렉

오래된 컴퓨터에서 그림판을 사용하고 있다. 그림판의 화면은 픽셀이라 부르는 칸을 가진 격자 모양이다. 가장 왼쪽 아래 픽셀의 좌표를 $(1, 1)$로 하고, 오른쪽으로 a번째 위쪽으로 b번째 픽셀의 좌표를 (a, b)로 한다. 초기 화면에는 수직, 수평 변을 가진 N개의 직사각형들이 그려져 있다. 직사각형은 이 구역안에 포함된 픽셀들로 표현된다.

N개의 직사각형에 M개의 이동 명령이 수행될 것이다. 직사각형의 이동은 동, 서, 남, 북의 4방향과 북동, 북서, 남동, 남서(수평축과 45도 방향) 4방향으로 이루어진다. 또한 이동 거리 d가 주어진다. 다시 말해서, 이동 명령은 방향과 거리로 주어진다. 구체적으로, 직사각형의 가장 왼쪽, 아래 모서리 픽셀의 좌표가 (a, b)라 하면, 동, 북, 서, 남 방향으로 거리 d만큼의 이동은 모서리가 각각 $(a+d, b)$, $(a, b+d)$, $(a-d, b)$, $(a, b-d)$가 된다. 또한 북동, 북서, 남서, 남동 방향으로 거리 d만큼의 이동은 각각 $(a+d, b+d)$, $(a-d, b+d)$, $(a-d, b-d)$, $(a+d, b-d)$가 된다 (**그림 1**).

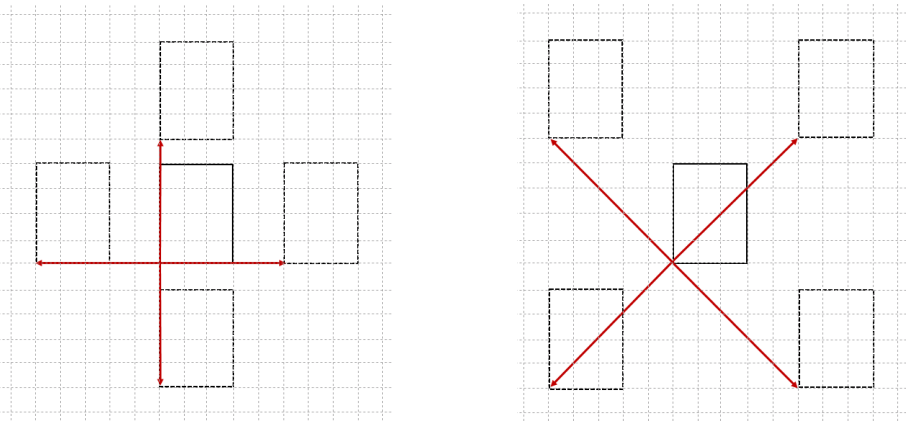


그림 1

화면에서 직사각형 R의 거리 d만큼 이동은 초기 위치를 포함해서 R이 거리 1 만큼 이동할 때마다 R의 모습을 순서대로 빠르게 나타냄으로서 구현된다. 하지만 우리의 컴퓨터는 아주 오래 되어서 R의 이동 시 렉이 심하게 걸린다. 결과적으로 R의 이동에서 그리게 되는 모든 R의 모습이 화면에 그대로 남아있게 된다. 따라서 R이 거리 d만큼 이동하면, d개의 직사각형들이 새롭게 화면에 만들어진다. 예를 들어, 아래 **그림 2**에서 직사각형이 북동방향으로 거리 3만큼 이동하면, 3개의 직사각형들이 만들어져서 총 4개의 직사각형이 화면 위에 남게 된다. 물론, 이동 후에는 북동 방향 끝에 있는 직사각형이 R 이 된다.

