

Problem D

Donut Drone

Submits: 60

Accepted: ?

Author: Luka Kalinovčić

1	2	6	1	1
2	4	1	2	2
5	5	5	3	5
3	1	2	5	3

1	2	6	1	1
2	4	1	2	2
5	5	5	3	5
3	1	2	5	3

1	2	6	1	1
2	4	1	2	2
5	5	5	3	5
3	1	2	5	3

1	2	6	1	1
2	4	1	2	2
5	5	5	3	5
3	1	2	5	3

1	2	6	1	1
2	4	1	2	2
5	5	5	3	5
3	1	2	5	3

1	2	6	1	1
2	4	1	2	2
5	5	5	3	5
3	1	2	5	3

1	2	6	1	1
2	4	1	2	2
5	5	5	3	5
3	1	2	5	3

1	2	6	1	1
2	4	1	2	2
5	5	5	3	5
3	1	2	5	3

1	2	6	1	1
2	4	1	2	2
5	5	5	3	5
3	1	2	5	3

1	2	6	1	1
2	4	1	2	2
5	5	5	3	5
3	1	2	5	3

The task is to implement two functions:

`move(k)`: Moves a drone k steps and reports the final coordinates.

`update(row, col, value)`: Updates the elevation at provided coordinates.

1	2	6	1	1
2	4	1	2	2
5	5	5	3	5
3	1	2	5	3

Let's start with a naive solution:

```
def simple_move(k):  
    for i in range(k):  
        coords = step(coords)  
    return coords
```

Complexity: $O(k)$ - too slow.

1	2	6	1	1
2	4	1	2	2
5	5	5	3	5
3	1	2	5	3

Observation: The drone will eventually enter a cycle.

1	2	6	1	1
2	4	1	2	2
5	5	5	3	5
3	1	2	5	3

Observation: The drone will eventually enter a cycle.

1	2	6	1	1
2	4	1	2	2
5	5	5	3	5
3	1	2	5	3

Observation: The drone will eventually enter a cycle.

1	2	6	1	1
2	4	1	2	2
5	5	5	3	5
3	1	2	5	3

Observation: The drone will eventually enter a cycle.

1	2	6	1	1
2	4	1	2	2
5	5	5	3	5
3	1	2	5	3

Observation: The drone will eventually enter a cycle.

1	2	6	1	1
2	4	1	2	2
5	5	5	3	5
3	1	2	5	3

Observation: The drone will eventually enter a cycle.

```
def smarter_move(k):  
    first_seen = dict()  
    for i in range(k):  
        if coords not in first_seen:  
            first_seen[coords] = i  
        else:  
            cycle_length = i - first_seen[coords]  
            steps_left = k - i  
            return simple_move(steps_left % cycle_length)  
        coords = step(coords)  
    return coords
```

Complexity $O(R \cdot C)$ - still too slow in the worst case.

Key idea: Maintain an array `jump[row]` that stores the cell we would end up if we moved C steps from a cell $(row, 1)$ in the first column.

As soon as we reach the first column we can start making jumps of size C that stay in the first column until there are less C steps to make.

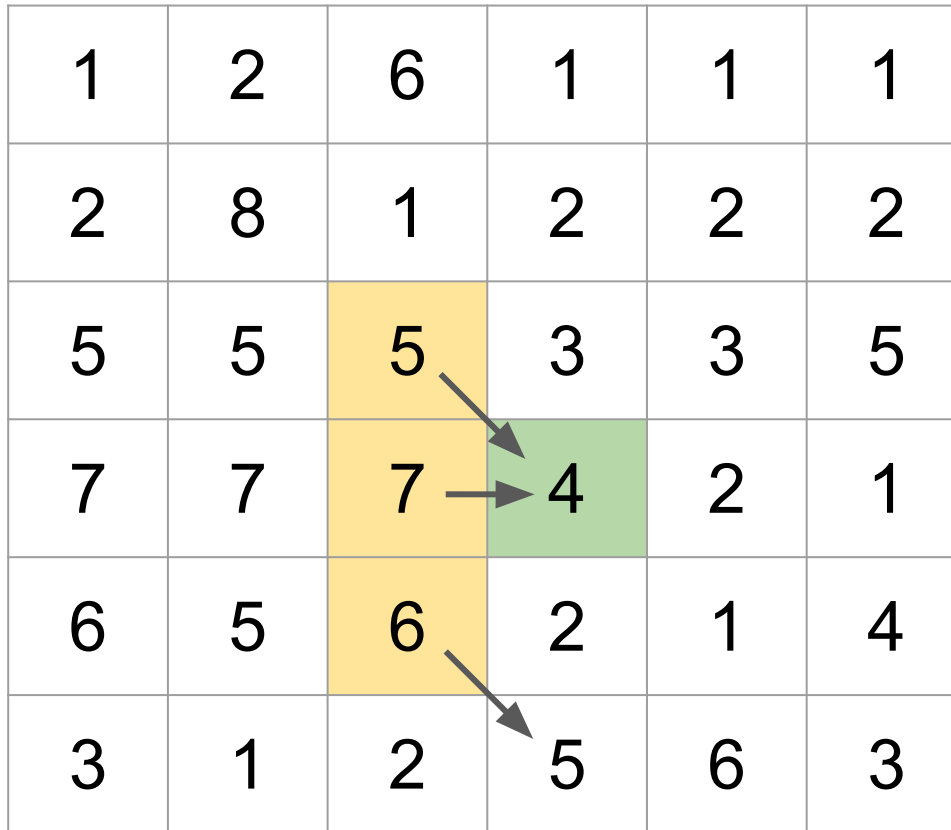
Then we proceed to make single steps again to find the final cell.

