"주차 타워" 문제 풀이

작성자: 김준원

부분문제 1

모든 차의 번호가 1이기 때문에, 차를 어떤 순서로 빼도 괜찮다. 버튼 A를 $N-1$번 누르거나, 버튼 B를 $N-1$번 누르면 모든 차를 뺄 수 있다. 이 방법이 가장 적은 횟수로 차를 빼는 방법이다.

따라서, $N-1$을 출력하면 된다.

부분문제 2

모든 차의 번호가 다르기 때문에, 차를 빼야 하는 순서가 유일하게 정해진다. 번호가 가장 작은 차부터 순서대로 빼어야 한다. 정해진 순서에 따라 이를 시뮬레이션하면 된다.

현재 맨 아래에 x번째 칸이 있고, y번째 칸에 있는 차를 빼야 하는 상황이라고 하자.

- $x \to y$ 방향으로 이동한다면 버튼을 $|x-y|$번 눌러야 한다.
- $x \to N \to 1 \to y$ 또는 $x \to 1 \to N \to y$ 방향으로 이동한다면 버튼을 $N-|x-y|$번 눌러야 한다.

따라서 차를 빼기 위해 버튼을 눌러야 하는 최소 횟수는 $\min(|x-y|, N-|x-y|)$번이다.

이 횟수를 N대의 차를 빼는 순서대로 계산해서 합치면 문제의 답이 된다.

부분문제 3

N개의 칸을 차례대로 모두 방문하는 순서는 최대 $N! = N \times (N-1) \times \cdots 2 \times 1$가지 존재한다.

$N!$가지의 순서를 백트래킹으로 모두 돌아보며, 어떤 순서로 방문하는 것이 가능한지, 가능하다면 이 때 버튼을 몇 번 누르게 되는지를 모두 계산하여 최적의 순서를 찾는다. 시간 복잡도는 $O(N \times N!)$이다.

부분문제 4

이제, 동적 계획법 (DP) 를 사용해야 이후의 부분문제들을 해결할 수 있다. 어떤 경로를 따라 차를 빼기 위해 버튼을 눌러야 하는 횟수를, 경로의 길이로 정의하자.

먼저, 지금은 차들의 번호의 대소관계만이 중요하므로, 좌표 압축 기법을 이용하여 번호를 1에서 N 사이에 있는 수들로 바꾸자. 그리고 번호가 1번인 차들의 목록, 번호가 2번인 차들의 목록, $\cdots$를 미리 저장해 두자.

$dp[X][E]$를, 번호가 1번에서 X번까지인 모든 차들을 빼고 나서, 맨 아래에 E번째 칸이 있도록 하기 위해 버튼을 눌러야 하는 최소 횟수로 정의하자. 그러면,

$$dp[X][E] = \min_{S=1}^{N}(dp[X-1][S] + \text{비용})$$

(비용: 현재 맨 아래에 S번째 칸이 있을 때, 번호가 X번인 모든 차를 빼고, 맨 아래에 E번째 칸이 놓이게 하는 경로의 길이)

로 쓸 수 있다. 번호가 X인 모든 차를 빼기 위해서는, 부분문제 1에서와 같이, 버튼 A를 계속 누르거나 버튼 B를 계속 눌러야 한다. 비용은 (1) S번째 칸에서 출발해서 번호가 X인 모든 차를 빼는 경로의 길이와, (2) 도착 지점에서 E번째 칸으로 이동하는 경로의 길이의 합과 같다.

비용을 $O(1)$에 계산할 수 있으므로, 총 $O(N^3)$의 시간 복잡도에 부분문제 4를 해결할 수 있다.

부분문제 5

부분문제 4의 풀이에서, dp 배열의 모든 칸을 채울 필요는 없다. $dp[X][E]$의 값은, E번째 칸에 있는 차의 번호가 X번일 때에만 필요하다. 전체 dp 배열에서 채워야 하는 칸의 개수는 $O(N)$개이다. 따라서, $O(N^2)$의 시간 복잡도에 부분문제 5를 해결할 수 있다.

