"점프 게임 (jumpgame)" 문제 풀이

작성자: 구재현

부분문제 1

N 이 작기 때문에 배열 $A[i]$ 를 변홧값 배열을 사용하여 직접 계산할 수 있다. $f(i)$ 를 주인공이 i 번 발판에서 출발했을 때 얻을 수 있는 최대 점수라고 할 때, $f(i) = \max(f(i+K) + A[i], f(i+1))$ 이 성립한다. 이 동적 계획법 점화식을 $O(N)$ 에 구현하면, 부분문제 1을 통과할 수 있다.

부분문제 2

점프를 하더라도 다음 칸에 떨어지기 때문에 모든 발판의 점수를 얻을 수 있다. 고로 답은 모든 $A[i]$ 의 합이다. 이는 아래와 같이 계산할 수 있다.

```
return accumulate(R.begin(), R.end(), 0ll) - accumulate(L.begin(), L.end(), 0ll) + Q;
```

부분문제 4

우리가 해결하고자 하는 문제는, 서로 간격이 K 이상인 점들을 $[0, N-1]$ 에서 적절히 골라서, $A[i]$ 의 합을 최대화하는 것으로 해석할 수 있다. 합을 최대화하는 점들의 번호를 $i_1, i_2, \ldots, i_m$ 라고 하자. 다음 사실이 성립한다:

Lemma. 모든 $1 \le x \le m$ 에 대해, $i_x = E[j] - tK$ 를 만족하는 어떠한 $0 \le j \le N-1, 0 \le t$ 가 존재한다. (즉, i_x 는 $E[j], E[j]-K, E[j]-2K, \ldots$ 중 하나이다)

증명. 어떠한 해의 i_x 가 위 형태를 만족하지 않는다고 하자. 이 때 위 형태를 만족하지 않는 i_x 중 x 를 최대화하는 점을 고른다.

- $x = m$ 인 경우, i_x 를 포함하는 구간이 존재한다면, 이들 중 끝점이 최소인 구간의 끝점으로 옮긴다. 포함하는 구간이 없다면 아예 점을 지워도 된다.
- $x < m$ 인 경우, $i_{x+1} = E[j] - tK$ 가 성립하며, 고로 가정에 의해 $i_x < E[j] - (t+1)K$ 이다. i_x 를 포함하고 $E[j]-(t+1)K$ 를 포함하지 않는 구간이 존재한다면, 이들 중 끝점이 최소인 구간의 끝점으로 옮긴다. 포함하는 구간이 없다면, $i_x = E[j] - (t+1)K$ 로 설정해 주면 된다.

이를 반복하면, 모든 해가 위 조건을 만족하게 할 수 있다. ■

부분문제 4에서는, K가 크기 때문에 $E[j] - tK$ 꼴의 점이 최대 $5Q$ 개이다. 위와 같은 점들의 정렬된 리스트를 $Q_1, Q_2, \ldots, Q_p$ 라고 하자. 이 점들만을 상태로 가지는, 다음과 같은 동적 계획법을 사용한다. $DP[i]$ 를, 서로 간격이 K 이상인 점들을 $Q_1, Q_2, \ldots, Q_i$ 에서 적절히 골라서 얻을 수 있는 $A[Q_i]$ 합의 최대라고 하면, 다음 식이 성립한다 ($Q_0 = -\infty$ 로 두자):

$$DP[i] = \max(DP[i-1], DP[j] + A[Q_i])$$
$$\text{where } j = \max_{0 \le k \le i-1, Q_k \le Q_i - K} k$$

이 동적 계획법 점화식을 계산하면 문제를 해결할 수 있다. 하지만 그대로 계산하면 비효율적일 수 있어 다음과 같은 최적화가 필요하다.

- 첫 번째 문제는 $A[Q_i]$ 를 계산하는 것이다. 부분문제 1에서 사용한 변홧값 배열을 똑같이 사용하는데, 증가시키는 위치를 이분 탐색으로 찾아주면 된다.
- 두 번째 문제는 j 를 계산하는 것이다. Two pointers를 사용하면 된다.

두 점을 고려해주면 부분문제 4를 $O(\frac{N}{K}Q\log Q)$ 에 해결할 수 있다.

부분문제 6

설명상 편의를 의해 부분문제 1의 동적 계획법을 다음과 같이 변형해서 사용한다. 부분문제 4의 관찰에 의해, 이렇게 변형하여도 답이 바뀌지 않는다.

$$f(i) = \max(f(i-K) + A[i], f(i-1))$$

관찰은, $f(i)$ 를 계산하는 과정을 1차원 DP의 형태로 보지 않고 2차원 DP의 형태로 보는 것이다. $i = qK+r$ ($0 \le r \le K-1$, q, r 은 음이 아닌 정수) 라고 하면, 위 점화식을 다음과 같이 표현할 수 있다. 편의상 N 이 K 의 배수라고 가정하자.

$$f(q,0) = \max(f(q-1,0) + A[q][0], f(q-1,K-1))$$
$$f(q,r) = \max(f(q-1,r) + A[q][r], f(q,r-1))$$

