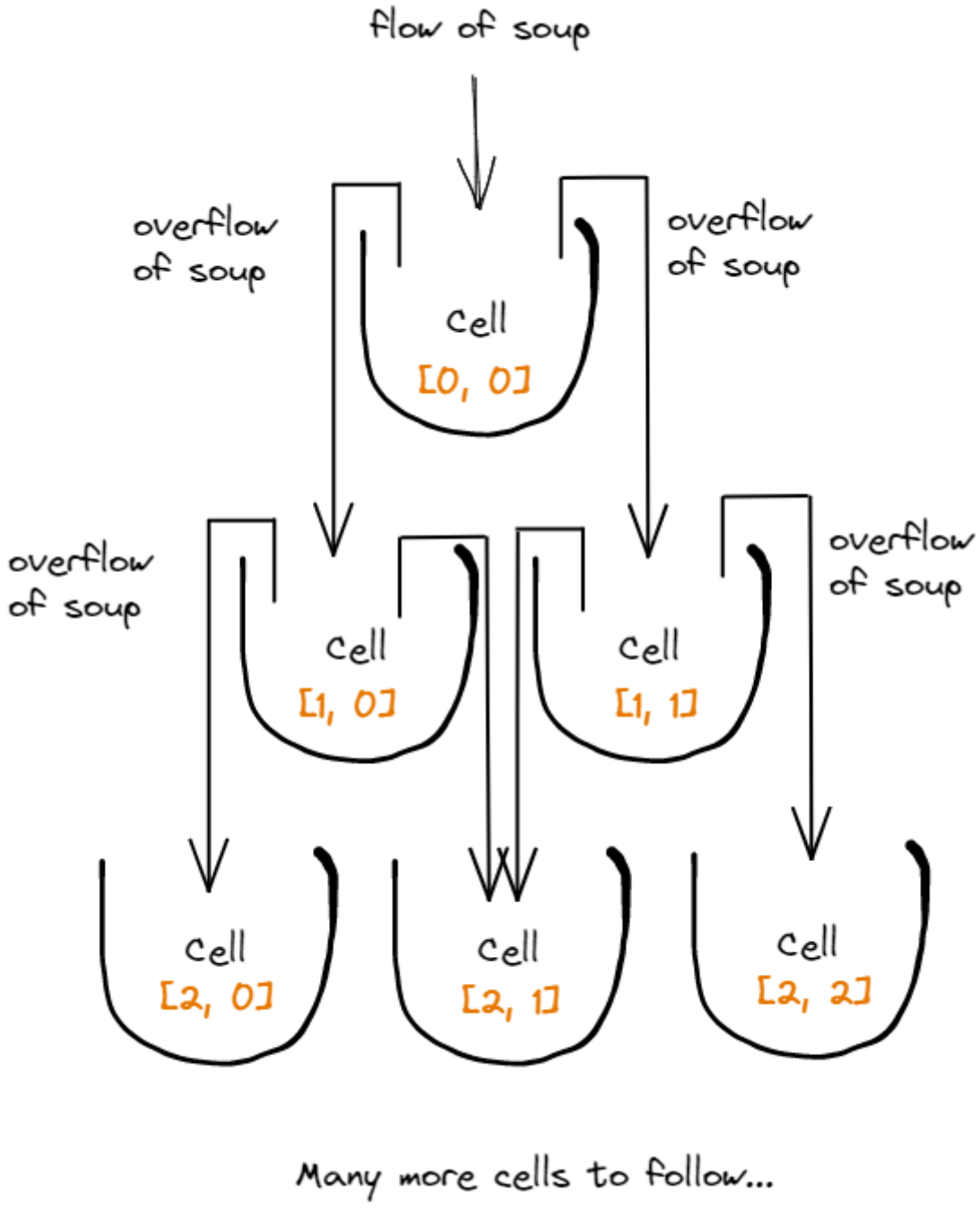
Magic Cauldrons

Problem Statement

You are in a room full of magic cauldrons. Each cauldron consists of many cells.
Below is a diagram of how one of them looks like.



Soup is poured from the top into the first cell (`[0, 0]`).
When the first cell is completely filled, extra soup overflows into the two cells below on the left (`[1, 0]`) and right (`[1, 1]`).
When any of the two cells at the second level is completely filled, extra soup also overflows into the cells at the third level (`[2, 0]`, `[2, 1]` and `[2, 2]`), so on and so forth.

