2023-2 SCCC 내부 대회 #3

A. 수업 참여도 (BOJ 7173)

단순 구현 문제입니다.

```
#include <bits/stdc++.h>
using namespace std;
constexpr int di[] = {1, -1, 0, 0};
constexpr int dj[] = {0, 0, 1, -1};

int N, M, A[222][222], S;
inline bool Bound(int i, int j){ return 1 <= i && i <= N && 1 <= j && j <= M; }

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    for(int i=1; i<=N; i++) for(int j=1; j<=M; j++) { char c; cin >> c; A[i][j]
= c - '0'; }
    for(int i=1; i<=N; i++){
        for(int j=1; j<=M; j++){
            int now = 0, cnt = 0;
            for(int k=0; k<4; k++){
                int r = i + di[k], c = j + dj[k];
                if(Bound(r, c)) cnt++, now += abs(A[i][j] - A[r][c]);
            }
            S += now * (12 / cnt);
        }
    }
    cout << fixed << setprecision(4) << S / 12.L;
}
```

B. 톱니바퀴 수열 (BOJ 7239)

입력으로 주어지는 수가 모두 다릅니다. 따라서 N개의 수를 모두 정렬한 다음, $2k-1$번째 원소와 $2k$번째 원소를 교환해서 조건을 만족하는 수열을 만들 수 있습니다.

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

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int N; cin >> N;
    vector<int> V(N);
    for(auto &i : V) cin >> i;
    sort(V.begin(), V.end());
    for(int i=2; i<N; i+=2) swap(V[i-1], V[i]);
    copy(V.begin(), V.end(), ostream_iterator<int>(cout, " "));
}
```

C. 방 탈출 (BOJ 28789)

만약 모든 문자가 서로 다르다면 정답은 $N(N-1)/2$입니다. 같은 문자가 여러 개 존재하면 두 위치를 선택하는 방법이 다르더라도 결과 문자열이 같아질 수 있기 때문에, 이 부분을 처리해야 합니다.

어떤 문자 i가 등장한 횟수를 $C[i]$라고 합시다. 서로 다른 두 문자를 선택해서 교환하는 방법의 개수는 $\{\sum C_i(N-C_i)\}/2$입니다. 만약 $C_i \geq 2$인 문자가 하나라도 존재한다면 두 문자를 교환해서 이전과 동일한 문자열을 얻을 수 있으므로 정답에 1을 더하면 됩니다.

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

ll N, C[333], R; string S;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> S; N = S.size();
    for(auto c : S) C[c]++;
    for(int i=0; i<333; i++) R += C[i] * (N - C[i]);
    cout << R / 2 + (count_if(C, C+333, [](ll v){ return v > 1; }) != 0);
}
```

D. 잔디깎기 (BOJ 7257)

매번 모든 잔디의 길이를 B_j씩 감소시키는 방식으로 구현하면 시간 복잡도가 $O(NM)$이 되어서 제한 시간 안에 문제를 해결할 수 없습니다.

길이가 짧은 잔디가 긴 잔디보다 먼저 잘려 나갈 것이니 잔디를 길이 내림차순으로 정렬하고, 완전히 잘려 나간 잔디를 `pop_back`하는 방식으로 구현할 것입니다. 매번 잔디의 길이를 감소시키는 것은 비효율적이므로 **아직 잘려 나가지 않은 잔디들의 길이 변화량** F를 관리합니다. 매일 F는 $1-B_j$만큼 증가합니다.

F를 알고 있으면 초기에 길이가 h였던 잔디가 완전히 잘려 나갔는지 판별하는 것은 $h+F \leq 0$으로 확인할 수 있고, 남아 있는 잔디의 길이도 (잔디의 초기 길이의 합) + (남아 있는 잔디의 수) * F로 구할 수 있습니다. 전체 시간 복잡도는 $O(N+M)$입니다.

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

ll N, Q, F, S;
vector<ll> V;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N; V.resize(N);
    for(auto &i : V) cin >> i;
    sort(V.begin(), V.end(), greater<>());
    S = accumulate(V.begin(), V.end(), 0LL);
    cin >> Q;
    for(int q=1; q<=Q; q++){
        ll t; cin >> t;
        while(!V.empty() && V.back() + F + 1 - t <= 0) S -= V.back() + F,
V.pop_back();
        F += 1 - t; S += (1 - t) * V.size();
```

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

E. 격자 색칠 (BOJ 30168)

$R_i = k$라는 것은 $(i,1),(i,2),\cdots,(i,k)$는 검은색, $(i,k+1)$은 흰색으로 칠해져 있고, 그 이후는 어떤 색이 되더라도 상관 없다는 것을 의미합니다. 마찬가지로, $C_j = k$라는 것은 $(1,j),(2,j),\cdots,(k,j)$는 검은색, $(k+1,j)$는 흰색으로 칠해져야 하고, 그 이후에 대한 제약은 없음을 의미합니다.

따라서 각 칸은 아래 네 가지로 분류할 수 있습니다.

0. 원하는 색으로 칠할 수 있음
1. 검은색으로 칠해야 함
2. 흰색으로 칠해야 함
3. 검은색으로 칠해야 하면서 흰색으로 칠해야 함 (모순)

(3)과 같은 칸이 있을 때의 정답은 0이고, 그렇지 않으면 2^(0 개수)가 정답입니다.

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

int N, M, R[1010], C[1010], A[1010][1010], X=1;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    for(int i=1; i<=N; i++) cin >> R[i];
    for(int i=1; i<=M; i++) cin >> C[i];
    for(int i=1; i<=N; i++){
        for(int j=1; j<=R[i]; j++) A[i][j] |= 1;
        A[i][R[i]+1] |= 2;
    }
    for(int j=1; j<=M; j++){
        for(int i=1; i<=C[j]; i++) A[i][j] |= 1;
        A[C[j]+1][j] |= 2;
    }
    for(int i=1; i<=N; i++){
        for(int j=1; j<=M; j++){
            if(A[i][j] == 3) X = 0;
            else if(A[i][j] == 0 && (X += X) >= MOD) X -= MOD;
        }
    }
    cout << X;
}
```