부분문제 6

풀이 1

문제를 푸는 데 필요한 실행 시간을 줄이려면, $dp[X][E]$ 한 칸을 계산하는 데 드는 시간을 $O(N)$ 미만으로 줄여야 한다. 풀이 1에서는 슬라이딩 윈도우 기법을 이용한다.

번호가 X인 차가 총 K개 있고, 각각 $a[0], \cdots, a[K-1]$번째 칸에 있다고 하자. 출발 지점과 도착 지점에 상관 없이, K대의 차들을 모두 빼는 방법은 아래 네 가지가 있다.

1. $a[0] \to a[K-1]$.
2. $a[K-1] \to a[0]$.
3. 어떤 인덱스 $0 \le i < K-1$에 대해, $a[i] \to 1 \to N \to a[i+1]$.
4. 어떤 인덱스 $0 \le i < K-1$에 대해, $a[i+1] \to N \to 1 \to a[i]$.

각각의 방법은 서로 다른 출발 지점과 도착 지점을 가진다. 앞서 부분문제 4에서 비용을 (1), (2) 두 가지로 분류했다. 각 방법에 대해, 우리는 $S =$ 해당 방법에서의 출발 지점인 경우만 고려하면 됨을 알고 있으므로, (1)을 $dp[X-1][$출발 지점$] +$ 해당 방법으로 이동하는 경로의 길이로 계산할 수 있다.

$dp[X][E]$는, 모든 도착 지점에 대해,

(1) 출발 지점에서 위 4가지 중 하나의 방법으로 도착 지점으로 도달하는 경로의 길이

(2) 도착 지점에서 E로 이동하는 경로의 길이

를 합한 값의 최솟값으로 나타난다.

각 도착 지점에 대해, (1)은 dp 배열을 이용한 간단한 수식으로 계산할 수 있다. 이 값들을 모으고, 슬라이딩 윈도우 기법을 이용해, 각 E에 대해 (1) + (2)의 최솟값을 구할 수 있다.

각 번호에 대해 $O(K)$ (K : 해당 번호를 가진 차의 수)의 시간 복잡도에 문제를 풀 수 있다. 따라서, 총 시간 복잡도는 $O(N)$이다.

풀이 2

앞서 풀이 1에서 $dp[X][:]$들을 한꺼번에 계산한 것과 다르게, $dp[X][S]$들을 일일히 계산할 것이다. 풀이 2에서는 게으르게 갱신하는 세그먼트 트리를 이용한다.

풀이 1에서, 같은 번호를 가진 차들을 모두 빼는 방법을 네 가지로 분류했었다. 이 차들을 빼기 위해 S번째 칸에서 출발한다고 하자. 각각의 방법은, S에서 출발해서 처음 빼는 차로 향해 이동할 때, 시계 방향으로

이동하는가 또는 시계 반대방향으로 이동하는가에 따라 두 가지씩으로 나누어진다. 각 방법에 대해 버튼을 눌러야 하는 최소 횟수는 시작하는 칸의 번호 S에 대해 (1) $S +$ 상수 또는 (2) $-S +$ 상수 형태로 나타난다.

배열 arr[1..N]에 대해 다음 쿼리들을 지원하는 세그먼트 트리를 두 개 구성한다. 두 개의 세그먼트 트리는 각각 $S +$ 상수와 $-S +$ 상수 로 나타내어지는 방법의 최소 이동횟수를 기록하기 위해 사용할 것이다.

1. init() : 트리를 초기화한다. (모든 값을 ∞로 교체한다)
2. update(L, R, V) : 각각의 i = L, L+1, ..., R-1, R에 대해, arr[i] = min(arr[i], V) 연산을 시행한다.
3. query(X) : 배열의 X번째 값을 구한다.

이제, 번호 1번을 가진 차들, 번호 2번을 가진 차들, $\cdots$를 차례대로 순회하면서, 다음 번호에 해당하는 dp 배열의 값들을 채우기 위한 정보들을 두 개의 세그먼트 트리에 구성해 나간다.

게으르게 갱신하는 세그먼트 트리의 시간 복잡도가 $O(\log N)$이기 때문에, 총 시간 복잡도 $O(N \log N)$에 문제를 풀 수 있다.

첨부된 공식 풀이 소스코드는 풀이 2에 대한 소스코드이다.