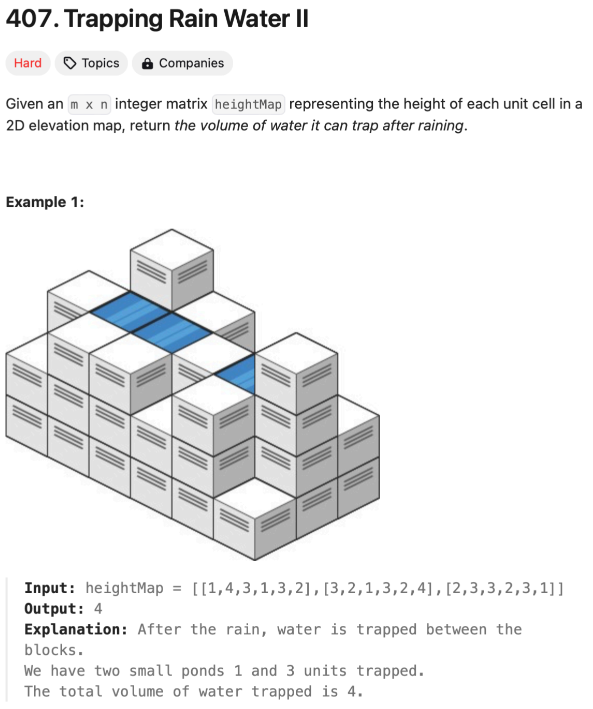


407. Trapping Rain Water II

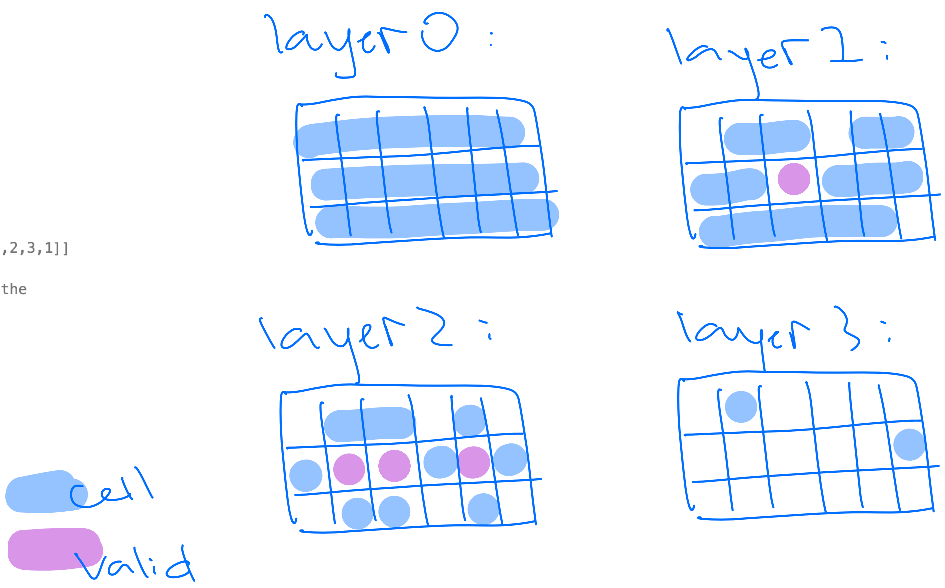
Given an `m x n` integer matrix `heightMap` representing the height of each unit cell in a 2D elevation map, return *the volume of water it can trap after raining*.

