2023 SCCC 봄 #3

BOJ 17939. Gazzzua

i번째 날에 구매한 코인은 $i, i+1, \cdots, N$번째 날 중 가장 비쌀 때 판매해야 합니다. 코인을 구매하지 않는 것은 구매한 날 다시 파는 것이라 생각할 수 있기 때문에, 일단 매일 코인을 구매하고 이후 시점에서 가장 비쌀 때 판매한다고 생각해도 무방합니다.

주어진 가격 배열을 뒤에서부터 차례대로 순회하면 $N, N-1, N-2, \cdots, i+1, i$번째 날 중 가장 비쌀 때의 가격 M_i을 알 수 있습니다. i번째 날에 A_i원에 구매한 코인을 M_i번째 날에 판매하면 $M_i - A_i \geq 0$ 만큼의 이득을 볼 수 있습니다. 따라서 가격 배열의 맨 뒤부터 차례대로 보면서 정답을 구할 수 있습니다.

```
#include <bits/stdc++.h>
using namespace std;

int N, A[101010], M, R;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i];
    for(int i=N; i>=1; i--) M = max(M, A[i]), R += M - A[i];
    cout << R;
}
```

BOJ 17953. 디저트

동적 계획법을 이용해 문제를 해결할 수 있습니다. $D(i, j)$를 $1, \cdots, i$번째 날까지 디저트를 먹었고 i번째 날 j번째 디저트를 먹었을 때의 최대 만족감이라고 정의합시다. 같은 디저트를 2일 연속으로 먹으면 만족도가 절반이 되기 때문에 상태 전이는 $D(i,j) = \begin{cases} D(i-1,k) + V_{i,j} & \text{if } j \neq k \\ D(i-1,k) + V_{i,j}/2 & \text{if } j = k \end{cases}$와 같이 나타낼 수 있습니다.

상태가 총 $O(NM)$가지 있고, 각 상태의 답을 계산하는데 $O(M)$개의 이전 상태를 고려해야 하므로 $O(NM^2)$ 시간에 문제를 해결할 수 있습니다.

```
#include <bits/stdc++.h>
using namespace std;

int N, M, A[101010][11], D[101010][11];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    for(int j=1; j<=M; j++) for(int i=1; i<=N; i++) cin >> A[i][j];
    for(int j=1; j<=M; j++) D[1][j] = A[1][j];
    for(int i=2; i<=N; i++) for(int j=1; j<=M; j++) for(int k=1; k<=M; k++) D[i]
[j] = max(D[i][j], D[i-1][k] + A[i][j] / (j == k ? 2 : 1));
    cout << *max_element(D[N]+1, D[N]+M+1);
}
```

BOJ 2107. 포함하는 구간

$O(N^2)$으로 안 풀릴 것 같지만 사실은 잘 풀리는 문제입니다. 하지만 이렇게 풀면 너무 재미없으니 이런 문제를 풀 때 자주 나오는 접근법에 대해 간단하게 소개하겠습니다.

선분의 포함 관계를 묻는 문제는 보통 선분들을 끝점 기준 오름차순으로 정렬하고 생각하는 것이 편합니다. 어떤 선분 $[s_i, e_i]$가 $[s_j, e_j]$에 포함되기 위한 조건은 $s_j \le s_i, e_i \le e_j$으로 두 개의 부등식을 봐야 하는데, 끝점 기준으로 정렬을 수행하면 $e_i \le e_j$가 항상 성립해서 $s_j \le s_i$만 고려해도 되기 때문입니다.

