열 패턴 매칭 알고리즘; 문자열 S, 패턴 P라면, O(S + P) 시간에 찾는다.
vector<int> kmp_fail(const string &s) {
    int sz = s.length();
    vector<int> fail(sz);

    for (int i = 1, j = 0; i < sz; i++) {
        while (j && s[i] != s[j]) j = fail[j - 1];
        if (s[i] == s[j]) fail[i] = ++j;
    }
    return fail;
}
// S에서, P가 몇 번 등장하는지를 구하는 kmp 함수.
// 0-index이므로, 1부터 인덱스가 시작한다면, match_index에 1을 더해 줘야 한다.
vector<int> kmp(const string &a, const string &b) {
    int sz_a = a.length(); int sz_b = b.length();
    vector<int> fail = kmp_fail(b), match_index;
    for (int i = 0, j = 0; i < sz_a; i++) {
        while (j && a[i] != b[j]) j = fail[j - 1];
        if (a[i] == b[j]) {
            if (j == sz_b - 1) {
                match_index.push_back(i - sz_b + 1);
                j = fail[j];
            } else j++;
        }
    }
    return match_index;
}
```

```cpp
int main() {
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    string S, W;
    getline(cin, S); getline(cin, W);
    vector<int> res = kmp(S, W);
    cout << res.size() << "\n";
    for(const auto& ele : res) cout << ele + 1 << "\n";
    return 0;
}
```

2.2 Trie

2.3 Manacher

```cpp
//가장 긴 팰린드롬을 O(n)에 찾는 알고리즘
int manacher(string inputStr){
  int strLen = inputStr.size(), r = 0, p = 0, ret = 0;
  string str = " ";
  for(int i = 0; i < strLen; i++){
    str += inputStr[i];
    str += " ";
  }
  strLen = str.size();
  vector<int> A(strLen + 1);
  for(int i = 0; i < strLen; i++){
    if(i <= r)  A[i] = min(r - i, A[2 * p - i]);
    while(0 <= i - A[i] - 1 && i + A[i] + 1 < strLen && str[i - A[i] - 1] == str[i + A[i] +
    1]) A[i]++;
    if(r < i + A[i]){
      r = i + A[i];
      p = i;
    }
  }

  for(int i = 1; i <= strLen; i++){
    ret = max(ret, A[i]);
  }

  return ret;
}

int main(){
  cin.tie(nullptr);
  ios::sync_with_stdio(false);
  string input;
  cin >> input;
  cout << manacher(input);
  return 0;
}
```

2.4 Z

```cpp
//S[i...] 의 prefix 들 중 S의 prefix이기도 한 녀석들 중 길이가 가장 긴 것의 길이에 대한 배열을 O(N)
에 구하기
void getZarray(vector<int> &zValue, string str){
    int n = str.size(), l = 0, r = 0;
    zValue.resize(str.size() + 1);
    reverse(str.begin(), str.end());
  zValue[0] = str.size();
  for(int i = 1; i < n; i++){
        if(r < i){
            zValue[i] = 0;
            while(str[zValue[i]] == str[i + zValue[i]]) zValue[i]++;
            l = i;
            r = i + zValue[i] - 1;
        }
        else{
            if(zValue[i - l] <= r - i)  zValue[i] = zValue[i - l];
      else{
                zValue[i] = r - i + 1;
        while(str[zValue[i]] == str[i + zValue[i]]) zValue[i]++;
        l = i;
        r = i + zValue[i] - 1;
      }
        }
  }
}

int main(){
    int q, x;
    cin.tie(nullptr); ios::sync_with_stdio(false);
    string str;
    vector<int> result;
    cin >> str;
    getZarray(result, str);
    cin >> q;
    while(q--){
        cin >> x;
        cout << result[str.size() - x] << '\n';
    }
    return 0;
}
```

2.5 Suffix Array

```cpp
string input;

int n;
int t;

vector<int>suffix, cur, nextGroup, lcp, revSuffix;

bool cmp(const int &a, const int &b){
    if(cur[a] == cur[b])    return cur[a + t] < cur[b + t];
    else return cur[a] < cur[b];
}

void getSuffix(){
    suffix.resize(n);
    cur.resize(n + 1);
    nextGroup.resize(n + 1);
    for(int i = 0; i < n; i++){
        suffix[i] = i;
```

```
        cur[i] = input[i] - 'a';
    }
    cur[n] = -1;
    for(t = 1; t <= n; t <<= 1){
        sort(suffix.begin(), suffix.end(), cmp);
        nextGroup[n] = -1;
        nextGroup[suffix[0]] = 0;
        for(int i = 1; i < n; i++){
            if(cmp(suffix[i - 1], suffix[i]))    nextGroup[suffix[i]] = nextGroup[suffix[i -
            1]] + 1;
            else nextGroup[suffix[i]] = nextGroup[suffix[i - 1]];
        }
        for(int i = 0; i < n; i++){
            cur[i] = nextGroup[i];
        }
    }
}

void getLcp(){
    int k = 0;
    lcp.resize(n + 1);
    revSuffix.resize(n + 1);
    for(int i = 0; i < n; i++){
        revSuffix[suffix[i]] = i;
    }

    for(int i = 0; i < n; i++){
        if(revSuffix[i]){
            int j = suffix[revSuffix[i] - 1];
            while(input[i + k] == input[j + k])    k++;
            lcp[revSuffix[i]] = k;
            if(k)    k--;
        }
    }
}

int main(){
    cin.tie(0); ios::sync_with_stdio(0);
    cin >> input;
    n = input.size();
    getSuffix();
    getLcp();
    for(int i = 0; i < n; i++){
        cout << suffix[i] + 1 << " ";
    }
    cout << "\nx ";
    for(int i = 1; i < n; i++){
        cout << lcp[i] << " ";
    }
    return 0;
}
```

2.6 Aho-Corasick

```
// 일대다 문자열 패턴 매칭 알고리즘, 문자열의 길이 N, k개의 패턴, 각각의 길이 m[i] (1 <= i <= k),
O(N + sigma(m[i], i : 1 ~ k))
struct Trie {
    Trie *go[26];
    Trie *fail;
    bool output;

    Trie() {
        fill(go, go + 26, nullptr);
        output = false;
    }
    ~Trie() {
        for(int i = 0; i < 26; i++)
            if(go[i]) delete go[i];
    }
    void insert(const char* key) {
        if(*key == '\0'){
            output = true;
            return;
        }
        int next = *key - 'a';
        if(!go[next]) {
            go[next] = new Trie;
        }
        go[next]->insert(key + 1);
    }
};

void buildTrieAndFailureLinks(Trie *root, int N) {
    for(int i = 0; i < N; i++) {
        char str[10001];
        cin >> str;
        root->insert(str);
    }

    queue<Trie*> Q;
    root->fail = root;
    Q.push(root);
    while(!Q.empty()) {
        Trie *current = Q.front();
        Q.pop();

        for(int i = 0; i < 26; i++) {
            Trie *next = current->go[i];
            if(!next) continue;

            if(current == root) next->fail = root;
            else {
                Trie *dest = current->fail;
                while(dest != root && !dest->go[i])
                    dest = dest->fail;
                if(dest->go[i]) dest = dest->go[i];
                next->fail = dest;
            }
            if(next->fail->output) next->output = true;
            Q.push(next);
        }
    }
}
```

```
bool searchInTrie(Trie *root, const char *str) {
    Trie* current = root;
    for(int c = 0; str[c]; c++){
        int next = str[c] - 'a';
        while(current != root && !current->go[next])
            current = current->fail;
        if(current->go[next])
            current = current->go[next];
        if(current->output)
            return true;
    }
    return false;
}

int main() {
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int N, M;
    Trie* root = new Trie;

    cin >> N;
    buildTrieAndFailureLinks(root, N);

    cin >> M;
    for(int i = 0; i < M; i++) {
        char str[10001];
        cin >> str;
        cout << (searchInTrie(root, str) ? "YES" : "NO") << "\n";
    }

    delete root;
}
```

2.7 Rabin-Karp

```
// 문자열 패턴 매칭 알고리즘, Rolling- Hash 기법
const int BASE = 401, MOD = 1e9 + 7, MAX = 1e6 + 6;
int HASH, len, HASH2, POWER[MAX];

void solve() {
     POWER[0] = 1;
     for (int i = 1; i < MAX; i++) POWER[i] = md(MOD, POWER[i - 1] * BASE);
     string s, t;
     getline(cin, s);
     getline(cin, t);
     len = sz(t);
     if (sz(s) < len) {
      cout << 0;
        return;
     }
     for (int i = len - 1; i >= 0; i--) {
        HASH = md(MOD, HASH + t[i] * POWER[len - 1 - i]);
        HASH2 = md(MOD, HASH2 + s[i] * POWER[len - 1 - i]);
     }
     vi ans;
     if (HASH == HASH2) ans.pb(0);
     for (int i = 1; i + len - 1 < sz(s); i++) {
        HASH2 = md(MOD, md(MOD, HASH2 - s[i - 1] * POWER[len - 1]) * BASE + s[i + len - 1]);
        if (HASH == HASH2) ans.pb(i);
     }
     cout << sz(ans) << endl;
     for (int i: ans) cout << i + 1 << ' ';
}
int main() {
  return 0;
}
```

3 그래프

3.1 SCC

