Count pairs Sum in matrices

Author:	Prakash JC

Problem Statement

Given two matrices `mat1[][]` and `mat2[][]` of size `n x n`, where the elements in each matrix are arranged in strictly ascending order. Specifically, each row is sorted from left to right, and the last element of a row is smaller than the first element of the next row.

You're given a target value `x`, your task is to find and count all pairs `{a, b}` such that `a` is from `mat1` and `b` is from `mat2` where the sum of `a + b` is equal to `x`.

Examples:

Input: `n = 3`, `x = 21`,

`mat1[][] = [[1, 5, 6], [8, 10, 11], [15, 16, 18]]`,

`mat2[][] = [[2, 4, 7], [9, 10, 12], [13, 16, 20]]`

Output: 4

Explanation: The pairs whose sum is found to be 21 are (1, 20), (5, 16), (8, 13) and (11, 10).

Input: `n = 2`, `x = 10`,

`mat1[][] = [[1, 2], [3, 4]]`

`mat2[][] = [[4, 5], [6, 7]]`

Output: 2

Explanation: The pairs whose sum found to be 10 are (4, 6) and (3, 7).

Constraints:

`1 ≤ n ≤ 100`

`1 ≤ x ≤ 10^5`

`1 ≤ mat1[i][j] , mat2[i][j] ≤ 10^5`

Editorial

Brute Force Approach

Approach

- For every element in `mat1`, iterate through every element in `mat2`.
- If sum matches, increment the count.

Code

```
class Solution {
public:
    int countPairs(vector<vector<int>> &mat1, vector<vector<int>> &mat2, int x) {
```

```
        int n = mat1.size();
        int count = 0;

        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j)
                for (int p = 0; p < n; ++p)
                    for (int q = 0; q < n; ++q)
                        if (mat1[i][j] + mat2[p][q] == x)
                            count++;

        return count;
    }
};
```

Complexity Analysis

- **Time Complexity:** `O(n^4)`
- **Space Complexity:** `O(1)`

Better Approach Using Hashing

Approach

- Insert all elements of `mat2` into a hash set.
- For each element `a` in `mat1`, check if `x - a` exists in the hash set.

This avoids redundant comparisons and gives linear time checking.

Code

```
class Solution {
public:
    int countPairs(vector<vector<int>> &mat1, vector<vector<int>> &mat2, int x) {
        int n = mat1.size();
        unordered_set<int> st;
        int count = 0;

        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j)
                st.insert(mat2[i][j]);

        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j)
                if (st.count(x - mat1[i][j]))
                    count++;

        return count;
    }
};
```

Complexity Analysis

- **Time Complexity:** `O(n^2)`
- **Space Complexity:** `O(n^2)`

Optimal Approach Using Flattening + Two Pointers

Approach

- Flatten `mat1` and `mat2` into 1D sorted arrays.
- Use two pointers:
  - One starting at the beginning of `mat1` (`i = 0`)
  - One starting at the end of `mat2` (`j = n^2 - 1`)
- While `i < n^2` and `j >= 0`:
  - If `mat1[i] + mat2[j] == x`: count++, move both pointers.
  - If sum < x: move `i++`
  - If sum > x: move `j--`

This leverages full sorted property across matrix.

Code

```
class Solution {
public:
    int countPairs(vector<vector<int>> &mat1, vector<vector<int>> &mat2, int x) {
        int n = mat1.size();
        vector<int> flat1, flat2;

        // Flatten mat1 and mat2
        for (int i = 0; i < n; ++i) {
            flat1.insert(flat1.end(), mat1[i].begin(), mat1[i].end());
            flat2.insert(flat2.end(), mat2[i].begin(), mat2[i].end());
        }

        int i = 0, j = flat2.size() - 1, count = 0;

        while (i < flat1.size() && j >= 0) {
            int sum = flat1[i] + flat2[j];
            if (sum == x) {
                count++;
                i++;
                j--;
            } else if (sum < x) {
                i++;
            } else {
                j--;
            }
        }

        return count;
```

```
    }
};
```

Complexity Analysis

- **Time Complexity:** `O(n^2)`
- **Space Complexity:** `O(n^2)` for flattened arrays

THE END