따라서 이 문제는 선분들을 끝점 기준으로 정렬한 다음, 자신보다 앞선 선분 중 시작점이 늦은 점들의 개수를 구하는 방식으로 해결할 수 있습니다. 단순하게 구현하면 $O(N^2)$이고, 세그먼트 트리나 펜윅 트리와 같은 구간 합 쿼리가 가능한 자료구조를 사용하면 $O(N \log N)$ 시간에 해결할 수도 있습니다.

```
#include <bits/stdc++.h>
using namespace std;

int N, R;
vector<pair<int,int>> V;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N; V.resize(N);
    for(auto &[a,b] : V) cin >> a >> b;
    sort(V.begin(), V.end(), [](auto a, auto b){ return tie(a.second,a.first) <
tie(b.second,b.first); });
    for(int i=1; i<N; i++){
        int cnt = 0;
        for(int j=0; j<i; j++) cnt += V[i].first <= V[j].first;
        R = max(R, cnt);
    }
    cout << R;
}
```

$O(N \log N)$ 코드

```
#include <bits/stdc++.h>
using namespace std;
constexpr int SZ = 1 << 16;

int T[SZ];
void Add(int x, int v){ for(x+=3; x<SZ; x+=x&-x) T[x] += v; }
int Get(int x){ int r = 0; for(x+=3; x; x-=x&-x) r += T[x]; return r; }
int Get(int l, int r){ return l <= r ? Get(r) - Get(l-1) : 0; }

int N, R;
vector<pair<int,int>> V;
vector<int> C;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N; V.resize(N); C.reserve(N+N);
    for(auto &[a,b] : V) cin >> a >> b;
    for(const auto &[a,b] : V) C.push_back(a), C.push_back(b);
    sort(C.begin(), C.end());
    C.erase(unique(C.begin(), C.end()), C.end());
    for(auto &[a,b] : V) a = lower_bound(C.begin(), C.end(), a) - C.begin() + 1;
```

```
    for(auto &[a,b] : V) b = lower_bound(C.begin(), C.end(), b) - C.begin() + 1;
    sort(V.begin(), V.end(), [](auto a, auto b){ return tie(a.second,a.first) <
tie(b.second,b.first); });
    for(const auto &[a,b] : V) R = max(R, Get(a, b)), Add(a, 1);
    cout << R;
}
```

BOJ 2871. 아름다운 단어

어떤 문자 c_1이 c_2보다 사전 순으로 앞선다는 것을 $c_1 < c_2$라고 표현하겠습니다.

희원이의 입장에서는 매번 남아있는 문자들 중에서 사전 순으로 가장 앞선 문자를 가져가는 것이 최선의 전략임을 쉽게 알 수 있습니다. 한 번이라도 상대방보다 사전 순으로 앞서는 문자를 가져가면 이기는 게임이기 때문입니다.

사전 순으로 가장 앞서는 문자가 유일하게 결정된다면 그 문자를 가져가면 되지만, 그러한 문자가 여러 개 있다면 어디에 있는 문자를 가져갈지 결정해야 합니다. 이때는 최대한 뒤에 있는 문자를 가져가는 것이 이득입니다. 상근이는 매번 가장 뒤에 있는 문자를 가져가기 때문에, 희원이는 최대한 뒤에 있는 문자부터 가져가서 상근이가 사전 순으로 가장 앞선 문자에 접근하는 것을 막아야 하기 때문입니다.

각 알파벳이 등장하는 위치를 저장하는 배열을 만들면 쉽게 구현할 수 있습니다.

```
#include <bits/stdc++.h>
using namespace std;

int N;
string S, A, B;
vector<int> V[26];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> S;
    for(int i=0; i<N; i++) V[S[i]-'a'].push_back(i);
    for(int i=0; i<N; i++){
        if(i % 2 == 0){
            int mx = -1, pos = -1;
            for(int j=0; j<26; j++) if(!V[j].empty() && V[j].back() > mx) mx =
V[j].back(), pos = j;
            A += char(pos+'a'); V[pos].pop_back();
        }
        else{
            int pos = -1;
            for(int j=0; j<26; j++) if(!V[j].empty()) { pos = j; break; }
            B += char(pos+'a'); V[pos].pop_back();
        }
    }
    cout << (A > B ? "DA" : "NE") << "\n" << B;
}
```

BOJ 12992. 홍준이의 교집합

$1 \le i_1 \le i_2 \le \cdots \le i_k \le N$인 모든 부분 집합에서 교집합의 크기를 구하는 것은 어려워 보이기 때문에 다른 방법을 찾는 것이 좋을 것 같습니다.

