**1. Matrix Addition**

**문제:** 임의의 값으로 초기화된 N ×N 행렬 A가 있다. 5개의 값 R1, R2, C1, C2 및 V가 주어지면

다음과 같은 프로그램이 있다. (1 ≤ R1, R2, C1, C2 ≤ N 정수, V : 실수 값)

for k := 1 to N do

read (R1,R2,C1,C2, V );

for i := R1 to R2 do

for j := C1 to C2 do

A[i][j] := A[i][j] + V ;

end

end

end

이 프로그램을 실행하면 O(N3) 시간이 걸린다. 더 효율적인 방법을 설계하라.

**설계:** 사각형의 네 모서리의 인덱스 값과 실수 값이 주어지면 사각형 내의 모든 위치에서 주어진

실수 값만큼 더해야 하며, 이 과정을 N번 반복한다. 본인은 입력 받는 R1, R2, C1, C2, V 값들을

이용해 행렬을 바로 바로 업데이트하는 것이 아니라 모든 요청을 누적한 후에 마지막에 한번에.

업데이트하는 방법을 적용할 것이다. 이를 위해 차분배열(2D Difference Array)을 이용하려고 한다.

먼저 모든 원소가 0으로 초기화되어 있는 2차원 배열을 준비한다. 그 다음 업데이트를 해야 하는

구간의 시작 부분 그리고 두 번 감소가 될 부분에 V만큼 더하고, 구간이 끝나는 부분+1에는 V만큼

빼 준다. 이 과정을 N번 입력 받으면 N번 실행해준 뒤 열 방향 누적합, 행 방향 누적합을 구해주면

원본 행렬에 업데이트해야 할 값들을 모두 얻게 된다. 따라서 마지막에 누적합을 구한 배열의 값들을

원본 행렬에 모두 더해주면 끝나게 된다. 아래의 그림을 보면 더 이해하기 쉬울 것이다.

스크린샷, 라인, 번호, 도표이(가) 표시된 사진

자동 생성된 설명 🡪 텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명 텍스트, 도표, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

입력에 대한

배열 업데이트

🡪 텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명 🡪 텍스트, 번호, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명 🡪 라인, 도표, 텍스트, 번호이(가) 표시된 사진

자동 생성된 설명

최종적인

누적합 배열

세로 방향에

대한

누적합 계산

가로 방향에

대한

누적합 계산

**시간 복잡도**: R1, R2, C1, C2, V에 대한 N번의 입력에 대해 배열에 표시를 하기 위해서는

4번의 연산이 N번 필요하므로 시간 복잡도가 4N이 된다.

그 다음 가로방향, 세로방향에 대한 누적합 계산은 각각 N2의 시간 복잡도가 걸린다.

마지막으로 누적합 배열을 원본 배열에 더하기 위해서는 N2의 시간 복잡도가 걸린다.

따라서 전체적인 시간복잡도는 T(n) = 4n + 3n2 🡪 O(n2) 이다.

**공간 복잡도**: 5개의 입력에 대한 변수 선언과 누적합을 저장할 N x N 배열에 대한 공간이 필요하기 때문에

이 알고리즘의 공간 복잡도는 T(n) = 5 + n2 🡪 O(n2)이다.

**2. Half-Circle Property**

**문제:** (0,0)을 중심으로 한 원 C가 있고 원의 둘레(원주)에 많은 점이 있다.

점들의 집합을 S로 표시할 때, 점 (0,0)을 통과하는 선 L이 존재하여 S의 모든 점이

L의 한 면에 놓여 있다면, S는 '반원 성질'을 만족한다고 말한다.

주어진 점 집합에 대해 집합이 반원 성질을 만족하는지 여부를 찾는 알고리즘을 설계하고,

그의 시간과 공간의 복잡성을 계산하라.

**설계:** 원의 중심좌표와 원의 둘레에 있는 점의 좌표를 가지고 아크 탄젠트(arctangent)를 활용하여

-π ~ π의 라디안 값으로 변환한다. 그 다음 이를 다시 0 ~ 360의 도(θ)로 변환한다.

이 값들을 정렬시킨 뒤 가장 작은 값과 가장 큰 값의 차이를 비교하여 180도가 넘으면 반원 성질을

만족하지 못하고, 안 넘으면 반원 성질을 만족한다고 할 수 있다.