```
//SCC 타잔 알고리즘 구현 (2-SAT에 응용할 때, x1 v x2 를 ¬x1 → x2, ¬x2 → x1 로 변환하여 그래프를 그린 다음)
//¬x1과 x1이 같은 SCC 안에 있으면 모순. (sccNo[2 * i] == sccNo[2 * i - 1])
int id, visited[10001], sccNo[4004];
bool finished[10001];
vector<int> graph[10001];
vector<vector<int> >SCC;
stack<int> S;

int v, e;

/*2-SAT 에서 필요한 index 매핑 함수*/
int getIdx(int k){
  return k > 0 ? 2 * k : 2 * (-k) - 1;
}

int dfs(int cur){
    visited[cur] = ++id;  //고유의 id값 할당
    S.push(cur);
    int parent = visited[cur];
    //가장 높은 위치를 parent에 저장
    for(int next:graph[cur]){
        if(!visited[next]){   //탐색되지 않은 경우
            parent = min(parent, dfs(next));
        }
        else if(!finished[next]){   //탐색이 되었으나 SCC에 포함이 되지 않은 경우
            parent = min(parent, visited[next]);
        }
    }
    //자기 자신이 부모인 경우 (자기 자신이 SCC에서 가장 높은 위치인 경우)
    if(parent == visited[cur]){
        vector<int> group;
        while(1){   //자기 자신이 나올때까지 stack에서 pop하고 SCC에 저장
            int p = S.top();
            S.pop();
            group.push_back(p);
            sccNo[p] = SCC.size() + 1;
            finished[p] = true; //SCC에 포함되었음을 체크
            if(p == cur)    break;
        }
        sort(group.begin(), group.end());
        SCC.push_back(group);
    }
    return parent;
```

```
}

int main(){
    cin.tie(nullptr); ios::sync_with_stdio(false);

    cin >> v >> e;
    int x, y;
    for(int i = 0; i < e; i++){
        cin >> x >> y;
        graph[x].push_back(y);
    }

    for(int i = 1; i <= v; i++){
        if(!visited[i])   dfs(i);
    }

    cout << SCC.size() << "\n";
    for(int i = 0 ; i < SCC.size(); i++){
        for(int data : SCC[i])    cout << data << " ";
        cout << "\n";
    }

    return 0;
}
```

3.2 BCC (단절점)

```
//단절점 탐색 dfs
//root 이면서 자식 수 2개 이상, 해당 정점을 거치지 않고 빠른 방문 번호를 가진 정점으로 이동 불가
const int MAXSIZE = 1e4 + 1;
int v, e;

vector<int> graph[MAXSIZE];
vector<int> ans;
int visited[MAXSIZE], cnt; //방문 여부 및 순서 저장
bool isArt[MAXSIZE]; //단절점(Articulation Point) 여부 저장

int dfs(int cur, bool isRoot){
  visited[cur] = ++cnt; //정점 방문 순서 저장
  int ret = visited[cur], prev, child = 0;
  for(int next : graph[cur]){
    if(!visited[next]){
      child++;
      prev = dfs(next, false);
      ret = min(ret, prev);
            //자식 노드들이 정점 A를 거치지 않고 정점 A보다 빠른 방문번호를 가진 정점으로 이동 불가
      if(!isRoot && prev >= visited[cur]){//해당 조건을 만족시 cur-next는 단절선
        isArt[cur] = true;
      }
    }
    else{
      ret = min(ret, visited[next]);
    }
  }
    //단절선 찾으려면 필요 없음
  if(isRoot && child >= 2){//루트 node이고 자식의 수가 2개 이상이면
    isArt[cur] = true;
  }
  return ret;
}

int main(){
    cin.tie(nullptr);
    ios::sync_with_stdio(false);
    cin >> v >> e;
  int x, y;
  for(int i = 0; i < e; i++){
    cin >> x >> y;
    graph[x].push_back(y);
    graph[y].push_back(x);
  }

  for(int i = 1; i <= v; i++) {
    if(!visited[i])  dfs(i, true);
  }

  for(int i = 1; i <= v; i++) {
    if(isArt[i])  ans.push_back(i);
  }

  sort(ans.begin(), ans.end());
  cout << ans.size() << "\n";
  for(int p : ans){
    cout << p << " ";
  }
    return 0;
}
```

3.3 LCA

```
// LCA -> main 함수는 boj 11437(https://www.acmicpc.net/problem/11437)
template<typename ValueType, typename IndexType>
struct LCA {
    int N;
    int MAXLN;
    vector<vector<int>> tree;
    vector<vector<int>> par;
    vector<int> depth;
    vector<ValueType> node_values;

    unordered_map<IndexType, int> idx_map;
    vector<IndexType> idx_reverse_map;

    int idx_counter;

    // 생성자
    LCA() : idx_counter(0) {
    }

    int get_idx(IndexType idx) {
        auto it = idx_map.find(idx);
        if(it != idx_map.end()) {
            return it->second;
        } else {
```

```
            int new_idx = idx_counter++;
            idx_map[idx] = new_idx;
            idx_reverse_map.push_back(idx);
            tree.emplace_back();
            depth.push_back(0);
            node_values.emplace_back();
            return new_idx;
        }
    }

    // LCA 구성 간선 추가.
    void add_edge(IndexType u_idx, IndexType v_idx) {
        int u = get_idx(u_idx);
        int v = get_idx(v_idx);
        tree[u].push_back(v);
        tree[v].push_back(u);
    }

    // 노드에 값 부여
    void set_node_value(IndexType idx, ValueType value) {
        int u = get_idx(idx);
        node_values[u] = value;
    }

    void dfs(int node, int parent) {
        for(int next : tree[node]) {
            if(next == parent) continue;
            depth[next] = depth[node] + 1;
            par[0][next] = node;
            dfs(next, node);
        }
    }

    // LCA 전처리 -> LCA 쿼리 날리기 전에 호출 필수
    void prepare_LCA(IndexType root_idx) {
        N = idx_counter;
        MAXLN = ceil(log2(N)) + 1;
        par.assign(MAXLN, vector<int>(N));
        int root = get_idx(root_idx);
        depth[root] = 0;
        par[0][root] = root;
        dfs(root, -1);
        for(int i = 1; i < MAXLN; i++) {
            for(int j = 0; j < N; j++) {
                par[i][j] = par[i - 1][par[i - 1][j]];
            }
        }
    }

    // LCA 구하기
    IndexType LCA_query(IndexType u_idx, IndexType v_idx) {
        int u = get_idx(u_idx);
        int v = get_idx(v_idx);

        if(depth[u] < depth[v]) swap(u, v);
        for(int i = MAXLN - 1; i >= 0; i--) {
            if(depth[u] - (1 << i) >= depth[v]) {
                u = par[i][u];
            }
        }
        if(u == v) return idx_reverse_map[u];
        for(int i = MAXLN - 1; i >= 0; i--) {
            if(par[i][u] != par[i][v]) {
                u = par[i][u];
                v = par[i][v];
            }
        }
        return idx_reverse_map[par[0][u]];
    }

    // 노드에 들어있는 값 참조
    ValueType get_node_value(IndexType idx) {
        int u = get_idx(idx);
        return node_values[u];
    }
};
int main() {
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    LCA<int, int> lca;
    int n; cin >> n;
    for(int i = 0; i < n - 1; i++) {
        int u, v; cin >> u >> v;
        lca.add_edge(u, v);
    }
    lca.prepare_LCA(1);
    int m; cin >> m;
    for(int i = 0; i < m; i++) {
        int u, v; cin >> u >> v;
        cout << lca.LCA_query(u, v) << "\n";
    }

    return 0;
}
```

3.4 이분매칭 (hopcroft-karp)

```
//O(sqrt(V) * E); 적정히 조정
const int MAXN = 2005, MAXM = 1005;

template <typename MatchType>
class HopcroftKarp {
    vector<int> gph[MAXN];
    int dis[MAXN], vis[MAXN];
    vector<MatchType> l, r;

    bool bfs(int n) {
        queue<int> que;
        bool ok = 0;
        memset(dis, 0, sizeof(dis));
        for(int i = 0; i < n; i++) {
            if(l[i] == -1 && !dis[i]) {
                que.push(i);
                dis[i] = 1;
```