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.
```

① Bottom-up

Find holes in each layer

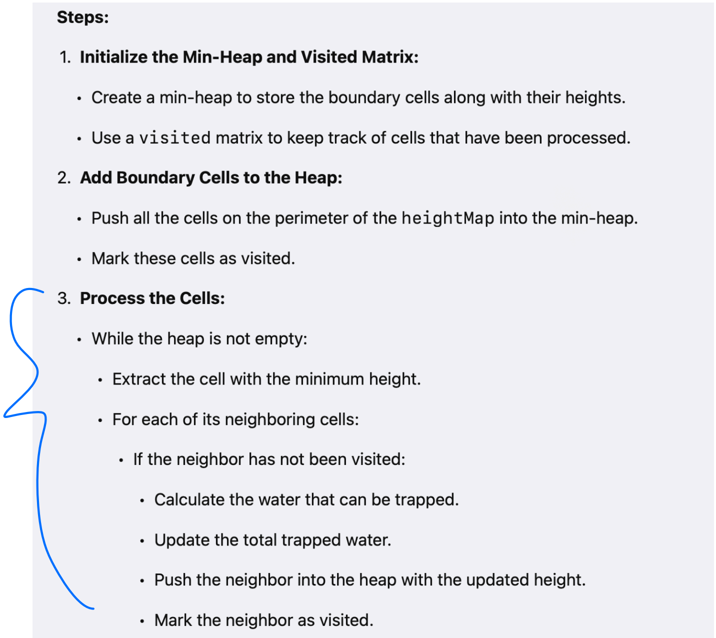


→ O(m×n×h) ,

② GPT4o

1	4	3	1	3	2
3	2	1	3	2	4
2	3	3	2	3	1

priority queue (Min-Heap) & BFS

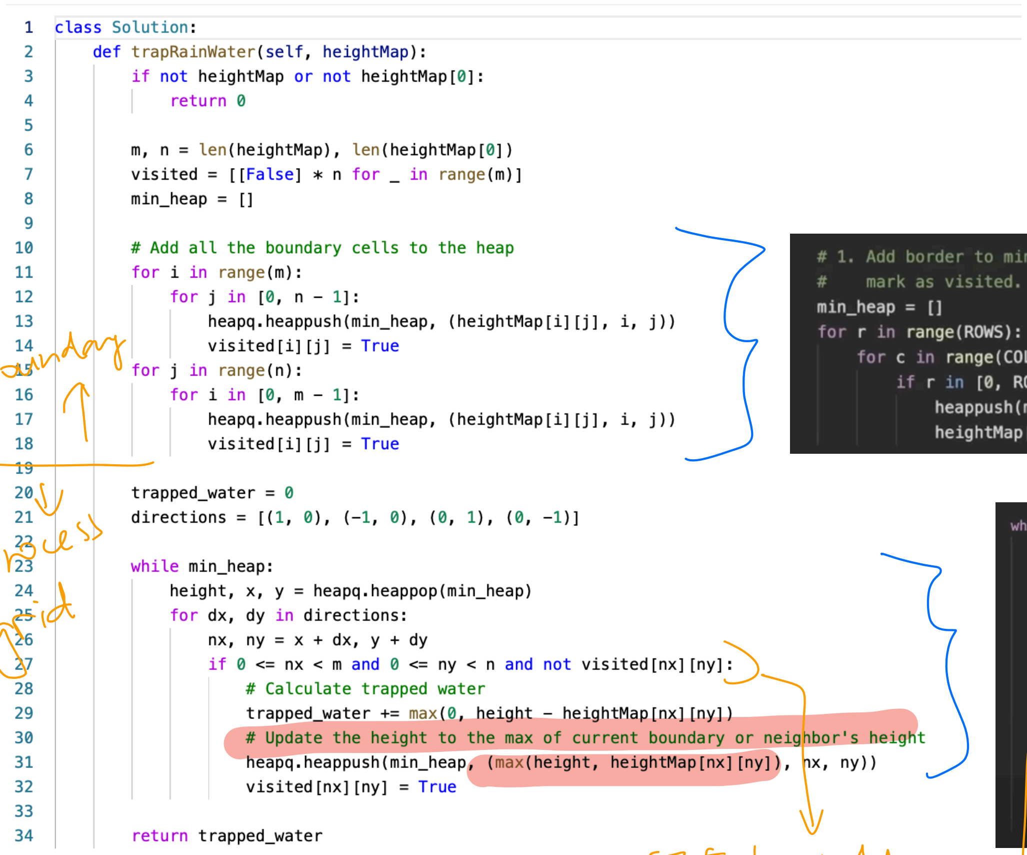


Steps:

1. **Initialize the Min-Heap and Visited Matrix:**
   - Create a min-heap to store the boundary cells along with their heights.
   - Use a `visited` matrix to keep track of cells that have been processed.
2. **Add Boundary Cells to the Heap:**
   - Push all the cells on the perimeter of the `heightMap` into the min-heap.
   - Mark these cells as visited.
3. **Process the Cells:**
   - While the heap is not empty:
     - Extract the cell with the minimum height.
     - For each of its neighboring cells:
       - If the neighbor has not been visited:
         - Calculate the water that can be trapped.
         - Update the total trapped water.
         - Push the neighbor into the heap with the updated height.
         - Mark the neighbor as visited.

可以参考 Q695

Max Area of Island



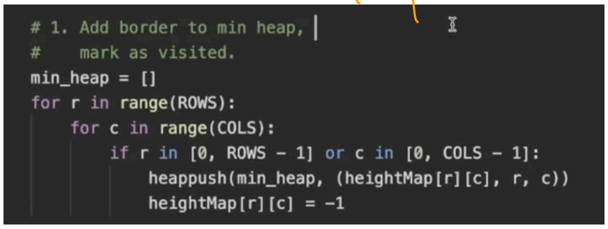
```python
class Solution:
    def trapRainWater(self, heightMap):
        if not heightMap or not heightMap[0]:
            return 0

