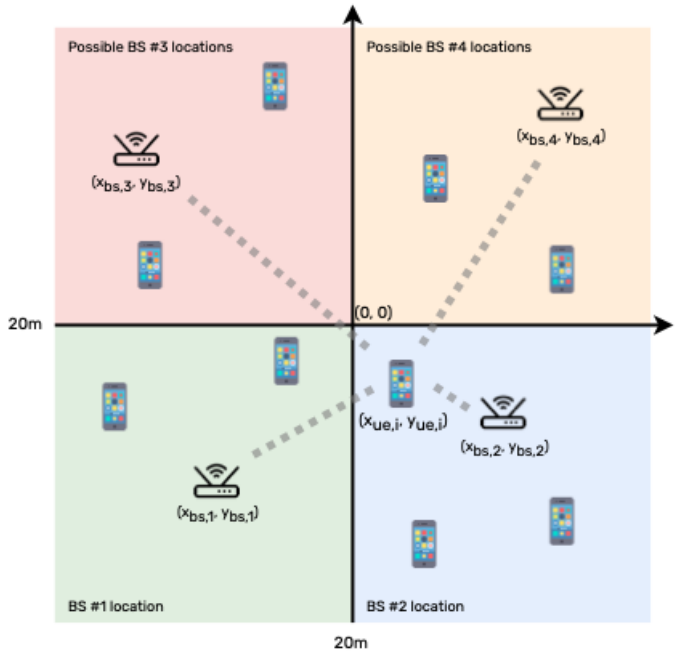
A. Indoor Positioning

Limits: 4 sec., 1024 MiB

Most indoor positioning algorithms depend on accurate base station locations. However, in actual scenarios, the cost of obtaining accurate base station locations is high. Many base stations are hidden in the ceiling and cannot be measured. It is a difficult problem in the industry to locate users and sites simultaneously by using only a few accurate sites and a large number of users' delay measurements.

Consider a room as 20×20 meters square in 2D. The center of the room has the coordinate (0, 0). There are four base stations (BS) and n user equipment (UE) devices located in the room. Both BSs and UEs can be considered points inside the room. Base stations are located in four different quadrants of the plane in the order shown in the image below.



You know the exact locations of the first two BSs, but you don't know the locations of the remaining two BSs as well as all UEs.

For each UE, you are given the time of arrival (ToA) in nanoseconds (ns) of signal from the UE to each of the four base stations. ToA is measured on the base station and includes transmission delay and noise. More information on how ToA is calculated can be found in the *Test generation* section below.

Based on the given information, your task is to estimate the coordinates of the remaining two BSs as well as all UEs as closely as possible to the actual positions.

Input

The first line contains a single integer n – the number of UEs.

The second line contains two space-separated floating-point numbers $x_{bs,1}$ and $y_{bs,1}$ – the coordinates of the first BS.

The third line contains two space-separated floating-point numbers $x_{bs,2}$ and $y_{bs,2}$ – the coordinates of the second BS.

The following n lines contain four space-separated floating-point numbers $t_{i,1}$ $t_{i,2}$ $t_{i,3}$ $t_{i,4}$ each, denoting the measured time of arrival from the i-th UE to the corresponding BS.

Output

In the first line print two space-separated floating-point numbers $x_{bs,3}$ and $y_{bs,3}$ – the estimated coordinates of the third BS.

In the second line print two space-separated floating-point numbers $x_{bs,4}$ and $y_{bs,4}$ – the estimated coordinates of the fourth BS.

In the next n lines print estimated coordinates $x_{ue,i}$ and $y_{ue,i}$ of the corresponding UEs.

Constraints

$20 \leq n \leq 200$,
$t_{i,j} > 0$,
$x_{bs,1} < 0$ and $y_{bs,1} < 0$,
$x_{bs,2} > 0$ and $y_{bs,2} < 0$,
$x_{bs,3} < 0$ and $y_{bs,3} > 0$,
$x_{bs,4} > 0$ and $y_{bs,4} > 0$,
$-10 \leq x_{bs,i}, y_{bs,i}, x_{ue,i}, y_{ue,i} \leq 10$.

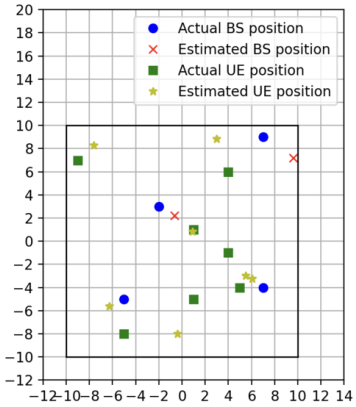
Notes

Sample test case

Check the following sample test case. Note that in this test case the number of UEs is less than the allowed lower limit in the constraints (20). This test case is for demonstration purposes only and is not included in the provisional or final test sets.

