Robotique
CRC
Robotics

CRC

PROGRAMMING COMPETITION

PRELIMINARY PROBLEM 4

A FEW NOTES

- The complete rules are in section 5 of the rulebook.
- You have until **Friday, February 10, 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:

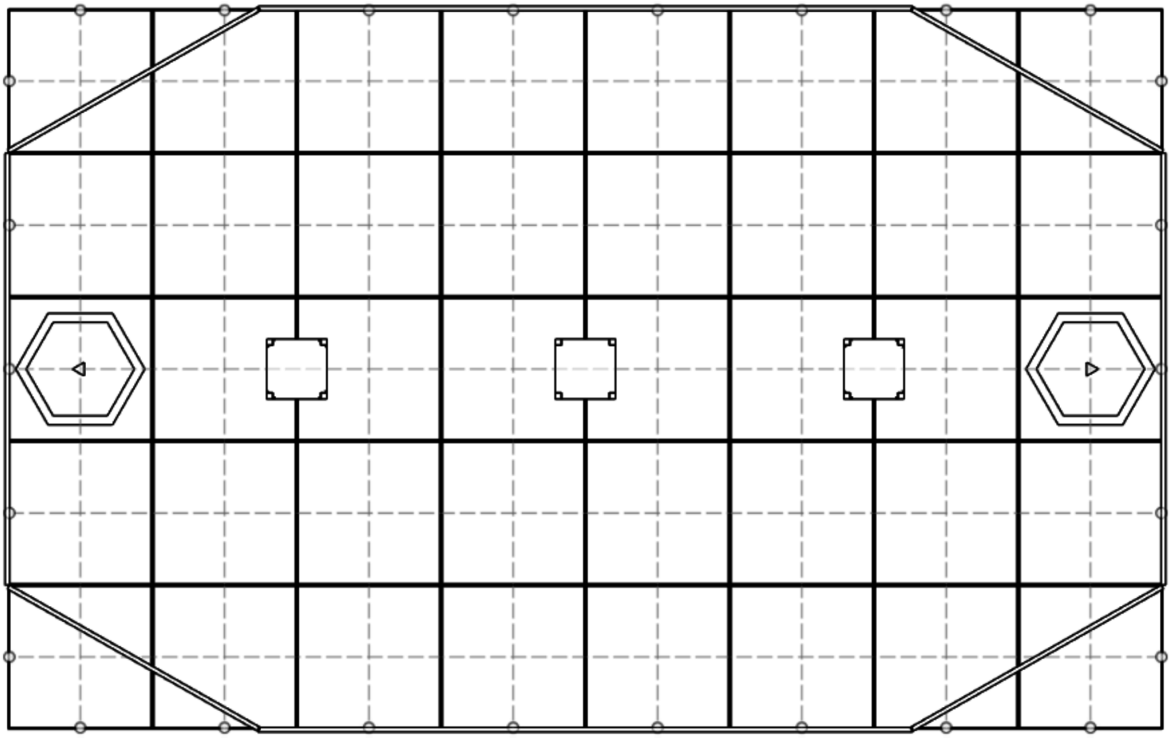
In these two sections, you will find a sample input (that often has multiple entries itself) in the sample input and what your program should produce for such an input in the sample output.

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.

AVIA THE GAME™

With the competition rapidly approaching, you are looking to aid your robot team by finding the best strategy to score the most points. Instead of doing all the math with a pen and paper, you decided that math that solves itself without a pen and paper is far more interesting. Thus, you simulate the Avia game with a few modifications:



- You will be placed in a position where your team is in the middle of a game and you will need to score as many points as possible with the remaining time to gain a comfortable lead and win the match. **You will be on the blue team in this situation.**
- You can only move vertically and horizontally on the grid above (composed of 2” x 2” squares). The grid contains 11 rows and 17 columns, and will be represented in your code by an 11x17 table containing information. Each cell symbolizes the crossing of two lines in the grid above.
- **The multiplication towers and the red piece are not activated in this simulator.**
- You can thus make points by flipping pieces to your color or placing them in the tower (see section 2 of the rulebook). The height of each level of the towers is found in the dimensions booklet given at the start of the season. Please note that you can go against your objective by flipping pieces yellow. A flip from blue to yellow will deduce 35 points from your score as the piece will not be worth 35 points for your team anymore. Placing a yellow piece in one of the towers just prevents making more points on the tower by blocking that level. You can still continue placing game pieces on that tower of course.
- You will need to avoid both yellow robots and your allied blue robot, which will remain stationary at certain points of the game field.