이런 상황에서 유용한 테크닉으로 더블 카운팅이 있습니다. 선분 k를 고른 다음 교집합의 길이를 더하는 대신, 특정 구간을 포함하도록 선분 k개를 선택하는 경우의 수를 세는 방식으로 문제를 해결합니다.

좌표 압축을 하면 $O(N)$개의 구간만 고려해도 됨을 알 수 있고, 변홧값 배열을 이용하면 특정 구간을 덮는 선분의 개수를 빠르게 구할 수 있습니다. 따라서 이 문제는 결국 $\binom{n}{r} \mod P$을 빠르게 여러 번 계산하는 문제가 되고, 이건 페르마의 소정리를 이용해 빠르게 계산할 수 있습니다. 실제로 구현할 때는 선분의 끝점이 놓이는 지점과 그렇지 않은 지점을 구분해서 계산하는 것이 편합니다.

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
constexpr int SZ = 808080;
constexpr ll MOD = 1e9+7;

ll Pow(ll a, ll b){
    ll res = 1;
    for(; b; b >>= 1, a = a * a % MOD) if(b & 1) res = res * a % MOD;
    return res;
}

ll Fac[SZ+1], Inv[SZ+1];
void Init(){
    Fac[0] = 1;
    for(int i=1; i<=SZ; i++) Fac[i] = Fac[i-1] * i % MOD;
    Inv[SZ] = Pow(Fac[SZ], MOD-2);
    for(int i=SZ-1; i>=0; i--) Inv[i] = Inv[i+1] * (i+1) % MOD;
}
ll Choose(int n, int r){ return r <= n ? Fac[n] * Inv[r] % MOD * Inv[n-r] % MOD
: 0; }

int N, K, L[202020], R[202020], S[808080];
vector<int> C;
ll X;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> K; C.reserve(N*6); Init();
    for(int i=1; i<=N; i++) cin >> L[i] >> R[i], C.push_back(L[i]),
C.push_back(R[i]);
    sort(C.begin(), C.end()); C.erase(unique(C.begin(), C.end()), C.end());
    for(int i=1; i<=N; i++){
        L[i] = lower_bound(C.begin(), C.end(), L[i]) - C.begin();
        R[i] = lower_bound(C.begin(), C.end(), R[i]) - C.begin();
        S[L[i]*2] += 1; S[R[i]*2+1] -= 1;
    }
    partial_sum(S, S+808080, S);
    for(int i=0; i<C.size(); i++){
        X = (X + Choose(S[i*2], K)) % MOD;
        if(i + 1 < C.size()){
            int len = C[i+1] - C[i] - 1, cnt = S[i*2+1];
            X = (X + Choose(cnt, K) * len) % MOD;
        }
    }
    cout << X;
}
```

BOJ 6101. 식당

어떤 합집합의 크기가 $\sqrt{N}$을 넘으면 그룹의 비용이 N보다 커지기 때문에 고려하지 않아도 됩니다. 모든 그룹의 크기를 1로 정해도 N 짜리 해를 얻을 수 있기 때문입니다. 따라서 합집합의 크기가 $\sqrt{N}$ 이하인 구간만 고려해도 충분합니다.

동적 계획법을 이용해 정답을 계산할 것입니다. 구체적으로, 정화식을 $D(i) := P[1 \cdots i]$만 고려했을 때의 정답으로 정의해서 계산합니다. $D(i)$는 $(j, i]$ 구간의 합집합의 크기가 $\sqrt{N}$ 이하가 되도록 하는 모든 j에 대해 $D(i) = \min\{D(j) + C(j+1, i)\}$로 계산할 수 있습니다.

이때 $D(i-1) \le D(i)$를 만족합니다. 만약 $D(i-1) > D(i)$인 i가 존재한다면
$D(i) = \min\{D(j) + C(j+1, i)\}$를 만족하는 j가 존재할 텐데,
$D(j) + C(j+1, i-1) \le D(k) + C(j+1, i)$이기 때문에 $D(i-1) > D(i)$가 성립하지 않습니다. 따라서 우리는 각 i마다 그룹의 크기가 $1 \le s \le \sqrt{N}$이 되도록 하는 가장 작은 j인 $J(s, i)$만 고려해도 충분합니다.

s가 고정되어 있을 때 $J(s, *)$는 투 포인터를 이용해 $O(N)$에 구할 수 있으므로 $O(N\sqrt{N})$ 시간에 전처리를 할 수 있습니다. 점화식은 $D(i) = \min\left\{D(J(s, i)) + s^2\right\}$으로 계산할 수 있으므로 전체 시간 복잡도는 $O(N\sqrt{N})$입니다.

