2023-2 SCCC 내부 대회 #1

A. 컨베이어 벨트 (BOJ 28938)

주어진 수들의 합이 양수면 `Right`, 음수면 `Left`, 0이면 `Stay`를 출력하면 됩니다.

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

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int N, S = 0; cin >> N;
    for(int i=0,t; i<N; i++) cin >> t, S += t;
    cout << (S > 0 ? "Right" : S < 0 ? "Left" : "Stay");
}
```

B. 축제 부스 기획하기 1 (BOJ 29558)

N이 홀수이면 $\cdots, D-2, D-1, D, D+1, D+2, \cdots$를 출력하면 된다는 것은 쉽게 알 수 있습니다. 짝수일 때도 비슷하게 $y = |x - D|$의 그래프 위에서 선택한 수들이 대칭을 이루도록 만들면 되고, 따라서 $\cdots, D-4, D-2, D+2, D+4, \cdots$와 같이 출력하면 됩니다.

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

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int N, X; cin >> N >> X;
    for(int i=1; i<=N/2; i++) cout << X + i * 2 << " " << X - i * 2 << " ";
    if(N % 2) cout << X;
}
```

C. 축제 부스 기획하기 2 (BOJ 28813)

(x, y)에 위치하는 드론은 $\max(|x|, |y|)$초가 지나야 원점에 도달할 수 있습니다. 따라서 만약 모든 드론을 없애는 방법이 존재한다면, N개의 드론을 원점에 도달할 수 있는 최소 시간인 $\max(|x|, |y|)$가 증가하는 순서대로 제거하는 방법도 정답이 될 수 있습니다. 드론의 좌표를 입력받고 정렬하는 데 $O(N \log N)$만큼의 시간이 걸리므로 전체 시간 복잡도는 $O(N \log N)$입니다.

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

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int N; cin >> N;
    vector<pair<int,int>> V;
    for(int i=1; i<=N; i++){
        int x, y; cin >> x >> y;
        V.emplace_back(max(abs(x),abs(y)), i);
    }
    sort(V.begin(), V.end());
```

```
    for(int i=0; i<V.size(); i++) if(V[i].first <= i) { cout << -1; return 0; }
    for(auto [a,b] : V) cout << b << " ";
}
```

D. 축제 부스 기획하기 3 (BOJ 28601)

원점을 사용하지 않고 삼각형을 만들었다면 그 삼각형을 평행이동해서 원점을 포함하도록 만들 수 있으므로, 최소한 한 꼭짓점은 원점에서 움직이지 않는다고 생각해도 괜찮습니다.

원점이 아닌 두 꼭짓점을 $A(x_1, y_1), B(x_2, y_2)$라고 합시다. 삼각형의 넓이는 $\frac{1}{2} \times \overline{OA} \times \overline{OB} \times \sin\angle AOB$이고, 우리는 이 값이 $S/2$ 이상이 되길 원합니다. 삼각형의 넓이가 늘어난다고 해서 손해를 보지 않으며 택시 거리에서는 축에 평행하게 이동하는 것이 좋으므로, A와 B를 각각 x축과 y축 위에 놓아서 $\sin\angle AOB$를 1로 만들 것입니다.

즉, 우리는 적당한 두 정수 x_1, y_2를 정해서 $x_1 y_2/2 \geq S/2$가 되도록 만들 때 $|x_1| + |y_2|$를 최소화하면 되고, 정수 조건이 없다면 산술 기하 부등식에 의해 $x_1 = y_2 = \sqrt{N}$일 때 $2\sqrt{N}$으로 최소가 된다는 것을 알 수 있습니다. 정수 조건이 붙어있으니 $\lceil 2\sqrt{N} \rceil$을 구하면 됩니다.

`math.h`(또는 `cmath`)에 있는 `sqrt` 함수는 부동소수점을 사용하기 때문에 오차가 있을 수 있습니다. 하지만 $< 2^{64}$ 범위에서는 그 오차가 별로 크지 않기 때문에, 단순히 `sqrt(S)`를 구한 다음 적당히 1씩 더하고 빼는 방식으로 보정할 수 있습니다. 자세한 구현은 아래 코드를 참고하세요.

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

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    ll N; cin >> N; N *= 4;
    if(N == 0){ cout << 0; return 0; }
    ll R = sqrt(N);
    while(R * R < N) R++;
    while((R - 1) * (R - 1) >= N) R--;
    cout << R;
}
```

E. 축제 부스 기획하기 4 (BOJ 28841)

만약 두 구간 $[l_1, r_1]$과 $[l_2, r_2]$가 겹치지 않는다면, 두 구간 사이의 구간을 모두 포함하도록($r_1 < l_2$이면 $[l_1, l_2 - 1], [r_1 + 1, r_2]$) 만드는 것으로 문제의 조건을 지키면서 구간의 길이를 늘일 수 있습니다. 따라서 두 구간의 교집합이 존재하는 경우만 고려해도 충분합니다.

$l < r$이면서 $A_l = A_r = 0$인 상황을 생각해 봅시다. 이때는 두 구간을 $[l, r-1], [l+1, r]$로 잡아서 maximal한(더 이상 확장할 수 없는) 후보해를 하나 구할 수 있습니다. 마찬가지로 $A_l = A_r = 1$일 때도 $[l, r-1], [l+1, r]$이 하나의 후보가 됩니다. 그리고 사실 잘 생각해 보면 이런 형태의 후보만 고려해도 정답을 구할 수 있다는 것을 알 수 있습니다.

