Max Increase to Keep City Skyline

In a 2 dimensional array `grid`, each value `grid[i][j]` represents the height of a building located there. We are allowed to increase the height of any number of buildings, by any amount (the amounts can be different for different buildings). Height 0 is considered to be a building as well.

At the end, the "skyline" when viewed from all four directions of the grid, i.e. top, bottom, left, and right, must be the same as the skyline of the original grid. A city's skyline is the outer contour of the rectangles formed by all the buildings when viewed from a distance. See the following example.

What is the maximum total sum that the height of the buildings can be increased?

```
Example:
Input: grid = [[3,0,8,4],[2,4,5,7],[9,2,6,3],[0,3,1,0]]
Output: 35
Explanation:
The grid is:
[ [3, 0, 8, 4],
  [2, 4, 5, 7],
  [9, 2, 6, 3],
  [0, 3, 1, 0] ]

The skyline viewed from top or bottom is: [9, 4, 8, 7]
The skyline viewed from left or right is: [8, 7, 9, 3]

The grid after increasing the height of buildings without affecting skylines is:

gridNew = [ [8, 4, 8, 7],
            [7, 4, 7, 7],
            [9, 4, 8, 7],
            [3, 3, 3, 3] ]
```

Notes:

- `1 < grid.length = grid[0].length <= 50`.
- All heights `grid[i][j]` are in the range `[0, 100]`.
- All buildings in `grid[i][j]` occupy the entire grid cell: that is, they are a `1 x 1 x grid[i][j]` rectangular prism.

Solution 1

Idea:
For `grid[i][j]`, it can't be higher than the maximun of its row nor the maximum of its col.
So the maximum increasing height for a building at `(i, j)` is `min(row[i], col[j]) - grid[i][j]`

Codes:
`row`: maximum for every row
`col`: maximum for every col
The fisrt loop of grid calcule maximum for every row and every col.
The second loop calculate the maximum increasing height for every building.

Complexity
O(N^2)

C++

```
    int maxIncreaseKeepingSkyline(vector<vector<int>>& grid) {
        int n = grid.size();
        vector<int> col(n, 0), row(n, 0);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                row[i] = max(row[i], grid[i][j]);
                col[j] = max(col[j], grid[i][j]);
            }
        }
        int res = 0;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                res += min(row[i], col[j]) - grid[i][j];
        return res;
    }
```

Java:

```
    public int maxIncreaseKeepingSkyline(int[][] grid) {
        int n = grid.length;
        int[] col = new int[n], row = new int[n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                row[i] = Math.max(row[i], grid[i][j]);
                col[j] = Math.max(col[j], grid[i][j]);
            }
        }
        int res = 0;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                res += Math.min(row[i], col[j]) - grid[i][j];
        return res;
    }
```

Python:

```python
    def maxIncreaseKeepingSkyline(self, grid):
        row, col = map(max, grid), map(max, zip(*grid))
        return sum(min(i, j) for i in row for j in col) - sum(map(sum, grid))
```

written by lee215 original link here

Solution 2

```
int maxIncreaseKeepingSkyline(vector<vector<int>>& grid) {
    vector<int> row_max(grid.size(), INT_MIN);
    vector<int> col_max(grid.size() ? grid[0].size() : 0, INT_MIN);
    int result = 0;

    for(int i = 0 ; i < grid.size(); i++) {
        for(int j = 0; j < grid[0].size(); j++) {
            row_max[i] = max(row_max[i], grid[i][j]);
            col_max[i] = max(col_max[i], grid[j][i]);
        }
    }

    for(int i = 0 ; i < grid.size(); i++) {
        for(int j = 0; j < grid[0].size(); j++) {
            if(grid[i][j] < row_max[i] && grid[i][j] < col_max[j])
                result += min(row_max[i] - grid[i][j], col_max[j] - grid[i][j]);
        }
    }

    return result;
}
```

written by DDev original link here

Solution 3

```
public int maxIncreaseKeepingSkyline(int[][] grid) {
        if (grid == null || grid.length == 0) return 0;
        int m = grid.length;
        int n = grid[0].length;
        int[] topbot = new int[n];
        int[] leftright = new int[m];
        int res = 0;
        for (int i = 0; i < topbot.length; i++) {
            for (int j = 0; j < grid[i].length; j++) {
                topbot[i] = Math.max(topbot[i], grid[i][j]);
            }
        }
        for (int i = 0; i < leftright.length; i++) {
            for (int j = 0; j < grid.length; j++) {
                leftright[i] = Math.max(grid[j][i], leftright[i]);
            }
        }
        int[][] dp = new int[m][n];
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                dp[i][j] = Math.min(topbot[j], leftright[i]);
                res += dp[i][j] - grid[i][j];
            }
        }
        return res;
    }
```

written by silin original link here