```
            }
        }
        while(!que.empty()) {
            int x = que.front();
            que.pop();
            for(auto &i : gph[x]) {
                if(r[i] == -1) ok = 1;
                else if(!dis[r[i]]) {
                    dis[r[i]] = dis[x] + 1;
                    que.push(r[i]);
                }
            }
        }
        return ok;
    }

    bool dfs(int x) {
        if(vis[x]) return 0;
        vis[x] = 1;
        for(auto &i : gph[x]) {
            if(r[i] == -1 || (!vis[r[i]] && dis[r[i]] == dis[x] + 1 && dfs(r[i]))) {
                l[x] = i;
                r[i] = x;
                return 1;
            }
        }
        return 0;
    }

public:
    HopcroftKarp() { l.resize(MAXN, -1); r.resize(MAXM, -1); }

    void clear() {
        for(int i = 0; i < MAXN; i++) gph[i].clear();
    }

    void addEdge(int l, int r) {
        gph[l].push_back(r);
    }
    // n에 들어가는 것은, 왼쪽 정점 집합 L의 크기이다.
    MatchType match(int n) {
        MatchType ret = 0;
        while(bfs(n)) {
            memset(vis, 0, sizeof(vis));
            for(int i = 0; i < n; i++) if(l[i] == -1 && dfs(i)) ret++;
        }
        return ret;
    }
};
int main() {
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int N, M; cin >> N >> M;
    HopcroftKarp<int> hk;
    for(int i = 0; i < N; i++) {
        int S; cin >> S;
        for(int j = 0; j < S; j++) {
            int k; cin >> k; k--;
            // 두 번의 노드에 모두 해당 일을 연결
            hk.addEdge(i, k);
            hk.addEdge(i + N, k);
        }
    }
    cout << hk.match(2 * N);
    return 0;
}
```

3.5 최대유량 (디닉)

```
// O(V^2E); Additional Implementation(민컷구하기 + 포화 간선 유량 구하기)
typedef long long ll;

template<typename FlowType>
struct Dinic {
    struct Edge {
        int v, rev;
        FlowType flow, cap;
    };

    int V;
    vector<int> level;
    vector<vector<Edge>> adj;
    map<pair<int, int>, int> edgeIndexMap;

    Dinic(int V) : V(V), adj(V), level(V) {}

    void addEdge(int u, int v, FlowType cap) {
        edgeIndexMap[{u, v}] = adj[u].size();
        Edge forward = {v, (int)adj[v].size(), 0, cap};
        Edge reverse = {u, (int)adj[u].size(), 0, 0};
        adj[u].push_back(forward);
        adj[v].push_back(reverse);
    }

    bool BFS_level_graph(int s, int t) {
        fill(level.begin(), level.end(), -1);
        level[s] = 0;
        queue<int> q;
        q.push(s);
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (Edge &e : adj[u]) {
                if (level[e.v] < 0 && e.flow < e.cap) {
                    level[e.v] = level[u] + 1;
                    q.push(e.v);
                }
            }
        }
        return level[t] >= 0;
    }

    FlowType DFS_blocking_flow(int u, FlowType flow, int t, vector<int> &start) {
        if (u == t) return flow;
```

```
        for (; start[u] < adj[u].size(); start[u]++) {
            Edge &e = adj[u][start[u]];
            if (level[e.v] == level[u] + 1 && e.flow < e.cap) {
                FlowType curr_flow = min(flow, e.cap - e.flow);
                FlowType temp_flow = DFS_blocking_flow(e.v, curr_flow, t, start);

                if (temp_flow > 0) {
                    e.flow += temp_flow;
                    adj[e.v][e.rev].flow -= temp_flow;
                    return temp_flow;
                }
            }
        }
        return 0;
    }

    FlowType Maxflow(int s, int t) {
        FlowType total = 0;
        while (BFS_level_graph(s, t)) {
            vector<int> start(V);
            while (FlowType flow = DFS_blocking_flow(s, numeric_limits<FlowType>::max(), t,
            start)) {
                total += flow;
            }
        }
        return total;
    }

    tuple<FlowType, vector<int>, vector<int>, vector<pair<int, int>>> getMincut(int s, int
    t) {
        FlowType maxflow = Maxflow(s, t);
        vector<int> S, T;
        vector<pair<int, int>> saturated_edges;
        BFS_level_graph(s, t);
        for(int i = 0; i < V; i++) (level[i] != -1 ? S : T).push_back(i);
        for(auto i : S) for(auto e : adj[i]) if(e.cap != 0 && level[e.v] == -1)
        saturated_edges.emplace_back(i, e.v);
        return {maxflow, S, T, saturated_edges};
    }

    FlowType getFlow(int u, int v) {
        auto it = edgeIndexMap.find({u, v});
        if (it != edgeIndexMap.end()) {
            return adj[u][it->second].flow;
        }
        return -1;
    }

    void clear() {
        for(int i = 0; i < V; i++) adj[i].clear();
        fill(level.begin(), level.end(), 0);
    }
};
int main() {
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int N, M; cin >> N >> M;

    Dinic<int> g(N + M + 2);
    int s = 0, t = N + M + 1;
    for(int i = 1; i <= N; i++) {
        int a; cin >> a;
        g.addEdge(s, i, a);
    }
    for(int i = 1; i <= M; i++) {
        int b; cin >> b;
        g.addEdge(N + i, t, b);
    }
    for(int i = 1; i <= M; i++) {
        for(int j = 1; j <= N; j++) {
            int c; cin >> c;
            g.addEdge(j, N + i, c);
        }
    }
    cout << g.Maxflow(s, t) << "\n";
    cout << "Additional implementation" << "\n";
    // 최소 컷 계산
    auto [maxflow, S, T, saturated_edges] = g.getMincut(s, t);
    // 결과 출력
    cout << "Maxflow: " << maxflow << "\n";
    cout << "S: ";
    for (auto v : S) cout << v << " ";
    cout << "\nT: ";
    for (auto v : T) cout << v << " ";
    cout << "\nSaturated Edges: \n";
    for (auto &[u, v] : saturated_edges) cout << "(" << u << ", " << v << ") , flow : " <<
    g.getFlow(u, v) << "\n";
    cout << "\n";
    return 0;
}
```

3.6 MCMF + SPFA

```
template<typename FlowType, typename CostType>
class MCMF {
    struct Edge {
        int to;
        FlowType capacity;
        CostType cost;
        int rev;
    };

    int V;
    vector<vector<Edge>> adj;
    vector<CostType> dist;
    vector<int> previous, edgeIndex;

public:
    MCMF(int V) : V(V), adj(V) {}

    void addEdge(int from, int to, FlowType cap, CostType cost) {
        adj[from].push_back({to, cap, cost, (int)adj[to].size()});
        adj[to].push_back({from, 0, -cost, (int)adj[from].size() - 1});
```

```
}

bool SPFA(int src, int sink) {
    dist.assign(V, numeric_limits<CostType>::max());
    previous.assign(V, -1);
    edgeIndex.assign(V, -1);
    vector<bool> inQueue(V, false);

    queue<int> Q;
    dist[src] = 0;
    Q.push(src);
    inQueue[src] = true;

    while (!Q.empty()) {
        int u = Q.front();
        Q.pop();
        inQueue[u] = false;

        for(int i = 0; i < adj[u].size(); ++i) {
            Edge &e = adj[u][i];
            if(e.capacity > 0 && dist[u] + e.cost < dist[e.to]) {
                dist[e.to] = dist[u] + e.cost;
                previous[e.to] = u;
                edgeIndex[e.to] = i;

                if(!inQueue[e.to]) {
                    Q.push(e.to);
                    inQueue[e.to] = true;
                }
            }
        }
    }
    return previous[sink] != -1;
}
pair<FlowType, CostType> getMCMF(int src, int sink) {
    FlowType maxFlow = 0;
    CostType minCost = 0;

    while (SPFA(src, sink)) {
        FlowType flow = numeric_limits<FlowType>::max();
        for(int v = sink; v != src; v = previous[v]) {
            int u = previous[v];
            flow = min(flow, adj[u][edgeIndex[v]].capacity);
        }
        for(int v = sink; v != src; v = previous[v]) {
            int u = previous[v];
            Edge &e = adj[u][edgeIndex[v]];
            e.capacity -= flow;
            adj[v][e.rev].capacity += flow;
            minCost += flow * e.cost;
        }