```
#include <bits/stdc++.h>
using namespace std;

int N, M, A[40404], C[40404], X, P[40404][222], D[40404];

void Ins(int x){ X += !C[A[x]]++; }
void Del(int x){ X -= !--C[A[x]]; }

void Get(int sz){
    memset(C, 0, sizeof C); X = 0;
    for(int i=1, j=1; j<=N; j++){
        Ins(j);
        while(i < j && X > sz) Del(i++);
        P[j][sz] = i;
    }
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    for(int i=1; i<=N; i++) cin >> A[i];
    for(int i=1; i*i<=N; i++) Get(i);
    memset(D, 0x3f, sizeof D); D[0] = 0;
    for(int i=1; i<=N; i++) for(int j=1; j*j<=N; j++) if(P[i][j]) D[i] =
min(D[i], D[P[i][j]-1] + j * j);
    cout << D[N];
}
```

BOJ 19614. Travelling Salesperson

정점을 하나씩 추가해서 경로를 만드는 방식으로 진행합니다.

지금까지 만든 경로를 $(v_1, v_2, \cdots, v_k)$, `RR...RBB...B` 형태라고 합시다.

새로운 정점 x를 추가할 때 만약 (x, v_1) 간선이 `R`이면 앞에 추가하고 (v_k, x) 간선이 `B`이면 뒤에 추가하면 됩니다. 두 가지 경우에 모두 해당하지 않는다면 (x, v_1)이 `B`, (v_k, x)가 `R`인 상태입니다.

만약 (v_1, v_k)가 `R`이면 v_k를 v_1 앞에 붙인 다음, (x, v_k)가 `R`이니까 x를 앞에 붙이면 됩니다. 그렇지 않다면 (v_1, v_k)가 `B`일 텐데, 이때는 v_k 뒤에 v_1을 붙인 다음 x를 뒤에 붙이면 됩니다.

남은 부분은 원하는 정점을 시작 정점으로 만드는 것입니다. 시작 정점을 가장 마지막에 추가한 다음, 만약 시작 정점이 맨 뒤로 갔다면 경로를 뒤집어서 시작 정점이 되도록 보장할 수 있습니다.

연결 리스트를 사용하면 시작점마다 $O(N)$ 만큼의 시간이 걸려서 총 $O(N^2)$ 시간에 문제를 해결할 수 있습니다.

