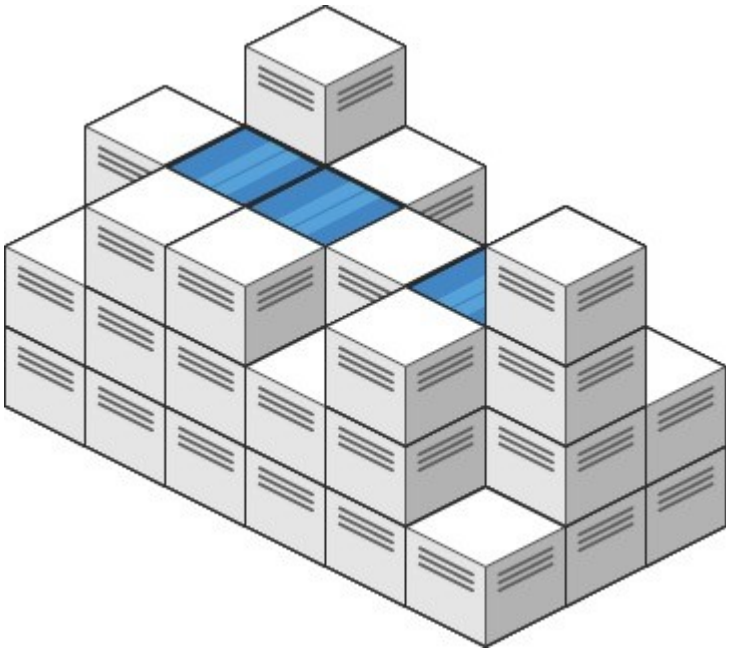
407. Trapping Rain Water II

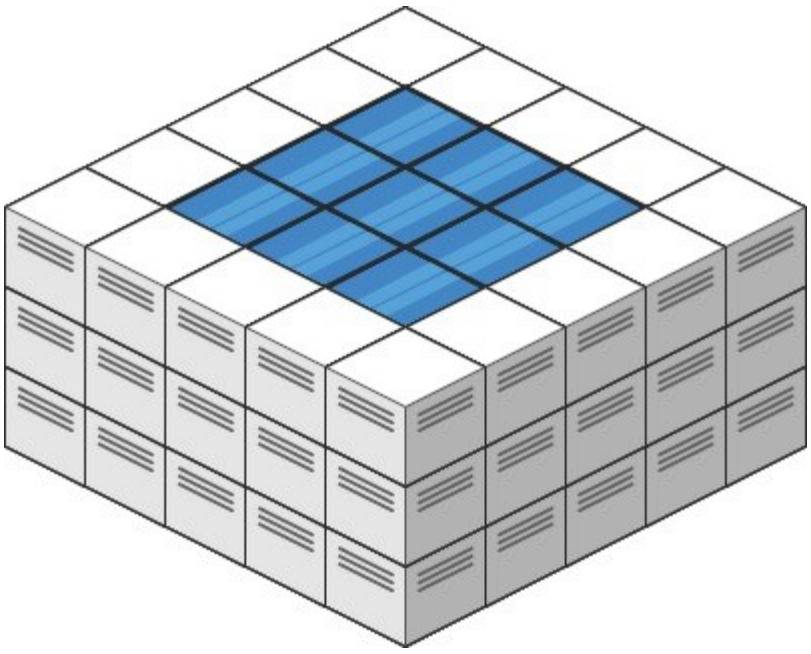
Given an `m x n` integer matrix `heightMap` representing the height of each unit cell in a 2D elevation map, return *the volume of water it can trap after raining*.

Example 1:



```
Input: heightMap = [[1,4,3,1,3,2],[3,2,1,3,2,4],[2,3,3,2,3,1]]
Output: 4
Explanation: After the rain, water is trapped between the blocks.
We have two small ponds 1 and 3 units trapped.
The total volume of water trapped is 4.
```

Example 2:



Input: heightMap = [[3,3,3,3,3],[3,2,2,2,3],[3,2,1,2,3],[3,2,2,2,3],[3,3,3,3,3]]
Output: 10

Constraints:

- `m == heightMap.length`
- `n == heightMap[i].length`
- `1 <= m, n <= 200`
- `0 <= heightMap[i][j] <= 2 * 10`4

407. Trapping Rain Water II

```
/*
   BFS + Min heap
   Time compelxity: O(2mn+mnlog(mn))
   Space compelxity: O(2mn)
*/

typedef std::pair<int,int> ii;
typedef std::pair<int,ii>iii;
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
typedef std::vector<ii> vii;
typedef std::vector<iii> viii;

class Solution {
   public:
      int trapRainWater(vector<vector<int>>& heightMap){
         int m=heightMap.size();
         int n=heightMap[0].size();

         // Directions for movement: right, left, down, up
         vvi directions={{0,1},{0,-1},{1,0},{-1,0}};

         // To mark visited cells
         vvi visited(m,vi(n,0));

         // Min heap to process cels in increasing height order
         std::priority_queue<iii,viii,std::greater<iii>> min_heap;
```

```
 /*Add the boundary to yhe min heap*/
// Add the first and the last row the the min heap
for(int i=0;i<n;++i){
   min_heap.push({heightMap[0][i],{0,i}});
   min_heap.push({heightMap[m-1][i],{m-1,i}});
   // and mark their cells as visited
   visited[0][i]=visited[m-1][i]=1;
}

// Add the first and the last column to the min heap
for(int i=0;i<m;++i){
   min_heap.push({heightMap[i][0],{i,0}});
   min_heap.push({heightMap[i][n-1],{i,n-1}});
   // and mark their cells as visited
   visited[i][0]=visited[i][n-1]=1;
}
```

```
        int ans=0;
        while(!min_heap.empty()){
            // Get the cell with min height from the boundary
            auto[cur_height,cur_cell]=min_heap.top();
            min_heap.pop();
            int cur_row=cur_cell.first;
            int cur_col=cur_cell.second;
            // Explore the four neighbor of the current cell
            for(auto& dir: directions){
                // For each neighbor
                int row=cur_row+dir[0];
                int col=cur_col+dir[1];
                // If it is not in the grid, no need to process it and pass to next
                if(row<0 || col<0 || row>m-1 || col>n-1) continue;

                // If it is visited, no need to process it and pass to next
                if(visited[row][col]) continue;

                /*Otherwise*/
                // Get its height
                int height=heightMap[row][col];

                // If its height is greater than or equal to the height of the current cell
                // means, it can not trap water
                // Otherwise, it can trap water
                // Add the volume to the answer
                ans+=std::max(0,cur_height-height);

                // Propagate the higher height max(current height, neighbor's height)
                // Add the neighbor to the min heap with the new height
                min_heap.push({std::max(cur_height,height),{row,col}});
                // Mark it as visited
                visited[row][col]=1;
            }
        }
        return ans;
    }
};
```