2023-2 SCCC 내부 대회 #2

A. 보자기 (BOJ 28960)

빨랫줄과 평행한 변이 없도록 보자기를 거는 방식은 고려할 필요가 없습니다. 따라서 길이가 A인 변이 빨랫줄과 평행하도록 거는 것과 길이가 B인 변이 빨랫줄과 평행하도록 거는 것 중 바닥에 끌리지 않게 거는 방법이 있는지 확인하면 됩니다.

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

int main(){
    int h, l, a, b; cin >> h >> l >> a >> b;
    if(a <= h * 2 && b <= l) cout << "YES";
    else if(b <= h * 2 && a <= l) cout << "YES";
    else cout << "NO";
}
```

B. 암호 분석 (BOJ 28967)

문제 풀이는 코드를 참고하세요.

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

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    string p, s; cin >> p >> s;
    int res = 0;
    for(int i=0, j=0; i<s.size(); i++) if(s[i] == p[j] && ++j == p.size())
res++, j = 0;
    cout << res;
}
```

C. 심부름 (BOJ 28836)

편의상 집을 0번 지역, 마트를 1번 지역, 경비실을 2번 지역이라고 합시다. i번 지역에서 j번 지역으로 가는 최단 거리 $D(i,j)$를 미리 구해놓으면, 이동 경로를 생각하는 대신 단순히 물건을 받고 집에 가져다 놓는 것만 생각해도 되므로, 문제를 조금 더 편하게 해결할 수 있습니다.
$D(0,1) = D(1,0) = \min(a, b+c)$, $D(0,2) = D(2,0) = \min(b, a+c)$,
$D(1,2) = D(2,1) = \min(c, a+b)$라는 것을 이용해 D 배열을 미리 채웁시다.

물건을 들고 다니는 것에 따라, 경로를 다음과 같이 4개로 분류할 수 있습니다.

1. 집 - 마트(식재료) - 집 - 경비실(택배) - 집
2. 집 - 경비실(택배) - 집 - 마트(식재료) - 집
3. 집 - 마트(식재료) - 경비실(택배) - 집
4. 집 - 경비실(택배) - 마트(식재료) - 집

이때 1, 2는 이동 시간이 같기 때문에 세 가지 경로만 고려하면 되고, 이는 D 배열을 이용해 쉽게 계산할 수 있습니다.

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

double A, B, C, V[3], D[3][3];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> A >> B >> C >> V[0] >> V[1] >> V[2];
    D[0][1] = D[1][0] = min(A, B + C);
    D[0][2] = D[2][0] = min(B, A + C);
    D[1][2] = D[2][1] = min(C, A + B);

    double r1 = D[0][1] / V[0] + D[1][0] / V[1] + D[0][2] / V[0] + D[2][0] /
V[1];
    double r2 = D[0][1] / V[0] + D[1][2] / V[1] + D[2][0] / V[2];
    double r3 = D[0][2] / V[0] + D[2][1] / V[1] + D[1][0] / V[2];
    cout << fixed << setprecision(10) << min({r1, r2, r3});
}
```

D. 평행 우주 (BOJ 28837)

만약 A에서는 켜져 있는데 B에서는 꺼져 있는 비트가 있으면 -1을 출력하고 프로그램을 종료하면 됩니다. 이제 그렇지 않은 경우, 즉 $A \vee B = B$인 경우만 생각합시다.

이미 켜진 비트는 or 연산을 해도 바뀌지 않기 때문에, A에서는 꺼져 있고 B에서는 켜져 있는 비트들만 신경써도 됩니다. 따라서 버튼을 눌렀을 때 비트가 하나 이상 켜지면서($A \vee a_i \neq A$) B에서 꺼져 있는 비트가 켜지지 않도록 하는($A \vee a_i \vee B = B$) 버튼을 찾아서 누르는 것을 반복하면 문제를 해결할 수 있습니다.

입력으로 주어지는 수는 모두 $10^9 < 2^{30}$ 이하이므로 비트가 하나 이상 켜지는 버튼만 누르면 30개 이하의 버튼을 눌러서 문제를 해결할 수 있습니다.

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

int N, S, E, A[101010];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> S >> E;
    for(int i=1; i<=N; i++) cin >> A[i];
    if((S | E) != E){ cout << -1; return 0; }

    vector<int> R;
    for(int i=1; i<=N; i++){
        if((S | A[i]) != S && (S | A[i] | E) == E){
            R.push_back(i); S |= A[i];
        }
    }
    if(S != E){ cout << -1; return 0; }

    cout << R.size() << "\n";
    copy(R.begin(), R.end(), ostream_iterator<int>(cout, " "));
}
```