예를 들어 원의 중심좌표(x1, y1)와 원의 둘레에 있는 점의 좌표(x2, y2)가 주어졌다고 해보자.

이를 이용해서 두 점 사이의 상대 좌표를 얻는다.(x2 - x1, y2 - y1)

이때 단순히 tan-1( (y2 - y1) / (x2 - x1) )를 사용하게 되면 -π/2 ~ π/2의 범위밖에 반환이 되지 않으며,

좌표가 어느 위치에 있는지 알 수 없다. 또한 x값이 0인 경우에는 계산이 불가능하다.

따라서 x값과 y값에 따라서 몇가지 조건을 걸려고 한다.

1. x > 0 : tan-1( y / x ) 🡨 기본적인 변환
2. x < 0 && y ≥ 0 : tan-1( y / x ) + π 🡨 90 ~ 180도 사이에 위치하므로 조정
3. x < 0 && y < 0 : tan-1( y / x ) – π 🡨 -180 ~ -90도 사이에 위치하므로 조정
4. x = 0 && y > 0 : π/2 🡨 90도에 있으므로 조정
5. x = 0 && y < 0 : - π/2 🡨 -90도에 있으므로 조정
6. x = 0 && y = 0 : 0 ( undefined ). 🡨 원점에 있기 때문에 각도가 존재하지 않음

위 조건들을 이용해서 라디안 값으로 반환해준다.

다시 이 값에 180/π를 곱하여 도(θ)로 변환해 준 뒤 배열에 저장해준다.

이때, 값이 음수인 경우에는 0도를 기준으로 반시계방향으로 회전한 각도를 의미하므로

360을 더해주어 0~360도의 범위내에 있도록 조정해준다.

이제 배열에 있는 값들을 오름차순으로 정렬해준다.

정렬해 주기 위한 알고리즘으로는 분할정복 기법을 활용한 병합 정렬 알고리즘을 사용하려고 한다.

1. **분할 (Divide)**:

배열을 절반으로 나누어 두 개의 하위 배열을 만든다. 이 과정을 하위 배열이 한 요소만 남을 때까지

반복한다. 즉, 크기가 1인 배열은 이미 정렬된 상태로 간주된다.

1. **정복 (Conquer)**:

하위 배열을 정렬하는 과정을 수행한다. 각 하위 배열이 정렬되면, 그 하위 배열들을 병합한다.

1. **병합 (Merge)**:

정렬된 두 하위 배열을 하나의 정렬된 배열로 병합한다. 이 과정에서 두 하위 배열의 첫 번째 요소를

비교하여 작은 요소를 결과 배열에 추가하고, 해당 요소를 제거한 후 이 과정을 반복한다.

한쪽 하위 배열의 모든 요소가 결과 배열에 추가된 후에는

나머지 하위 배열의 요소들을 모두 추가한다.

이렇게 정렬된 배열을 모두 순회하면서 현재 위치의 점과 다음 위치의 점사이의 각도 차이가

180도를 넘는지 확인한다. 만약 180도가 넘는다면 그 두 점 사이에는 점들이 존재하지 않기 때문에

중심을 가로지르는 직선의 한 면에만 점들이 존재한다고 말할 수 있다.

다시 말해 ‘반원 성질'을 만족한다고 말할 수 있다.

**시간 복잡도**: 중심좌표와 각 점들의 좌표를 이용해 라디안 값으로 변환하기 위해 n의 연산이 필요하고,

이를 다시 도(θ)로 변환하여 배열에 저장하기 위해 n의 연산이 필요하다.

그 다음 배열을 정렬하기 위한 병합 정렬 알고리즘의 연산은 nlogn만큼 필요하다.

마지막으로 두 배열 값들을 비교하기 위해서는 n-1의 연산이 필요하므로

최종적인 시간 복잡도는 T(n) = 2n + nlogn + n-1 = nlogn + 3n - 1 🡪 O(nlogn)

**공간 복잡도**: 라디안 값을 저장할 변수 하나와 도(θ)로 변환하여 저장하기 위한 n 크기의 배열이 필요하고,

병합 정렬 알고리즘에서 사용되는 n 크기의 새로운 배열이 필요하다.

따라서 공간 복잡도는 T(n) = 2n + 1 🡪 O(n) 이다.