

59m left

ALL

1

2

1. Question 1

You are working with a model represented as a binary texture image *image*. This texture is represented as an $n \times n$ grid where each 1 in the grid is a black pixel, and 0 is a white pixel.

There are three transformations to apply to the model's texture in this order:

1. Rotation: The texture image can be rotated by 90, 180, or 270 degrees clockwise specified in the variable *rotation*.
2. Vertical Flip: If *verticalFlip* = 1, flip the image along its horizontal axis. The pixels will appear in reverse order from top to bottom.
3. Horizontal Flip: If *horizontalFlip* = 1, flip the image along its vertical axis. The pixels will appear in reverse order from left to right

Implement a function that applies the transformations and returns the final image.

The function *getFinalImage* will take the following inputs:
int *image*[*n*][*n*]: the texture image to process
int *rotation*: the rotation parameter
int *verticalFlip*: the vertical flip parameter
int *horizontalFlip*: the horizontal flip parameter

The function should return a binary matrix representing the final texture image after performing all three operations in order.

Example
Suppose *n* = 3, *rotation* = 270, *verticalFlip* = 0, *horizontalFlip* = 1, and *image* =

```

1 0 0
0 1 1
0 0 1

```

Rotate the image by 270 degrees.

```

0 1 1
0 1 0
1 0 0

```

Since *verticalFlip* is 0, the image is not flipped vertically.

Perform the horizontal flip.

```

1 1 0
0 1 0
0 0 1

```

Python 3

Environment

Autocomplete Ready

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

```

1 > #!/bin/python3 -
10
11 #
12 # Complete the 'getFinalImage' function below.
13 #
14 # The function is expected to return a 2D_INTEGER_ARRAY.
15 # The function accepts following parameters:
16 # 1. 2D_INTEGER_ARRAY image
17 # 2. INTEGER rotation
18 # 3. INTEGER verticalFlip
19 # 4. INTEGER horizontalFlip
20 #
21
22 def getFinalImage(image, rotation, verticalFlip, horizontalFlip):
23     # Write your code here
24
25
26 > if __name__ == '__main__': -

```

Test Results

Custom Input

Run Code

Run Tests

Submit

helper functions:

- Rotate:
 - 90 degree: `m = [list(row)[::-1] for row in zip(*m)]`
 - `zip(*m)` : transposing a matrix

```

python

matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
zip(*matrix)  →  (1, 4, 7), (2, 5, 8), (3, 6, 9)

```

- 90 degree = transpose + reverse each row
- 180 degree = vertical flip + horizontal flip = 90 degree * 2
- 270 degree = vertical flip + transpose = 90 degree * 3
- else, no change
- Vertical
 - `m[::-1]`
- Horizontal
 - `[row[::-1] for row in m]`

43m left

ALL

2

In a multi-model machine learning system, different models are trained sequentially on a single GPU. If a higher-priority task arrives during training, the lower-priority task is paused until the higher-priority task finishes.

There are n different AI models to be trained on the GPU, each with a unique ID between 0 and $n-1$. A list of m training logs is represented as an array of strings, $logs[m]$. Log entries follow the format: $\{modelid\}:\{start/end\}:\{timestamp\}$, indicating that the model with ID = $modelid$, either starts or ends at a time identified by the *timestamp* value. If any model is introduced while the previous one is running, the previous model is put on hold, and the current model is trained until it is completed or put on hold.

Implement a function that determines each model's training time.

The function *getTotalTrainingTime* will take two inputs:
int n : the number of models train
string $logs[m]$: each string is a training log

The function should return an array specifying the exclusive training times of each model.

Example
Suppose $n = 3$, $logs = ["0:start:0", "2:start:4", "2:end:5", "1:start:7", "1:end:10", "0:end:11"]$

Timestamp	Model Running	Remarks
0	0	Model 0 starts
1	0	
2	0	
3	0	
4	2	Model 0 is paused and Model 2 starts
5	2	Model 2 ends
6	0	Model 0 resumes

Language Python 3 Environment

Autocomplete Ready

```

1 > #!/bin/python3~
10
11 #
12 # Complete the 'getTotalTrainingTime' function below.
13 #
14 # The function is expected to return an INTEGER_ARRAY.
15 # The function accepts following parameters:
16 # 1. INTEGER n
17 # 2. STRING_ARRAY logs
18 #
19
20 def getTotalTrainingTime(n, logs):
21     # Write your code here
22
23 > if __name__ == '__main__': ...

```

Line: 10 Col: 1

Test Results Custom Input

Run Code Run Tests Submit

use stack, obviously

steps:

- split log into segments: model, action, timestamp
- for each log, if start: pause current and push new one; else, pop current and resume prev if any

- track time deltas and add to corresponding model's counter

```

def getTotalTrainingTime(n, logs):
    stack = []
    res = []
    prev = 0

    for l in logs:
        seg = l.split(":")
        model, action, timestamp = int(seg[0]), seg[1], int(seg[2])

        if stack: #check if currently has a model running
            res[stack[-1]] += timestamp - prev # time btw the new model start and current
            running is available for current model to run
        if action == "start":
            stack.append(model)
        else: # a model finish running
            res[stack.pop()] += 1 #pop from stack and add to res, +=1 for account inclusive end
            timestamp += 1 # update timestamp to next logical time
        prev = timestamp
    return res

```