E. 쓸어담기 (BOJ 28729)

많이 구매하게 되는 물건부터 가격을 줄이는 것이 최적이라는 것은 쉽게 알 수 있습니다. 각 물건을 구매하는 횟수를 구하는 방법을 먼저 살펴 봅시다.

쓸어담기를 할 때마다 L_i번째 물건부터 R_i번째 물건까지를 한 번씩 구매하게 됩니다. 따라서 $S[L_i], S[L_i+1], \cdots, S[R_i]$를 1씩 증가시켜야 하며, 이는 $S[L_i]$에 +1을 더하고 $S[R_i+1]$에 -1을 더한 다음, 마지막에 배열 S의 누적 합을 구하는 것으로 처리할 수 있습니다. $O(N+M)$ 시간에 각 물건의 구매 횟수를 구할 수 있습니다.

각 물건의 구매 횟수를 구했으니 이제 많이 구매하는 물건부터 가격을 최대한 많이 깎으면 됩니다. 물건을 구매 횟수의 내림차순으로 정렬해서 차례대로 순회하는 것으로 정답을 구할 수 있습니다.

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

ll N, M, K, A[101010], S[101010], R;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M >> K;
    for(int i=1; i<=N; i++) cin >> A[i];
    for(int i=1,u,v; i<=M; i++) cin >> u >> v, S[u]++, S[v+1]--;
    partial_sum(S+1, S+N+1, S+1); // for(int i=1; i<=N; i++) S[i] += S[i-1];

    vector<int> O(N);
    iota(O.begin(), O.end(), 1); // for(int i=0; i<N; i++) O[i] = i+1;
    sort(O.begin(), O.end(), [](int x, int y){ return S[x] > S[y]; });

    for(auto i : O){
        ll now = min(K, A[i]);
        A[i] -= now; K -= now;
        R += A[i] * S[i];
    }
    cout << R;
}
```

F. 배열 자르기 (BOJ 9788)

$1..n$번째 원소를 k개의 부분 배열로 나누었을 때의 최솟값을 $D(k,n)$이라고 정의합시다. $l_s, l_{s+1}, \cdots, l_e$의 최댓값과 최솟값을 각각 $M(s,e), m(s,e)$라고 하면, 점화식은
$D(k,n) = \min_{i<n} \{D(k-1,i) + M(i+1,n) - m(i+1,n)\}$과 같이 나타낼 수 있습니다. 따라서 $O(KN^2)$ 시간에 배열 D의 모든 원소를 채우고 정답을 구할 수 있습니다.

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

int N, K, A[444], D[444][444];

void Solve(){
    cin >> N >> K;
    for(int i=1; i<=N; i++) cin >> A[i];
    memset(D, 0x3f, sizeof D);
    D[0][0] = 0;
    for(int k=1; k<=K; k++){
```

```
        for(int i=1; i<=N; i++){
            int mx = -1e9, mn = 1e9;
            for(int j=i; j>=1; j--){
                mx = max(mx, A[j]);
                mn = min(mn, A[j]);
                D[k][i] = min(D[k][i], D[k-1][j-1] + mx - mn);
            }
        }
    }
    cout << D[K][N] << "\n";
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int TC; cin >> TC;
    for(int tc=1; tc<=TC; tc++) Solve();
}
```

G. 격자 게임 (BOJ 28754)

만약 게임을 먼저 시작한 사람(선공)이 정확히 게임판의 중앙을 차지할 수 있다면, 그 이후로 선공은 항상 후공의 행동을 점대칭으로 따라하는 것으로 승리할 수 있습니다. 반대로 선공이 게임판의 중앙을 차지하지 못한다면 후공이 선공의 행동을 점대칭으로 따라해서 승리할 수 있습니다.