점화식을 위와 같이 표현할 경우, 두 가지 점을 활용할 수 있다. 첫 번째로, 부분문제 4에서 사용한 Lemma에 따라서, r은 $E[j] \pmod K$ 중 하나고, 고로 r 의 후보가 $O(N)$ 개로 줄어든다. 두 번째로, $A[q][r]$ 의 형태가 보통은 그렇게 복잡하지 않다. 예를 들어, *대다수의* 행 q 에 대해서 $A[q][r]$ 의 값은 모두 동일하다. 각 구간이 추가될 때마다 위 조건이 만족하지 않게 되는 행이 최대 두 개이기 때문이다. 구체적으로, 입력으로 받은 쿼리 구간들을 스위핑하듯이 처리하여 $A[i]$ 가 같은 구간끼리 묶게 되면, 최대 $2Q$ 개의 구간이 나온다. 만약 한 구간의 길이가 충분히 크다면, 이 구간은 여러 행을 전부 같은 값으로 덮게 된다.

각 행을 두 가지 종류로 구분하자. 첫 번째 종류는, 해당 행에 서로 다른 $A[q][r]$ 값이 존재하는 경우고, 두 번째 종류는, 해당 행의 모든 $A[q][r]$ 값이 같은 경우이다. 첫 번째 종류는 최대 $2Q$ 번 등장하니, $f(q,r)$ 을 배열 등으로 관리한다면 단순히 $O(Q)$ 정도에 계산해 줄 수 있다. $Q \le 5000$ 이라 이렇게 하여도 시간 복잡도가 $O(Q^2)$ 가 되어 시간 안에 동작한다.

두 번째 종류는, 해당 행의 모든 $A[q][r]$ 값이 같은 경우이다. 이 경우, 이 행을 완전히 포함하는 구간을 생각해 보면, 이 행 뿐만 아니라 이 행과 인접한 몇 개의 구간들에 대해서 $A[q][r]$ 값이 모두 같음을 알 수 있다. 고로 문제를 다음과 같은 식으로 환원할 수 있다.

- i 에 따라 증가하는 함수 $f(0,i)$가 있다. 모든 $A[i][j]$ 의 값이 X 일 때, $f(T,i)$ 함수를 계산하여라.

이 문제는 T 의 크기에 상관없이 $O(Q)$ 시간에 해결할 수 있다. $f(1,i)$ 를 단순하게 정의에 따라서 계산하면, $f(1,K-1) - f(1,0) \le X$ 가 성립한다. 고로 이 시점부터는 $f(q,i) = f(q-1,i) + X$ 의 형태로 점화식을 단순화할 수 있고, 이렇게 되면 단순히 구간에 $(T-1)X$ 를 더해주면 되기 때문이다.

위 방법을 사용하면 $f(\frac{N}{K},i)$ 를 모두 계산할 수 있고, 문제의 정답 역시 계산할 수 있다. 시간 복잡도는 $O(Q^2)$ 이다.

부분문제 7

$f(*, i)$ 를 효율적으로 관리하는 자료구조를 구성하고, 부분문제 6의 풀이를 사용하면 만점을 받을 수 있다. 함수 형태를 효율적인 자료구조로 관리하여 동적 계획법을 최적화한다는 점에서, Slope trick을 사용한 풀이라고 생각할 수도 있다.

자료구조는, 다음과 같은 정보를 관리한다:

- $f(i, 0)$ 의 값 (L)
- $f(i, K-1)$ 의 값 (R)
- $f(i, j) \neq f(i, j+1)$ 일 경우, $(j, f(i, j+1) - f(i, j))$ 의 쌍 (D)

세 번째 값 (D) 은 `std::map` 과 같은 자료구조로 관리한다.

두 번째 종류에서 나오는, 전체 $f(i, j)$ 에 특정 수 X 를 더하는 것은 L, R 을 조정하는 것으로 $O(1)$ 시간에 해결할 수 있다. 첫 번째 종류의 쿼리는, 구간에 특정 수를 더한 후, 구간 오른쪽에서 $f(i, j) > f(i, j+1)$ 인 지점이 발생하면 이 부분을 $f(i, j)$ 로 맞춰주어 정렬 조건을 유지하는 것임을 관찰할 수 있다. 만약 수를 더하는 구간이 $f(i, *)$ 의 Suffix라면, 연산 이후에도 정렬성이 깨지지 않아 D 에 값 하나를 추가하는 식으로 쉽게 구현할 수 있다. Suffix가 아닌 경우에는 쿼리의 결과가 $f(i, j)$ 의 정렬성을 깨서 문제가 되는데, 구간의 오른쪽 끝에서부터 순서대로 D 의 값을 제거하면서 정렬성을 보장하는 식으로 구현할 수 있다. 각 쿼리에서 D 에 삽입하는 값의 수가 $O(1)$ 개이기 때문에 총 $O(Q \log Q)$ 시간 복잡도에 문제를 해결할 수 있다.