```
#include <bits/stdc++.h>
using namespace std;

int N, A[2020][2020];

void Solve(int src){
    list<int> res;
    function<void(int)> add = [&res](int v){
        if(res.size() <= 1){ res.push_back(v); return; }
        int a = *res.begin(), b = *res.rbegin();
        if(A[v][a] == 0) res.push_front(v);
        else if(A[v][b] == 1) res.push_back(v);
        else if(A[a][b] == 0) res.pop_back(), res.push_front(b),
res.push_front(v);
        else res.pop_front(), res.push_back(a), res.push_back(v);
    };

    for(int i=1; i<=N; i++) if(i != src) add(i);
    add(src);
    if(*res.begin() != src) res.reverse();

    cout << N << "\n";
    for(auto i : res) cout << i << " ";
    cout << "\n";
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=2; i<=N; i++) for(int j=1; j<i; j++) { char c; cin >> c; A[i][j] =
A[j][i] = c == 'B'; }
    for(int i=1; i<=N; i++) Solve(i);
}
```

BOJ 21824. Weird Numeral System

진법의 개념에서 크게 벗어나지 않기 위해서 일단 $M \le K, n \neq 0$인 상황만 생각해 봅시다.

이런 상황에서는 매 단계마다 n이 존재할 수 있는 구간을 $[-K^t, K^t]$에서 $[-K^{t-1}, K^{t-1}]$으로 축소시킬 수 있습니다. $n \equiv a_i \pmod K$인 a_i를 마지막에 붙인다고 가정한 다음, $(n - a_i)/K$에 대해서 문제를 해결하면 되기 때문입니다. 따라서 $M \le K$이면 재귀적으로 문제를 해결할 수 있습니다.

$n = 0$일 때 공집합을 출력하면 안 된다는 이상한 조건이 붙어있습니다. 공집합이 아니라는 것은 마지막에 어떤 원소 a_i를 붙여야 한다는 것이고, 당연히 $a_i \equiv 0 \pmod K$를 만족해야 합니다.

마지막에 a_i를 붙여서 0이 되기 위해서는 $(-a_i/K) \times K + a_i$ 꼴이 되어야 합니다. 따라서 위에서 설명

한 풀이를 이용해 $-a_i/K$를 만드는 방법을 구하면 됩니다.

마지막으로 $M > K$인 경우를 생각해 봅시다. 이 경우에는 구간을 K배만큼 축소가 불가능할 수도 있기 때문에 위에서 설명한 풀이를 적용하지 못합니다.

어떤 상황에서 축소에 실패하는지 살펴봅시다. 대부분의 경우에는 a_i를 더하는 것보다 K로 나누는 것의 영향력이 더 크기 때문에 축소를 할 수 있습니다. 하지만 $|n| \le M$이면 a_i를 더하는 것으로 인해서 $[-M, M]$ 범위 밖으로 벗어나는 일이 발생할 수 있습니다. 따라서 절댓값이 M 이하인 수들에 대해서는 다른 방법으로 경로를 탐색해야 합니다.

다행히도 M이 $2\,500$ 정도로 굉장히 작기 때문에 단순히 BFS를 이용해 전처리해도 괜찮습니다. n을 계속 축소하다가 $|n| \le M$이 되면 BFS로 전처리한 경로를 사용하면 됩니다.

재귀적으로 탐색하는 깊이가 $O(\log_K N)$ 정도로 매우 작기 때문에 시간 제한 안에 문제를 해결할 수 있습니다.

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

int K, Q, D, M, A[5050];
vector<int> G[1010101];
unordered_map<ll, int> Vst, Prv;
inline ll Mod(ll t){ return (t %= K) >= 0 ? t : t + K; }

void SmallBFS(){
    for(int i=-M; i<=M; i++) Vst[i] = 0;
    queue<ll> Que; Que.push(0); Vst[0] = 1;
    while(!Que.empty()){
        ll v = Que.front(); Que.pop();
        for(int i=1; i<=D; i++){
            ll nxt = v * K + A[i];
            if(abs(nxt) > M || Vst[nxt]) continue;
            Que.push(nxt); Vst[nxt] = 1; Prv[nxt] = A[i];
        }
    }
}

int Check(ll n){
    if(auto it=Vst.find(n); it != Vst.end()) return it->second;
    for(auto i : G[Mod(n)]) if(Check((n - A[i]) / K)) { Prv[n] = A[i]; return
Vst[n] = 1; }
    return Vst[n] = 0;
}

vector<int> Path(ll n){
    vector<int> res;
    if(n != 0){
        if(!Check(n)) return {};
        while(n != 0) res.push_back(Prv[n]), n = (n - Prv[n]) / K;
        reverse(res.begin(), res.end());
        return res;
    }
    else{
        for(int i=1; i<=D; i++) if(A[i] == 0) return {A[i]};
        for(auto i : G[0]){
            if(!Check(-A[i] / K)) continue;
```

```
            res = Path(-A[i] / K); res.push_back(A[i]);
            return res;
        }
        return {};
    }
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> K >> Q >> D >> M;
    for(int i=1; i<=D; i++) cin >> A[i], G[Mod(A[i])].push_back(i);
    SmallBFS();
    for(int q=1; q<=Q; q++){
        ll n; cin >> n;
        auto path = Path(n);
        if(path.empty()) cout << "IMPOSSIBLE\n";
        else for(int i=0; i<path.size(); i++) cout << path[i] << " \n"
[i+1==path.size()];
    }
}
```