Note: Each cell has a maximum capacity of 100 grams of soup.

Part 1

You observe the flow of soup in amusement, and realize that given a constant flow rate of soup from the top, the amount of soup in any given cell at any specific point in time is deterministic.

Given the rate of flow of soup from the top and the duration of time passed, can you calculate the amount of soup at a specific cell?

Input Description:
- `flow_rate`: rate of flow of soup from the top of the cauldron (gram/second), an integer
- `time`: time (in seconds) that has passed since the beginning (when soup started pouring into the cauldron), an integer
- `row_number`: row index of specified cell (0-indexed), an integer
- `col_number`: column index of specified cell (0-indexed), an integer

Output Description:
- `amount_of_soup`: current units of soup in specified cell (gram), accurate to two decimal places (with half-even rounding mode)

Part 2

After pondering over the previous problem, you've lost track of time!

Can you calculate how much time has passed, given the amount of soup at a specific cell, and the rate of flow of soup from the top?

Input Description:
- `flow_rate`: rate of flow of soup from the top of the cauldron (gram/second), an integer
- `amount_of_soup`: current units of soup in specified cell (gram), accurate to two decimal places (with half-even rounding mode)
- `row_number`: row index of specified cell (0-indexed), an integer
- `col_number`: column index of specified cell (0-indexed), an integer

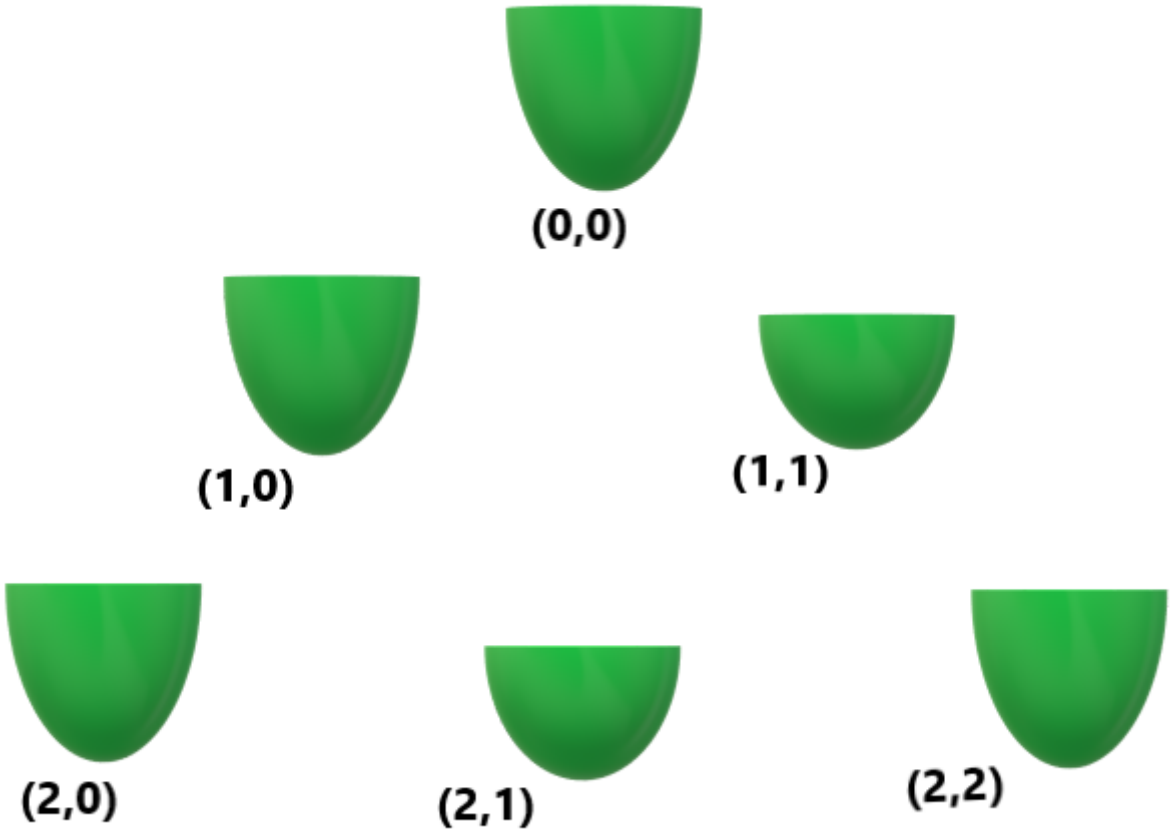
Output Description:
- `time`: time (in seconds) that has passed since the beginning (when soup started pouring into the cauldron), accurate to the nearest integer (with half-even rounding mode)

Part 3

You roamed around the room and found some even weirder magic cauldrons.

