```
/*
在二维数组grid中，grid[i][j]代表位于某处的建筑物的高度。 我们被允许增加任何数量（不同建筑物的数量可能不同）的建筑物的高度。 高度 0 也被认为是建筑物。
最后，从新数组的所有四个方向（即顶部，底部，左侧和右侧）观看的“天际线”必须与原始数组的天际线相同。 城市的天际线是从远处观看时，由所有建筑物形成的矩形的外部轮廓。 请看下面的例子。
建筑物高度可以增加的最大总和是多少？
例子：
输入： grid = [[3,0,8,4],[2,4,5,7],[9,2,6,3],[0,3,1,0]]
输出： 35
解释：
The grid is:
[ [3, 0, 8, 4],
  [2, 4, 5, 7],
  [9, 2, 6, 3],
  [0, 3, 1, 0] ]

从数组竖直方向（即顶部，底部）看“天际线”是：[9, 4, 8, 7]
从水平水平方向（即左侧，右侧）看“天际线”是：[8, 7, 9, 3]

在不影响天际线的情况下对建筑物进行增高后，新数组如下：
gridNew = [ [8, 4, 8, 7],
            [7, 4, 7, 7],
            [9, 4, 8, 7],
            [3, 3, 3, 3] ]
说明:

    1 < grid.length = grid[0].length <= 50。
     grid[i][j] 的高度范围是： [0, 100]。
    一座建筑物占据一个grid[i][j]：换言之，它们是 1 x 1 x grid[i][j] 的长方体。

来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/max-increase-to-keep-city-skyline
著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。
*/
```

分析:

- 分析题意,遍历每一个元素,求出该个元素与(该元素所在行的最大值和所在列的最大值)之间的较小值的差值累加起来即可.

方法一:C_常规方法

```
int maxIncreaseKeepingSkyline(  int**   grid        ,
                                int     gridSize    ,
                                int*    gridColSize
                              )
{
    int     i           =   0;
    int     j           =   0;
    int     ret_val     =   0;
    int*    rows_max    =   (int*)malloc(gridSize*sizeof(int));
    int*    cols_max    =   (int*)malloc((*gridColSize)*sizeof(int));
```

```
11
12
13      /* 求出行最大值   */
14      for(i = 0 ; i < gridSize ; i++)
15      {
16          rows_max[i] = grid[i][0];
17          for(j = 1; j < *gridColSize ; j++)
18          {
19              if(grid[i][j] > rows_max[i])
20              {
21                  rows_max[i] = grid[i][j];
22              }
23          }
24      }
25
26      /* 求出列最大值   */
27      for(i = 0 ; i < *gridColSize ; i++)
28      {
29          cols_max[i] = grid[0][i];
30          for(j = 1 ; j < gridSize ; j++)
31          {
32              if(grid[j][i] > cols_max[i])
33              {
34                  cols_max[i] = grid[j][i];
35              }
36          }
37      }
38
39      for(i = 0 ; i < gridSize ; i++) /* 遍历行*/
40      {
41          for(j = 0 ; j < *gridColSize ; j++) /* 遍历列*/
42          {
43              //ret_val += __min(rows_max[i],cols_max[j]) - grid[i][j];
44              ret_val +=  (rows_max[i]>cols_max[j] ? cols_max[j] :
rows_max[i]) - grid[i][j];
45          }
46      }
47
48      free(rows_max);
49      free(cols_max);
50
51      return ret_val;
52  }
53
54  /*
55  执行结果:
56  通过
57  显示详情
58  执行用时 :12 ms, 在所有 C 提交中击败了88.89% 的用户
59  内存消耗 :7.4 MB, 在所有 C 提交中击败了100.00%的用户
60  */
```

AlimyBreak
2019.09.19