F. 도로 건설하기 (BOJ 7255)

문제 지문을 읽으면 알 수 있듯이, 최소 신장 트리를 구하는 것과 비슷한 방식으로 해결할 수 있습니다.

먼저 Union-Find를 이용해 입력으로 주어진 간선으로 연결되어 있는 정점을 합친 다음 크루스칼 알고리즘을 이용해 MST를 구하면 됩니다. 다만, 서로 다른 컴포넌트를 연결하는 간선을 모두 구한 다음 크루스칼 알고리즘을 수행하면 간선 개수가 너무 많아서 시간 초과를 받게 되고, 가중치가 가장 작은 정점과 다른 모든 컴포넌트를 연결하는 방식으로 MST를 구해야 합니다.

Union-Find에서 각 집합에 속한 원소 가중치의 최솟값을 함께 관리하면 쉽게 구현할 수 있습니다.

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

int N, M, P[101010], A[101010]; long long R;
int Find(int v){ return v == P[v] ? v : P[v] = Find(P[v]); }
bool Merge(int u, int v){ return Find(u) != Find(v) && (A[P[v]]=min(A[P[v]],
A[P[u]]), P[P[u]]=P[v], true); }

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M; iota(P+1, P+N+1, 1);
    for(int i=1; i<=N; i++) cin >> A[i];
    for(int i=1,u,v; i<=M; i++) cin >> u >> v, Merge(u, v);
    vector<int> V;
    for(int i=1; i<=N; i++) if(i == Find(i)) V.push_back(A[i]);
    sort(V.begin(), V.end());
    for(int i=1; i<V.size(); i++) R += 1LL * V[0] * V[i];
    cout << R;
}
```

G. 수열 만들기 (BOJ 28608)

$N \leq 3$이면 조건을 만족하는 수열을 만들 수 없다는 것은 어렵지 않게 알 수 있습니다.

만약 N이 소수가 아니면서 소수의 제곱도 아니라면 N의 약수 d를 하나 잡아서 $B = \{d, N/d\}$로 둘 수 있습니다. $N > 4$면 $d + N/d < N$이므로 A는 $d + N/d$와 1 여러 개로 채우면 됩니다.

N이 소수 또는 소수의 제곱이라면 $N > 3$인 경우만 생각해도 되므로 $N = 4$인 경우를 제외하면 $N + 1$은 항상 짝수입니다. 따라서 $B = \{1, N\}$, $A = \{2, (N+1)/2, 1, \cdots\}$으로 하면 됩니다.

이제 $N = 4$인 경우만 남았고, 이건 손으로 적당히 만들면 문제를 해결할 수 있습니다.

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

int FindDivisor(int n){
    if(n < 2) return 0;
    for(int i=2; i*i<=n; i++) if(n % i == 0) return i;
    return 0;
}

pair<vector<int>, vector<int>> Solve(int n){
    if(n <= 3) return {};
    if(n == 4) return { vector<int>{2, 2}, vector<int>{4} };
    if(int d=FindDivisor(n); d != 0 && d * d != n){
        int one = n - d - n/d;
        vector<int> a(one+1, 1), b = {d, n/d};
        a[0] = d + n / d;
        return {std::move(a), std::move(b)};
    }
    else{ // assert(n % 2 == 1);
        int one = n - 2 - (n+1)/2;
        vector<int> a(one+2, 1), b = {1, n};
        a[0] = 2; a[1] = (n + 1) / 2;
        return {std::move(a), std::move(b)};
    }
}
```

```
int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int N; cin >> N;
    auto [A,B] = Solve(N);
    if(A.empty()){ cout << "-1 -1\n"; return 0; }
    cout << A.size() << " " << B.size() << "\n";
    for(auto i : A) cout << i << " "; cout << "\n";
    for(auto i : B) cout << i << " "; cout << "\n";
}
```

H. 블록 게임 (BOJ 28753)

점화식을 $D(i) = \max_{i \le j \le N} \{A_j - D_{j+1}\}$와 같이 세우면 정답은 $D(1)$이 됩니다. 점화식을 naive하게 계산하면 $O(N^2)$이 돼서 제한 시간 안에 문제를 해결할 수 없습니다. 세그먼트 트리를 이용해 $A_j - D_{j+1}$의 최댓값을 관리하면 $O(N \log N)$에 계산할 수 있고, 최댓값을 구해야 하는 구간의 끝점이 항상 N이라는 점을 이용하면 $O(N)$ 시간에도 해결할 수 있습니다.

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

ll N, A[202020], D[202020], M=-9e18;

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--) D[i] = M = max(M, A[i] - D[i+1]);
    cout << D[1];
}
```