‘‘빨강 파랑’’ 문제 풀이

작성자: 김동현

$[a, b] \times [c, d]$ 상자란, 왼쪽 아래 꼭짓점이 (a, b)이고 오른쪽 위 꼭짓점이 (c, d)인 각 변이 좌표축에 평행한 직사각형(내부 및 경계 모두 포함)을 의미한다.

부분문제 1

W, H 및 모든 좌표의 값이 1 이상 50 이하이므로, 직사각형의 왼쪽 꼭짓점이 $[-50, 50] \times [-50, 50]$ 상자 밖에 있으면 점을 하나도 포함하지 못하고, 따라서 고려할 필요가 없음을 알 수 있다.

직사각형의 왼쪽 꼭짓점 좌표를 고정하면 그 직사각형이 빨간 점과 파란 점을 각각 몇 개 포함하는지를 $O(N + M)$ 시간에 구할 수 있다.

이제 이 범위 내의 모든 가능한 정수 좌표에 직사각형의 왼쪽 꼭짓점을 놓는 경우를 다 시도해 보면 $O(N + M)$ 짜리 과정을 10000번 시도하므로 제한 시간 내에 문제를 해결할 수 있다.

부분문제 2

좌표의 값이 1 이상 1000 이하이므로, 직사각형의 왼쪽 꼭짓점이 $[-1000, 1000] \times [-1000, 1000]$ 상자 내에 있는 경우만 고려하면 된다. 대신, 이제는 각 좌표마다 해당하는 직사각형이 포함하는 빨간 점 개수와 파란 점 개수의 차를 빠르게 구할 수 있어야 한다.

(x, y) 좌표에 위치한 점을 포함하는 직사각형은 왼쪽 아래 꼭짓점 좌표를 (a, b)라고 할 때 $x - W \le a \le x$, $y - H \le b \le y$를 만족해야 한다. 즉, (a, b)가 $[x - W, x] \times [y - H, y]$ 상자에 속해야 한다.

편의상 모든 좌표에 1000을 우선 더했다고 하고, 2001×2001 크기의 2차원 배열이 있다고 하자. 입력의 각 점 (x, y)에 대해 배열의 $[x - W, x] \times [y - H, y]$ 상자 부분에 (그 점이 빨간색일 경우 1, 파란색일 경우 -1)을 더했다고 하자. 그렇다면 문제의 답은 배열에 들어있는 값들 중 절댓값의 최댓값, 즉 $\max($최댓값, $-$최솟값$)$이 된다. 정답에 해당하는 값이 적혀있는 위치의 인덱스에서 각각 1000을 다시 빼면 직사각형 왼쪽 아래 꼭짓점의 좌표 역시 얻을 수 있다.

특정 직사각형 위치의 Cell들에 일정한 값을 더하고 최댓값 및 최솟값을 구하는 것을 서브태스크 제약 조건 하에서 빠르게 처리하기 위해서는 크게 두 가지 방법을 사용할 수 있다.

첫 번째 방법은 일명 "2차원 누적합" 이라고 부르는, 아래와 같은 알고리즘을 사용하는 것이다. 배열이 $[0, R - 1] \times [0, C - 1]$ 상자라고 하자.

- $[a, b] \times [c, d]$ 상자에 값 v를 더할 때에는 일단 다음의 4가지 연산을 수행한다.

  ```
  arr[a][b] += v; arr[a][d + 1] -= v; arr[c + 1][b] -= v; arr[c + 1][d + 1] += v;
  ```

- 모든 더하기 연산이 끝난 뒤, 마지막으로 다음을 수행한다.

  ```
  for i in [0, R - 1]: for j in [1, C - 1]: arr[i][j] += arr[i][j - 1];
  ```

  ```
  for i in [1, R - 1]: for j in [0, C - 1]: arr[i][j] += arr[i - 1][j];
  ```

더하기 연산 하나당 $O(1)$, 마지막 처리는 $O(RC)$이다. 이후 배열의 값은 단순 참조로 $O(1)$에 알 수 있다. 모든 더하기 연산이 끝난 뒤에야 마지막 처리를 하고 각 배열 요소의 값을 알 수 있음에 유의하라.

더하기 연산은 $O(N+M)$번 수행하고 배열의 크기가 2001×2001이므로 제한시간 안에 문제를 풀 수 있다.

실제 구현 시에는 Out-of-Bound error를 예방하기 위해 배열 각 차원의 크기를 조금 더 여유있게 할당하는 편이 좋다.