        maxFlow += flow;
    }
    return {maxFlow, minCost};
}
};
int main() {
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int N;
    cin >> N;
    vector<int> A(N + 1, 0), H(N + 1, 0), L(N + 1, 0);
    MCMF<int, int> mcmf(2 * N + 2);

    for(int i = 1; i <= N; i++) cin >> A[i];
    for(int i = 1; i <= N; i++) cin >> H[i];
    for(int i = 1; i <= N; i++) cin >> L[i];

    int src = 0, sink = 2 * N + 1;
    int max_A = *max_element(A.begin() + 1, A.end());

    for(int i = 1; i <= N; i++) {
        if(A[i] == max_A) {
            mcmf.addEdge(src, i, L[i], H[i]);
            mcmf.addEdge(i + N, sink, 0, H[i]);
        } else {
            mcmf.addEdge(src, i, L[i] - 1, H[i]);
            mcmf.addEdge(i + N, sink, 1, H[i]);
        }
    }

    for(int i = 1; i <= N; i++) {
        for(int j = 1; j <= N; j++) {
            if(A[i] > A[j]) {
                int cost = A[i] ^ A[j];
                mcmf.addEdge(i, j + N, 1, -cost);
            }
        }
    }

    auto ans = mcmf.getMCMF(src, sink);
    cout << -ans.second << "\n";

    return 0;
}
```

4 구간쿼리

4.1 Fenw

```
// PURQ, RUPQ, RURQ 모두 지원, 코드 자체는 1-based로 구현. 0-based 호출한다고 가정.; O(log N) +
상수 이점(bit 연산)
typedef long long ll;

// Point Update, Range Query
template<typename ValueType, typename IndexType>
struct fenwick_PURQ {
    IndexType n;
    vector<ValueType> tree;
    fenwick_PURQ(IndexType n) : n(n) { tree.resize(n + 1); }

    ValueType sum(IndexType i) {
        ValueType ret = 0;
        for (; i; i -= i & -i) ret += tree[i];
```

```
        return ret;
    }

    void update(IndexType i, ValueType x) { for (i++; i <= n; i += i & -i) tree[i] += x; }

    ValueType query(IndexType l, IndexType r) { return l > r ? 0 : sum(r + 1) - sum(l); }
};

// Range Update, Point Query, PURQ 필요.
template<typename ValueType, typename IndexType>
struct fenwick_RUPQ {
    fenwick_PURQ<ValueType, IndexType> f;
    fenwick_RUPQ(IndexType n) : f(fenwick_PURQ<ValueType, IndexType>(n + 1)) {}

    void update(IndexType l, IndexType r, ValueType x) { f.update(l, x), f.update(r + 1,
    -x); }

    ValueType query(IndexType i) { return f.query(0, i); }
};

// Range Update, Range Query, PURQ 필요.
template<typename ValueType, typename IndexType>
struct fenwick_RURQ {
    IndexType N;
    fenwick_PURQ<ValueType, IndexType> A, B;
    fenwick_RURQ(IndexType N) : N(N), A(fenwick_PURQ<ValueType, IndexType>(N + 1)),
    B(fenwick_PURQ<ValueType, IndexType>(N + 1)) {}

    void update(IndexType L, IndexType R, ValueType d) {
        A.update(L, d);
        A.update(R + 1, -d);

        B.update(L, (-L + 1) * d);
        B.update(R + 1, R * d);
    }

    ValueType query(IndexType L, IndexType R) {
        ValueType R_Value = A.query(0, R) * R + B.query(0, R);
        ValueType L_Value = 0;
        if (L > 0) {
            L_Value = A.query(0, L - 1) * (L - 1) + B.query(0, L - 1);
        }
        return R_Value - L_Value;
    }
};
int main() {
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int n, m; cin >> n >> m;
    fenwick_PURQ<ll, ll> fenw(n);
    vector<ll> a(n, 0);
    // initialize
    for (int i = 0; i < n; i++) fenw.update(i, a[i]);

    for (int i = 0; i < m; i++) {
        ll x; cin >> x;
        if (x == 1) {
            ll idx, val; cin >> idx >> val;
            ll diff = val - a[idx - 1];
            a[idx - 1] = val;
            fenw.update(idx - 1, diff);
        } else {
            ll l, r; cin >> l >> r;
            if(l > r) swap(l, r);
            cout << fenw.query(l - 1, r - 1) << "\n";
        }
    }
    return 0;
}
```

4.2 sqrt decomposition

```
// 주어진 쿼리를, O(sqrt(N))개의 그룹으로 분할해 결과를 구한다. -> 시간복잡도  O(sqrt(N)), 세그보다
공간 복잡도 측면에서 우수.
// 1 i j k: Ai, Ai+1, ..., Aj로 이루어진 부분 수열 중에서 k보다 큰 원소의 개수를 출력한다. 2 i k:
Ai를 k로 바꾼다.
typedef long long ll;

template<typename ValueType>
struct SqrtDecomposition {
    vector<ValueType> arr;
    vector<vector<ValueType>> buckets;
    ll sq;

  SqrtDecomposition(ll n, const vector<ValueType>& input) : arr(input) {
        sq = sqrt(n);
        buckets.resize((n + sq - 1) / sq); // 올림 처리를 통한 크기 설정
        for(ll i = 0; i < n; i++){
            buckets[i / sq].push_back(arr[i]);
        }
    // 이 부분은, 문제에 따라서 추가한 부분이다. -> upper_bound 함수 이용해서 빠르게 k보다 큰 원소의
    개수 count.
        for(ll i = 0; i < buckets.size(); i++) { // buckets.size()를 사용
            sort(buckets[i].begin(), buckets[i].end());
        }
    }

    void update(ll idx, ValueType val) {
        ll id = idx / sq; // 그룹 번호
        auto &bucket = buckets[id];
        auto it = lower_bound(bucket.begin(), bucket.end(), arr[idx]);
        bucket.erase(it);
        bucket.insert(upper_bound(bucket.begin(), bucket.end(), val), val);
        arr[idx] = val;
    }

    ValueType query(ValueType l, ValueType r, ValueType k) {
        ValueType ret = 0;
        while(l % sq != 0 && l <= r) {
            if(arr[l] > k) ret++;
            l++;
        }
        while((r + 1) % sq != 0 && l <= r) {
            if(arr[r] > k) ret++;
```

```
            r--;
        }
        while(l <= r){
            ret += buckets[l / sq].end() - upper_bound(buckets[l / sq].begin(), buckets[l /
            sq].end(), k);
            l += sq;
        }
        return ret;
    }
};

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    ll n, m; cin >> n;
    vector<ll> arr(n);
    for(int i = 0; i < n; i++) cin >> arr[i];

    SqrtDecomposition<ll> root(n, arr);

    cin >> m;
    for(int i = 0; i < m; i++){
        int a; cin >> a;
        if(a == 2) {
            ll b, c; cin >> b >> c; b--;
            root.update(b, c);
        } else {
            ll l, r, val; cin >> l >> r >> val; l--; r--;
            cout << root.query(l, r, val) << "\n";
        }
    }
    return 0;
}
```

4.3 mo's

```
// Offline query, N개의 원소에 대한 쿼리를 Q개 처리, 만약 시작점과 끝점을 이동하는데 걸리는 시간이
T(N)이라면, 최종 시간복잡도는 O((N + Q)√ N * T(N)).
// i j: Ai, Ai+1, ..., Aj에 존재하는 서로 다른 수의 개수를 출력
//가장 많이 등장하는 수, 서로 다른 수의 개수, abs(Ai - Aj) ≤ K인 (i, j) 쌍의 수(펜윅 트리 결합)
등에 사용
typedef long long ll;

struct query{
    int index, s, e;
};
int n, m, sqrtN, cnt;
vector<query> v;
int ans[100001], num[100001], check[1000001];

bool cmp(query &lhs, query &rhs){
    if(lhs.s/sqrtN == rhs.s/sqrtN)    return lhs.e < rhs.e;
    else return lhs.s/sqrtN < rhs.s/sqrtN;
}
int main(){
    cin.tie(nullptr); ios::sync_with_stdio(false);
    int s, e;
    cin >> n;
    sqrtN = sqrt(n);

    for(int i = 1; i <= n; i++) cin >> num[i];

    cin >> m;
    for(int i = 0; i < m; i++){
        cin >> s >> e;
        v.push_back({i, s, e});
    }
    sort(v.begin(), v.end(), cmp);
    s = v[0].s; e = v[0].e;

