[CS 11 25.1] Lab 4d – Koyuki and Bombs 5

Cheatsheet is available here: https://oj.dcs.upd.edu.ph/cs11cheatsheet/

Submit solution [CS 11 25.1] Lab Exercise 4

Points: 160 (partial)
Time limit: 15.0s
Memory limit: 2G

Problem type

Allowed languages
py3

Problem Statement

Koyuki is still throwing bombs around; Noa and Yuuka have just accepted reality at this point.

She is *still* on an infinite grid of cells. Each cell has position (i, j), where the i represents the vertical location and increases as you go down the grid, and the j represents the horizontal location and increases as you go right.

Koyuki throws bombs at various cells, and she has programmed the bombs to all have a *power* of p. If Koyuki is currently standing on the cell with posiiton (k_i, k_j), visualize the bomb traces in the $(2n + 1) \times (2n + 1)$ area of cells centered at Koyuki (see **Task Details** for specifics).

Note. When we say *power*, we visually mean this, for $p = 0,1,2,3$:

```
p=0     p=1     p=2     p=3
.......  .......  .......  ...X...
.......  .......  ...X...  ..X.X..
.......  ...X...  ..X.X..  .X...X.
...X...  ..X.X..  .X...X.  X.....X
.......  ...X...  ..X.X..  .X...X.
.......  .......  ...X...  ..X.X..
.......  .......  .......  ...X...
```

Here, the bomb is thrown at the cell in the fourth row and the fourth column, and the cells marked with `X` are the cells caught in the blast.

Task Details

Your task is to implement a function named `draw_bomb_traces`, which should have the following *signature*:

```
def draw_bomb_traces(bombs, p, koyuki, n):
```

The above says that it has four arguments `bombs`, p, `koyuki`, and n.

- `bombs` is a `tuple` of pairs (`tuple`s of length 2) of integers (`int`s) denoting the positions of the bombs.
- p is an integer denoting the power of the bombs.
- `koyuki` is a pair of `int`s corresponding to k_i and k_j.
- n is an integer.

The function must return a string (`str`) denoting the traces of the thrown bombs in a square area centered at Koyuki. In particular,

- It must have exactly $2n + 1$ rows separated by newlines (`\n`), and each row must consist of exactly $2n + 1$ characters (aside from the newline).
- The character at the n^{th} row and n^{th} column (0-indexed) should correspond to the position Koyuki is in.
- Each character should be marked as `X` if it is caught in the blast of some bomb, and `.` otherwise.

Restrictions

- The following symbols can now be used: `list`, `set`, `dict`, `enumerate`, `append`, `pop`, `extend`, `remove`, `sort`, `sorted`, `insert`, `clear`, `reverse`, `reversed`.
- Loops are allowed.
- Recursion is *disallowed*.
- Comprehensions are *disallowed*.
- Your source code must have at most $1{,}200$ bytes.

Examples

Example 1 Function Call

```
draw_bomb_traces(((1, 1), (3, 2)), 1, (2, 2), 2)
```

Example 1 Return Value

```
"""\
.X...
X.X..
.XX..
.X.X.
..X..
"""
```

Example 2 Function Call

```
draw_bomb_traces(((-4, 6), (-1, 14), (-4, 15)), 4, (-3, 8), 4
```

Example 2 Return Value

```
"""\
.X.X.....
X...X....
.....X..X
......XX.
.....X..X
X...X..X.
.X.X..X..
..X....X.
........X
"""
```

Constraints

- The function `draw_bomb_traces` will be called at most 300 times.
- $|i|, |j|, |k_i|, |k_j| \le 10^{20}$
- $0 \le p \le 10^9$
- $0 \le n \le 100$
- The sum of n across all calls to `draw_bomb_traces` will be ≤ 300.
- The length of `bombs` will be at most 100.

Scoring

Note: New tests may be added and all submissions may be rejudged at a later time. (All future tests will satisfy the constraints.)

- You get 10 ❤️ points if you solve all test cases where:
  - $p \le 1$
  - $n \ge 1$
- You get 25 ❤️ points if you solve all test cases where:
  - $p \le 100$
  - $n \ge 1$
- You get 25 ❤️ points if you solve all test cases where:
  - $n \ge 1$
- You get 100 🔴 points if you solve all test cases.

Clarifications

Report an issue

No clarifications have been made at this time.