Problem 3242: Design Neighbor Sum Service

Problem Information

Difficulty: Easy
Acceptance Rate: 76.14%
Paid Only: No
Tags: Array, Hash Table, Design, Matrix, Simulation

Problem Description

You are given a `n x n` 2D array `grid` containing **distinct** elements in the range `[0, n2 - 1]`.

Implement the `NeighborSum` class:

* `NeighborSum(int [][]grid)` initializes the object. * `int adjacentSum(int value)` returns the **sum** of elements which are adjacent neighbors of `value`, that is either to the top, left, right, or bottom of `value` in `grid`. * `int diagonalSum(int value)` returns the **sum** of elements which are diagonal neighbors of `value`, that is either to the top-left, top-right, bottom-left, or bottom-right of `value` in `grid`.

![](https://assets.leetcode.com/uploads/2024/06/24/design.png)

Example 1:

Input:

["NeighborSum", "adjacentSum", "adjacentSum", "diagonalSum", "diagonalSum"]

[[[[0, 1, 2], [3, 4, 5], [6, 7, 8]]], [1], [4], [4], [8]]

Output: [null, 6, 16, 16, 4]

Explanation:

![](https://assets.leetcode.com/uploads/2024/06/24/designexample0.png)

* The adjacent neighbors of 1 are 0, 2, and 4. * The adjacent neighbors of 4 are 1, 3, 5, and 7. * The diagonal neighbors of 4 are 0, 2, 6, and 8. * The diagonal neighbor of 8 is 4.

Example 2:

Input:

["NeighborSum", "adjacentSum", "diagonalSum"]

[[[[1, 2, 0, 3], [4, 7, 15, 6], [8, 9, 10, 11], [12, 13, 14, 5]]], [15], [9]]

Output: [null, 23, 45]

Explanation:

![](https://assets.leetcode.com/uploads/2024/06/24/designexample2.png)

* The adjacent neighbors of 15 are 0, 10, 7, and 6. * The diagonal neighbors of 9 are 4, 12, 14, and 15.

Constraints:

* `3 <= n == grid.length == grid[0].length <= 10` * `0 <= grid[i][j] <= n2 - 1` * All `grid[i][j]` are distinct. * `value` in `adjacentSum` and `diagonalSum` will be in the range `[0, n2 - 1]`. * At most `2 * n2` calls will be made to `adjacentSum` and `diagonalSum`.

Code Snippets

C++:

```
class NeighborSum {
public:
NeighborSum(vector<vector<int>>& grid) {

}

int adjacentSum(int value) {

}
int diagonalSum(int value) {

}
};

/**
* Your NeighborSum object will be instantiated and called as such:
* NeighborSum* obj = new NeighborSum(grid);
* int param_1 = obj->adjacentSum(value);
* int param_2 = obj->diagonalSum(value);
*/
```

Java:

```
class NeighborSum {

public NeighborSum(int[][] grid) {

}

public int adjacentSum(int value) {

}

public int diagonalSum(int value) {

}
}

/**
* Your NeighborSum object will be instantiated and called as such:
* NeighborSum obj = new NeighborSum(grid);
* int param_1 = obj.adjacentSum(value);
* int param_2 = obj.diagonalSum(value);
*/
```

Python3:

```
class NeighborSum:

def __init__(self, grid: List[List[int]]):
```

```
def adjacentSum(self, value: int) -> int:


def diagonalSum(self, value: int) -> int:



# Your NeighborSum object will be instantiated and called as such:
# obj = NeighborSum(grid)
# param_1 = obj.adjacentSum(value)
# param_2 = obj.diagonalSum(value)
```