    /*최초 구간 계산하기*/
    for(int i = s; i <= e; i++){
        if(!check[num[i]])    cnt++;
        check[num[i]]++;
    }
    ans[v[0].index] = cnt;
    for(int i = 1; i < m; i++){
        while(s < v[i].s){
            /*update(구간 축소)*/
            // 수의 개수를 하나 줄임
            check[num[s]]--;
            // 수의 개수가 0이 되면 "서로 다른 수"의 개수 감소
            if(!check[num[s]])    cnt--;
            /*------*/
            s++;
        }
        while(v[i].s < s){
            s--;
            /*update(구간 확장)*/
            // 새로운 수라면 서로 다른 수의 개수 증가
            if(!check[num[s]])    cnt++;
            // 해당 수의 개수 증가
            check[num[s]]++;
            /*------*/
        }
        while(e < v[i].e){
            e++;
            /*update(구간 확장)*/
            if(!check[num[e]])    cnt++;
            check[num[e]]++;
            /*------*/
        }
        while(v[i].e < e){
            /*update(구간 축소)*/
            check[num[e]]--;
            if(!check[num[e]])    cnt--;
            /*------*/
            e--;
        }
        ans[v[i].index] = cnt; //쿼리에 대한 정답 저장
    }
    for(int i = 0; i < m; i++) cout << ans[i] << "\n";
    return 0;
}
```

4.4 dynamic seg

```
// pointer 이용, Segtree 메모리 사용량 줄여주는 방식.  공간복잡도 O(Q log N) -> N >= 10^6 일때 사용.
typedef long long ll;

template<typename ValueType, typename IndexType>
class DynamicSegmentTree {
public:
    struct Node {
        ValueType value = ValueType();
        Node *left = nullptr, *right = nullptr;
    };

private:
    Node *root = new Node();
    IndexType N; // 인덱스의 최대값 (배열 크기 - 1)

    void update(Node *node, IndexType nodeLeft, IndexType nodeRight, IndexType updateIndex,
    ValueType value) {
        if (updateIndex < nodeLeft || updateIndex > nodeRight) return;
        if (nodeLeft == nodeRight) {
            node->value = value;
            return;
        }
        IndexType mid = nodeLeft + (nodeRight - nodeLeft) / 2;
        if (updateIndex <= mid) {
            if (!node->left) node->left = new Node();
            update(node->left, nodeLeft, mid, updateIndex, value);
        } else {
            if (!node->right) node->right = new Node();
            update(node->right, mid + 1, nodeRight, updateIndex, value);
        }
        node->value = (node->left ? node->left->value : ValueType()) + (node->right ?
        node->right->value : ValueType());
    }

    ValueType query(Node *node, IndexType nodeLeft, IndexType nodeRight, IndexType
    queryLeft, IndexType queryRight) {
        if (!node || nodeRight < queryLeft || nodeLeft > queryRight) return ValueType();
        if (queryLeft <= nodeLeft && nodeRight <= queryRight) return node->value;
        IndexType mid = nodeLeft + (nodeRight - nodeLeft) / 2;
        return query(node->left, nodeLeft, mid, queryLeft, queryRight) + query(node->right,
        mid + 1, nodeRight, queryLeft, queryRight);
    }

public:
    DynamicSegmentTree(IndexType size) : N(size - 1) {}

    void update(IndexType index, ValueType value) {
        update(root, (IndexType)0, N, index, value);
    }

    ValueType query(IndexType left, IndexType right) {
        return query(root, (IndexType)0, N, left, right);
    }
};

int main() {
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    ll n, m, k; cin >> n >> m >> k;
    DynamicSegmentTree<ll, ll> dynaseg(n);
    for (ll i = 0; i < n; i++) {
        ll v; cin >> v;
        dynaseg.update(i, v);
    }
    for (ll i = 0; i < m + k; i++) {
        ll op; cin >> op;
        if (op == 1) {
            ll b, c; cin >> b >> c; b--;
            dynaseg.update(b, c);
        }
        if (op == 2) {
            ll b, c; cin >> b >> c; b--; c--;
            cout << dynaseg.query(b, c) << "\n";
        }
    }
    return 0;
}
```

4.5 PST

```
// 2차원 영역 쿼리
#define mp(a, b) make_pair((a),(b))
#define lbi(X, n) int(lower_bound(X.begin(), X.end(), n) - begin(X))
#define ubi(X, n) int(upper_bound(X.begin(), X.end(), n) - begin(X))

const int YMAX = 1e5 + 5, inf = 1e9;
typedef long long ll;

// l = 왼쪽 자식, r = 오른쪽 자식, v = 현재 노드
struct Node { int l = -1, r = -1, v = 0; };

template<typename IndexType, typename ValueType>
struct PST {
    // version 관리를 위한, vector<int>
    vector<ValueType> version;
    IndexType N;
    vector<Node> tree;

    PST(IndexType n) : N(n) {
        tree.push_back({});
        version.push_back(0);
    }
    IndexType update(IndexType i, ValueType v) {
        IndexType prev_root = version.back();
        // 업데이트 과정에서의 root은, 기존의 PST의 마지막 index의 다음 값을 가지게 되므로,
        tree.size() 를 할당하면 된다.
        IndexType root = tree.size();
        tree.push_back({});
        // version 벡터에, 현재 PST의 root를 저장한다.
        version.push_back(root);
        update(prev_root, root, 0, N - 1, i, v);
```

```
        // 업데이트가 끝난 후, 새로운 버전의 index를 반환한다. version.size()-1을 반환하는 이유는,
        0-based index이기 때문이다.
        return version.size() - 1;
    }
    ValueType query(IndexType version_idx, IndexType l, IndexType r) {
        // version[version_idx]에는, version_idx에 해당하는 버전의 루트 노드의 번호가 저장된다.
        return query(version[version_idx], 0, N - 1, l, r);
    }
private:
    void update(IndexType prev, IndexType cur, IndexType nl, IndexType nr, IndexType i,
    IndexType v) {
        // 현재 노드가 없거나, 현재 노드가 표현하는 구간에, 업데이트하려는 인덱스가 포함되지 않는
        경우, 아무 작업도 하지 않는다.
        if (cur == -1 || nr < i || nl > i) return;
        // leaf node인 경우.
        if (nl == nr) {
            tree[cur].v += v;
            return;
        }
        // For convenience, makes previous tree always has node in this place
        if (tree[prev].l == -1) {
            tree[prev].l = tree.size();
            tree[prev].r = tree.size() + 1;
            tree.push_back({}), tree.push_back({});
        }

        IndexType m = nl + (nr - nl) / 2;
        // 왼쪽 자식 노드를 업데이트해야 하는 경우. 오른쪽 자식 노드는 그대로 사용한다.
        if (i <= m) {
            IndexType new_child = tree.size();
            tree.push_back(tree[tree[prev].l]);
            tree[cur].l = new_child;
            tree[cur].r = tree[prev].r;
            update(tree[prev].l, tree[cur].l, nl, m, i, v);
        }
        // 오른쪽 자식 노드를 업데이트해야 하는 경우. 왼쪽 자식 노드는 그대로 사용한다.
        else {
            IndexType new_child = tree.size();
            tree.push_back(tree[tree[prev].r]);
            tree[cur].l = tree[prev].l;
            tree[cur].r = new_child;
            update(tree[prev].r, tree[cur].r, m + 1, nr, i, v);
        }

        tree[cur].v = (~tree[cur].l ? tree[tree[cur].l].v : 0) + (~tree[cur].r ?
        tree[tree[cur].r].v : 0);
    }
    // n = 노드 번호, nl, nr = 노드가 관할하는 구간, l, r = 쿼리를 날리는 구간
    ValueType query(IndexType n, IndexType nl, IndexType nr, IndexType l, IndexType r) {

        if (n == -1 || nr < l || nl > r) return 0;
        if (nl >= l && nr <= r) return tree[n].v;
        IndexType m = nl + (nr - nl) / 2;
        return query(tree[n].l, nl, m, l, r) + query(tree[n].r, m + 1, nr, l, r);
    }
};
```

```
int main() {
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
  PST<int, int> pst(YMAX);
  int T; cin >> T;
  vector<int> res;
  while(T--) {
    int n, m; cin >> n >> m;
    vector<pair<int, int>> query(n);
    for (auto&[x, y]: query)cin >> x >> y;
    sort(query.begin(), query.end());

    vector<pair<int, int>> versions;
    for (auto&[x, y]: query) {
      int version_idx = pst.update(y, 1);
      versions.push_back({x, version_idx});
    }
    int vidx = pst.update(YMAX - 1, 0);
    versions.push_back({1e9, vidx});

    ll ans = 0;
    while (m--) {
      int x1, x2, y1, y2; cin >> x1 >> x2 >> y1 >> y2;
      if (x1 > x2) swap(x1, x2);
      if (y1 > y2) swap(y1, y2);

      // x2 보다 큰 버전들 중 가장 먼저 나오는 것의 바로 이전 버전
      int version_right = versions[ubi(versions, mp(x2, inf))].second - 1;
      int t = pst.query(version_right, y1, y2);

