Robotique
CRC
Robotics

CRC

PROGRAMMING COMPETITION

PRELIMINARY PROBLEM 2

A FEW NOTES

- The complete rules are in section 5 of the rulebook.
- You have until **Friday, December 9, 11:59 pm** to submit your code
- Feel free to use the programming forum on the CRC discord to ask questions and discuss the problem with other teams. That's what it's there for!
- **We are giving you quick and easy-to-use template files for all languages allowed. Please use them.**

USING THE TEMPLATE FILE

- Basically, the template file uses your solve() code to transform a test case into its output and allows you to quickly check if it did what was expected or not. **All your code (except additional functions and classes, which should go right above it in the file) should be written in the solve() function.**
- **Do not touch anything lower than the solve().** There will be a list of input and a list of expected outputs, and a showing of how they would look in the problem statement for your convenience. There will also be a function that checks your answers by looping over all test cases of that input list. **Your solve() takes only one test case (arrays/lists in this case) at a time.**
- You can swap the **console's language to French** at the top of the file.

STRUCTURE

Every problem contains a small introduction like this about the basics of the problem and what is required to solve it. Points distribution is also given here for the preliminary problems.

Input and output specification:

In these two sections, we specify what the inputs will be and what form they will take, and we also say what outputs are required for the code to produce and in what format they shall be.

Sample input and output:

In these two sections, you will find a sample input (that often has multiple entries itself) in the sample input and what your program should produce for such an input in the sample output.

Explanation of the first output:

Sometimes, the problem might still be hard to understand after those sections, which is why there will also be a usually brief explanation of the logic that was used to reach the first output from the first input.

MERCH DROP

A merchandise deliverer's shipping list has unfortunately been shredded. It is of utmost importance to remake this list as fast as possible, as it's actually the most precious merchandise in the world: the Avia 2023 game pieces! However, when rival companies learned of the loss of this list, they decided to join the fray. The CRC has thus organized a competition for the distribution and will award the delivery contract to the transporter that can generate the most profit. Your job will be to create the most profitable list of orders so you can win the contract for your company. Your orders have for follow some rules:

- Each city possesses a sale value and a buying value per unit of merchandise. These values are constant and independent.
- Each city also has a maximum amount of merchandise it can buy and sell. These values change, but are still independent. For example: if city "A" sends 30 units to city "B", the number of units city A can sell goes down by 30, and the number of units city B can buy also goes down by 30. **The order is not permitted if either of those numbers was already below 30. This also implies that units that have already been delivered cannot be resold.**
- Every possible order also has an associate transport cost per unit of merchandise, which you can find in the table T. The rows of the table correspond to the sender while the columns correspond to the receiver. Therefore, the cost of transportation from city A to city B would be in the first row, second column, while the cost of B to A would be in the second row, first column. **It is important to note that these two values may not necessarily be equal.** It is also important to highlight that the table's diagonal is null, which evidently means that the merchandise hasn't moved, but also means that the order isn't allowed. **A city cannot sell to itself.**
- Null trades and negative trades are not allowed.
- The profit is calculated by a 3rd part based on your list of orders (in the template file) with the following formula:

$$Profit = (Buying\ value\ -\ Selling\ price\ -\ Transport\ cost)\ *\ Quantity$$

- **It is forbidden to directly modify the list of tests in the template file. You should only have to use what is happening in the solve() function. It is also forbidden to use symbolic calculation libraries/modules.**

Scoring:

A series of 5 tests will be done on your code, similar to what you'll find in the template file. The 5 tests will however be unknown to you as to best test the power of your code. Each "company" (team) will get a result, and the best of these results for a same test will be established as the maximal profit for the test. Each team will receive points based on the following calculation if the list of orders generates a not null profit:

$$Score = 5 + (total\ profit/maximal\ profit) * 10$$

The team(s) that obtain the maximal profit would therefore get 15 points for the test. The total score for a team for this problem will be the average of the 5 tests.

Input specification:

In python, your code will receive the table C with dimensions N x 5, containing the cities' information. The information, in order, is: the name of the city, its buying quantity, its buying price, its sales quantity, its sales price. Your code will also receive table T with N x N dimensions containing all the costs of transportation between the cities. N represents the number of cities and will be a whole number between 4 and 20. For Java and C++ files, as tables don't allow us to mix the types inside them, table C will be separated into a table V with dimensions N containing the name of the cities in order and a "real" table C with dimensions N x 4 containing all the whole numbers representing the cities' information.

Output specification:

Your solve() method must return a list of orders in the format [order1, order2, …] so that we can test the validity of these orders. Even if you return only one order, return under the form [order1], as this will avoid bugs. Each order must itself be a table of these 3 values, in order: buying city name, selling city name, units of merchandise traded. For python, the units of merchandise traded can be a whole number, but for Java and C++, the number will have to be of type String so we can have an array of type String.

First input in template file (python example):

```
[["Pékin",150,103,160,98],
 ["Séoul",90,101,250,97],
 ["Pyongyang",200,104,120,100],
 ["Kyoto",70,104,140,97]]
[[0,2,2,3],
 [2,0,1,2],
 [3,3,0,3],
 [3,2,2,0]]
```

Sample output for first input (python example again):

```
[["Pyongyang","Séoul",200],
 ["Pékin","Séoul",50]]
```

Explanation of the first output:

As Seoul can sell a lot of units, we decide to sell some of them to Pyongyang until they can't take anymore, then sell the rest to Beijing. The first sale will net 1200 profit, and the second one will net 200 profit. The total profit for this list of orders is 1400.