**Code Jam 2010 - Round 2**

# Analysis: Grazing Google Goats

This problem is tough from both the implementation side and the algorithm side. In the analysis, we will begin by explaining some general concepts that show how the task is actually very similar to convex hull. We will then go into the particulars needed to build a solution and to prove that it is right. The resulting code isn't as scary as the length of the analysis makes you think.

Let us just focus on one bucket **Q** at a time. The problem asks us to choose rope lengths: i.e., for each pole position, we need to choose the radius of the circle centered at that pole. The common area will be the intersection of all these circles. It is clear that decreasing the radius of any circle will not increase the intersection. Therefore, we want the lengths to be as small as possible. Obviously, for each pole $\mathbf{P}_i$, $\mathbf{P}_i\mathbf{Q}$ is the smallest length we can pick so that the goat can still reach the bucket. Our main task is to efficiently compute the common intersection area of these circles.

From the limits of this problem, it is also clear that computing the intersection in $\Omega(\mathbf{N}^2)$ will be too slow. We are aiming for an algorithm that will, for each bucket, compute the intersection in time $O(\mathbf{N} \log \mathbf{N})$.

The intersection is a convex shape where the boundary consists of arcs of the circles. One can prove in various ways (some of the reason will be obvious below) that each circle will only contribute at most one such arc. We will mostly focus on how to compute these arcs -- to which circle does it belong, where does it stop and where does it start. Once all this is computed, it is relatively easy to get the area.

## Some theoretical background

While this section is not absolutely necessary for solving the problem, it will provide some nice insights. Once we have seen the problem from a few different angles, the solution will be conceptually clear and very natural.

So, first of all, note that all the circles we consider pass through a common point **Q**. If you have not seen it before, it is now our honor to introduce to you the beautiful geometric transformation called *inversion*. It takes **Q** as the center. And each point $\mathbf{X} \neq \mathbf{Q}$ is mapped to a point **X'** where **QX** and **QX'** are in the same direction, and the lengths satisfy $|\mathbf{QX}|\,|\mathbf{QX'}| = 1$.

Here is one very nice property of inversion: Every circle passing through the center point **Q** is mapped to a line that does not pass **Q**, and vice versa. And the interior of the circle is mapped to a half-plane that does not include **Q**. For our problem, the intersection of the **N** circles is then mapped to the intersection of **N** half-planes. For more details on inversion, we refer you to the Wikipedia page on inversive geometry.

We leave it to the reader to verify that, if the intersection is not empty, then we can rotate the plane such that **Q** is above all the half-planes. And the intersection of the half-planes is bounded by something called the *lower envelope of line arrangements*. The segments of the lower envelope are exactly the images of the arcs in the circle intersection from our original problem.

Then there is a concept of *duality* in computational geometry that maps each line `y = ax + b` to the point `(a, -b)`. The lower envelope is mapped by this transformation to the upper convex hull of the set of corresponding points.

So, our problem is indeed equivalent to the convex hull problem and the lower envelope of line arrangements problem. Both subjects are well studied and there are simple O(**N** log **N**) algorithms for both.

Out of these three settings, perhaps the algorithm for line arrangement is the simplest to visualize: Sort the lines by their slopes, and add them one by one. In each step, the existing lower envelope will be cut by the new line in two parts. The time for sorting is O(**N** log **N**), and the amortized complexity for the rest is O(**N**).

## Solution to our problem

The previous section suggests a couple approaches that begin with explicitly doing an inversion about **Q**. In this section, we will discuss a direct solution to the problem, where we do not need to view the problem using the transformations from the previous section. Note that it will be in some ways equivalent to the above algorithm we introduced for line arrangements.

Let's look from **Q** along a straight ray. The intersection of this ray with each circle will be either just **Q**, or a segment between **Q** and the second intersection point of this ray with the boundary of the circle. That means that in order to build the area required in the problem statement, we need to find the circle "closest" to **Q** in each direction from it. Let's denote each direction by its polar angle, which is defined up to a multiple of $2\pi$.

We start by considering just one circle. Suppose the polar angle of a ray that goes from **Q** towards the center of that circle is $\varphi$. When the polar angle of a ray goes from $\varphi-\pi/2$ towards $\varphi$, the distance from **Q** to the second intersection point with that ray increases from 0 to the diameter of the circle. When the polar angle continues from $\varphi$ towards $\varphi+\pi/2$, the distance decreases back to 0. When the polar angle changes from $\varphi+\pi/2$ to $\varphi+3\pi/2$ (the latter is the same as $\varphi-\pi/2$, where we started), the intersection of the ray with the circle is just **Q**.