      // x1 보다 같거나 큰 버전들 중 가장 먼저 나오는 것의 바로 이전 버전
      int version_left = versions[lbi(versions, mp(x1, -inf))].second - 1;
      t -= pst.query(version_left, y1, y2);
      ans += t;
    }
    res.push_back(ans);
  }
  for(const auto& ele : res) cout << ele << "\n";
  return 0;
}
```

4.6 merge sort tree

```
typedef long long ll;

const int MAX_ST = 1 << 18;

// 머지 소트 트리
template<typename ValueType>
struct MergesortTree {
    vector<ValueType> arr[MAX_ST];

    void construct() {
        for(int i = MAX_ST/2-1; i > 0; i--) {
            vector<ValueType> &c = arr[i], &l = arr[i*2], &r = arr[i*2+1];
            arr[i].resize(l.size()+r.size());
            for(int j = 0, p = 0, q = 0; j < c.size(); ++j) {
```

```
                if(q == r.size() || (p < l.size() && l[p] < r[q])) c[j] = l[p++];
                else c[j] = r[q++];
            }
        }
    }

    int greater(int s, int e, int k, int node = 1, int ns = 0, int ne = MAX_ST / 2) {
        if(ne <= s || e <= ns) return 0;
        if(s <= ns && ne <= e)
            return arr[node].end() - upper_bound(arr[node].begin(), arr[node].end(), k);
        int mid = (ns + ne) / 2;
        return greater(s, e, k, node * 2, ns, mid) + greater(s, e, k, node * 2 + 1, mid,
        ne);
    }
};

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

    int N, M, L = 0;
    MergesortTree<ll> ST;

    cin >> N;
    for(int i = 0; i < N; i++) {
        ll val; cin >> val;
        ST.arr[MAX_ST / 2 + i].push_back(val);
    }

    ST.construct();

    cin >> M;
    for(int i = 0; i < M; i++) {
        ll s, e, k; cin >> s >> e >> k;
        cout << (L = ST.greater((s^L) - 1, e^L, k^L)) << "\n";
    }
    return 0;
}
```

4.7 ETT

```
// ETT(Euler-Tour-Technique) + fenw, 쿼리개수 Q, 노드개수 N -> O(N + QlogN)
// p a x가 주어진 경우 a의 모든 부하의 월급을 x만큼 증가시킨다. (-10,000 ≤ x ≤ 10,000), u a가
주어진 경우에는 a의 월급을 출력한다.
typedef vector<int> vi;
typedef vector<vector<int>> vvi;
int n, m;
vi a, par, in, out;
vvi edges;

int dfsn = 0;
template<typename ValueType>
struct fenwick {
    int n;
    vector<ValueType> tree;
    fenwick(int n) : n(n) { tree.resize(n + 1); }
    ValueType sum(int i) {
        ValueType ret = 0;
        for (; i; i -= i & -i) ret += tree[i];
        return ret;
    }
    void update(int i, ValueType x) { for (i++; i <= n; i += i & -i) tree[i] += x; }
    ValueType query(int l, int r) { return l > r ? 0 : sum(r + 1) - sum(l); }
};
template<typename ValueType>
struct fenwick_point {
    fenwick<ValueType> f;
    fenwick_point(int n) : f(fenwick<ValueType>(n + 1)) {}
    void update(int l, int r, ValueType x) { f.update(l, x), f.update(r + 1, -x); }
    ValueType query(int i) { return f.query(0, i); }
};

// dfs ordering
void dfs(int i) {
    in[i] = dfsn++;
    for (int child: edges[i]) {
        dfs(child);
    }
    out[i] = dfsn - 1;
}

int main() {
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> n >> m;
    a.resize(n); edges.resize(n);
    in.resize(n); out.resize(n);

    for (int i = 0; i < n; i++) {
        int x, p;
        cin >> x;
        if (i) cin >> p;
        a[i] = x;
        if (i)
            edges[p - 1].push_back(i);
    }

    // ETT 먼저 하고 fenw 적용해야함.
    dfs(0);
    fenwick_point<int> fw(n);
    for (int i = 0; i < n; i++) {
        fw.update(in[i], in[i], a[i]);
    }

    for (int i = 0; i < m; i++) {
        string cmd;
        cin >> cmd;
        if (cmd == "p") {
            int k, x;
            cin >> k >> x, k--;
            // 자기 자신은 제외하고 업데이트 해야 한다.
            fw.update(in[k] + 1, out[k], x);
        } else {
            int k;
            cin >> k, k--;
```

```
            cout << fw.query(in[k]) << "\n";
        }
    }
    return 0;
}
```

4.8 HLD

```
// O(log^2 N); "트리" 에서 이 작업을 수행.
// update(v, w) -> 정점 v의 가중치에 w를 더함, query(s, e) -> s->e 경로의 모든 정점의 가중치의 합
(변경가능) 출력.
vector<int> inp[101010]; //입력 / 양방향 그래프

template<typename ValueType>
struct Seg{
    ValueType tree[1 << 18];
    ValueType sz = 1 << 17;

    void update(int x, ValueType v){
        x |= sz; tree[x] += v;
        while(x >>= 1){
            tree[x] = tree[x << 1] + tree[x << 1 | 1];
        }
    }

    ValueType query(int l, int r){
        l |= sz, r |= sz;
        ValueType ret = 0;
        while(l <= r){
            if(l & 1) ret += tree[l++];
            if(~r & 1) ret += tree[r--];
            l >>= 1, r >>= 1;
        }
        return ret;
    }
};

template<typename ValueType>
struct HLD {
    ValueType sz[101010], dep[101010], par[101010], top[101010], in[101010], out[101010];
    vector<ValueType> g[101010];
    Seg<int> seg;

    ValueType chk[101010];
    void dfs(int v = 1) {
        chk[v] = 1;
        for(auto i : inp[v]) {
            if(chk[i]) continue;
            chk[i] = 1;
            g[v].push_back(i);
            dfs(i);
        }
    }

    void dfs1(int v = 1) {
        sz[v] = 1;
        for(auto &i : g[v]) {
            dep[i] = dep[v] + 1; par[i] = v;
            dfs1(i); sz[v] += sz[i];
            if(sz[i] > sz[g[v][0]]) swap(i, g[v][0]);
        }
    }

    ValueType pv;
    void dfs2(int v = 1) {
        in[v] = ++pv;
        for(auto i : g[v]) {
            top[i] = i == g[v][0] ? top[v] : i;
            dfs2(i);
        }
        out[v] = pv;
    }

    void update(int v, ValueType w) {
        seg.update(in[v], w);
    }

    ValueType query(int a, int b) {
        ValueType ret = 0;
        while(top[a] ^ top[b]) {
            if(dep[top[a]] < dep[top[b]]) swap(a, b);
            ValueType st = top[a];
            ret += seg.query(in[st], in[a]);
            a = par[st];
        }
        if(dep[a] > dep[b]) swap(a, b);
        ret += seg.query(in[a], in[b]);
        return ret;
    }
};

int main() {
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int n, q; cin >> n >> q; //정점 개수, 쿼리 개수
    HLD<int> hld;
    // 간선정보 입력
    for(int i = 1; i < n; i++) {
        int s, e; cin >> s >> e;
        inp[s].push_back(e);
        inp[e].push_back(s);
    }
    hld.dfs(); hld.dfs1(); hld.dfs2();
    while(q--) {
        //1 v w : update v w
        //2 s e : query s e
        int op, a, b; cin >> op >> a >> b;
        if(op == 1) hld.update(a, b);
        else cout << hld.query(a, b) << "\n";
    }
}
```

5 기하

5.1 geometry init

```
typedef long long ll;
const double EPS = 0;

// 2D Point 구조체 정의
struct Point {
    double x, y;

    Point() : x(0), y(0) {}
    Point(double x_, double y_) : x(x_), y(y_) {}

    // 벡터 덧셈
    Point operator + (const Point& other) const {return Point(x + other.x, y + other.y);}
    // 벡터 뺄셈
    Point operator - (const Point& other) const {return Point(x - other.x, y - other.y);}
    // 내적 (Dot product)
    double operator * (const Point& other) const {return x * other.x + y * other.y;}
    // 외적 (Cross product)
    double operator ^ (const Point& other) const {return x * other.y - y * other.x;}
    // 스칼라 곱셈 (Point * 스칼라)
    Point operator * (double scalar) const {return Point(x * scalar, y * scalar);}
    // 스칼라 곱셈 (스칼라 * Point)
    friend Point operator * (double scalar, const Point& p) {return Point(p.x * scalar, p.y
    * scalar);}
    // 비교 연산자 (사전순 정렬을 위해)
    bool operator == (const Point& other) const {return fabs(x - other.x) < EPS && fabs(y -
    other.y) < EPS;}
    bool operator < (const Point& other) const {if (fabs(x - other.x) > EPS) return x <
    other.x;return y < other.y;}
};