BOJ 25009. 뚫기

Subtask 1. $N \leq 3\,000, M \leq 3\,000, Q = 1$ (7점)

사실 $M \leq 3\,000$ 조건이 없더라도 좌표 압축을 하면 $M = O(N)$으로 생각할 수 있습니다.

$i-1$번째 줄에서 (i, j)로 이동하는 것은 $(i-1, k)$에서 앞으로 한 칸 전진해 (i, k)로 이동한 뒤, 비용이 A인 연산을 이용해 (i, j)로 이동하는, 두 개의 단계로 나눌 수 있습니다. DP 배열을 다음과 같이 정의하겠습니다.

- $D(2i, j) =$ 마지막에 앞으로 한 칸 전진해서 (i, j)에 도달하는 최소 비용
- $D(2i+1, j) = (i, j)$에 도달하는 최소 비용

$D(2i, j) = D(2i-1, j) + (0 \ or \ B)$이고, $D(2i+1, j) = \min\{D(2i, j), D(2i, k) + A\}$ 입니다.
$O(N^2)$에 계산할 수 있습니다.

Subtask 2. $Q \leq 50$ (29점)

$D(i, *)$를 하나의 세그먼트 트리로 관리합니다. $D(i-1, *)$를 빠르게 $D(i, *)$로 바꾸는 방법을 찾아야 합니다.

$D(2i, j) = D(2i-1, j) + B$는 막이 존재하는 구간에 B를 더하는 것과 동일하고, 이는 세그먼트 트리와 레이지 프로퍼게이션을 이용해 $O(\log N)$에 수행할 수 있습니다.
$D(2i+1, j) = \min\{D(2i, j), D(2i, k) + A\}$는 $D(2i, *)$의 최솟값 $D(2i, k)$를 구한 다음, 트리의 기존 값과 $D(2i, k) + A$ 중 더 작은 값으로 갱신해야 합니다. 이는 세그먼트 트리 비츠를 이용해 $O(\log N)$에 수행할 수 있습니다.

각 쿼리를 $O(N \log N)$에 처리할 수 있으므로 전체 시간 복잡도는 $O(QN \log N)$입니다.

Subtask 5. $N \leq 10\,000$ (100점)

Subtask 1/2의 풀이를 아무리 최적화해도 쿼리를 $O(N \log N)$보다 빠르게 처리할 수는 없어 보입니다. 대신, 각 연산을 각각 최대 N번만 사용해도 최적해를 찾을 수 있다는 점을 이용해 다른 풀이를 생각해 봅시다.