Input *(stdin)*	**Output** *(stdout)*
7 -5.000 -5.000 7.000 -4.000 126.299 95.354 120.525 119.727 135.098 180.551 172.184 194.325 234.749 259.073 214.143 242.686 229.045 226.359 212.143 234.152 88.203 56.870 66.165 95.202 244.898 246.406 219.213 211.507 165.713 182.486 188.144 209.985	-0.657574772 2.202897216 9.612727847 7.184456177 6.052894344 -3.230646577 -6.258069892 -5.628141926 -7.630356154 8.287634824 0.906085783 0.813700552 5.519233285 -2.995729624 3.011525986 8.822308221 -0.400688838 -7.997504188

The actual positions of the four BSs are (-5, -5), (7, -4), (-2, 3) and (7, 9). There are 7 UEs in positions (5, -4), (-5, -8), (-9, 7), (1, 1), (4, -1), (4, 6) and (1, -5). All actual and estimated positions for this sample test case are shown on the image below.



Scoring

If your output for the test is invalid, i.e. the coordinates are missing, out of bounds of the room or corresponding base station quadrant, your score for a test is 0. Otherwise, your score for the test is

$$\left\lfloor \left(\frac{1}{1+D_{bs}} + \frac{1}{1+D_{ue}}\right) \cdot 5 \cdot 10^6 \right\rfloor$$

where:

$$D_{bs} = \sqrt{\frac{1}{2}\sum_{i=3}^{4}(x_{bs,i} - \hat{x}_{bs,i})^2 + (y_{bs,i} - \hat{y}_{bs,i})^2}$$

$$D_{ue} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ue,i} - \hat{x}_{ue,i})^2 + (y_{ue,i} - \hat{y}_{ue,i})^2}$$

$(\hat{x}_{bs,i}, \hat{y}_{bs,i})$ – the actual coordinates of the BSs,
$(\hat{x}_{ue,i}, \hat{y}_{ue,i})$ – the actual coordinates of the UEs.

The score for the sample test case above is 2918539.

Submissions

- The execution time limit is 4 seconds per test case and the memory limit is 1024 mebibytes.
- The code size limit is 64 kibibytes.
- The compilation time limit is 1 minute.
- There are 50 provisional test cases. Your submissions will be evaluated on the provisional set during the submission phase.
- You can submit your code once per 10 minutes and you will get feedback with your score for each of the provisional tests.
- There will be 500 test cases in the final testing after the submission phase is over. The final results will be announced in one week.

Test generation

All test cases including provisional and final sets are generated by the generator. Check the generator source code.

The generator randomly positions BSs and UEs and calculates the measured time of arrival based on the formula below.

$$T = T_{ue} + T_{bs,ue} + N_0 + N_{nLoS}$$

where:

- T is the measured time of arrival given to you as input.
- T_{ue} is the UE transmission time (between 0 and 200).
- $T_{bs,ue}$ is the signal traveling time. The speed of the signal is considered to be the speed of light.
- N_0 is the additive Gaussian noise (between 0 and 5).
- Non-line-of-sight noise $N_{nLoS} = \begin{cases} 0 & \text{if LoS} \\ 5 \leq N_{nLoS} \leq 50 & \text{if nLoS} \end{cases}$
- $nLoS$ probability is between 0 and 0.5 from any UE to any BS and is fixed for a single test case.
- All values are in nanoseconds.

A test case is produced by providing the generator with a seed number.

1. Download the generator source code
2. Save it to a file named `generator.java`
3. Compile the source code with Java

   ```
   javac generator.java
   ```

4. Generate a test case for some seed number

   ```
   java generator <seed>
   ```

   This will print the test case for `<seed>` to *standard output*.

Quickstart

Check the sample solution, which iteratively assigns random coordinates for the two unknown BSs, then iteratively assigns random coordinates for UEs, and minimizes the total time of arrival difference.

The source code is available for some of the contest programming languages:

- C++
- Python
- Java
- C#

References

Four of the most popular methods: Chan's [1], Foy's [2], Fang's [3] and Friedlander's [4] can be considered. These algorithms enable the calculation of the geographical position of a user equipment using the time differences of arrival (TDoA) between several base stations and UE.

[1] Chan, Y.T., Ho, K.C.: A Simple and Efficient Estimator for Hyperbolic Location. IEEE Trans. on Signal Proc. 42(8), 1905–1915 (1994)

[2] Foy, W.H.: Position-Location Solutions by Taylor-Series Estimation. IEEE Trans. on Aero. and Elec. Systems AES-12(2), 187–194 (1976)

[3] Fang, B.T.: Simple Solution for Hyperbolic and Related Position Fixes. IEEE Trans. on Aero. and Elec. Systems 26(5), 748–753 (1990)

[4] Friedlander, B.: A Passive Localization Algorithm and Its Accuracy Analysis. IEEE Jour. of Oceanic Engineer. OE-12(1), 234–245 (1987)