// 출력 연산자 오버로딩
ostream& operator << (ostream& os, const Point& p) {
    os << "(" << p.x << ", " << p.y << ")";
    return os;
}
// CCW 함수
int CCW(const Point& a, const Point& b, const Point& c) {
    double area = (b - a) ^ (c - a);
    if (area > EPS) return 1;   // 반시계 방향
    if (area < -EPS) return -1; // 시계 방향
    return 0;                   // 일직선 상
}

// 두 점 사이의 거리의 제곱을 계산
double squaredDistance(const Point& a, const Point& b) {
    return (a - b) * (a - b);
}
```

5.2 convexHull

```
// 볼록 껍질을 구하는 함수 (Andrew's Monotone Chain Algorithm)
vector<Point> convexHull(vector<Point> points) {
    int n = points.size();
    if (n <= 1) return points;

    sort(points.begin(), points.end());

    vector<Point> lower, upper;
    for (const Point& p : points) {
        while (lower.size() >= 2 && CCW(lower[lower.size() - 2], lower[lower.size() - 1], p)
        <= 0) {
            lower.pop_back();
        }
        lower.push_back(p);
    }
    for (int i = n - 1; i >= 0; i--) {
        const Point& p = points[i];
        while (upper.size() >= 2 && CCW(upper[upper.size() - 2], upper[upper.size() - 1], p)
        <= 0) {
            upper.pop_back();
        }
        upper.push_back(p);
    }

    // 마지막 점은 중복되므로 제외
    lower.pop_back();
    upper.pop_back();

    // 볼록 껍질의 점들을 연결
    vector<Point> hull = lower;
    hull.insert(hull.end(), upper.begin(), upper.end());

    return hull;
}
```

5.3 perpendicularFoot

```
// 점 c에서 선분 ab에 내린 수선의 발
Point perpendicularFoot(const Point& a, const Point& b, const Point& c) {
    double dx = b.x - a.x;
    double dy = b.y - a.y;

    if (fabs(dx) < EPS && fabs(dy) < EPS) {
        return a;
    }

    double t = ((c - a) * (b - a)) / (dx * dx + dy * dy);
    return a + (b - a) * t;
}

// 점 c를 선분 ab에 대하여 반사(reflection)한 점 d를 계산
Point reflectOverLine(const Point& a, const Point& b, const Point& c) {
    // 점 c에서 선분 ab에 내린 수선의 발 p 계산
    Point p = perpendicularFoot(a, b, c);
    // 점 c를 점 p에 대하여 반사한 점 d를 계산
    Point d = p * 2 - c;
    return d;
}
```

5.4 segmentsIntersect

```
// 두 선분이 교차하는지 확인하고, 교점 계산 (정점이 선분 위에 있으면 교차하지 않는다 판정)
bool segmentsIntersect(const Point& a, const Point& b, const Point& c, const Point& d,
Point& intersection) {
```

```
    double det = (b - a) ^ (d - c);
    if (fabs(det) < EPS) {
        // 두 직선이 평행하거나 일치하는 경우
        // 선분이 일직선 상에 있는지 확인
        if (CCW(a, b, c) != 0) {
            return false; // 평행하지만 다른 직선
        }

        // 일직선 상에 있을 때, 선분이 겹치는지 확인
        if (max(a.x, b.x) + EPS < min(c.x, d.x) - EPS || max(c.x, d.x) + EPS < min(a.x, b.x)
        - EPS ||
            max(a.y, b.y) + EPS < min(c.y, d.y) - EPS || max(c.y, d.y) + EPS < min(a.y, b.y)
            - EPS) {
            return false; // 선분이 겹치지 않음
        }

        // 선분이 겹침 (교점이 여러 개일 수 있음)
        // 이 경우 특정한 교점을 반환하기 어렵지만, 필요에 따라 처리 가능
        //return true;
        return false;
    } else {
        // 크래머의 공식 이용하여 교점 계산
        double t = ((c - a) ^ (d - c)) / det;
        double u = ((c - a) ^ (b - a)) / det;

        // 수정된 조건문 (등호 미포함) (정점이 선분 위에 있으면 교차한다 판정)
        if (t < EPS || t > 1 - EPS || u < EPS || u > 1 - EPS) {
            return false; // 선분이 교차하지 않음
        }
        intersection = a + (b - a) * t;
        return true;
    }
}
```

5.5 polygonArea

```
// 다각형의 넓이를 계산하는 함수 (신발끈 공식)
double polygonArea(const vector<Point>& points) {
    double area = 0;
    int n = points.size();

    for (int i = 0; i < n; i++) {
        area += points[i] ^ points[(i + 1) % n];
    }

    return fabs(area) / 2.0;
}
```

5.6 isPointInConvexPolygon

```
// 점 p가 선분 ab 위에 있는지 확인하는 함수
bool onSegment(const Point& a, const Point& b, const Point& p) {
    double minX = min(a.x, b.x);
    double maxX = max(a.x, b.x);
    double minY = min(a.y, b.y);
    double maxY = max(a.y, b.y);

    return (minX - EPS <= p.x && p.x <= maxX + EPS) &&
           (minY - EPS <= p.y && p.y <= maxY + EPS);
}

// 점 p가 볼록 다각형 polygon 내부에 있는지 판별하는 함수 (점이 변 위에 있는 경우 내부 판단) //O(log
M), M은 polygon.size()
bool isPointInConvexPolygon(const vector<Point>& polygon, const Point& p) {
    int m = polygon.size();
    if (m < 2) return false;  // 다각형이 아닌 경우

    // 볼록 다각형 첫 번째 점과 마지막 점 기준으로 각을 이분 탐색
    if (CCW(polygon[m-1], polygon[0], p) < 0) return false;  // 점이 다각형 외부
    if (CCW(polygon[1], polygon[0], p) > 0) return false;    // 점이 다각형 외부

    // 이분 탐색으로 점이 다각형 내부에 있는지 확인
    int low = 1, high = m - 1;
    while (low + 1 < high) {
        int mid = (low + high) / 2;
        if (CCW(polygon[mid], polygon[0], p) < 0)
            low = mid;
        else
            high = mid;
    }

    // 마지막으로 점이 다각형의 해당 구간에 있는지 확인
    return CCW(polygon[low], p, polygon[low + 1]) < 0 ||
           (CCW(polygon[low], p, polygon[low + 1]) == 0 &&
           onSegment(polygon[low], polygon[low + 1], p));
}
```

5.7 rotatingCalipers

```
// 회전 캘리퍼스 알고리즘
double rotatingCalipers(const vector<Point>& convexHull) {
    int n = convexHull.size();
    if (n == 2) {
        return sqrt(squaredDistance(convexHull[0], convexHull[1]));
    }

    double maxDist = 0.0;
    int j = 1;
    for (int i = 0; i < n; i++) {
        int ni = (i + 1) % n;
        while (true) {
            int nj = (j + 1) % n;
            double cross = (convexHull[ni] - convexHull[i]) ^ (convexHull[nj] -
            convexHull[j]);
            if (cross > EPS) {
                j = nj;
            } else {
                break;
            }
        }
        double dist = sqrt(squaredDistance(convexHull[i], convexHull[j]));
        if (dist > maxDist + EPS) {
            maxDist = dist;
        }
    }
```

```
    return maxDist;
}
```

6 MISC

6.1 nlogn LIS

```
//LIS 크기 및 무작위 예시 1개를 찾음
const int MAXSIZE = 1e6;

int num[MAXSIZE + 1];
int trace[MAXSIZE + 1]; //무작위 예시 찾기

vector<int> v;
vector<int> LIS; //무작위 예시 찾기

int n, sizeofLIS;

int main(){
    cin.tie(NULL);  ios::sync_with_stdio(false);

    cin >> n;

    for(int i = 0; i < n; i++){
        cin >> num[i];
    }

    for(int i = 0; i < n; i++){
        if(v.empty() || num[i] > v.back()){
      v.push_back(num[i]);
      trace[i] = v.size(); //무작위 예시 찾기
    }
    else{
      auto t = lower_bound(v.begin(), v.end(), num[i]);
      *t = num[i];
      trace[i] = t - v.begin() + 1;  //무작위 예시 찾기
    }
    }

    sizeofLIS = v.size();