두 번째 방법은 Plane Sweeping 기법을 사용하는 것이다. Plane Sweeping이란, 주로 2차원 공간에서 특정 축을 기준으로 잡고 직선을 그 축에 대해 한 방향으로 쭉 움직이면서 직선 위에서 나타나는 변화를 효율적으로 관리하는 방식의 기법을 총칭한다.

2차원 배열 위에서 이를 다른 말로 표현하면 `row[i] = {arr[i][0], arr[i][1], ..., arr[i][C-1]}`이라고 할 때 `k`를 0부터 `(R-1)`까지 하나씩 늘려가면서 row 배열에 나타나는 변화를 관리하는 것이다.

$[a,b] \times [c,d]$ 상자에 값 v를 더한다고 할 때, 이는 다음 2개의 변화로 설명이 가능하다.

- `k = a`가 되는 순간 `row[]` 배열의 `[c, d]` 구간에 `v`를 더한다.
- `k = b+1`이 되는 순간 `row[]` 배열의 `[c, d]` 구간에 `v`를 뺀다.

가장 단순하게 이를 구현하면, `row[]` 배열을 단순 1차원 배열로 관리하고, 변화가 일어날 때마다 해당하는 Cell에 일일이 값을 더하는 방식이 될 것이다. 최댓값 및 최솟값의 위치는 `k = 0, 1, ..., (R-1)`에 대해 `row[]` 배열의 값을 모두 확인하면 알 수 있다.

값을 더하는 상자가 $N+M$개 있으므로 변화는 총 $2(N+M)$개가 있고, 하나의 변화를 적용하는 데 $O(C)$ 만큼의 시간이 걸리고, 모든 Cell을 한 번씩 확인하는 데 $O(RC)$ 만큼 걸리므로 총 시간복잡도가 $O((R+N+M)C)$ 가 되어 이 방식으로도 제한시간 내에 문제를 해결할 수 있다.

부분문제 3

직사각형의 왼쪽 아래 꼭짓점의 후보를 줄일 수 있다. 만약 직사각형의 경계에 점이 놓이지 않았다면, 경계에 점이 하나 이상 놓일 때까지 직사각형을 적절히 "당겨도" 포함하는 점의 집합이 변하지 않기 때문이다.

구체적으로 따져 보면, 직사각형의 왼쪽 아래 꼭짓점 (a,b)가 다음 조건을 만족하는 경우만 보아도 충분하다.

- 입력에서 주어진 어떤 점 (x_i, y_i)에 대해 $a = x_i + 1$ 또는 $a = x_i - W$를 만족
- 입력에서 주어진 어떤 점 (x_j, y_j)에 대해 $b = y_j + 1$ 또는 $b = y_j - H$를 만족

x좌표와 y좌표의 후보가 각각 $O(N+M)$개 있으므로, 가능한 좌표는 총 $O((N+M)^2)$가지가 있다. 좌표를 하나 고정하면 $O(N+M)$ 시간에 빨간 점과 파란 점 개수의 차를 구할 수 있으므로 $O((N+M)^3)$ 시간복잡도로 문제가 해결된다.

부분문제 4

부분문제 2와 3을 푸는 데 쓰인 아이디어들을 결합하면 부분문제 4를 해결할 수 있다.

부분문제 3을 풀 때 x, y좌표의 후보가 각각 최대 $2(N+M)$개 뿐임을 관찰하였다. 이 후보 x, y 좌표들을 각각 $\{x_0, x_1, \cdots, x_{R-1}\}$, $\{y_0, y_1, \cdots, y_{C-1}\}$이라고 하자. 이제 다른 좌표값들은 중요하지 않으므로 (x_i, y_j)

좌표를 $R \times C$ 2차원 배열의 (R, C) Cell에 대응시키면 $R, C \leq 4000$이 성립하게 된다. 이제 부분문제 2를 풀 때 사용하였던 방법을 마찬가지로 적용할 수 있다.

좌표 압축 이후에는 직사각형의 높이 및 너비가 더 이상 일정하지 않음에 유의하라.

부분문제 5

부분문제 2, 4의 풀이에 쓰인 두 가지 방법 중 Plane Sweeping 기법을 자료구조를 통해 최적화할 수 있다.

구체적으로, Plane Sweeping 과정에서 필요한 다음의 2가지 연산을 빠르게 지원하는 자료구조가 필요하다.

- 특정 구간의 Cell에 일정한 값 더하기
- 최댓값(또는 최솟값)이 적힌 Cell의 위치 구하기

이는 Lazy Propagation을 사용한 Segment Tree로 구현할 수 있다. 이 경우 두 가지 연산을 모두 $O(\log(N+M))$에 처리할 수 있으므로 전체 문제를 $O((N+M)\log(N+M))$에 해결할 수 있다.