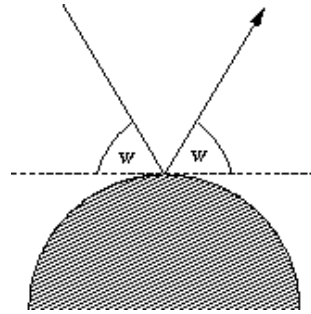
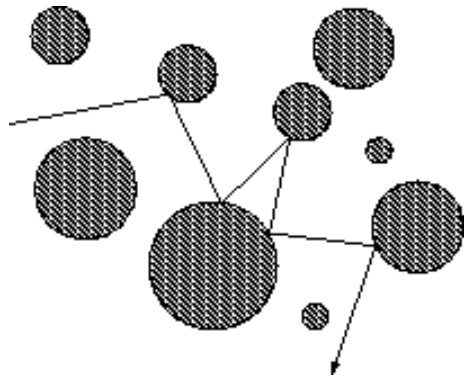


Rendering realistic images of imaginary environments or objects is an interesting topic in computer graphics. One of the most popular methods for this purpose is *ray-tracing*.

To render images using ray-tracing, one computes (traces) the path that rays of light entering a scene will take. We ask you to write a program that computes such paths in a restricted environment.

For simplicity, we will consider only two-dimensional scenes. All objects in the scene are totally reflective (mirror) spheres. When a ray of light hits such a sphere, it is reflected such that the angle of the incoming ray and the leaving ray against the tangent are the same:

The figure on the right shows a typical path that a ray of light may take in such a scene.



Your task is to write a program, that given a scene description and a ray entering the scene, determines which spheres are hit by the ray.

Input

The input consists of a series of scene descriptions. Each description starts with a line containing the number n ($n \leq 25$) of spheres in the scene. The following n lines contain three integers x_i, y_i, r_i each, where (x_i, y_i) is the center, and $r_i > 0$ is the radius of the i-th sphere. Following this is a line containing four integers x, y, d_x, d_y, which describe the ray. The ray originates from the point (x, y) and initially points in the direction (d_x, d_y). At least one of d_x and d_y will be non-zero.

The spheres will be disjoint and non-touching. The ray will not start within a sphere, and never touch a sphere tangentially.

A test case starting with $n = 0$ terminates the input. This case should not be processed.

Output

For each scene first output the number of the scene. Then print the numbers of the spheres that the ray hits in its first ten deflections (the numbering of spheres is according to their order in the input).

If the ray hits at most ten spheres (and then heads towards infinity), print `inf` after the last sphere it hits. If the ray hits more than 10 spheres, print three points (`...`) after the tenth sphere.

Output a blank line after each test case.

Sample Input

```
3
3 3 2
7 7 1
8 1 1
3 8 1 -4
2
0 0 1
5 0 2
2 0 1 0
0
```

Sample Output

```
Scene 1
1 2 1 3 inf

Scene 2
2 1 2 1 2 1 2 1 2 1 ...
```