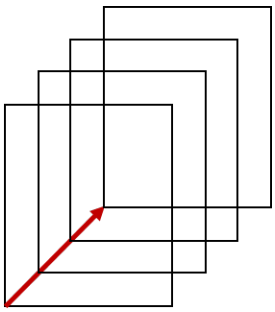


그림 2

M개의 이동 명령을 수행한 후 Q개의 질의가 주어질 것이다. 각 질의는 평면 상의 픽셀 p로 주어진다. 질의에 대한 대답으로 픽셀 p를 포함하는 직사각형들의 개수를 출력한다.

함수 목록 및 정의

여러분은 아래 함수를 구현해야 한다.

```
 vector<long long int> count_enclosing_rectangle(vector< pair<int, int> >
R1, vector< pair<int, int> > R2, vector<int> V, vector<int> I, vector<int>
D, vector< pair<int, int> > P )
```

- 이 함수는 단 한 번만 호출된다.
- 인자로 주어지는 배열 R1, R2의 크기는 N이다. 배열의 각 원소는 초기에 주어진 N개의 직사각형 중 하나를 나타내고, R1[i]와 R2[i]는 직사각형 $i+1$의 각각 가장 왼쪽 아래 픽셀과 가장 오른쪽 위 픽셀의 좌표를 나타내는 순서쌍이다. 순서쌍이 (a, b) 로 주어진다면, 좌표의 위치는 (a, b) 이다. 여기서, 직사각형은 1부터 N의 정수로 나타낸다.
- 인자로 주어지는 배열 V, I, D의 크기는 M이다. 배열의 각 원소는 M개의 직사각형 이동 중 하나를 나타내고, 직사각형 I[i]가 V[i] 방향으로 거리 D[i] 만큼 이동함을 나타낸다.
- 인자로 주어지는 배열 P의 크기는 Q이다. 배열 P의 각 원소는 질의에 해당하는 평면 상의 픽셀 p의 좌표를 나타내는 순서쌍이다. 순서쌍이 (a, b) 로 주어진다면, 좌표의 위치는 (a, b) 이다.
- 이 함수는 각 질의의 픽셀 p를 포함하는 직사각형들의 개수를 알아내어, 길이 Q인 배열에 담아 반환해야 한다. i 번째 값은 i 번 질의의 결과여야 한다. $(0 \le i \le Q-1)$

제출하는 소스 코드의 어느 부분에서도 입출력 함수를 실행해서는 안 된다.

제약 조건

- $1 \le N \le 250,000$
- $0 \le M \le 250,000$
- $1 \le Q \le 250,000$
- $1 \le$ R1[i].first $\le$ R2[i].first $\le 250,000$
- $1 \le$ R1[i].second $\le$ R2[i].second $\le 250,000$
- $0 \le$ V[i] ≤ 7
- $1 \le$ I[i] $\le N$
- $1 \le$ D[i] $\le 250,000$
- 화면의 좌표 값은 1이상 $250,000$이하이다. 임의의 직사각형에 포함되는 모든 픽셀들의 좌표값이 항상 이 범위안에 있다. 이는 이동이 일어난 이후에도 만족한다. 쿼리로 주어지는 픽셀 역시 이 조건을 만족한다.
- 직사각형의 이동 방향 V[i]의 값은 0(동쪽), 1(북동), 2(북쪽), 3(북서), 4(서쪽), 5(서남), 6(남쪽), 7(남동) 이다.

부분문제

1. (8점)

- $N \leq 100, M = 0$

2. (8점)
   - $M = 0$
3. (11점)
   - $M \leq 100$
4. (13점)
   - $V[i] \in \{0, 2, 4, 6\}$ $(0 \leq i \leq M - 1)$ 이다. 즉, 직사각형은 상하좌우로만 움직인다.
5. (12점)
   - $R1[i] = R2[i]$ $(0 \leq i \leq N - 1)$
6. (48점)
   - 추가적인 제약 조건이 없다.

채점 기준

`count_enclosing_rectangle` 함수가 반환한 수열의 길이가 Q와 같고, 정답 수열과 리턴한 수열의 원소들이 모두 같은 경우에만 해당 데이터를 맞춘 것으로 채점된다.