        m, n = len(heightMap), len(heightMap[0])
        visited = [[False] * n for _ in range(m)]
        min_heap = []

        # Add all the boundary cells to the heap
        for i in range(m):
            for j in [0, n - 1]:
                heapq.heappush(min_heap, (heightMap[i][j], i, j))
                visited[i][j] = True
        for j in range(n):
            for i in [0, m - 1]:
                heapq.heappush(min_heap, (heightMap[i][j], i, j))
                visited[i][j] = True

        trapped_water = 0
        directions = [(1, 0), (-1, 0), (0, 1), (0, -1)]

        while min_heap:
            height, x, y = heapq.heappop(min_heap)
            for dx, dy in directions:
                nx, ny = x + dx, y + dy
                if 0 <= nx < m and 0 <= ny < n and not visited[nx][ny]:
                    # Calculate trapped water
                    trapped_water += max(0, height - heightMap[nx][ny])
                    # Update the height to the max of current boundary or neighbor's height
                    heapq.heappush(min_heap, (max(height, heightMap[nx][ny]), nx, ny))
                    visited[nx][ny] = True

        return trapped_water
```

boundary

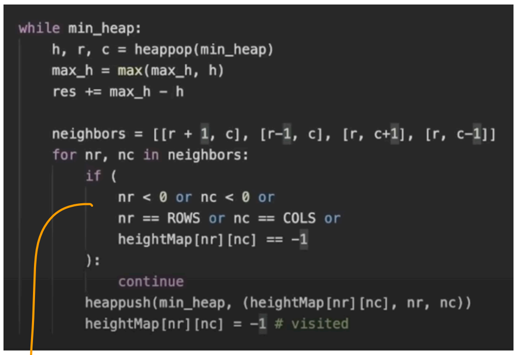
process grid

The boundary & not visited.

两个条件一起



```python
# 1. Add border to min heap,
#    mark as visited.
min_heap = []
for r in range(ROWS):
    for c in range(COLS):
        if r in [0, ROWS - 1] or c in [0, COLS - 1]:
            heappush(min_heap, (heightMap[r][c], r, c))
            heightMap[r][c] = -1
```



```python
while min_heap:
    h, r, c = heappop(min_heap)
    max_h = max(max_h, h)
    res += max_h - h

    neighbors = [[r + 1, c], [r-1, c], [r, c+1], [r, c-1]]
    for nr, nc in neighbors:
        if (
            nr < 0 or nc < 0 or
            nr == ROWS or nc == COLS or
            heightMap[nr][nc] == -1
        ):
            continue
        heappush(min_heap, (heightMap[nr][nc], nr, nc))
        heightMap[nr][nc] = -1 # visited
```

NOT DO in condition

整体而言没变. 思路更efficient.