Problem 154: The Last Place You Look

Difficulty: Hard

Author: Ben Fenton, Ampthill, Reddings Wood, United Kingdom

Originally Published: Code Quest 2021

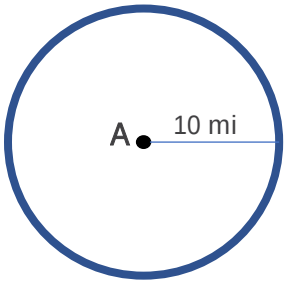
Problem Background

It's said that when you're looking for something, you always find it in the last place you look. This is invariably true, because once you find the item, you stop looking for it, but more often than not it seems that the item also happens to be in the last place you'd ever think of looking. Let's reduce the guesswork somewhat and create some devices that can guarantee that whenever our items walk off on their own, they're always in the first place we look!

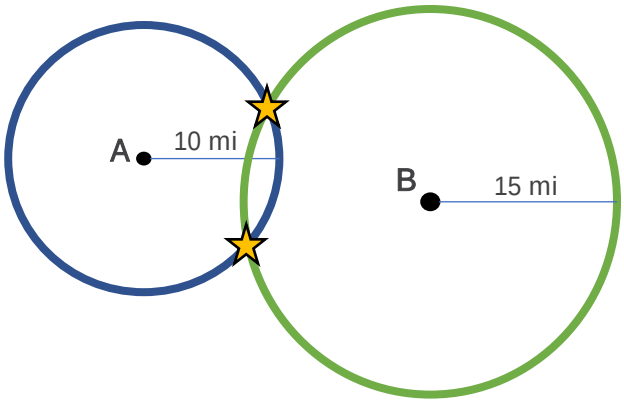
Geopositioning tags were among the first products to make up the rapidly-growing "Internet of Things;" an industry of normally common-place items that have a wide range of useful features added to them by virtue of a constant wireless internet connection. In this case, a keychain could have a small fob connected to it that contains a miniaturized GPS receiver. This fob constantly reports its location to an app on your smartphone, allowing you to easily find your keys if they should get lost. The concept may sound simple, but a lot of software design goes into something like this.

GPS, or Global Positioning Systems, rely on a network of satellites orbiting far above the earth. Each satellite is able to track its position over the Earth, and constantly broadcasts its location and the current time. A GPS receiver can use this information to determine how far away it is from the satellite's position. However, a signal from a single satellite is not enough to determine location; in actuality, a receiver needs signals from at least four different satellites to pinpoint its own location.

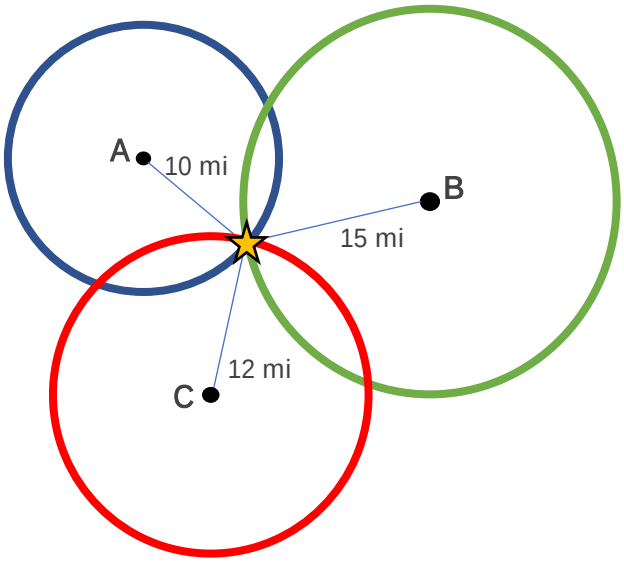
Let's look at a simplified example of how GPS receivers work. If a receiver only knows it is 10 miles away from point A, it could be located anywhere along the circumference of this circle:



Even a signal from a second satellite can't narrow down an exact position; the receiver could be in either of the two locations where these circles intersect:



Once we add a third signal, we can get a better idea of our location. The receiver must be located at the one point at which it can be the indicated distance from each of the three satellites:



Again, in practice, a receiver would actually need four satellite signals to confirm its location; since GPS satellites work in three dimensions (latitude, longitude, and altitude), you need four readings to get an exact location. For this problem, we'll ignore the altitude, allowing us to focus on the first two dimensions.

Problem Description

Lockheed Martin is working on a set of devices that can work as a sort of local GPS to help aircraft maintainers find tools they've misplaced; a set of three transmitters that act as "GPS satellites," and small receiver tags that attach to tools. Since using latitude and longitude over an area as small as an aircraft hangar isn't going to be terribly accurate, your team is developing an alternative coordinate system for use with these tags.

The hangar will be divided into a grid using an integer-based X,Y coordinate system. Receiver tags will be able to determine the "taxicab distance" between them and each of the three transmitters spaced around the hangar. "Taxicab distance" is how many spaces in the coordinate grid must be traversed to reach your destination while moving only vertically and horizontally, as though you were a taxi

navigating city streets. The grid below shows the "taxicab distance" between each cell and the target cell, marked with an X.

8	7	6	5	4	5	6	7	8
7	6	5	4	3	4	5	6	7
6	5	4	3	2	3	4	5	6
5	4	3	2	1	2	3	4	5
4	3	2	1	X	1	2	3	4
5	4	3	2	1	2	3	4	5
6	5	4	3	2	3	4	5	6
7	6	5	4	3	4	5	6	7
8	7	6	5	4	5	6	7	8

As you can see, each number creates a diamond pattern around the central X, approximating the circles shown in the GPS diagrams above.

Using the known locations of the three transmitters, and the "taxicab distance" to each of them, you must determine the location of the receiver tag (and the tool to which it is attached).

Sample Input

The first line of your program's input, received from the standard input channel, will contain a positive integer representing the number of test cases. Each test case will include nine integer values, separated by spaces:

- The X-coordinate of the first transmitter. This will be an integer value.
- The Y-coordinate of the first transmitter. This will be an integer value.
- The taxicab distance of the receiver from the first transmitter. This will be a positive integer value.
- The X-coordinate of the second transmitter. This will be an integer value.
- The Y-coordinate of the second transmitter. This will be an integer value.
- The taxicab distance of the receiver from the second transmitter. This will be a positive integer value.
- The X-coordinate of the third transmitter. This will be an integer value.
- The Y-coordinate of the third transmitter. This will be an integer value.
- The taxicab distance of the receiver from the third transmitter. This will be a positive integer value.

```
3
10 10 20 -5 -5 10 8 -8 16
8 8 15 -10 5 10 6 -2 13
-4 5 6 9 4 10 5 -4 14
```

Sample Output

For each test case, your program must print a single line containing the integer X- and Y-coordinates of the receiver tag, in (X,Y) format.

```
(0,0)
(-2,3)
(1,6)
```