- Each square will be either empty or will contain information. An empty square is only symbolized by an empty space " ". A piece with the yellow side facing upwards is symbolized by a "J" while a piece with the blue side up is symbolized by a "B". The three other robots are symbolized by a "#" while your starting position is symbolized by a "&". The number of pieces contained in each tower will be shown in the squares the towers occupy.
- As an input, you will receive the game field grid at the beginning of the situation, the time remaining in the match in the format "M:SS" and your robot statistics. Your robot statistics are:
    - horizontal (ground) speed (inches/second)
    - the lift speed (inches/second); **The time to lift a piece to a certain height should always be rounded up to the nearest integer, as your simulator will treat seconds as integers**; if the vertical speed is 2.7 inches/second and the height is 8.2 inches, the time taken to put the piece in the tower is 4 seconds and not 3.04 seconds. **The time taken to lift the piece to the right height is the total time necessary to complete the action.**
    - time to flip a piece (seconds)
    - time to pick up a piece (seconds)
- Your options for instructions are the following:
    - Robot movement **(you must never go on a square occupied by another robot)**:
        - "U" to go up.
        - "D" to go down.
        - "L" to go left.
        - "R" to go right.
    - Interact with a game piece on the square occupied by your robot:
        - "F" to flip a piece. **You can only flip a piece if your robot contains fewer than 2 game pieces.**
        - "G" to pick up a piece. As explained in the rules, your robot may hold up to two pieces at a time. **A piece keeps its face up color when picked up and will be put in the tower with that color facing upwards. It will need to be flipped before being picked up if it is the wrong color.**
    - Interact with a tower on the same square as your robot:
        - "T" places the piece in the tower. It is the only instruction that allows you to remove a piece from your robot. The resulting number of points depends strictly on the number of pieces already in the tower. **The piece that will be placed is the last one to have been picked up by the robot.** If the tower is full, you cannot add pieces.
- The time required to complete each instruction depends entirely on the robot performance statistics given in the input.

Scoring:

Your code will be tested with 5 unknown test cases. Each case will be worth 6 points, for a total of 30 points for the problem. For every case, 2 points will be attributed all non-zero points, and the 4 other points will be attributed following a normalization according to the best team, like in preliminary problem #2. For example, if a team leaves a scenario with 750 points, those points will be inserted in the following formula:

$$Performance\ Points\ =\ (scored\ points/max\ scored\ points)\ *\ 4$$

Added up, that means 20 points will be attributed to your performance and 10 points for the success of each scenario.

Input specification:

The input will contain all parameters needed to simulate the situation. First, you will be given the grid of the playing field, an array made of 11 strings of 17 chars each containing information on every square. Next will be the game's remaining time given as a string in the format "M:SS", the robot's moving speed in inches/second given as an integer, the lifting speed of the robot given as a real number, the time required to flip a game piece and, finally, the time required to pick up a game piece, both given as integers in seconds.

Output specification:

Your solve() function will return a string with the sequence of actions performed by your robot to obtain the best score. The possible actions are "U", "D", "R", "L", "F", "G", and "T", as explained above.

First input of template file (Python example) :

```
[["          BJ      "
  "       BJ  JJ J   "
  "    J   BBB    J  "
  "    &#JJ   B  JJ  "
  "    BB    BJ   B  "
  " B  2 J 7   5    "
  "    J J   # JB J  "
  "    #  BJ BJ  J   "
  " J   BJ    JBB   "
  "      J  J   JJJ  "
  "             J    "],
  "1:31", 24, 20, 2, 1]
```

Output example for the first input:

"UFRRDFRFGDDFGLLTTDRF"

This is not the most optimal solution: it is only there for demonstrative purposes.

Example of a first output:

Here is the output you will get from the console in c++:
The information given on every line consists of the position (starts at (0,0) in the top-left corner of the grid). The first part of the position is your position measured downwards and the second part is left/right. Then, you have the remaining time after your action. You then have the current score followed by a description of the action.

```
The starting position is: 3 : 3
Pos: (2,3) time: 90s score: 0 action: Up
Pos: (2,3) time: 88s score: 35 action: You have flipped a GP from yellow to blue
Pos: (2,4) time: 87s score: 35 action: Right
Pos: (2,5) time: 86s score: 35 action: Right
Pos: (3,5) time: 85s score: 35 action: Down
Pos: (3,5) time: 83s score: 70 action: You have flipped a GP from yellow to blue
Pos: (3,6) time: 82s score: 70 action: Right
Pos: (3,6) time: 80s score: 105 action: You have flipped a GP from yellow to blue
Pos: (3,6) time: 79s score: 70 action: You have picked up a blue game piece
Pos: (4,6) time: 78s score: 70 action: Down
Pos: (5,6) time: 77s score: 70 action: Down
Pos: (5,6) time: 75s score: 105 action: You have flipped a GP from yellow to blue
Pos: (5,6) time: 74s score: 70 action: You have picked up a blue game piece
Pos: (5,5) time: 73s score: 70 action: Left
Pos: (5,4) time: 72s score: 70 action: Left
Pos: (5,4) time: 71s score: 110 action: You have put a game piece in the tower
Pos: (5,4) time: 69s score: 190 action: You have put a game piece in the tower
Pos: (6,4) time: 68s score: 190 action: Down
Pos: (6,5) time: 67s score: 190 action: Right
Pos: (6,5) time: 65s score: 225 action: You have flipped a GP from yellow to blue

Here is the time left and the total score: 65s & 225 points
```