만약 모든 $0 \le i \le N$에 대해, 비용이 A인 연산을 정확히 i번 사용했을 때 필요한 B 연산의 횟수를 알고 있다면, i가 증가할수록 필요한 B 연산의 횟수는 단조감소하므로 이분 탐색을 이용해 각 쿼리를 $O(\log N)$에 처리할 수 있습니다. 모든 i에 대해 직접 계산해보면 계산 방법에 따라 Subtask 3/4를 해결할 수 있습니다.

도착점까지 갈 때 필요한 A 연산의 횟수 a와 B 연산의 횟수 b를 2차원 점 (a, b)로 나타내보면, 볼록 껍질의 **왼쪽 아래 부분**을 구성하는 점만 실제 정답이 될 수 있다는 것을 알 수 있습니다. 좌표 범위가 X일 때 볼록 껍질의 꼭짓점이 될 수 있는 정수 격자점은 최대 $O(X^{2/3})$개이기 때문에, $N+1$개의 점에서 모두 DP를 하는 것 대신 왼쪽 아래 껍질 상의 점에 대해서만 DP를 하면 시간 복잡도를 줄일 수 있습니다.

필요한 점을 빠르게 찾는 방법을 알아봅시다. 먼저, x좌표가 가장 작은 왼쪽에 있는 점 L과 y좌표가 가장 작은 아래에 있는 점 U는 항상 껍질에 포함됩니다. 그러므로 두 점을 먼저 구합니다.
그 다음에는 L과 U를 잇는 직선과 평행한 껍질의 접선의 접점 M을 찾습니다. 다시 L과 M을 잇는 직선, M과 U를 잇는 직선의 기울기를 재귀적으로 처리하는 과정을 통해 모든 점을 찾아줄 수 있습니다.

만약 **주어진 기울기에 대한 볼록 껍질의 접점**을 $T(N)$ 시간에 찾을 수 있다면 모든 점을 $O(N^{2/3}T(N))$에 구할 수 있습니다. 이때 L과 U는 기울기가 $1/0, 0/1$인 접선의 접점입니다.
기울기가 $-p/q$ $(p, q \ge 0)$인 접점을 구하는 것은 A 연산의 비용이 p, B 연산의 비용이 q인 문제를 해결하는 것으로 생각할 수 있습니다. 이는 Subtask 2의 풀이를 이용해 $O(N \log N)$에 해결할 수 있습니다.