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

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int N, M, S; cin >> N >> M >> S;
    if(S == 1) cout << (N * M % 2 ? "YES" : "NO");
    else if(N % 2 == 1 || M % 2 == 1 || S >= 4) cout << "YES";
    else cout << "NO";
}
```

H. 부분 문자열과 쿼리 (BOJ 28947)

i번째 쿼리로 $1\ l\ r\ s$가 주어진 상황을 생각해 봅시다. 이건 x좌표가 i이고 y좌표가 $[l, r]$인 길쭉한 직사각형 영역에 $|s|$를 더하는 것이라고 생각할 수 있습니다. 1번 쿼리를 이런 관점에서 바라보면, 2번 쿼리를 해결하는 것은 쿼리로 주어진 행에서 이분 탐색을 하면서, 몇 번째 쿼리에서 추가된 문자열인지 확인해서 출력하면 됩니다. 2차원 세그먼트 트리에서 레이지 프로퍼게이션을 하는 것은 귀찮으므로, 분할 정복이나 스위핑같은 걸 적용하는 방법을 고민해 봅시다.

쿼리를 오프라인으로 처리할 것입니다. 모든 쿼리를 전부 입력받은 다음 y축 방향으로 밀고 올라가면서, 1번 쿼리는 $y = l$인 시점에 $x = i$인 지점에 $|s|$를 더하고 $y = r + 1$인 시점에 $x = i$인 지점에 $|s|$를 빼는 것으로 처리할 수 있습니다. 따라서 구간의 합을 구하는 연산과 k번째 원소를 구하는 연산을 지원하는 세그먼트 트리를 이용해 문제를 해결할 수 있습니다.

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
constexpr int SZ = 1 << 18;

ll T[SZ<<1];
```

```cpp
void Add(int x, ll v){
    for(x|=SZ; x; x>>=1) T[x] += v;
}

int Kth(ll k){
    int x = 1;
    while(x < SZ){
        if(k <= T[x<<1]) x = x << 1;
        else k -= T[x<<1], x = x << 1 | 1;
    }
    return x ^ SZ;
}

ll Sum(int l, int r){
    ll res = 0;
    for(l|=SZ, r|=SZ; l<=r; l>>=1, r>>=1){
        if(l & 1) res += T[l++];
        if(~r & 1) res += T[r--];
    }
    return res;
}

int N, Q, X, Y;
string S[SZ], R[SZ];
vector<int> Ins[SZ], Del[SZ];
vector<tuple<int,int,int>> Ask[SZ];

string Print(ll l, ll r){
    int pos = Kth(l);
    int idx = l - Sum(0, pos-1) - 1;
    string res;
    while(true){
        res += S[pos][idx]; if(++l > r) break;
        if(++idx == S[pos].size()) idx = 0, pos = Kth(l);
    }
    return res;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> Q;
    for(int q=1; q<=Q; q++){
        int op; cin >> op;
        if(op == 2){
            ll i, x, y; cin >> i >> x >> y;
            Ask[i].emplace_back(x, y, ++X);
        }
        else{
            ll l, r; Y++; cin >> l >> r >> S[Y];
            Ins[l].push_back(Y); Del[r].push_back(Y);
        }
    }
    for(int i=1; i<=N; i++){
        for(auto s : Ins[i]) Add(s, +(int)S[s].size());
        for(auto [l,r,idx] : Ask[i]) R[idx] = Print(l, r);
        for(auto s : Del[i]) Add(s, -(int)S[s].size());
    }
    for(int i=1; i<=X; i++) cout << R[i] << "\n";
```

```
}
```