They are similar to the one described above, but the cells at even columns (0-indexed) are 1.5 times bigger than other cells.

i.e., cells at even columns have maximum capacity of 150 grams of soup, whereas cells at odd columns have maximum capacity of 100 grams of soup.



Given the rate of flow of soup from the top and the duration of time passed, can you calculate the amount of soup at a specific cell?

Input Description:
- `flow_rate`: rate of flow of soup from the top of the cauldron (gram/second), an integer
- `time`: time (in seconds) that has passed since the beginning (when soup started pouring into the cauldron), an integer
- `row_number`: row index of specified cell (0-indexed), an integer
- `col_number`: column index of specified cell (0-indexed), an integer

Output Description:
- `amount_of_soup`: current units of soup in specified cell (gram), accurate to two decimal places (with half-even rounding mode)

Part 4

For the irregular cauldrons as described in Part 3, can you calculate how much time has passed, given the amount of soup at a specific cell, and the rate of flow of soup from the top?

Input Description:
- `flow_rate`: rate of flow of soup from the top of the cauldron (gram/second), an integer
- `amount_of_soup`: current units of soup in specified cell (gram), accurate to two decimal places (with half-even rounding mode)
- `row_number`: row index of specified cell (0-indexed), an integer
- `col_number`: column index of specified cell (0-indexed), an integer

Output Description:
- `time`: time (in seconds) that has passed since the beginning (when soup started pouring into the cauldron), accurate to the nearest integer (with half-even rounding mode)

Sample Input

You would be given a series of tests in the form of a json array.


```
[
  {
    "part1": {
      "flow_rate": 20,
      "time": 1,
      "row_number": 0,
      "col_number": 0
    },
    "part2": {
      "flow_rate": 10,
      "amount_of_soup": 10.00,
      "row_number": 0,
      "col_number": 0
    },
    "part3": {
      "flow_rate": 30,
      "time": 2,
      "row_number": 0,
      "col_number": 0
    },
    "part4": {
      "flow_rate": 50,
      "amount_of_soup": 100.00,
      "row_number": 0,
      "col_number": 0
    }
  },
  {
    "part1": {
      "flow_rate": 23,
      "time": 1,
      "row_number": 0,
      "col_number": 0
    },
    "part2": {
      "flow_rate": 17,
      "amount_of_soup": 34.00,
      "row_number": 0,
      "col_number": 0
    },
    "part3": {
      "flow_rate": 36,
      "time": 1,
      "row_number": 0,
      "col_number": 0
    },
    "part4": {
      "flow_rate": 5,
      "amount_of_soup": 20.00,
      "row_number": 0,
```

```
      "col_number": 0
    }
  }
]
```

Note:
This is a sample input with only two test cases.
Actual input array will have many more elements.

Sample Output

Please return your answers in the form of a json array.

```
[
  {
    "part1": 20.00,
    "part2": 1,
    "part3": 60.00,
    "part4": 2
  },
  {
    "part1": 23.00,
    "part2": 2,
    "part3": 36.00,
    "part4": 4
  }
]
```

Note: Please make sure your output array has the same sequence as the input array.

Boundary conditions

- 1 < `flow_rate` < 999
- 1 < `time` < 999
- 0 <= `row_number` <= 99
- 0 <= `col_number` <= 99
- 0.00 < `amount_of_soup` < 100.00 (for Part 1, Part 2, and odd-column cells for Part 3, Part 4)
- 0.00 < `amount_of_soup` < 150.00 (for even-column cells for Part 3, Part 4)

Timeout: 4 seconds
Number of test cases per evaluation round: 25

Grading Criteria

Expose a POST endpoint `/magiccauldrons` in your registered solution server.
For each evaluation round, we will make a POST request with the json input (as described above) to your endpoint.

Partial marks will be given when answers for some but not all parts are correct.