$T(N) = O(N \log N)$이므로 $O(N^{2/3} \cdot N \log N)$에 모든 점을 계산할 수 있습니다. 쿼리는 $O(\log N)$에 처리할 수 있으므로 전체 시간 복잡도는 $O((N^{5/3} + Q) \log N)$입니다. 만약 세그먼트 트리 대신 평방 분할을 사용하면 $O(N^{13/6} + Q \log N)$이 됩니다.

```
#include "breakthru.h"
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;
using ll = long long;
using Point = pair<ll, ll>;
constexpr ll INF = 0x3f3f3f3f3f3f3f3f;

ll CCW(const Point &p1, const Point &p2, const Point &p3){
    return (p2.x - p1.x) * (p3.y - p2.y) - (p3.x - p2.x) * (p2.y - p1.y);
}

int N, M, S[10101], E[10101];
vector<int> C;
vector<Point> R;

struct Tree{
    pair<ll,ll> T[1<<16], M[1<<16];
    int S[1<<16];
    void clear(){
        fill(T, T+(1<<16), make_pair(0LL,0LL));
        fill(S, S+(1<<16), 0);
        fill(M, M+(1<<16), make_pair(INF,INF));
    }
    void push(int node, int s, int e){
        T[node].x += S[node];
        if(s != e){
            S[node<<1] += S[node];
            S[node<<1|1] += S[node];
            M[node<<1].x += S[node];
            M[node<<1|1].x += S[node];
        }
```

```
        T[node] = std::min(T[node], M[node]);
        if(s != e){
            M[node<<1] = std::min(M[node<<1], M[node]);
            M[node<<1|1] = std::min(M[node<<1|1], M[node]);
        }
        S[node] = 0;
        M[node] = make_pair(INF,INF);
    }
    void add(int l, int r, ll v, int node=1, int s=0, int e=(1<<15)-1){
        push(node, s, e);
        if(r < s || e < l) return;
        if(l <= s && e <= r){ S[node] += v; push(node, s, e); return; }
        int m = s + e >> 1;
        add(l, r, v, node<<1, s, m);
        add(l, r, v, node<<1|1, m+1, e);
        T[node] = std::min(T[node<<1], T[node<<1|1]);
    }
    void min(int l, int r, pair<ll,ll> v, int node=1, int s=0, int e=(1<<15)-1){
        push(node, s, e);
        if(r < s || e < l) return;
        if(l <= s && e <= r){ M[node] = std::min(M[node], v); push(node, s, e);
return; }
        int m = s + e >> 1;
        min(l, r, v, node<<1, s, m);
        min(l, r, v, node<<1|1, m+1, e);
        T[node] = std::min(T[node<<1], T[node<<1|1]);
    }
    pair<ll,ll> query(int l, int r, int node=1, int s=0, int e=(1<<15)-1){
        push(node, s, e);
        if(r < s || e < l) return make_pair(INF,INF);
        if(l <= s && e <= r) return T[node];
        int m = s + e >> 1;
        return std::min(query(l, r, node<<1, s, m), query(l, r, node<<1|1, m+1,
e));
    }
} T;

ll f(const Point &pt, int a, int b){
    return pt.x*a + pt.y*b;
}

Point Optimize(int a, int b){
    T.clear();
    for(int i=1; i<=N; i++){
        T.add(S[i], E[i], b);
        auto [cst,cnt] = T.query(0, C.size() - 2);
        T.min(S[i], E[i], make_pair(cst+a, cnt+1));
    }
    auto [cst,cnt] = T.query(0, C.size() - 2);
    return {cnt, (cst - cnt*a) / b};
}

void DnC(Point le, Point dw){
    ll dx = dw.x - le.x, dy = le.y - dw.y;
    auto pt = Optimize(dy, dx);
    if(CCW(le, pt, dw) > 0){
        R.push_back(pt);
        DnC(le, pt);
```

```
        DnC(pt, dw);
    }
}

void init(int n, int m, vector<int> s, vector<int> e){
    C.clear(); R.clear();
    N = n; M = m; C.reserve(N*2+2);
    C.push_back(0); C.push_back(M);
    copy(s.begin(), s.end(), S+1);
    copy(e.begin(), e.end(), E+1);
    for(int i=1; i<=N; i++) C.push_back(S[i]), C.push_back(E[i]+1);
    sort(C.begin(), C.end()); C.erase(unique(C.begin(), C.end()), C.end());
    for(int i=1; i<=N; i++) S[i] = lower_bound(C.begin(), C.end(), S[i]) -
C.begin();
    for(int i=1; i<=N; i++) E[i] = lower_bound(C.begin(), C.end(), E[i]+1) -
C.begin() - 1;

    Point L = Optimize(N*2, 1), D = Optimize(1, N*2);
    R.push_back(L); R.push_back(D);
    if(L != D) DnC(L, D);
    sort(R.begin(), R.end());
}

ll minimize(int a, int b){
    int l = 0, r = R.size() - 1;
    while(l+3 < r){
        int m1 = (l + l + r) / 3, m2 = (l + r + r) / 3;
        ll f1 = f(R[m1], a, b), f2 = f(R[m2], a, b);
        if(f1 < f2) r = m2;
        else l = m1;
    }
    ll res = 0x3f3f3f3f3f3f3f3f;
    for(int i=l; i<=r; i++) res = min(res, f(R[i], a, b));
    return res;
}
```