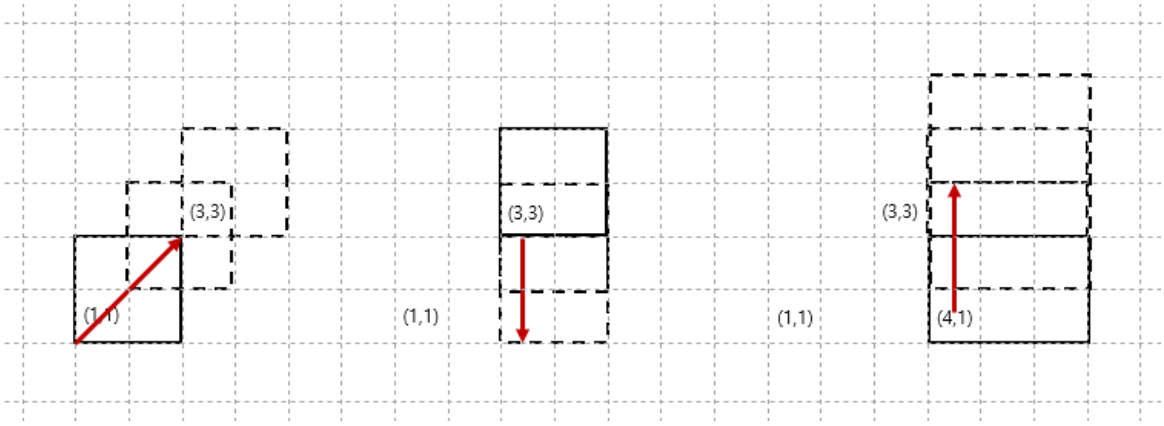
예제

- $N = 3$, $M = 3$, $Q = 4$, `R1` $= [(1, 1), (3, 3), (4, 1)]$, `R2` $= [(2, 2), (4, 4), (6, 2)]$, `V` $= [1, 6, 2]$, `I` $= [1, 2, 3]$, `D` $= [2, 2, 3]$, `P` $= [(3, 3), (4, 3), (3, 2), (5, 3)]$인 경우를 생각해 보자.

  그레이더는 다음 함수를 호출한다.

```
count_enclosing_rectangle(R1, R2, V, I, D, P)
```

  이 예제에서는 아래 그림과 같이 3개 직사각형의 3번의 이동에서 총 7개의 직사각형들이 만들어진다. 고로, 최종적으로 10개의 작사각형이 남게 된다. 픽셀 $(3, 3)$은 직사각형 1이 만드는 직사각형 2개와 직사각형 2가 만드는 직사각형 2개에 포함되서 총 4개의 직사각형들에 포함된다. 여기서, 직사각형 1의 이동의 세번째 직사각형과 직사각형 2는 동일한 영역을 포함하는 직사각형이지만 별개의 직사각형으로 간주함에 주의한다.



`count_enclosing_rectangle` 함수는 종료하면서 배열 `[4, 5, 3, 2]`를 반환해야 한다.

Sample grader

Sample grader는 아래와 같은 형식으로 입력을 받는다.

- Line 1: N M Q
- Line $1+i$ $(1 \le i \le N)$: `R1[`$i-1$`].first` `R1[`$i-1$`].second` `R2[`$i-1$`].first` `R2[`$i-1$`].second`
- Line $1+N+i$ $(1 \le i \le M)$: `V[`$i-1$`]` `I[`$i-1$`]` `D[`$i-1$`]`
- Line $1+N+M+i$ $(1 \le i \le Q)$: `P[`$i-1$`].first` `P[`$i-1$`].second`

Sample grader는 다음을 출력한다.

- Line i $(1 \le i \le Q)$: 함수 `count_enclosing_rectangle`가 반환한 배열의 i 번째 원소.

Sample grader는 실제 채점에서 사용하는 그레이더와 다를 수 있음에 유의하라.