    /*-----무작위 예시를 찾을 때 사용-----*/
    int idx = v.size();
    for(int i = n - 1; i >= 0; i--){
        if(trace[i] == idx){
            idx--;
            LIS.push_back(num[i]);
        }
    }
    reverse(LIS.begin(), LIS.end());
    /*---------------------------------*/

    cout << sizeofLIS << '\n';
    for(int x : LIS){
        cout << x << ' ';
    }
    return 0;
}
```

6.2 분할정복 트릭 (DNC)

```
//dp[t][i]=min{k<i}(dp[t-1][k]+C[k][i])
//K[t][i]는 dp[t][i]를 만족시키는 최소 k라 할 때 다음 부등식을 만족 K[t][i]≤K[t][i+1]
//혹은 비용 C에 대해 C[a][c]+C[b][d]<=C[a][d]+C[b][c]
using ll = long long;

const ll INF = 1e18;
int L, G, K[801][8001];
ll psum[8001], C[801][1001];
ll dp[801][8001], ans;
//dncOpt 기본형, dp[mid] < (i - mid) * t[i] + v[mid]처럼 t가 없는 경우도 가능
void dncOptOriginal(int t, int s, int e, int optS, int optE){
    // dp[t][s], dp[t][s+1], ..., dp[t][e]를 계산하는 함수. [optS, optE] 가능한 k의 범위
    if(s > e) return;
        int mid = s + e>> 1;
        dp[t][mid] = 1e18;
        for (int k = optS; k <= optE && k < mid; k++){// dp[t][m]에 대해 가능한 답을 선형
        시간에 계산
            ll value = dp[t-1][k] + C[k][mid];
            if (dp[t][mid] > value)
                dp[t][mid] = value, K[t][mid] = k;
        }
        //dp[t][s], ..., dp[t][m-1]에 대해서 계산하기 위한 재귀 호출
        dncOptOriginal(t, s, mid - 1, optS, K[t][mid]);
        //dp[t][m+1], ..., dp[t][e]에 대해서 계산하기 위한 재귀 호출
        dncOptOriginal(t, mid + 1, e, K[t][mid], optE);
        // 2, 3에 대해 가능한 k의 후보는 조건에 의해 줄어든다.
}
//13261 풀이
void dncOptEx(int t, int s, int e, int optS, int optE){
  if(s > e)  return;
  int mid = s + e >> 1;
  int opt = 0;
  dp[t][mid] = 1e18;
  for(int i = optS; i <= min(mid, optE); i++){
    if(dp[t][mid] > dp[t - 1][i] + (mid - i) * (psum[mid] - psum[i])){
      dp[t][mid] = dp[t - 1][i] + (mid - i) * (psum[mid] - psum[i]);
      opt = i;
    }
  }
  dncOptEx(t, s, mid - 1, optS, opt);
  dncOptEx(t, mid + 1, e, opt, optE);
}

int main(){
  int x;
  cin.tie(nullptr);
  ios::sync_with_stdio(false);
  cin >> L >> G;
  for(int i = 1; i <= L; i++){
    cin >> x;
    psum[i] = psum[i - 1] + x;
    dp[1][i] = psum[i] * i;
```

```
  }
  for(int i = 2; i <= G; i++){
    dncOptEx(i, 1, L, 1, L);
  }
  cout << dp[G][L];
  return 0;
}
```

6.3 ConvexHull Tric

```
// dp optimization; convex hull trick, O(N logN)
typedef long long ll;

template<typename ValueType>
struct Line {
    ValueType slope, intercept;
    double xLeft;

    Line(ValueType _slope, ValueType _intercept) : slope(_slope), intercept(_intercept),
    xLeft(-1e18) {}

    ValueType getY(ValueType x) const {
        return slope * x + intercept;
    }

    double intersectX(const Line& other) const {
        return (double)(other.intercept - intercept) / (slope - other.slope);
    }
};

template<typename ValueType>
struct CHT {
    vector<Line<ValueType>> hull;

    void addLine(ValueType slope, ValueType intercept) {
        Line newLine(slope, intercept);
        while (!hull.empty()) {
            newLine.xLeft = newLine.intersectX(hull.back());
            if (hull.back().xLeft < newLine.xLeft)
                break;
            hull.pop_back();
        }
        hull.push_back(newLine);
    }

    ValueType query(ValueType x) {
        int l = 0, r = hull.size() - 1, ans = 0;
        while (l <= r) {
            int mid = (l + r) / 2;
            if (hull[mid].xLeft <= x) {
                ans = mid;
                l = mid + 1;
            } else {
                r = mid - 1;
            }
        }
        return hull[ans].getY(x);
    }
};

int main() {
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int N; cin >> N;
    vector<ll> a(N), b(N), dp(N);
    for (int i = 0; i < N; ++i) cin >> a[i];
    for (int i = 0; i < N; ++i) cin >> b[i];

    CHT<ll> cht;
    dp[0] = 0;
    cht.addLine(b[0], dp[0]);

    for (int i = 1; i < N; ++i) {
        dp[i] = cht.query(a[i]);
        cht.addLine(b[i], dp[i]);
    }

    cout << dp[N - 1] << '\n';
    return 0;
}
```

6.4 삼분탐색

```
// Trenary search(삼분 탐색); O(logN)
// 아래로 볼록, 혹은 위로 볼록(unimodal) 함 함수에 대해, 극값 혹은 최대/최소를 구하는 테크닉.
typedef long long ll;

template<typename ValueType>
class TernarySearch {
private:
    vector<ValueType> data;
    function<ValueType(int)> costFunction;

public:
    TernarySearch(const vector<ValueType>& data, function<ValueType(int)> costFunction)
        : data(data), costFunction(costFunction) {}

    ValueType FindMin() {
        ValueType lo = 0, hi = data.back();
        while (hi - lo >= 3) {
            ValueType p = (lo * 2 + hi) / 3, q = (lo + hi * 2) / 3;
            if (costFunction(p) <= costFunction(q)) hi = q;
            else lo = p;
        }

        ValueType result = numeric_limits<ValueType>::max();
        for (ValueType i = lo; i <= hi; ++i)
            result = min(costFunction(i), result);
        return result;
    }

    ValueType FindMax() {
        ValueType lo = 0, hi = data.back();
        while (hi - lo >= 3) {
```

```
            ValueType p = (lo * 2 + hi) / 3, q = (lo + hi * 2) / 3;
            if (costFunction(p) >= costFunction(q)) hi = q;
            else lo = p;
        }

        ValueType result = numeric_limits<ValueType>::min();
        for (ValueType i = lo; i <= hi; ++i)
            result = max(costFunction(i), result);
        return result;
    }
};

int main() {
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int N; cin >> N;
    vector<ll> x(N);
    for (int i = 0; i < N; ++i)
        cin >> x[i];

    auto costFunction = [&](int dist) -> ll {
        ll totalCost = 0;
        for (int i = 1; i < x.size(); ++i)
            totalCost += abs(1LL * dist * i - x[i]);
        return totalCost;
    };

    TernarySearch<ll> search(x, costFunction);
    cout << search.FindMin();
    return 0;
}
```

6.5 Sparse table

```
/*
합성함수 문제예제; table의 계산(전처리)은 Θ(N·log(K))만큼의 시간/공간을 소요
solve(v, k)의 문제 해결은 Θ(log(K))만큼의 시간을 소요
*/
const int MAX_N = 200000;
const int MAX_LOG_K = 19; // n의 최대값이 500,000이므로 2^19 > 500,000

int table[MAX_LOG_K + 1][MAX_N + 1];

// table 생성 함수
void init(int m, const vector<int>& nxt) {
    // k=0, 한 칸 이동하는 경우 초기화
    for (int i = 1; i <= m; ++i) {
        table[0][i] = nxt[i];
    }

    // k>0인 경우, 2^k번 이동한 결과를 계산
    for (int k = 1; k <= MAX_LOG_K; ++k) {
        for (int i = 1; i <= m; ++i) {
            int tmp = table[k - 1][i];
            table[k][i] = table[k - 1][tmp];
        }
    }
}

// 쿼리 처리 함수
int solve(int v, int k) {
    for (int i = MAX_LOG_K; i >= 0; --i) {
        if (k & (1 << i)) {
            v = table[i][v];
        }
    }
    return v;
}

int main() {
    ios::sync_with_stdio(false);cin.tie(nullptr);

    int m;
    cin >> m;

    // 함수 f(i)를 입력받아 nxt 배열에 저장
    vector<int> nxt(m + 1);
    for (int i = 1; i <= m; ++i) {
        cin >> nxt[i];
    }

    // table 생성
    init(m, nxt);

    int Q;
    cin >> Q;
    while (Q--) {
        int n, x;
        cin >> n >> x;
        int result = solve(x, n);
        cout << result << '\n';
    }

    return 0;
}
```