Now consider two circles, one with center at polar angle $\varphi$ (remember, all polar angles discussed here are relative to **Q**) , another one with center at polar angle $\xi$, and suppose $0 < \varphi - \xi < \pi$. Then we'll see the following as we turn the ray originating from **Q**: starting from polar angle $\xi-\pi/2$ until $\varphi-\pi/2$ the ray will intersect only the second circle; from $\varphi-\pi/2$ the ray will intersect both circles, but the intersection point with the first circle will be closer to **Q** until the circles intersect; after the intersection and until $\xi+\pi/2$ the ray will still intersect both circles but the second is now closer; and from $\xi+\pi/2$ until $\varphi+\pi/2$ the ray will only intersect the first circle.

The important thing about the two circles case discussed above is that in the range of polar angles where the ray intersects both circles (and that's exactly the range we care about in finding the intersection area). The situation is very simple: before the intersection point one circle is closer to **Q**, after the intersection point another circle is closer to **Q**.

Now suppose we have many circles. First, we find the polar angle of the center of each circle, and thus also find the range of polar angles where a ray going from **Q** intersects each of the circles. We can then intersect all those ranges of polar angles to obtain a small range $[\alpha, \beta]$ of polar angles where the ray would intersect all circles. If this range is empty, then we already know there's no intersection.

Now let's start adding circles one by one, starting from the one with the smallest polar angle. You might ask, what does "smallest" mean when we're on a circle? Luckily, here we have less than full circle: since all circles intersect all rays in $[\alpha, \beta]$ range, the polar angles of all centers are between $\beta-\pi/2$ and $\alpha+\pi/2$. That leaves us a segment of length less than $\pi$ where we have a definite order.

As we're adding the circles, we'll maintain which circle is closest for each angle from $[\alpha, \beta]$. For example, after adding the first circle it will be the closest for the entire $[\alpha, \beta]$ range. Now we add

the second circle. Let's say the polar angle of the intersection point between two circles is γ. If you look carefully at the above analysis, you'll find that:

- When γ is less than α, the first circle is still the closest for the entire [α, β] .
- When γ is between α and β, the second circle is the closest for [α, γ], and the first circle is the closest for [γ, β].
- When γ is more than β, the second circle is now closest for the entire [α, β] .

Here we rely on the fact that we process circles in increasing order of the polar angle of their center.

Now look at the general case. Suppose before adding circle number $i$ we have the following picture: on [α, $\gamma_1$] circle $j_1$ is the closest; on [$\gamma_1$, $\gamma_2$] circle $j_2$ is the closest; ...; on [$\gamma_{k-1}$, β] circle $j_k$ is the closest.

Consider the polar angle δ of the intersection point between circle $i$ and circle $j_1$. There are 3 ways it could compare with the range where $j_1$ is the closest:

- When δ is less than α, it means that circle $j_1$ is closer than circle $i$ on the entire [α β] segment, and we can just stop processing circle $i$ since it doesn't affect the answer.
- When δ is between α and $\gamma_1$, we now have that circle $i$ is the closest on [α, δ], and circle $j_1$ is the closest on [δ, $\gamma_1$]. After doing this change, we can also stop processing circle $i$ since circle $j_1$ will be closer to the center all the remaining way.
- Finally, when δ is more than $\gamma_1$, we can forget about circle $j_1$ since circle $i$ is closer than it on the entire [α, $\gamma_1$] segment. In that case we continue processing circle $i$ by comparing it with circle $j_2$, and so on.

That's it! After we do this processing for all circles, we know the contour of the intersection figure, and we continue by computing its area.

The above algorithm requires a data structure that maintains a list of arcs with integer tags, allowing us to change the first arc, remove the first arc, and add a new first arc. The simplest way to get this data structure is to just use a stack where the first ac would be on the top, and the last one would be on the bottom.

The running time for the above algorithm can be estimated using amortized analysis as follows. When processing each circle, we do at most two "push to stack" operations, and one or more "pop from stack" operations. That means the total number of push operations is O(**N**), and since the number of pop operations can't be greater than the number of push operations (there'd be nothing to pop!), it's also O(**N**), giving us the total runtime of O(**N**). However, we must also remember the step where we sort all circles by the polar angle of their center, so overall runtime is O(**N** log **N**).

We've so far avoided discussing the low-level computational geometry needed for this problem. In the above solution we required two geometric routines: find the polar angle of the other intersection point of two circles which have the first intersection point at origin; and find the area of a figure that is bounded by arcs.

The first routine is straightforward if you can already intersect circles (a useful thing to know how to do!); alternatively, you could derive the formulas for the polar angle directly. The second one is slightly more tricky: first, we split the figure into "rounded triangles" with rays going from **Q** and having polar angles $\gamma_1$, $\gamma_2$, ..., $\gamma_{k-1}$. Each rounded triangle has two straight sides (one of those might have zero length) and one circular side. We can then split the rounded triangle into the corresponding triangle plus the round part (a sliced off part of a circle). You have probably seen how to compute the area of a triangle before - one good approach is the "shoelace formula". To

calculate the rounded area, it helps to start with a "pie slice" from the center, whose area is a simple fraction of the total circle area, and then add or subtract triangle areas.

Of course there is a lot to do here, but that's why we made it Problem #4!