If we also implement the cycle detection among the cells in the first column we end up with a $O(R + C)$ move operation.

1	2	6	1	1	1
2	8	1	2	2	2
5	5	5	3	3	5
7	7	7	4	2	1
6	5	6	2	1	4
3	1	2	5	6	3

However, the `update(row, col, value)` becomes tricky, as we may need to update the `jump[row]` array.

1	2	6	1	1	1
2	8	1	2	2	2
5	5	5	3	3	5
7	7	7	4	2	1
6	5	6	2	1	4
3	1	2	5	6	3



Up to three cells may be directly affected.

1	2	6	1	1	1
2	8	1	2	2	2
5	5	5	3	3	5
7	7	7	1	2	1
6	5	6	2	1	4
3	1	2	5	6	3

Up to three cells may be directly affected.

1	2	6	1	1	1
2	8	1	2	2	2
5	5	5	3	3	5
7	7	7	1	2	1
6	5	6	2	1	4
3	1	2	5	6	3

For each affected cell, we'll run the update algorithm.

1	2	6	1	1	1
2	8	1	2	2	2
5	5	5	3	3	5
7	7	7	1	2	1
6	5	6	2	1	4
3	1	2	5	6	3

For each affected cell, we'll run the update algorithm.
1) Repeatedly make steps to find in which cell in the first column we'll end up.

1	2	6	1	1	1
2	8	1	2	2	2
5	5	5	3	3	5
7	7	7	1	2	1
6	5	6	2	1	4
3	1	2	5	6	3

For each affected cell, we'll run the update algorithm.
1) Repeatedly make steps to find in which cell in the first column we'll end up.

1	2	6	1	1	1
2	8	1	2	2	2
5	5	5	3	3	5
7	7	7	1	2	1
6	5	6	2	1	4
3	1	2	5	6	3

For each affected cell, we'll run the update algorithm.
1) Repeatedly make steps to find in which cell in the first column we'll end up.

1	2	6	1	1	1
2	8	1	2	2	2
5	5	5	3	3	5
7	7	7	1	2	1
6	5	6	2	1	4
3	1	2	5	6	3

For each affected cell, we'll run the update algorithm.
1) Repeatedly make steps to find in which cell in the first column we'll end up.

1	2	6	1	1	1
2	8	1	2	2	2
5	5	5	3	3	5
7	7	7	1	2	1
6	5	6	2	1	4
3	1	2	5	6	3

For each affected cell, we'll run the update algorithm.
1) Repeatedly make steps to find in which cell in the first column we'll end up.

1	2	6	1	1	1
2	8	1	2	2	2
5	5	5	3	3	5
7	7	7	1	2	1
6	5	6	2	1	4
3	1	2	5	6	3

For each affected cell, we'll run the update algorithm.
2) Starting from the affected cell, backtrack to the first column, maintaining an interval of affected rows.

1	2	6	1	1	1
2	8	1	2	2	2
5	5	5	3	3	5
7	7	7	1	2	1
6	5	6	2	1	4
3	1	2	5	6	3

For each affected cell, we'll run the update algorithm.
2) Starting from the affected cell, backtrack to the first column, maintaining an interval of affected rows.

1	2	6	1	1	1
2	8	1	2	2	2
5	5	5	3	3	5
7	7	7	1	2	1
6	5	6	2	1	4
3	1	2	5	6	3

For each affected cell, we'll run the update algorithm.
2) Starting from the affected cell, backtrack to the first column, maintaining an interval of affected rows.

1	2	6	1	1	1
2	8	1	2	2	2
5	5	5	3	3	5
7	7	7	1	2	1
6	5	6	2	1	4
3	1	2	5	6	3

For each affected cell, we'll run the update algorithm.
3) If we reach the first column, we have an interval of rows to update `jump[row]` for.

1	2	6	1	1	1
2	8	1	2	2	2
5	5	5	3	3	5
7	7	7	1	2	1
6	5	6	2	1	4
3	1	2	5	6	3

Interval bounds may only move by ± 1 between neighbouring columns, so we can maintain the affected interval in $O(1)$ per column as we backtrack. Overall update complexity is $O(C)$.