따라서 0과 1 모두 최대 한 번만 등장하는 수열에서는 조건을 만족하는 두 구간을 찾을 수 없고, 그렇지 않은 경우에는 가장 먼 두 0의 거리와 가장 먼 두 1의 거리 중 더 큰 것을 골라서 $[l, r-1]$과 $[l+1, r]$을 출력하면 됩니다.

누적 합 배열과 이분 탐색을 이용한 풀이도 있습니다.

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

```
int N; string S;
vector<int> P[2];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> S; N = S.size();
    for(int i=0; i<N; i++) P[S[i]-'0'].push_back(i+1);
    if(P[0].size() < 2 && P[1].size() < 2){ cout << -1; return 0; }
    if(P[0].size() < 2 || P[1].size() >= 2 && P[0].back() - P[0].front() <
P[1].back() - P[1].front()) swap(P[0], P[1]);
    cout << P[0].front() << " " << P[0].back() - 1 << " ";
    cout << P[0].front() + 1 << " " << P[0].back();
}
```

F. 단순한 문제 (BOJ 28744)

열심히 식을 정리합시다.

$$\sum_{i=1}^{N-2}\sum_{j=i+1}^{N-1}\sum_{k=j+1}^{N} A_i \times A_j \times A_k$$

$$= \sum_{i=1}^{N-2} A_i \times \left\{ \sum_{j=i+1}^{N-1} A_j \times (A_{j+1} + A_{j+2} + \cdots + A_N) \right\}$$

이때 $S_i = A_i + A_{i+1} + \cdots + A_N = \sum_{k=i}^{N} A_k$라고 정의하면

$$\sum_{i=1}^{N-2} A_i \times \left\{ \sum_{j=i+1}^{N-1} A_j \times (A_{j+1} + A_{j+2} + \cdots + A_N) \right\}$$

$$= \sum_{i=1}^{N-2} A_i \times \left\{ \sum_{j=i+1}^{N-1} A_j \times S_{j+1} \right\}$$

$F_i = A_i S_{i+1} + A_{i+1} S_{i+2} + \cdots + A_{N-1} S_N = \sum_{k=i}^{N-1} A_k S_{k+1}$이라고 정의하면

$$\sum_{i=1}^{N-2} A_i \times \left\{ \sum_{j=i+1}^{N-1} A_j \times S_{j+1} \right\}$$

$$= \sum_{i=1}^{N-2} A_i \times F_{i+1}$$

따라서 S와 F를 각각 $O(N)$ 시간에 전처리하면 전체 문제를 $O(N)$에 해결할 수 있습니다.

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

ll N, A[1010101], S[1010101], F[1010101], 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--) S[i] = (S[i+1] + A[i]) % MOD;
    for(int i=N; i>=1; i--) F[i] = (F[i+1] + A[i] * S[i+1]) % MOD;
    for(int i=1; i<=N; i++) R = (R + A[i] * F[i+1]) % MOD;
    cout << R;
}
```

G. 간단한 문제 (BOJ 28944)

$A_i - A_j < j - i$를 잘 정리하면 $A_i + i < A_j + j$가 됩니다. $i < j$인 모든 i, j에 대해 $A_i + i < A_j + j$가 성립하도록 원소를 최대한 많이 뽑는 것은, 수열 $B = \{A_0 + 0, A_1 + 1, A_2 + 2, \cdots, A_{N-1} + (N-1)\}$의 LIS를 구하는 것과 같습니다. 따라서 $O(N \log N)$ 시간에 문제를 해결할 수 있습니다.

```cpp
#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), R;
    for(auto &i : V) cin >> i;
    for(int i=0; i<N; i++) V[i] += i;
    for(auto i : V){
        if(R.empty() || R.back() < i) R.push_back(i);
        else *lower_bound(R.begin(), R.end(), i) = i;
    }
    cout << R.size();
}
```

H. 바둑알 배치하기 (BOJ 28730)

$N \times N$ 크기의 격자에 N개의 바둑알을 배치해야 하므로, 바둑알을 한 줄에 하나씩 배치해야 할 것 같다는 추측을 할 수 있습니다.

어떤 칸 (i, j)와 이웃한 4칸 $(i-1, j), (i+1, j), (i, j-1), (i, j+1)$ 중 3칸이 막혀있다면, (i, j)에 도미노를 놓는 방법은 유일하게 결정됩니다. 이 관찰과 한 줄에 하나씩 배치한다는 아이디어를 잘 적용하면 N이 홀수일 때의 답은 어렵지 않게 찾을 수 있습니다.

```
#....
....#
#....
....#
#....
```

N이 짝수일 때는 한 줄에 `.`이 홀수 개씩 있기 때문에 지금의 풀이를 적용할 수 없습니다. 하지만 이 풀이는 행의 개수와 관계없이 열의 개수가 홀수이면 항상 적용할 수 있기 때문에, N이 짝수일 때는 단순히 제일 오른쪽에 `.`으로만 구성된 열을 하나 추가하면 문제를 해결할 수 있습니다.

```
#.....
....#.
#.....
....#.
#.....
....#.
```

아래 코드에서 `std::string(n, c)`는 `std::string`을 길이가 `n`이고 모든 문자가 `c`인 문자열로 초기화하는 생성자입니다.

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

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int N; cin >> N;
    for(int i=1; i<=N; i++){
        if(i % 2 == 1) cout << "#" + string(N-1, '.') << "\n";
        else cout << string(N-2+N%2, '.') + "#" + (N % 2 ? "" : ".") << "\n";
    }
}
```