알고리즘 과제 보고서

* HW2

2013147550 김지호

2013147544 조인

1. 선분의 교점 탐색 과정

두 선분의 교점을 찾기 위해 우선 각 선분을 연장한 직선들의 교점을 찾는다. 두 직선의 기울기가 같은 경우 실제로는 평행하거나 일치하는 경우로 하나의 교점을 가지지 않는다. 이외의 경우 다음과 같은 식으로 교점의 좌표를 구할 수 있다.

직선 A는 서로 다른 두 점 (x1,y1), (x2,y2) 을 지나고, 직선 B는 서로 다른 두 점 (x3,y3), (x4,y4) 을 지날 때, 두 직선의 교점 (Px,Py) = (  
(x1y2−y1x2)(x3−x4)−(x1−x2)(x3y4−y3x4)(x1−x2)(y3−y4)−(y1−y2)(x3−x4),  
(x1y2−y1x2)(y3−y4)−(y1−y2)(x3y4−y3x4)(x1−x2)(y3−y4)−(y1−y2)(x3−x4))

직선의 교점을 찾은 후 이 점이 두 선분 위에 모두 존재하는지를 알아본다. 교점에서 선분의 양 끝점까지 거리는 선분의 길이보다 작아야 한다. 그렇지 않을 경우 선분 밖에 있는 점이다.

1. Skyline의 merge 과정

[1] 개괄적인 알고리즘

사이즈가 n인 (skyline을 구성하고 있는 line의 갯수) 두 skyline을 merge할 때에 시간복잡도가 O(n) 이 나와야 divide and conquer로 구현하였을 때 O(nlogn)이 나온다.

시간복잡도 O(n)의 merge 알고리즘 구현을 위해 우선적으로 아래와 같은 사실들을 확인하고 넘어간다.

skyline은 일대일함수이다.

그렇기 때문에, 하나의 선분과 skyline을 비교할 때 선분은 자신의 끝점 x좌표보다

시작점의 x좌표가 뒤에있는 skyline의 선분들과 절대로 만날 수 없다.

위와 같은 사실들을 바탕으로, 다음과 같은 결과를 도출할 수 있다.

사이즈가 n인 두 skyline A, B를 merge하는 경우, A를 구성하는 line 중 하나 a와 B를 구성하는 line중 하나 b를 비교한다. 이 때, 적어도 둘 중 하나는 다른 skyline의 나머지 line들과 절대 만나지 않을 것임을 알 수 있다. 따라서, 한번의 비교로 두 skyline을 구성하는 2n개의 line들 중 적어도 하나를 비교선상에서 지워버릴 수 있다.

한번의 비교로 2n개의 line들 중 하나를 지울 수 있으므로, 최대 2n번 안에 두 skyline의 merge가 끝나게 된다.

그렇기에 사이즈 n의 두 skyline의 merge 알고리즘은 O(n) 시간복잡도를 가지게 된다.

[2] 알고리즘 상세 내용

우선적으로, merge 과정 중에는 현재 upper skyline, lower skyline이 무엇인지를 항상 저장하고 있는다. 이를 통해서 upper skyline만 merge된 skyline에 들어가도록 한다.

본격적인 merge 과정에서는, 루프문 안에서 두 skyline에서 각각 선분을 하나씩 (순서대로) 가져와서 비교한다.

두 선분을 비교하는 과정은 크게 두 단계로 나뉜다. 첫번째는 교점의 존재 여부 및 이에 따른 처리이고, 두번째는 둘 중 비교선상에서 지워버릴 line을 고르는 과정이다.

[2-1] 교점의 존재 여부 및 이에 따른 처리

두 직선의 교점이 존재하는 지와 그 교점이 어떠한 모양으로 생기는 지에 따라 처리해줘야 할 사항들이 달라진다.

교점이 어떻게 생기는 지에 따라 많은 case들이 나오는데, 이는 아래와 같다.

교점이 발생하지 않는 경우: 별도의 작업 없이 바로 두번째 단계로 넘어간다.

교점이 양끝점이 아닌 지점에서 발생하는 경우: 현재 upper skyline을 결과 skyline에 추가한 후 upper, lower skyline을 변경한다.

교점이 끝점에서 발생하는 경우: 다음 iteration에서 upper, lower가 바뀌는 지 확인할 수 있다. 따라서 현재 upper line을 delayed segment로 저장하고, 다음 iteration에 바뀌면 해당 delayed segment를 결과 lines에 추가한다.

교점이 시작점에서 발생하는 경우: upper, lower를 두 line의 기울기에 따라 재설정한다.

일치하는 경우: 결과 lines에 추가하고 바로 다음 iteration으로 간다.

하나의 직선이 다른 직선을 포함하는 경우: 짧은 line을 delayed segment로 저장하고, 긴 line을 upper, 짧은 line을 lower skyline으로 지정한다. 다음 iteration에 upper, lower가 변경되면 delayed를 결과 lines에 추가한다.

[2-2] 둘 중 비교선상에서 지울 line 고르기

기본적으로 두 line중 끝점의 x좌표가 작은 것을 지운다. 다만, 두 직선이 수직으로 만나는 등 특정 예외적인 상황이 있으므로 예외상황에 대한 처리는 따로 해준다. 지울 line을 고른 후, 만약 지울 line이 현재 upper skyline에 속해 있으면 결과 lines에 추가한다. 그러나 이 역시 특정 예외적인 상황에서는 upper skyline에 속하더라도 결과 lines에 추가하면 안되므로, 이러한 특수 case는 따로 처리를 해준다.

위의 두 단계를 루프문 안에서 반복하여 두 skyline의 모든 line들을 처리하고 나면, merge된 skyline을 구할 수 있다.

1. Divide And Conquer

skyline의 list를 merge하는 재귀함수를 구현한다. list를 원소의 크기가 하나가 될 때까지 절반으로 나누어 재귀함수를 호출한다. 결과적으로 삼각형 하나를 하나의 skyline으로 하여 merge해 나간다. 총 logN 개의 레벨이 생긴다. 두 개의 skyline의 선분 개수가 m이라고 할 때 merge의 시간 복잡도는 O(m)이다. 전체 input 삼각형의 선분의 개수를 n이라고 하면 k 레벨에서 m은 n/2^(k-1) 이고 이 때 merge 함수가 호출되는 수는 2^(k-1) 번이므로 한 레벨에서의 시간복잡도는 항상 O(n)이다. 전체 log(n)의 레벨이 있으므로 전체 알고리즘의 시간복잡도는 O(nlog(n))이다.